PROMETEUS

preterm brain-oxygenation and metabolic eu-sensing

D5.3

Communication Protocol

Partner:	DAVE
Lead Author:	Scarpa (DAVE)
Version: F: final; D: draft; RD: revised draft	F
Date:	22/11/2024

European Innovation Council

Funded by the European Union

This project has received funding from the European Union's Horizon Europe research and innovation programme under grant agreement No 101099093

Grant Agreement	101099093
Acronym	Prometeus
Project full title	Preterm Brain-Oxygenation and Metabolic EU-Sensing: Feed the Brain

Deliverable	**D5.3**
Deliverable Name	**Communication Protocol**
Nature of deliverable	R: Document, Report
Dissemination level	SEN (Sensitive – limited under the conditions of the Grant Agreement)
Scheduled delivery date	30/11/2024
Actual delivery date	22/11/2024

Prepared by	Alberto Scarpa (DAVE)
Reviewed by	Manuele Papais (DAVE)
Verified by	Sabrina Brigadoi (UniPD)

History of Changes

Revision	Date (dd/mm/yyyy)	Author	Changes	Status (Draft/In review/Submitted)
V 1	28/10/2024	Alberto Scarpa (DAVE)	First draft	Draft
V 1.1	04/11/2024	Alberto Scarpa, Manuele Papais (DAVE)	General improvements	Submitted to partners
V 1.2	15/11/2024	Alberto Scarpa, Manuele Papais (DAVE)	Formatting and UI updates	Submitted to partners
V 1.3	21/11/2024	Alberto Scarpa (DAVE)	Formatting	Submitted to partners
V 2	22/11/2024	Alberto Scarpa (DAVE), Sabrina Brigadoi, Marta Pozza (UniPD)		Final version

Table of contents

1 Executive Summary

This document is the deliverable D5.3 "Communication Protocol" of the Prometeus Project.

The aim is to present the description of the specification and implementation of the communication protocols between the cloud-based app and the other component of the Prometeus project within the activities of Task 5.1 and Task 5.2 of WP5.

The contents of this document are:

I. High level HW and SW architecture of the Prometeus cloud app
II. Protocol specification

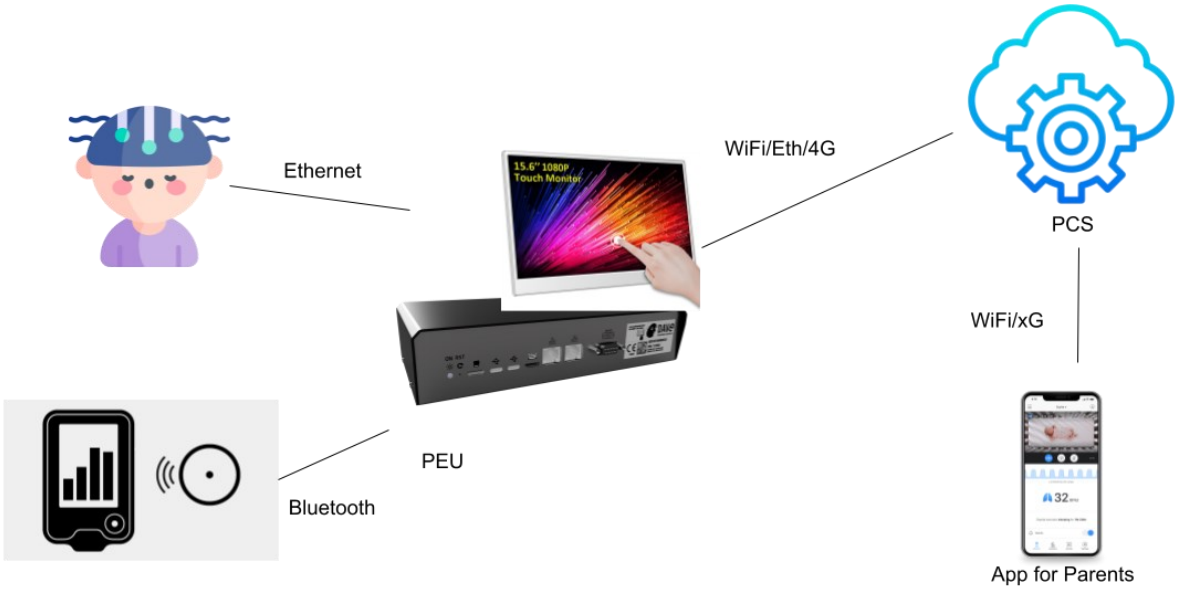
2 List of abbreviations

BR	Business Requirement
UR	User Requirement
SR	System Requirement
HCP	Health Care Professional
CMM	Continuous Metabolic Monitoring
NICU	Neonatal Intensive Care Unit
NCA	Nutritional Clinical Advisor
PEU	Prometeus Edge Unit
PCS	Prometeus Cloud Service

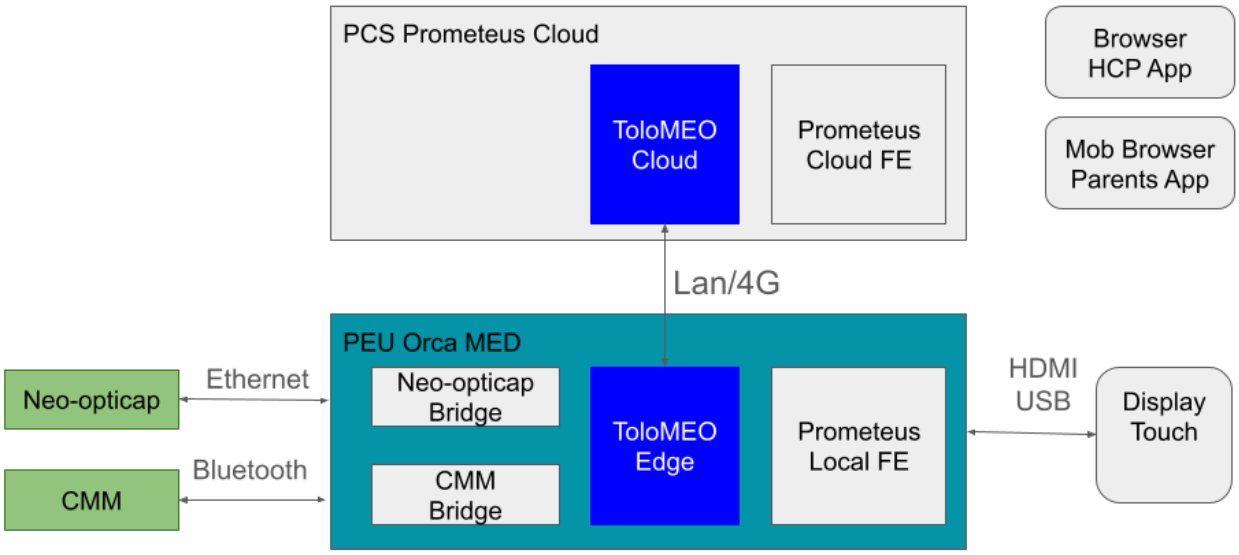
3 Architecture

3.1 High level architecture

The high-level architecture proposed in D5.1 was confirmed. The monitoring neo-opticap and CMM devices are connected via Ethernet/REST API and via Bluetooth with the PEU. The PEU integrates the connection with the display and is connected to the PCS via WiFi, Eth or 4G. The App for Parents is connected with the PCS.



On the software side, the PEU and PCS are connected using ToloMEO IoT libraries provided by DAVE Embedded Systems. On the PEU there are two software bridges for the connection with the Prometeus devices.



3.2 Components

3.2.1 Neo-opticap sensor

The neo-opticap is a cap for the baby. The cap measures 6 parameters per each of the 6 brain regions:

- FR: frontal right,
- FL: frontal left
- BR: back right
- BL: back left
- R: right
- L: left

Raw signal from the cap is elaborated with a custom electronic system that sends elaborated data to the PEU. It communicates via REST API over the network with the PEU. The sensor samples the 6x6 measurements at least every minute and after processing sends them to the PEU every 5 mins.

3.2.2 CMM sensor

The CMM is a patch sensor worn by the baby. It measures 3 analytes: glucose, lactate, beta hydroxybutyrate. The sensor communicates via Bluetooth with the PEU. The sensor generates a sample every 5 mins.

3.2.3 PEU (Prometeus Edge Unit)

PEU (Prometeus Edge Unit): field device which is in proximity to the sensing devices and the patients and interoperates with the sensors through their docking stations, manages the Controller execution, and interoperates with field equipment and other ICT platforms.

3.2.4 PCS (Prometeus Cloud Service)

PCS (Prometeus Cloud Service) is the cloud platform of the Prometeus system. It provides functionalities like user management, data storage, frontend for HCP, and frontend for parents.

3.2.5 Metabolic Model and NCA (Nutritional Clinical Advisor)

The Metabolic Model is a software component that receives the data from sensors and the information about the baby as input. Internally the model creates the digital avatar of the metabolism of the baby. The model is used to simulate the effect of active actions (like nutrition) on the baby.

NCA is the algorithm responsible to find the best nutrition for the baby based on the current health status and using the metabolic model of the baby to forecast his response.

3.2.6 Local App for HCP

HCP has a local interface to interact with the Prometeus system. The local interface guarantees the system can work even if the internet connection is not available. With the local app, the HCP monitors the information coming from the sensors and receives suggestions from the NCA (in this case, NCA shall run locally). The local app shows the alarms from the whole system to the HCP.

3.2.7 Webapp for HCP

The WebApp is the Prometeus' front-end UI for HCP users both with local and remote sessions. It is used to review old monitoring sessions and to perform advanced data analysis.

3.2.8 WebApp for Parents

The webapp for parents is used by the parents to monitor the overall status of their baby while he/she is in the NICU. The webapp for parents may be the same one used for HCP with different levels of access/functional availability.

4 Demo

A demo of the Prometeus project is always available at the following link:

https://prometeus.tolomeo.io

5 Protocols

The communication protocols between the PEU and the other components are designed to guarantee the interoperability of all components. The main design principles are:

- interoperability: the connection between the device might be as decoupled as possible
- standard: the protocols should follow the standards and best practices in the market
- robustness: the protocols should be robust to every type of errors

Based on the Prometeus architecture there are the following communication protocols between the components:

- Neo-opticap <-> PEU
- CMM <-> PEU
- NCA <-> PEU
- PEU <-> PCS

6 Neo-opticap <-> PEU Protocol

This section outlines the message exchange format and commands for communication between the neo-opticap sensor and the PEU edge device using a REST protocol over an ethernet physical connection. The protocol supports sending data, commands and errors from the neo-opticap sensor to the PEU edge device and vice versa.

This protocol ensures robust and reliable communication between the sensor and the edge device, enabling efficient data exchange and command execution. The standardised message format is used for interoperability.

6.1 Communication Protocol

The physical communication between PEU and the neo-opticap sensor uses a standard ethernet link. This protocol guarantees high throughputs and high reliability.

The application level communication uses REST and websocket server on the PEU. The sensor device can use the endpoints provided by the PEU for the connection.

6.2 Base URL

The PEU server runs on a local network with a predefined IP address 192.168.1.10 and a base url:

https://peu.prometeus.tolomeo.io/v1/

6.3 REST Endpoint

6.3.1 Post sensor data

- **Description**: Allow the sensor to send its data to the server.
- **Endpoint**: `/sensors/{sensor_id}/data`
- **Method**: `POST`
- **Path Parameters**:

  - `sensor_id` (string): Unique identifier of the sensor. We will use "neopticap".
- **Request Body**:
  - `timestamp` (string, ISO 8601): Time the data was collected.
  - `value` (array of arrays): 6x6 matrix of float values. The convention is a row for regions in this order: FR, R, BR, FL, L, BL. In the row the values are StO2, O2Hb, Hhb, Bfi, TotHb.
  - `accuracy` (array of arrays) optional: 6x6 matrix of int representing the accuracy of the measure. The convention is a row for regions in this order: FR, R, BR, FL, L, BL. In the row the values are StO2, O2Hb, Hhb, Bfi, TotHb. The value are
    - 0: bad accuracy
    - 1: medium accuracy
    - 2 good accuracy
- **Example Request Body**: (json)

```
{
  "timestamp": "2023-11-04T15:30:00Z",
  "matrix": [
    [1.1, 2.2, 3.3, 4.4, 5.5, 6.6],
    [7.1, 8.2, 9.3, 10.4, 11.5, 12.6],
    [13.1, 14.2, 15.3, 16.4, 17.5, 18.6],
    [19.1, 20.2, 21.3, 22.4, 23.5, 24.6],
    [25.1, 26.2, 27.3, 28.4, 29.5, 30.6],
    [31.1, 32.2, 33.3, 34.4, 35.5, 36.6]
  ]
}
```

- **Response**:
  - **Status Code**: `201 Created`

Body:(json)

```
{
  "sensor_id": "string",
  "status": "success",
  "message": "Data received successfully."
}
```

- **Errors**:
  - `400 Bad Request`: If the data is improperly formatted or missing.
  - `404 Not Found`: If `sensor_id` does not exist.

6.3.2 Get Sensor Configuration

- **Description**: Retrieve the current configuration of the sensor.
- **Endpoint**: `/sensors/{sensor_id}/config`
- **Method**: `GET`
- **Path Parameters**:
  - `sensor_id` (string): Unique identifier of the sensor.
- **Response**:
  - **Status Code**: `200 OK`
  - **Body**: `parameters of "config" json will be updated during the development of the project`

    Body:(json)

```
{
  "sensor_id": "string",
  "config": {
    "sampling_rate": "5",

  }
}
```

- **Errors**:
  - `404 Not Found`: If `sensor_id` does not exist.

6.3.3 Post File with bulk raw data

We have an additional endpoint to upload raw bulk data useful for the debug of the sensor.

This endpoint allows for efficient data uploads by enabling the sensor or an associated system to upload a CSV file. This method is optimal for batch processing or synchronizing large amounts of sensor data collected offline. The server will parse and process the CSV file upon upload, storing the data as needed.

- **Description**: Allows the sensor to upload a CSV file containing raw data for debugging.
- **Endpoint**: `/sensors/{sensor_id}/upload-raw-data`
- **Method**: `POST`
- **Path Parameters**:
  - `sensor_id` (string): Unique identifier of the sensor.
- **Headers**:
  - `Content-Type`: `multipart/form-data`
- **Request Body**:
  - **File**: A CSV file uploaded with key `file`.
  - **CSV File Format**:
    - **Columns**:
      - `timestamp`: ISO 8601 formatted timestamp.
      - `data` .

Example CSV File: (csv)

```
timestamp,data
2023-11-04T15:30:00Z,ABC43CB234
2023-11-04T15:35:00Z,2395E62F947
...
```

- **Response**:
  - **Status Code**: `201 Created`

Body: (json)

```
{
  "sensor_id": "string",
  "status": "success",
  "message": "CSV file uploaded and processed successfully."
}
```

- **Errors**:
  - `400 Bad Request`: If the file format is invalid or the file is missing required fields.
  - `415 Unsupported Media Type`: If the file is not a CSV.
  - `413 Payload Too Large`: If the file size exceeds the server's limit.
  - `404 Not Found`: If `sensor_id` does not exist.

6.4 Web socket

To support real-time bidirectional communication between the server and the sensor, we have an endpoint to establish a WebSocket connection. This WebSocket connection can enable the sensor to send live data or receive commands and configuration changes from the server without needing separate HTTP requests for each interaction.

This WebSocket endpoint facilitates real-time, continuous communication between the sensor and the server, allowing for instantaneous data transmission and control without needing individual HTTP requests. This is especially useful for high-frequency data transfer and immediate command responsiveness.

- **Description**: Establish a WebSocket connection for real-time, bidirectional communication between the PEU and the sensor.
- **Endpoint**: `/sensors/{sensor_id}/ws`
- **Method**: `GET` (used to initiate the WebSocket handshake)
- **Path Parameters**:
  - `sensor_id` (string): Unique identifier of the sensor.
- **WebSocket Events**:
  - `data`:
    - **Direction**: Sensor → Server
    - **Description**: Sensor sends real-time data in the form of individual data

points or matrices.

Payload:(json)

```
{
  "event": "data",
  "timestamp": "ISO 8601 string",
  "value": "float" or "6x6 matrix of floats"
}
```

- **command**:
  - **Direction**: Server → Sensor
  - **Description**: Server sends commands to the sensor for real-time control (e.g., "start", "stop", "calibrate").

Payload: (json)

```
{
  "event": "command",
  "command": "string",
  "parameters": {
    "key": "value"
  }
}
```

  - Available commands are:
    - **Start Measurement**
      - **Description**: Instructs the sensor to start taking measurements.
      - **Payload**: None
    - **Stop Measurement**
      - **Description**: Instructs the sensor to stop taking measurements.
      - **Payload**: None
    - **Set Measurement Interval**
      - **Description**: Sets the interval at which the sensor takes measurements.
      - **Payload**: Interval time in seconds
    - **Request Sensor Status**
      - **Description**: Requests the current status of the sensor.
      - **Payload**: None
- **config**:
  - **Direction**: Server → Sensor
  - **Description**: Server sends configuration changes to the sensor (e.g., updating sampling rate or threshold).

Payload:(json)

```
{
  "event": "config",
  "sampling_rate": "float",
  "threshold": "float",
  "mode": "string"
}
```

- **Expected WebSocket Connection Flow**:
  - **Handshake**: The WebSocket connection is established with a `GET` request to `/sensors/{sensor_id}/ws`.
  - **Data Transmission**:
    - The sensor sends data events to the server at intervals or based on conditions.
    - The server can send commands or configuration updates to the sensor as needed.
- **Response**:
  - **Status Code**: `101 Switching Protocols` (WebSocket connection is successfully established).
- **Errors**:
  - `404 Not Found`: If `sensor_id` does not exist.
  - `403 Forbidden`: If the sensor does not have permission to establish a WebSocket connection.
  - `500 Internal Server Error`: If there is an error during connection setup.

6.5 Error Responses

Common errors across endpoints:

`400 Bad Request`: Malformed request syntax or invalid parameters.
json

```
{
  "error": "string",
  "message": "string"
}
```

`404 Not Found`: Resource not found.
json

```
{
  "error": "Resource Not Found",
  "message": "The specified sensor or resource does not exist."
}
```

500 Internal Server Error: Server-side error.

json

```
{
  "error": "Internal Server Error",
  "message": "An error occurred on the server. Please try again later."
}
```

6.6 Security

- **Authentication**: Use API keys in the headers (`Authorization: Bearer <API_KEY>`).
- **HTTPS Required**: Ensure all requests are made over HTTPS.

7 CMM <-> PEU Protocol

This section outlines the message exchange format and commands for communication between the CMM sensor and the PEU edge device using the Bluetooth low energy. The protocol supports sending data, commands and errors from the CMM sensor to the edge device and vice versa.

This protocol ensures robust and reliable communication between the sensor and the edge device, enabling efficient data exchange and command execution. The standardised message format and checksum mechanism help maintain data integrity and streamline communication processes.

7.1 Physical layer

The CMM sensor and the PEU are connected using Bluetooth low energy radio protocol.

The PEU device uses a AzureWave AW-CM276MA-SUR (datasheet) module compliant with Bluetooth 2.1 and 3.0 + Enhanced Data Rate (EDR) + BT 5.3.

1.3.3 Bluetooth

<table>
<tr><th>Features</th><th>Description</th></tr>
<tr><td>Bluetooth Standard</td><td>Bluetooth 2.1 and 3.0+Enhanced Data Rate (EDR) + BT 5.3</td></tr>
<tr><td>Bluetooth VID/PID</td><td>1286/204E</td></tr>
<tr><td>Frequency Rage</td><td>2402~2480MHz</td></tr>
<tr><td>Modulation</td><td>GFSK (1Mbps), Π/4 DQPSK (2Mbps) and 8DPSK (3Mbps)</td></tr>
<tr><td>Output Power</td><td>
<table>
<tr><th></th><th>Min</th><th>Typ</th><th>Max</th><th>Unit</th></tr>
<tr><td>BDR</td><td>0</td><td>2</td><td>4</td><td>dBm</td></tr>
<tr><td>EDR</td><td>0</td><td>2</td><td>4</td><td>dBm</td></tr>
<tr><td>BLE</td><td>0</td><td>2</td><td>4</td><td>dBm</td></tr>
</table>
</td></tr>
<tr><td>Receiver Sensitivity</td><td>BER < 0.1%
<table>
<tr><th></th><th>Min</th><th>Typ</th><th>Max</th><th>Unit</th></tr>
<tr><td>BDR</td><td></td><td>-83</td><td></td><td>dBm</td></tr>
</table>
</td></tr>
</table>

7.2 BLE communication structure

BLE communication is organised around the Generic Attribute Profile (GATT). The protocol uses GATT services and characteristics to define how data is exchanged between devices.

The GATT structure is:

- **Service**: A collection of characteristics and relationships to other services that encapsulate the behaviour of part of a device.
- **Characteristic**: A value used in a service, which includes properties, a value, and optional descriptors.

7.2.1 Services and Characteristics

The service is configured with:

- **UUID**: `0000XXXX-0000-1000-8000-00805F9B34FB`

7.2.1.1 Characteristics

1. **Sensor Data Characteristic**
   - **UUID**: `0000XXXX-0001-1000-8000-00805F9B34FB`
   - **Properties**: Notify, Read
   - **Description**: Used by the sensor to send data to the edge device.
2. **Command Characteristic**
   - **UUID**: `0000XXXX-0002-1000-8000-00805F9B34FB`
   - **Properties**: Write
   - **Description**: Used by the edge device to send commands to the sensor.

7.2.1.2 Message Format

Messages exchanged via BLE are encapsulated in the value field of characteristics.

7.2.1.3 Sensor Data Message

- **Format**: `[Data Type] [Data Length] [Data Payload] [Checksum]`
  - **Data Type**: 1 byte (indicates the type of sensor data, e.g., glucose)
  - **Data Length**: 1 byte (length of the data payload)
  - **Data Payload**: Variable length (actual sensor data)
  - **Checksum**: 1 byte (simple checksum for error-checking)

7.2.1.4 Command Message

- **Format**: `[Command ID] [Command Payload Length] [Command Payload] [Checksum]`
  - **Command ID**: 1 byte (identifies the specific command)
  - **Command Payload Length**: 1 byte (length of the command payload)
  - **Command Payload**: Variable length (command-specific data)
  - **Checksum**: 1 byte (simple checksum for error-checking)

7.2.1.5 Supported Commands

1. **Start Measurement**
   - **Command ID**: 0xA1
   - **Description**: Instructs the sensor to start taking measurements.
   - **Payload**: None
2. **Stop Measurement**
   - **Command ID**: 0xA2
   - **Description**: Instructs the sensor to stop taking measurements.
   - **Payload**: None

3. **Set Measurement Interval**
    - **Command ID**: 0xA3
    - **Description**: Sets the interval at which the sensor takes measurements.
    - **Payload**: Interval time in seconds (1 byte)
4. **Request Sensor Status**
    - **Command ID**: 0xA4
    - **Description**: Requests the current status of the sensor.
    - **Payload**: None

7.2.1.6 Checksum Calculation

The checksum is calculated by summing all bytes of the message (excluding the start and end bytes for BLE, which is the characteristic value itself) and taking the least significant byte of the result.

7.2.2 Example Messages

7.2.2.1 Sensor Data Message Example

- **Description**: A message from the sensor sending temperature data.
- **Format**: `[Data Type] [Data Length] [Data Payload] [Checksum]`
- **Example**: `0x01 0x03 0x1E 0x2A 0x35`
    - Data Type: 0x01 (Temperature)
    - Data Length: 0x03 (3 bytes of data)
    - Data Payload: 0x1E 0x2A 0x35 (example temperature data)
    - Checksum: Calculated as (0x01 + 0x03 + 0x1E + 0x2A + 0x35) & 0xFF

7.2.2.2 Command Message Example

- **Description**: A message from the edge device to start measurements.
- **Format**: `[Command ID] [Command Payload Length] [Command Payload] [Checksum]`
- **Example**: `0xA1 0x00 0xA1`
    - Command ID: 0xA1 (Start Measurement)
    - Command Payload Length: 0x00 (no payload)
    - Checksum: Calculated as (0xA1 + 0x00) & 0xFF

7.3 Error Handling

- **Invalid Checksum**: If the checksum does not match, the message is discarded and an error message is logged.
- **Unsupported Commands**: If an unsupported command is received, the sensor responds with an error message indicating the invalid command.

8 NCA <-> PEU Protocol

The NCA (Nutritional Clinical Advisor) and the mathematical model of the baby metabolism are under development by the partners of WP 3 and WP4.

Based on the complexity of the final algorithms and on the memory usage it can be executed on the PEU or on the PCS. To be independent of the running setup the communication between the PEU and the NCA will be based on REST API and websocket.

8.1 Web Socket

- To support real-time bidirectional communication between the server and the NCA we have an endpoint to establish a WebSocket connection. This WebSocket connection can enable the PEU to send live sensor data to the NCA and to receive the advice for the nutritional feed.**Description**: Establish a WebSocket connection for real-time, bidirectional communication between the PEU and the NCA.
- **Endpoint**: `/nca/{nca_id}/ws`
- **Method**: `GET` (used to initiate the WebSocket handshake)
- **Path Parameters**:
  - `nca_id` (string): Unique identifier of the nca algorithm.
- **WebSocket Events**:
  - **`data`**:
    - **Direction**: PEU → NCA
    - **Description**: PEU sends real-time data received by the sensors.

    Payload:(json)

```
{
  "event": "data",
  "timestamp": "ISO 8601 string",
  "sensor-type": "neopticap",
  "value": "float" or "6x6 matrix of floats"
}
```

  - **`command`**:
    - **Direction**: PEU → NCA
    - **Description**: PEU sends commands to the NCA for real-time control (e.g., "start", "stop", "calibrate").

    Payload: (json)

```
{
  "event": "command",
  "command": "string",
  "parameters": {
    "key": "value"
  }
}
```

    - Available commands are:
      - **Start Prediction**
        - **Description**: Instructs the sensor to start taking measurements.
        - **Payload**: None
      - **Stop Prediction**
        - **Description**: Instructs the sensor to stop taking

measurements.
    - **Payload**: None
- **Set Patient info**
    - **Description**: Sets the information about the patient.
    - **Payload**: gender, weight, height, gestational age.
- **Request NCA Status**
    - **Description**: Requests the current status of the NCA.
    - **Payload**: None

- `config`:
    - **Direction**: PEU → NCA
    - **Description**: PEU sends configuration changes to the NCA (e.g., thresholds, internal parameters, models).

Payload:(json)

```
{
  "event": "config",
  "sampling_rate": "float",
  "threshold": "float",
  "mode": "string"
}
```

- `advice`:
    - **Direction**: NCA → PEU
    - **Description**: NCA sends advice to the PEU.

Payload: (json)

```
{
  "event": "advice",
  "description": "adjust infusion",
  "advice": {
    "key": "value"
  }
}
```

    - **Advice**: keys are the nutrition components to change in the infusion.

- **Expected WebSocket Connection Flow**:
    - **Handshake**: The WebSocket connection is established with a `GET` request to `/nca/{nca_id}/ws`.
    - **Data Transmission**:
        - The PEU sends data events to the NCA at intervals or based on conditions.
        - The PEU can send commands or configuration updates to the NCA as needed.
        - The NCA sends nutritional advice as needed.
- **Response**:
    - **Status Code**: `101 Switching Protocols` (WebSocket connection is successfully established).
- **Errors**:

- `404 Not Found`: If `nca_id` does not exist.
- `403 Forbidden`: If the nca does not have permission to establish a WebSocket connection.
- `500 Internal Server Error`: If there is an error during connection setup.

9 PEU <-> PCS Protocol

The PEU and PCS are connected and synchronised using the IoT platform named ToloMEO developed by DAVE. The detailed documentation of the platform is available on the website tolomeo.io .

The communication protocol is based on MQTT and on REST protocols.

The MQTT protocol is used to manage data from the sensors.

10 The REST protocol is used for the login of the users and to manage the patient information.USB Protocol

The PEU supports data exchange via USB storage.

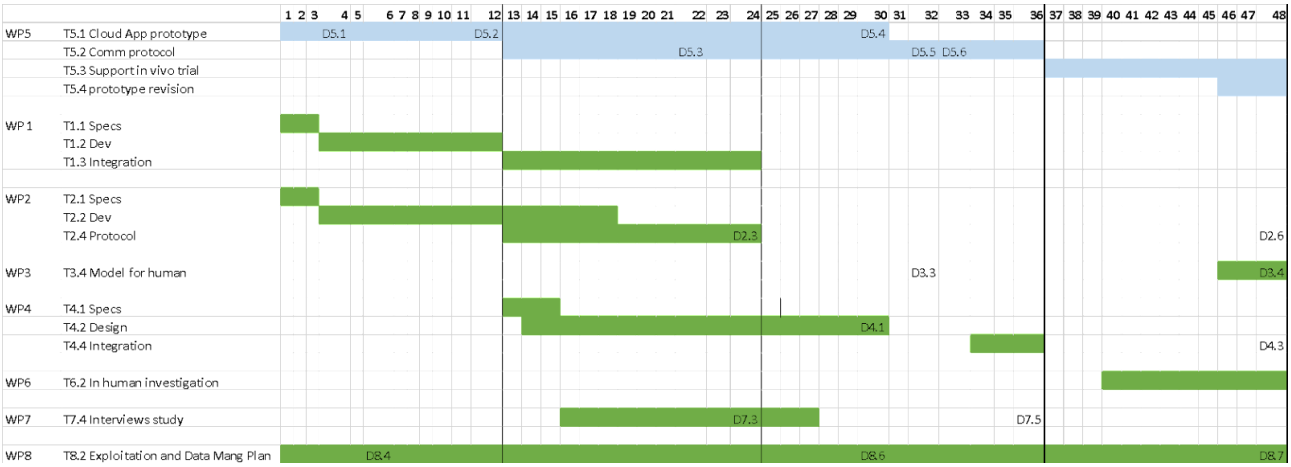
When a USB storage is inserted into the PEU the firmware searches for one of the following files to start the corresponding activity.

File name	**Action**	**Content**	**Output**
backup.txt	create a backup of all local data	signature	folder with name"backup_YYYYMMDD_HHmm" with inside the files: config, data, logs
update	update the fw of the PEU unit. This action works only if the USB is inserted at boot time.	signed binary update file	–
reset	delete all data and reset device to factory configuration	signature	–
console			kill FE and open console

11 Development Plan

The description of the communication protocol is well described in this document and part of the development is already started on the PEU side.

The following Gantt shows in blue the tasks and deliverables of WP5 and in green its direct related tasks and deliverables.



The sensors (WP1 and WP2) and the NCA (WP3 and WP4) are under development. An integration test will be performed as soon as the sensors will be available.