# Emotibit API Documentation

***Written by Matthew Yamen, Samuele Reyes***

## Database

## *How do I access the database?*

Follow these steps:

1. **Download and Install MySQL Client**:

If you don't already have a MySQL client installed on your local machine, you can download and install one. Some popular MySQL clients are:

- MySQL Workbench: A graphical client for managing MySQL databases.
- HeidiSQL: A lightweight, open-source client for MySQL.
- DBeaver: A universal database client that supports MySQL and other databases.

Choose the MySQL client that suits your preferences and install it on your local machine.

2. **Launch the MySQL Client**:

Open the MySQL client you've installed.

3. **Create a New Connection**:

- In the MySQL client, find the option to create a new database connection or connection profile.
- You will be prompted to provide connection details. Fill in the following information based on the details you provided:
  - **Connection Name**: A user-defined name for the connection (e.g., "Emotibit RDS").
  - **Host**: emotibit-data.cxx7fbhxbyig.ap-southeast-2.rds.amazonaws.com
  - **Port**: 3306
  - **Username**: admin
  - **Password**: kristyadmin

4. **Test the Connection**:

After entering the connection details, you can usually test the connection to ensure it works. This may involve clicking a "Test" button or something similar in the MySQL client.

5. **Connect to the Database**:

Once the connection is tested successfully, you can connect to the database by selecting the connection profile you created and clicking the "Connect" button.

6. **Explore the Database**:

After connecting to the database, you can explore the database schema, view tables, run SQL queries, and perform any database operations as needed. Ensure to use the command “USE emotibit” to access the emotibit database.

For example, you can use SQL commands list:

- SHOW TABLES; to list the tables in the current database.
- DESCRIBE <<table name>>; to show a table’s structure
- Run any CRUD SQL queries to retrieve or modify data.

## ***How do I add more datastreams to the database?***

Follow these steps:

1. **Connect to the database**:

As listed in “How do I access the database?” section 3. Ensure that the database used is “emotibit”.

2. **Create a new table for the datastream**

Run the command below with the name of the table according to the datastream to be added.

```
CREATE TABLE test (
   id BIGINT NOT NULL AUTO_INCREMENT PRIMARY KEY,
   date_time_recorded DATETIME(6) NOT NULL,
   patientid BIGINT NOT NULL,
   sessionid BIGINT NOT NULL,
   value FLOAT NOT NULL,
   FOREIGN KEY (patientid) REFERENCES patient(ID)
);
```

# *Database Schema*

**Notation**

**PK = Primary Key**
**FK = Foreign Key**

### Patient

| | |
|---|---|
| PK BIGINT | id |
| VARCHAR(255) | email |
| VARCHAR(255) | first_name |
| VARCHAR(255) | last_name |
| VARCHAR(255) | password |
| VARCHAR(255) | emotibit_name |
| FK BIGINT | clinicianid |

### Clinician

| | |
|---|---|
| PK BIGINT | id |
| VARCHAR(255) | email |
| VARCHAR(255) | first_name |
| VARCHAR(255) | last_name |
| VARCHAR(255) | password |
| VARCHAR(255) | clinic_name |

### HeartRate

| | |
|---|---|
| PK BIGINT | id |
| DATETIME(6) | date_time_recorded |
| FLOAT | value |
| BIGINT | sessionid |
| FK BIGINT | patientid |

### HeartRateVariability

| | |
|---|---|
| PK BIGINT | id |
| DATETIME(6) | date_time_recorded |
| FLOAT | value |
| BIGINT | sessionid |
| FK BIGINT | patientid |

### Temperature

| | |
|---|---|
| PK BIGINT | id |
| DATETIME(6) | date_time_recorded |
| FLOAT | value |
| BIGINT | sessionid |
| FK BIGINT | patientid |

### ElectrodermalLevel

| | |
|---|---|
| PK BIGINT | id |
| DATETIME(6) | date_time_recorded |
| FLOAT | value |
| BIGINT | sessionid |
| FK BIGINT | patientid |

### **ElectrodermalResponse**

| | |
|---|---|
| PK BIGINT | id |
| DATETIME(6) | date_time_recorded |
| FLOAT | value |
| BIGINT | sessionid |
| FK BIGINT | patientid |

### **SkinConductanceResponse**

| | |
|---|---|
| PK BIGINT | id |
| DATETIME(6) | date_time_recorded |
| FLOAT | value |
| BIGINT | sessionid |
| FK BIGINT | patientid |

# API

## *How do I use the API?*

In this guide, we will demonstrate how to use Node.js and the axios library to make HTTP requests to the Emotibit API endpoints. We'll cover GET and POST requests as examples (that's all the API offers at the moment and all we need). Ensure you have Node.js installed and the axios library installed by running npm install axios if you haven't already.

### Dependencies

Before you begin, make sure you have the required dependencies installed:

1. Node.js - Install Node.js from nodejs.org (should already have this).
2. Axios - Install the axios library using the following command: "npm install axios"

### Example: GET Request to Retrieve Latest DataStream for a Patient

This example demonstrates how to make a GET request to retrieve the latest value of a specific datastream, being electrodermal level (el) for a patient with ID 1. Replace the URL and endpoint as needed.

```
const axios = require('axios');

// Define the URL of your API endpoint
const apiUrl = 'http://3.27.158.164:8080/datastream/el/latest/1';

// Define the headers for the request
const headers = {
   'Accept': 'application/json'
};

// Make the GET request
axios.get(apiUrl, { headers })
   .then(response => { console.log('Latest DataStream (el) for Patient 1:', response.data)
;})
   .catch(error => { console.error('Error:', error);
});
```

### Example: POST Request to Register a Patient

This example demonstrates how to make a POST request to register a new patient with the provided information. Replace the URL and endpoint as needed.

```
const axios = require('axios');

// Define the URL of your API endpoint
const apiUrl = 'http://3.27.158.164:8080/auth/patient';

// Define the request data and headers
const requestData = {
   firstName: 'Tested',
   lastName: 'Tester',
   email: 'testj@gmail.com',
   password: 'test'
};

const headers = {
   'Content-Type': 'application/json'
};

// Make the POST
request axios.post(apiUrl, requestData, { headers })
   .then(response => {console.log('Patient Registration Response:', response.data);
})
   .catch(error => { console.error('Error:', error);
});
```

**NOTE:** The code is exactly the same for registering a new clinician. The only difference is the payload passed via the *requestData* constant. Change the dictionary passed to *requestData* according to the Clinician object specification in the API endpoints section.

## Example: Handling a Login Request and Checking for Success

In this example, we'll make a POST request to log in a clinician, and then we'll check the response to determine if the login was successful or not.

```
const axios = require('axios');

// Define the URL of your API endpoint
const apiUrl = 'http://3.27.158.164:8080/auth/login';

// Define the request data and headers const requestData = {
   email: 'bobj@gmail.com', passwordProvided: 'chips'
};

const headers = {
   'Content-Type': 'application/json'
};
```

```
// Make the POST request to log in the patient
axios.post(apiUrl, requestData, { headers })
   .then(response => {
      if (response.status === 200) { console.log('Login successful');
      // You can access user data from response.data, if provided
      }  else {
         console.log('Login failed');
      }
   })
   .catch(error => { console.error('Error:', error);
});
```

# API Endpoints

As you can see from the above tutorial, all you need to do is call an endpoint and (if necessary) pass it data. The process is very simple and it is identical with the following exhaustive list of endpoints.

Note that our AWS EC2 instance's public IPv4 address is: 3.27.158.164 and the port is 8080. You will use both of these to connect to the EC2 instance (i.e., where the backend API application is being hosted and currently running). These can change when the EC2 instance is restarted. All of these endpoints will then need to be updated with the new IPv4 public address.

## Authentication

**Endpoint for Patient Registration**:

- **URL Path:** http://3.27.158.164:8080/auth/patient
- **HTTP Method:** POST
- **Consumes:**
    - JSON with all data of a patient
    - Example payload:

```
{
   "firstName": "James",
   "lastName": "Smith",
   "email": "jamessmith@gmail.com",
   "password": "goldfish",
   "emotibitId": "MD-V5-0000215",
   "clinicianId": 1
}
```

- **Produces:**
    - JSON with all data of a patient with password removed
    - Example payload:

```json
{
   "firstName": "James",
   "lastName": "Smith",
   "email": "jamessmith@gmail.com",
   "password": "",
   "emotibitId": "MD-V5-0000215",
   "clinicianId": 1
}
```

- **Description:** This endpoint is used for registering a new patient. It expects a JSON payload with patient details, as specified above. The response contains the registered patient's data with the password field empty for security.

**Endpoint for Clinician Registration**:

- **URL Path:** http://3.27.158.164:8080/auth/clinician
- **HTTP Method:** POST
- **Consumes:**
    - JSON with all data of a clinician
    - Example payload:

```json
{
   "firstName": "Bob",
   "lastName": "Jalal",
   "email": "bobj@gmail.com",
   "password": "chips",
   "clinicName": "Wangaratta Medical Centre"
}
```

- **Produces:**
    - JSON with all data of a clinician with password removed
    - Example payload:

```json
{
   "firstName": "Bob",
   "lastName": "Jalal",
   "email": "bobj@gmail.com",
   "password": "",
   "clinicName": "Wangaratta Medical Centre"
}
```

- **Description:** This endpoint is used for registering a new clinician. It expects a JSON payload with clinician details, as specified above. The response contains the registered clinician's data with the password field empty for security.

### Endpoint for User Login:

- **URL Path:** http://3.27.158.164:8080/auth/login
- **HTTP Method:** POST
- **Consumes:**
  - JSON with the login details of a user
  - Example payload:

```
{
   "email": "jamessmith@gmail.com",
   "passwordProvided": "goldfish"
}
```

- **Produces:**
  - JSON with all data of a user with password removed
  - Example payload:

```
{
   "firstName": "James",
   "lastName": "Smith",
   "email": "jamessmith@gmail.com",
   "password": "",
   "emotibitId": "MD-V5-0000215",
   "clinicianId": 1
}
```

- **Description:** This endpoint is used for user authentication and login. It expects a JSON payload with login attempt details, including the user's email and provided password. The response contains the specific user's data with the password field cleared for security.

## Clinicians

### Endpoint to Get All Clinicians:

- **URL Path:** http://3.27.158.164:8080/clinician
- **HTTP Method:** GET
- **Produces:**
  - List of clinicians in JSON format with all data of each clinician, password removed
- **Description:** This endpoint is used to retrieve a list of all clinicians. When a GET request is made to this endpoint, it returns a JSON response containing information about all the clinicians in the system. This can be used to fetch a list of clinicians for further processing or display.

### Endpoint to Delete a Clinician:

- **URL Path:** http://3.27.158.164:8080/clinician/{clinicianId}

- **HTTP Method:** DELETE
- **Path Variable:** clinicianId - ID of clinician to delete
- **Description:** This endpoint is used to delete a particular clinician from the database. Be careful when using this as patient's might then point to a clinician id that no longer exists! Make sure to update the patient's clinicianId field when they are assigned a new clinician.

## Patient

### Endpoint to Get All Patients:

- **URL Path:** http://3.27.158.164:8080/patient
- **HTTP Method:** GET
- **Produces:**
    - List of clinicians in JSON format with all data of each patient, password removed
- **Description:** This endpoint is used to retrieve a list of all patients. When a GET request is made to this endpoint, it returns a JSON response containing information about all the patients in the system. This can be used to fetch a list of patients for further processing or display

### Endpoint to Get Clinician of a Patient:

- **URL Path:** http://3.27.158.164:8080/patient/{patientId}/clinician
- **HTTP Method:** GET
- **Path Variable:** patientId - ID of patient
- **Produces:**
    - JSON with all data of the respective clinician with password removed
- **Description:** This endpoint is used to retrieve the clinician assigned to a specific patient. Making a GET request to this endpoint with the patientId in the URL will return the clinician associated with that patient

### Endpoint to get a patient associated with a particular emotibitId:

- **URL Path:** http://3.27.158.164:8080/patient/{emotibitId}
- **HTTP Method:** GET
- **Path Variable:** emotibitId - ID of patient you want to find by their respective emotibitId
- **Produces:**
    - JSON with all data of the respective clinician with password removed
- **Description:** This endpoint is used to retrieve the emotobitId assigned to a specific patient. Making a GET request to this endpoint with the patientId in the URL will return the patient object associated with that emotibitId (if one exists).

### Endpoint to Delete a Patient:

- **URL Path:** http://3.27.158.164:8080/patient/{patientId}
- **HTTP Method:** DELETE

- **Path Variable:** patientId -  ID of patient to delete
- **Description:** This endpoint is used to delete a particular patient from the database.

## Datastreams

### Endpoint to Add Heart Rate Data of a Patient:

- **URL Path:** http:/3.27.158.164:8080/datastream/hr/
- **HTTP Method:** POST
- **Path Variable:** patientId - ID of the patient for whom you want to add new heart rate data
- **Consumes:**
  - JSON with all the heart rate data of a patient
  - Example payload:

```
{
   "id": "1",
   "patientid": "1",
   "dateTimeRecorded": "2024-04-27 12:30:59.833000",
   "value": "58",
   "sessionid": "1714221051236076000"
}
```

- **Description:** This endpoint is used to add new heart rate data to the heart rate table. It expects a JSON payload with some heart rate data with the specific format as seen above.

### Endpoint to Get Latest Heart Rate Data of a Patient:

- **URL Path:** http:/3.27.158.164:8080/datastream/hr/latest/{patientId}
- **HTTP Method:** GET
- **Path Variable:** patientId - ID of the patient for whom you want to get their latest heart rate data
- **Produces:**
  - JSON with the latest heart rate data of a patient
- **Description:** This endpoint is used to retrieve the latest heart rate data of a specific patient. You can make a GET request to this endpoint, providing the patientId in the URL to get the latest heart rate data for that patient.

### Endpoint to Get All Heart Rate History of a Patient:

- **URL Path:** http:/3.27.158.164:8080/datastream/hr/all/{patientId}
- **HTTP Method:** GET
- **Path Variable:** patientId - ID of the patient for whom you want to get all heart rate data
- **Produces:**
  - List of heart rate data in JSON format

- **Description:** This endpoint is used to retrieve the complete history of heart rate data for a specific patient. Making a GET request to this endpoint with the patientId in the URL will return all heart rate data entries for that patient.

### **Endpoint to Add new Data to a datastream for a Patient**:

- **URL Path:** http:/3.27.158.164:8080/datastream/<<endpoint of datastream>>/
- **HTTP Method:** POST
- **Path Variable:** patientId - ID of the patient you want to add data of a specified datastream
- **Consumes:**
  - JSON with all the new data of a datastream of a patient
  - Example payload:

```
{
   "id": "2",
   "patientid": "2",
   "dateTimeRecorded": "2024-06-12 11:20:39.235957",
   "value": "0.13",
   "sessionid": "17235263765975623100"
}
```

- **Description:** This endpoint is used to add new data to their respective table. It expects a JSON payload with some data with the specific format as seen above. The endpoint will change to either: hr, bi, t1,edr, edl, scrf.

### **Endpoint to Get Latest Data of a Datastream for a Patient**:

- **URL Path:** http:/3.27.158.164:8080/datastream/<<endpoint of datastream>>/latest/{patientId}
- **HTTP Method:** GET
- **Path Variable:** patientId - ID of the patient for whom you want to get their latest data of a specified datastream
- **Produces:**
  - JSON with the data of a datastream for a patient
- **Description:** This endpoint is used to retrieve the latest data of a specified datastream for a specific patient. A GET request with the patientId in the URL will return the latest data of a datastream for that patient. The endpoint will change to either: hr, bi, t1,edr, edl, scrf.

### **Endpoint to Get All Data History of a Datastream for a Patient:**

- **URL Path:** http:/3.27.158.164:8080/datastream/<<endpoint of datastream>>/all/{patientId}
- **HTTP Method:** GET
- **Path Variable:** patientId - ID of the patient for whom you want to get all their data of a specified datastream

- **Produces:**
    - List of data according to specified datastream in JSON format
- **Description:** This endpoint is used to retrieve the complete history of a specified datastream for a specific patient. A GET request with the patientId in the URL will return all data entries for that patient. The endpoint will change to either: hr, bi, t1,edr, edl, scrf.