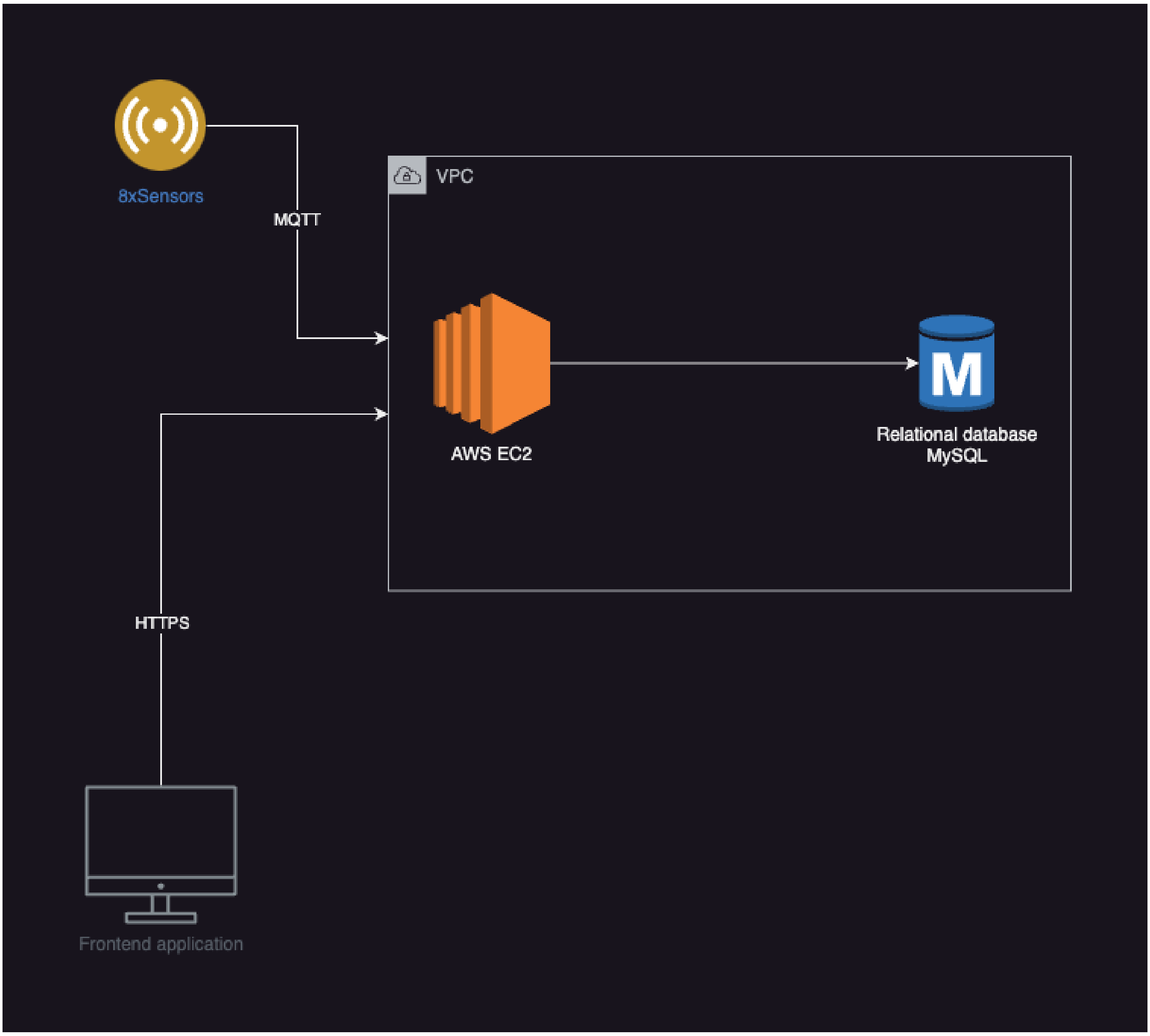
PaceB application documentation



API Documentation

Introduction

The EC2 application utilizes the PM2 process manager to run multiple applications concurrently. The following applications are included:

1. **Iot (Express JS App)**
2. **Iot Mqtt (Node.js MQTT Listener)**
3. **Gait (Python Flask Application for Gait Parameter Calculation)**
4. **reportGenerator (Python Flask Application for Report PDF Generation)**

This documentation provides an overview of the RESTful API endpoints exposed by the EC2 application and their respective functionalities.

Iot (Node.js Application)

The Iot application is built using Node.js (v16.17.0) and follows a token-based authentication mechanism using Firebase SDK. It runs on port 5004.

Endpoints

The root URL for accessing the Iot application is `http://<server_ip>:5004/`.

Create a Patient Record

- URL: `/patients`
- Method: `POST`
- Description: Creates a new patient record.
- Request Body:

  ```
  {
    "doctor_id": "<doctor_id>",
    "patient_name": "<patient_name>",
    "sex": "<gender>",
    "dob": "<date_of_birth>",
    "pincode": "<patient_pincode>",
    "age": <patient_age>
  }
  ```

- Response:
  - HTTP Status Code: 200 (OK)
  - Body:

    ```
    {
      "id": "<patient_id>",
      "doctor_id": "<doctor_id>",
      "patient_name": "<patient_name>",
      "sex": "<gender>",
      "dob": "<date_of_birth>",
      "pincode": "<patient_pincode>",
      "age": <patient_age>
    }
    ```

- Error Response:
  - HTTP Status Code: 500 (Internal Server Error)
  - Body:

    ```
    {
      "error": "<error_message>"
    }
    ```

Retrieve Patients by Doctor

- URL: `/patients/:doctor_id`
- Method: `GET`
- Description: Retrieves all patients associated with a specific doctor.
- Path Parameter:
  - `doctor_id`: The ID of the doctor.
- Response:
  - HTTP Status Code: 200 (OK)
  - Body:

    ```
    [
      {
        "id": "<patient_id>",
        "doctor_id": "<doctor_id>",
        "patient_name": "<patient_name>",
        "sex": "<gender>",
        "dob": "<date_of_birth>",
        "pincode": "<patient_pincode>",
        "age": <patient_age>
      },
      ...
    ]
    ```

- Error Response:
  - HTTP Status Code: 500 (Internal Server Error)
  - Body:

    ```
    {
      "error": "<error_message>"
    }
    ```

Medical History

The `/medicalHistory` endpoint allows creating and retrieving medical history records.

Create a Medical History Record

- URL: `/medicalHistory`

- Method: `POST`

- Description: Creates a new medical history record.

- Request Body:

```
{
  "patient_condition": "<condition>",
  "patient_symptom": "<symptom>",
  "patientId": "<patient_id>"
}
```

- Response:
    - HTTP Status Code: 200 (OK)
    - Body:

    ```
    {
      "id": "<record_id>",
      "patient_condition": "<condition>",
      "patient_symptom": "<symptom>",
      "patientId": "<patient_id>"
    }

    - Error Response:
    ```

    - HTTP Status Code: 500 (Internal Server Error)
    - Body:

    ```
    {
      "error": "<error_message>"
    }
    ```

Retrieve Medical Records by Patient

- URL: `/medicalHistory/:patient_id`
- Method: `GET`
- Description: Retrieves all medical records associated with a specific patient.
- Path Parameter:
    - `patient_id`: The ID of the patient.
- Response:
    - HTTP Status Code: 200 (OK)
    - Body:

    ```
    {
      "patient": {
        "id": "<patient_id>",
        "doctor_id": "<doctor_id>",
        "patient_name": "<patient_name>",
    ```

```
    "sex": "<gender>",
    "dob": "<date_of_birth>",
    "pincode": "<patient_pincode>",
    "age": <patient_age>
  },
  "medical_records": [
    {
      "id": "<record_id>",
      "patient_condition": "<condition>",
      "patient_symptom": "<symptom>",
      "patientId": "<patient_id>"
    },
    ...
  ]
}
```

- Error Response:
  - HTTP Status Code: 500 (Internal Server Error)
  - Body:

```
{
  "error": "<error_message>"
}
```

Shoe Registry

The `/shoeRegistry` endpoint allows managing the association and registration of shoe devices.

Associate Shoe with Doctor

- URL: `/shoeRegistry/associate`
- Method: `POST`
- Description: Associates a shoe with a doctor.
- Request Body:

```
{
  "doctor_id": "<doctor_id>",
  "device_id": "<device_id>"
}
```

- Response:
  - HTTP Status Code: 200 (OK)
  - Body:

```
{
  "doctor_id": "<doctor_id>",
  "device_id": "<device_id>"
}
```

- Error Response:
    - HTTP Status Code: 500 (Internal Server Error)
    - Body:

    ```
    {
      "error": "<error_message>"
    }
    ```

Register a Shoe Device

- URL: `/shoeRegistry/register`
- Method: `POST`
- Description: Registers a shoe device in the shoe registry.
- Request Body:

```
{
  "device_id": "<device_id>"
}
```

- Response:
    - HTTP Status Code: 200 (OK)
    - Body:

    ```
    {
      "device_id": "<device_id>"
    }
    ```

- Error Responses:
    - HTTP Status Code: 500 (Internal Server Error)
    - Body:

    ```
    {
      "error": "<error_message>"
    }
    ```

    - HTTP Status Code: 403 (Forbidden)

  - Body:

    ```
    {
      "message": "Device already registered"
    }
    ```

Get Unassociated Shoes

- URL: `/shoeRegistry/`
- Method: `GET`
- Description: Retrieves all shoes that are not associated with a doctor.
- Response:
  - HTTP Status Code: 200 (OK)
  - Body:

    ```
    [
      {
        "device_id": "<device_id>"
      },
      ...
    ]
    ```

- Error Response:
  - HTTP Status Code: 500 (Internal Server Error)
  - Body:

    ```
    {
      "error": "<error_message>"
    }
    ```

Get Associated Shoes by Doctor

- URL: `/shoeRegistry/:doctor_id`
- Method: `GET`
- Description: Retrieves all shoes associated with a specific doctor.
- Path Parameter:
  - `doctor_id`: The ID of the doctor.
- Response:
  - HTTP Status Code: 200 (OK

)

- Body:

```
[
  {
    "doctor_id": "<doctor_id>",
    "device_id": "<device_id>"
  },
  ...
]
```

- Error Response:
  - HTTP Status Code: 500 (Internal Server Error)
  - Body:

    ```
    {
      "error": "<error_message>"
    }
    ```

Set Shoe State

The `/setShoeState` endpoint allows updating the recording state of a medical history record ans turs the shoe ON/OFF, STATE: 1-ON, 0-OFF.

Update Recording State

- URL: `/setShoeState/`
- Method: `POST`
- Description: Updates the recording state of a medical history record.
- Request Body:

```
{
  "id": "<record_id>",
  "state": "<state>",
  "recording_id": "<recording_id>"
}
```

- Response:
  - HTTP Status Code: 200 (OK)
  - Body:

    ```
    {
      "id": "<record_id>",
      "state": "<state>",
      "recording_id": "<recording_id>"
    }
    ```

- Error Response:
  - HTTP Status Code: 500 (Internal Server Error)
  - Body:

    ```
    {
      "error": "<error_message>"
    }
    ```

Mosquitto Broker v2.0.11 Documentation

The Mosquitto broker facilitates communication using the MQTT. The broker captures data from a gait analysis shoe published on the topic `/sensor/#/#` and stores the data in an RDS database. Additionally, the provided Mosquitto configuration file is explained to help with customization and understanding.

The `mqttListener.js` as `iotMqtt` in PM2,is the main file that is used to connect to the Mosquitto Broker and subscribe to the topic `/sensor/#/#` and store the data in the RDS database.

Mosquitto Configuration File

The Mosquitto broker's configuration file controls its behavior and settings.

```
# Plain WebSockets configuration
listener 9001
protocol websockets
```

- WebSockets allow MQTT communication over a WebSocket connection. This configuration sets up a listener on port `9001` to handle MQTT over WebSockets connections.

```
# WebSockets over TLS/SSL
listener 9883
protocol websockets
```

- This configuration sets up a listener on port `9883` for secure MQTT over WebSockets connections. It enables encrypted communication using TLS/SSL.

Data Capture and Organisation

The captured data is stored in an RDS (Relational Database Service) database. A `recording ID` is generated from the frontend and associated with the captured data before being stored in the RDS database.

Database Schema Documentation

The schema consists of four tables: `medicalHistories`, `patients`, `shoeRawData`, and `shoeRegistries`. The structure and details of each table are described below.

medicalHistories Table

```
+-------------------+--------------+------+-----+---------+----------------+
| Field             | Type         | Null | Key | Default | Extra          |
+-------------------+--------------+------+-----+---------+----------------+
| id                | int          | NO   | PRI | NULL    | auto_increment |
| patient_condition | varchar(255) | YES  |     | NULL    |                |
| patient_symptom   | varchar(255) | YES  |     | NULL    |                |
| recording_id      | varchar(255) | YES  |     | NULL    |                |
| createdAt         | datetime     | NO   |     | NULL    |                |
| updatedAt         | datetime     | NO   |     | NULL    |                |
| patientId         | int          | YES  | MUL | NULL    |                |
+-------------------+--------------+------+-----+---------+----------------+
```

- `medicalHistories` table stores medical histories associated with patients.
- The `id` field is an auto-incrementing primary key.
- `patient_condition` and `patient_symptom` fields store the condition and symptom of the patient, respectively.
- `recording_id` field stores the recording ID associated with the medical history.
- `createdAt` and `updatedAt` fields store the creation and update timestamps.
- `patientId` field references the `id` field of the `patients` table.

patients Table

```
+--------------+-----------------------+------+-----+---------+----------------+
| Field        | Type                  | Null | Key | Default | Extra          |
+--------------+-----------------------+------+-----+---------+-----------
-----+
| id           | int                   | NO   | PRI | NULL    |
auto_increment |
| doctor_id    | varchar(255)          | NO   |     | NULL    |
|
| patient_name | varchar(255)          | NO   |     | NULL    |
|
| sex          | enum('M','F','T','O') | NO   |     | NULL    |
|
| dob          | datetime              | NO   |     | NULL    |
|
| pincode      | int                   | NO   |     | NULL    |
|
| age          | int                   | NO   |     | NULL    |
|
| createdAt    | datetime              | NO   |     | NULL    |
|
| updatedAt    | datetime              | NO   |     | NULL    |
|
+--------------+-----------------------+------+-----+---------+-----------
-----+
```

- `patients` table stores information about patients.
- The `id` field is an auto-incrementing primary key.
- `doctor_id` field stores the ID of the associated doctor.
- `patient_name` field stores the name of the patient.
- `sex` field stores the gender of the patient (`M` for male, `F` for female, `T` for transgender, `O` for other).
- `dob` field stores the date of birth of the patient.
- `pincode` field stores the pincode of the patient's location.
- `age` field stores the age of the patient.
- `createdAt` and `updatedAt` fields store the creation and update timestamps.

shoeRawData Table

```
+--------------+--------------+------+-----+---------+----------------+
| Field        | Type         | Null | Key | Default | Extra          |
+--------------+--------------+------+-----+---------+----------------+
| id           | int          | NO   | PRI | NULL    | auto_increment |
| time         | varchar(255) | NO   |     | NULL    |                |
| batt_percent | varchar(255) | NO   |     | NULL    |                |
| recording_id | varchar(255) | NO   |     | NULL    |                |
| shoe_side    | varchar(255) | NO   |     | NULL    |                |
| device_id    | varchar(255) | NO   |     | NULL    |                |
| vout1        | varchar(255) | NO   |     | NULL    |                |
| vout2        | varchar(255) | NO   |     | NULL    |                |
| vout3        | varchar(255) | NO   |     | NULL    |                |
| vout4        | varchar(255) | NO   |     | NULL    |                |
| vout5        | varchar(255) | NO   |     | NULL    |                |
| vout6        | varchar(255) | NO   |     | NULL    |                |
| vout7          | varchar(255) | NO    |     | NULL     |                 |
| vout8          | varchar(255) | NO    |     | NULL     |                 |
| speed          | varchar(255) | NO    |     | NULL     |                 |
+----------------+--------------+-------+-----+----------+-----------------+
```

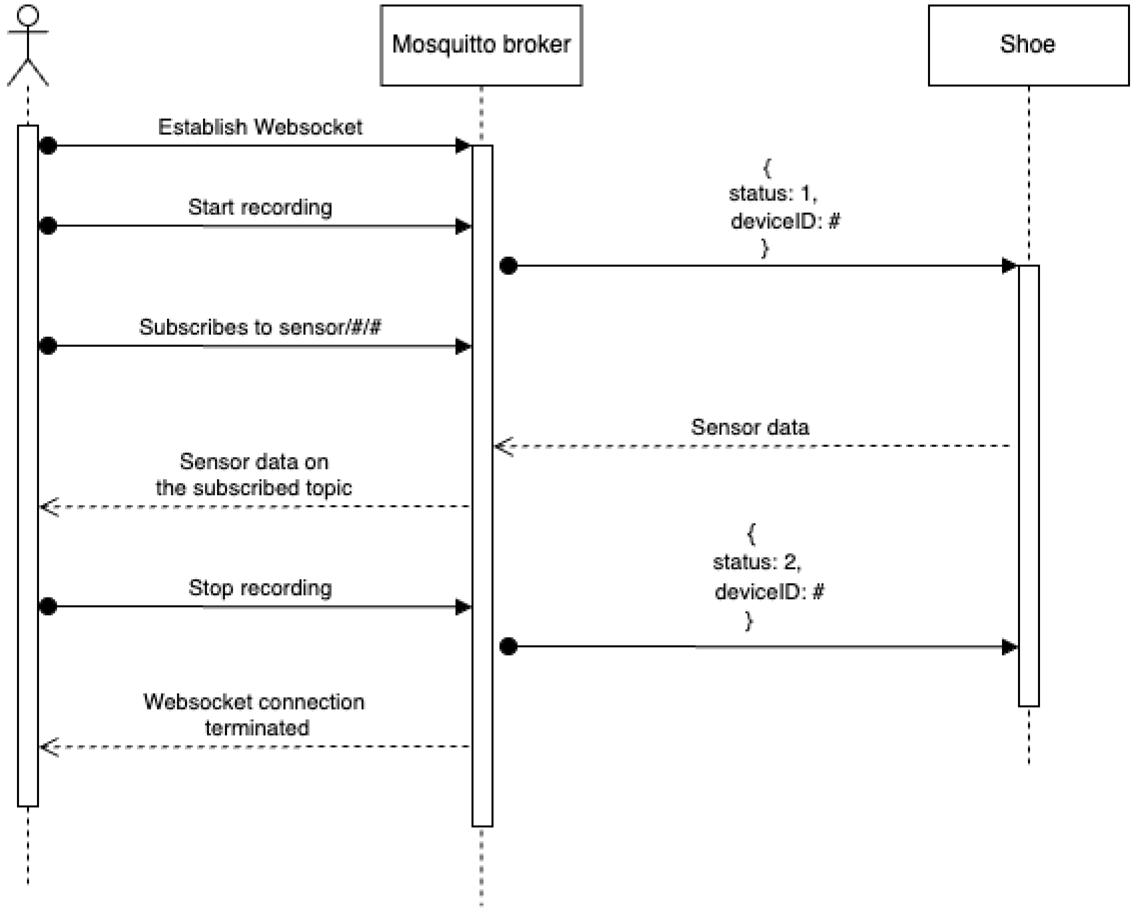
- `shoeRawData` table stores raw data captured from a gait analysis shoe.
- The `id` field is an auto-incrementing primary key.
- `time` field stores the time of the captured data.
- `batt_percent` field stores the battery percentage.
- `recording_id` field stores the recording ID associated with the captured data.
- `shoe_side` field stores the side of the shoe.
- `device_id` field stores the ID of the device.
- `vout1` to `vout8` fields store specific voltage readings.
- `speed` field stores the speed value.

shoeRegistries Table

```
+------------+--------------+------+-----+---------+----------------+
| Field      | Type         | Null | Key | Default | Extra          |
+------------+--------------+------+-----+---------+----------------+
| id         | int          | NO   | PRI | NULL    | auto_increment |
| doctor_id  | varchar(255) | YES  |     | NULL    |                |
| device_id  | varchar(255) | YES  | UNI | NULL    |                |
| createdAt  | datetime     | NO   |     | NULL    |                |
| updatedAt  | datetime     | NO   |     | NULL    |                |
+------------+--------------+------+-----+---------+----------------+
```

- `shoeRegistries` table stores information about registered shoes.
- The `id` field is an auto-incrementing primary key.
- `doctor_id` field stores the ID of the associated doctor.
- `device_id` field stores the ID of the shoe device. It is unique (`UNI` key constraint).
- `createdAt` and `updatedAt` fields store the creation and update timestamps.

Data Flow Documentation: Recording Sensor Data via MQTT



1. User Interaction:

    - The user initiates the recording process by pressing the "Start Recording" button on the frontend application.

2. Frontend Application Connection:

    - The frontend application establishes a WebSocket connection with the Mosquitto broker, enabling real-time communication.

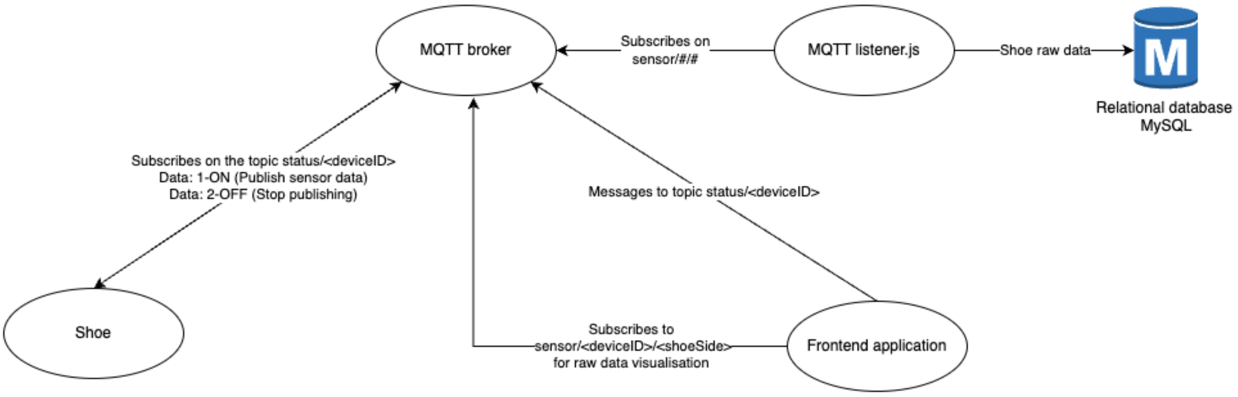
3. Start Recording Message:

    - The frontend application publishes a message on the topic "status/#" to trigger the start of data recording.
    - The "#" in the topic represents the selected shoe ID, specified by the doctor.
    - The message contains the device ID and its state (e.g., "start recording").

4. Device Activation:

    - Upon receiving the start recording message, the gait analysis device (shoe) associated with the selected ID activates and begins collecting sensor data.
    - The device posts its sensor data to the topic "sensor/#/#".

  - The first "#" in the topic represents the device ID, and the second "#" represents the shoe side (left or right).
  - Each data message includes the device ID, shoe side, and the provided recording ID.


MQTT broker
Subscribes on sensor/#/#
MQTT listener.js
Shoe raw data
Relational database MySQL
Subscribes on the topic status/<deviceID>
Data: 1-ON (Publish sensor data)
Data: 2-OFF (Stop publishing)
Messages to topic status/<deviceID>
Shoe
Subscribes to sensor/<deviceID>/<shoeSide> for raw data visualisation
Frontend application

5. Recording ID Message:
   - The device publishes a message on the topic, including the recording ID it generated.
   - This message helps in associating the subsequent sensor data with the specific recording ID.
   - The MQTT listener captures this message.
6. Sensor Data Capture:
   - The MQTT listener receives the sensor data messages posted by the device on the "sensor/#/#" topic.
   - The listener extracts the recording ID, device ID, shoe side, and other sensor data values from each message.
7. Data Storage:
   - The MQTT listener stores the captured sensor data in the RDS (Relational Database Service) associated with the application.
   - The data is stored in the appropriate table, likely the "shoeRawData" table, using the received recording ID to maintain the data's association.
8. Frontend Graph Component Subscription:
   - The frontend application's graph component subscribes to the topic "sensor/#/#" based on the selected shoe and shoe side.
   - This subscription enables the graph component to receive real-time sensor data updates from the MQTT broker.
9. Stop Recording Message:
   - When the user decides to stop the recording, the frontend application publishes a message on the "status/#" topic.
   - This message indicates the recording's end and contains a state number (e.g., 2 representing the "off" state).

10. Device Deactivation:

    - Upon receiving the stop recording message, the gait analysis device (shoe) associated with the selected ID deactivates and stops collecting sensor data.

PDF Generation

The application uses the Flask framework and PDFKit library to render HTML templates and convert them to PDF files. The API endpoint `/download` is responsible for generating and serving the PDF report.

Endpoint: `/download`

- Method: GET, POST
- Description: Generates and downloads a PDF report based on the provided data.
- Request Payload:
    - Content-Type: application/json
    - Body: JSON object containing the following data:
        - `patientId` (integer): ID of the patient to generate the report for.
        - `gaitData` (object): Gait analysis data in JSON format.
- Response:
    - Content-Type: application/pdf
    - Content-Disposition: inline; filename=output.pdf

Request Example:

```
curl -X POST -H "Content-Type: application/json" -d '{
  "patientId": 123,
  "gaitData": {
    "AverageForceApplied": {
      "left": 266.1249949385,
      "right": 190.959672511375
    },
    "Cadence": {
      "left": 252.63157894736844,
      "right": 186.33540372670808
    },
    ...
  }
}'
```

Response Example:

The response will be a PDF file that can be downloaded and viewed by the client.

Error Responses:

- Status: 400 Bad Request
  - Description: Invalid request payload or missing required data.

Database Connection:

The application connects to a MySQL database to retrieve patient and medical history data. The connection details are as follows:

- Host: database-2.ch7qblmju1la.ap-south-1.rds.amazonaws.com
- User: admin
- Password: Asdfgh2014$$
- Database: paceBdb

HTML Template:

The PDF report is generated by rendering an HTML template named "report.html". This template uses Jinja2 templating engine to dynamically populate the patient information and medical history data.

CORS Support:

The application includes CORS (Cross-Origin Resource Sharing) support using the Flask-CORS extension. This allows cross-origin requests to be made to the `/download` endpoint.