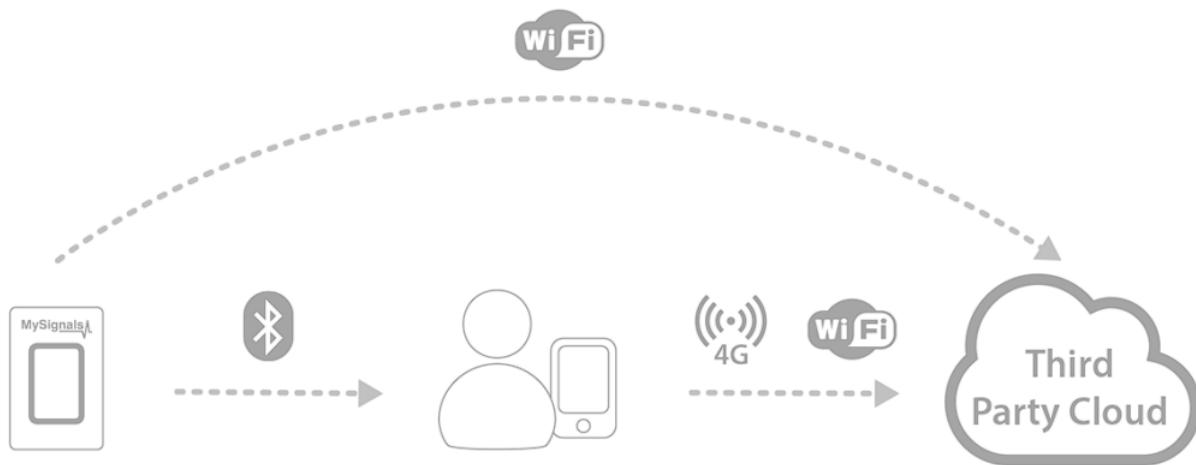


My Signals project

Aboubacar BAH
Taoufiq Boussraf

Objectives

This project involves collecting health data in real time from a "MySignals" IoT device to which several sensors are connected. Once collected, this data needs to be stored in a personal cloud API to be visualised in the form of graphs.

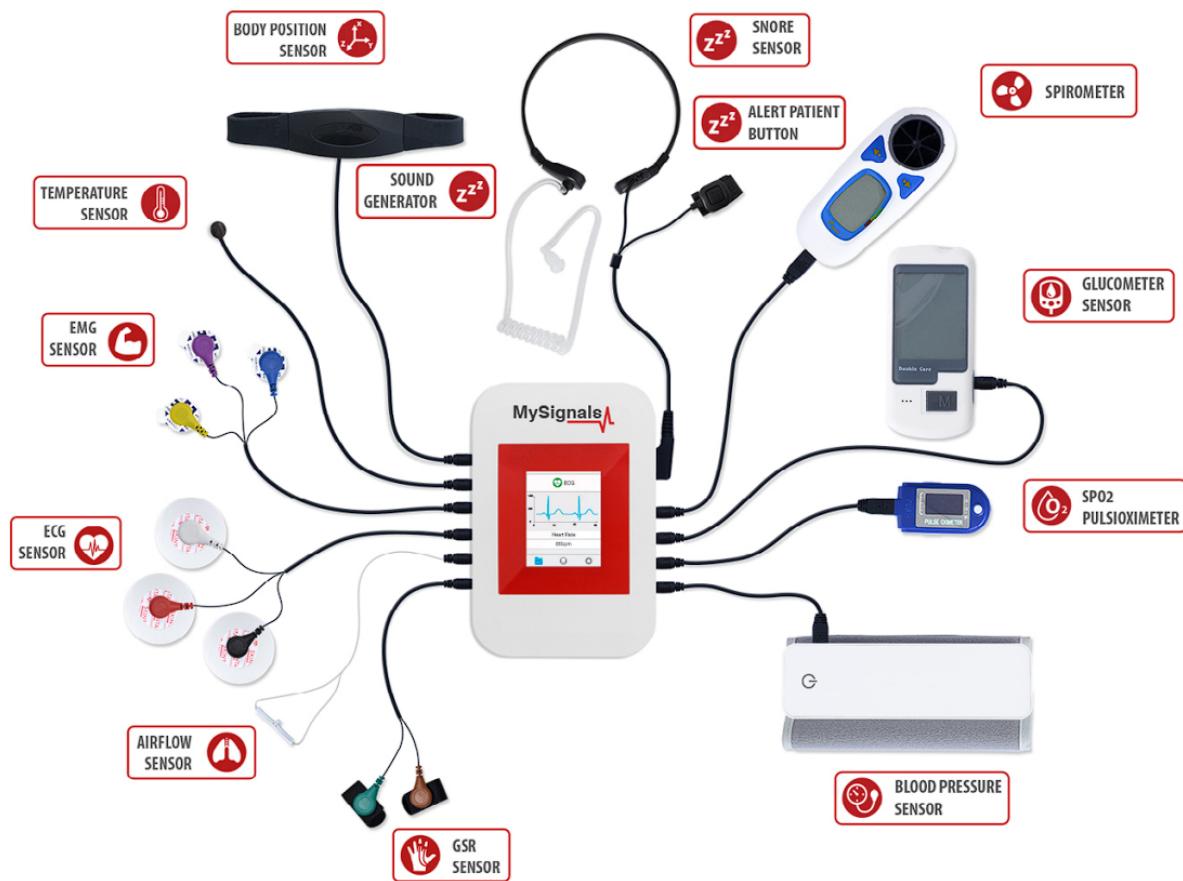


The main objectives are :

1. **Real-Time Data Collection:** Implement a solution to collect real-time data from health sensors connected to the MySignals IoT device.
2. **Integration of Multiple Sensors:** Enable the connection and integration of various types of health sensors such as ECG, EEG, temperature, blood pressure, glucose level, etc.
3. **Storage in Personal Cloud:** Develop a cloud API to securely store and manage collected data in a scalable manner.
4. **Data Visualization:** Create visualization features for data in the form of graphs and tables, allowing users to monitor and analyze collected health data.
5. **User-Friendly Interface:** Design a user-friendly interface that enables users to log in, view their data, and add new sensors data to the system.
6. **Data Security:** Implement security measures to protect sensitive health data stored in the personal cloud, using robust encryption and authentication methods.
7. **Multi-Platform Compatibility:** Ensure compatibility with different platforms, including mobile devices and computers, allowing users to access data from any device.

- 8. Notifications and Alerts:** Integrate notification and alert features to inform users of abnormal fluctuations or critical situations in health data.

Specifications



- 1. MySignals IoT Configuration:** Configure and set up the MySignals IoT device by connecting appropriate sensors and configuring communication settings.
- 2. Cloud API Development:** Design and establish a secure cloud API for storing and managing collected health data.
- 3. Android App Development:** Create an attractive and intuitive user interface that allows users to log in, connecting to MySignals Device, view health data, and collect sensors data from the device.
- 4. Data Visualization:** Develop interactive graphs and charts for visualizing health data provided by The Cloud API (By using an android App or a Web Application).
- 5. Cloud Connectivity Integration:** Implement bidirectional communication between the MySignals device and the cloud API for real-time data transmission.
- 6. Security and Authentication:** Implement security measures, including data encryption and authentication mechanisms, to ensure data confidentiality and integrity.
- 7. Notifications and Alerts:** Integrate notification and alert mechanisms to inform users of important events related to their health data.

8. **Testing and Validation:** Perform comprehensive testing to ensure that data collection, storage, visualization, and management functions correctly and securely.
9. **Documentation and Training:** Provide a documentation on how to use the application, along with user training on adding sensors, viewing data, and interacting with the user interface.
10. **Deployment and Maintenance:** Deploy the cloud application and ensure ongoing maintenance to ensure optimal performance and updates as needed.

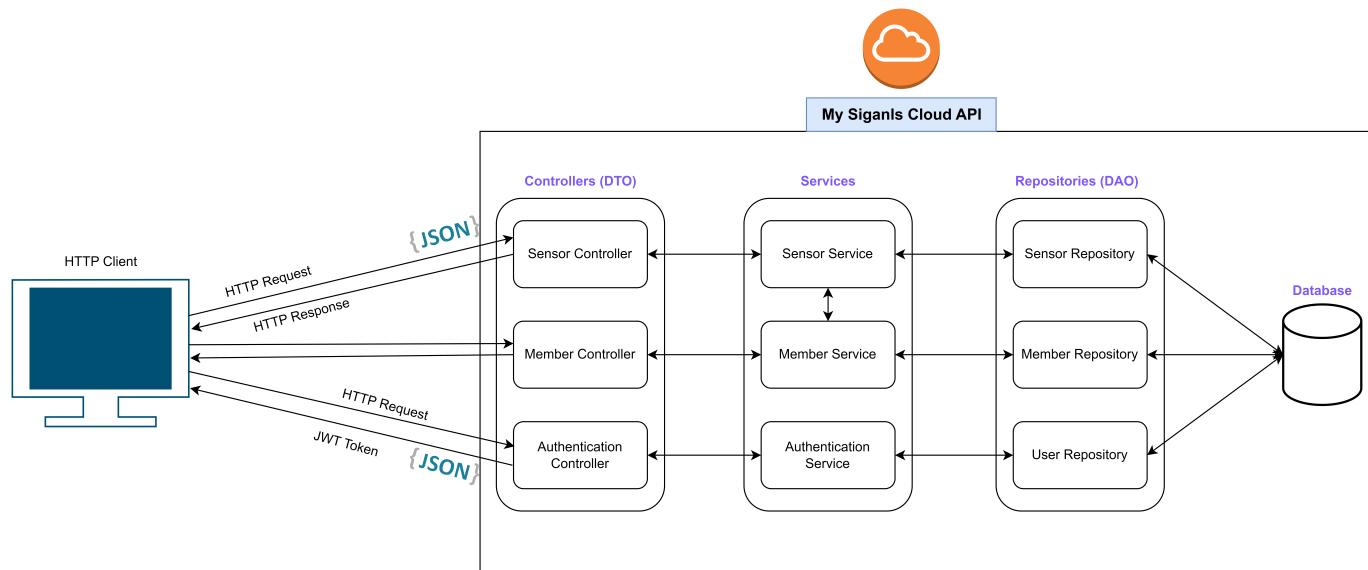
API Development

The API is based SpringBoot API

Tools

- IntelliJ IDEA, Visual Studio Code...
- Java JDK & JRE (Version ≥ 8)
- Maven

Architecture



Services

In the SpringBoot API, we have three services :

Sensors Service

The Sensors Service is responsible for managing the data related to various sensors used in the system. It implements CRUD (Create, Read, Update, Delete) operations to handle sensor information. This service allows you to perform the following actions:

- **Create:** Add new sensor information to the database, including details like sensor type, data units, and any other relevant metadata.
- **Read:** Retrieve sensor information from the database based on different criteria, such as sensor type or sensor ID.

- **Update:** Modify existing sensor information, such as updating sensor metadata or changing sensor properties.
- **Delete:** Remove sensor information from the database when a sensor is no longer needed or is replaced.

The Sensors Service plays a crucial role in storing and managing the characteristics and properties of different sensors within the system.

Member Service

The Member Service handles the management of member data within the system. Similar to the Sensors Service, it provides CRUD operations for member-related information. This service allows you to perform the following actions:

- **Create:** Add new member profiles to the system, including details like name, surname, profile picture, description, height, weight, and other relevant attributes.
- **Read:** Retrieve member profiles from the database based on different criteria, such as member ID or specific attributes.
- **Update:** Modify existing member profiles, including updating personal information or adjusting attributes like height and weight.
- **Delete:** Remove member profiles from the system when necessary.

The Member Service is essential for maintaining records of individuals associated with the health data and providing the necessary details for data analysis and visualization.

Authentication Service (JWT Authentication)

The Authentication Service is responsible for handling user authentication and authorization using JSON Web Tokens (JWT). It provides the necessary functionality to securely manage user access to the API endpoints. This service includes the following features:

- **User Registration:** Allow users to create accounts by providing necessary details, such as username, email, and password.
- **User Login:** Authenticate users by verifying their credentials and generating JWT tokens upon successful login.
- **Token Validation:** Verify the authenticity and integrity of incoming JWT tokens to ensure that users have valid access.
- **Authorization:** Control user access to specific endpoints and resources based on their roles and permissions stored within the JWT.
- **Token Refresh:** Provide the ability to refresh expired tokens, enabling users to extend their session without the need for frequent login.

The Authentication Service ensures that only authorized users can access the API endpoints and perform actions, contributing to the security and integrity of the system.

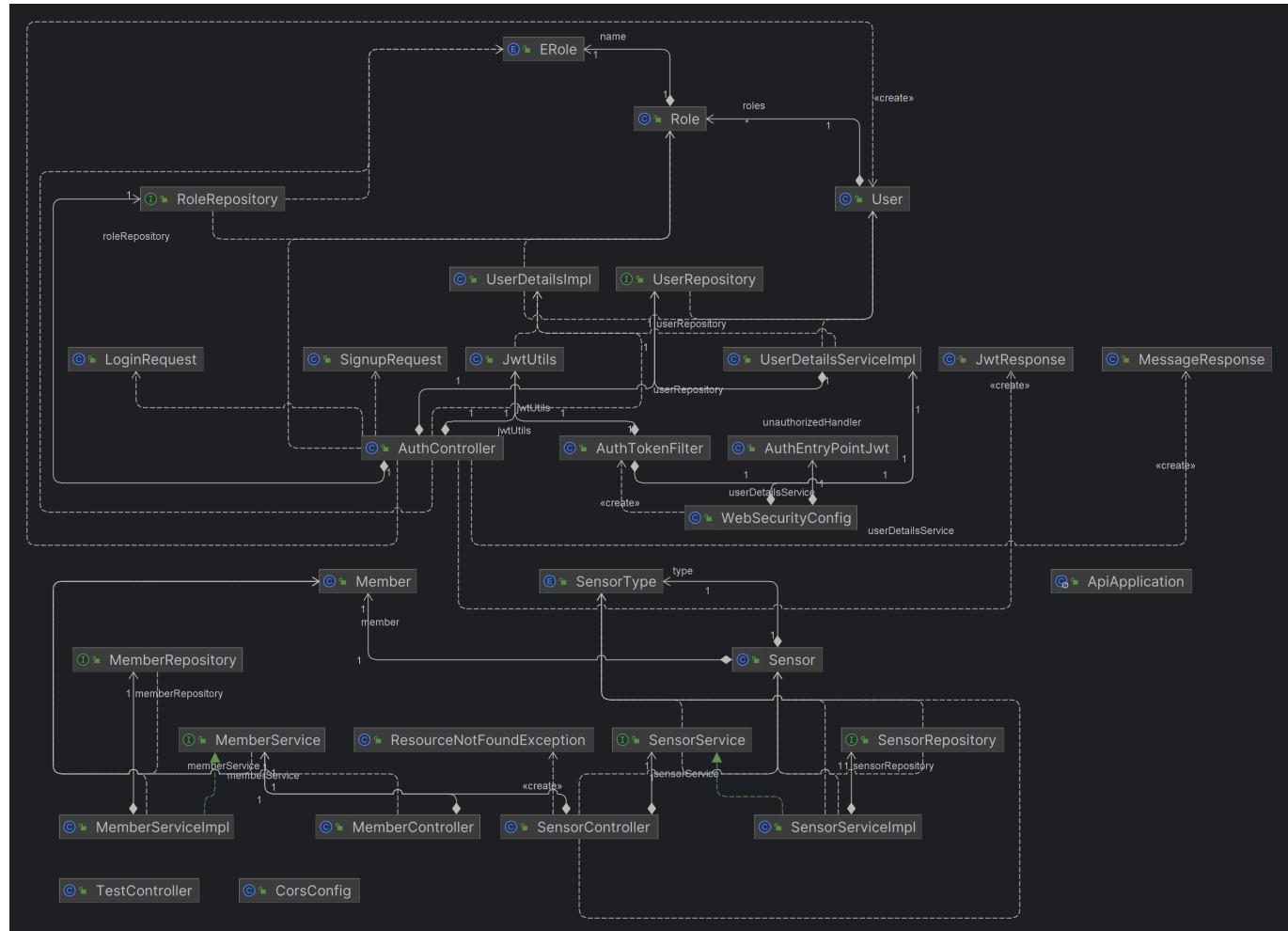
Overall, these three services work together to provide a comprehensive and secure platform for managing sensor data, member profiles, and user authentication within your Spring Boot API.

Database Management System (DBMS)

H2 Database (in developpment)

POSTGRES + PGADMIN4 (deployment)

Classes Diagram



Services Deployment

Docker compose file

Github Pipeline (TODO)

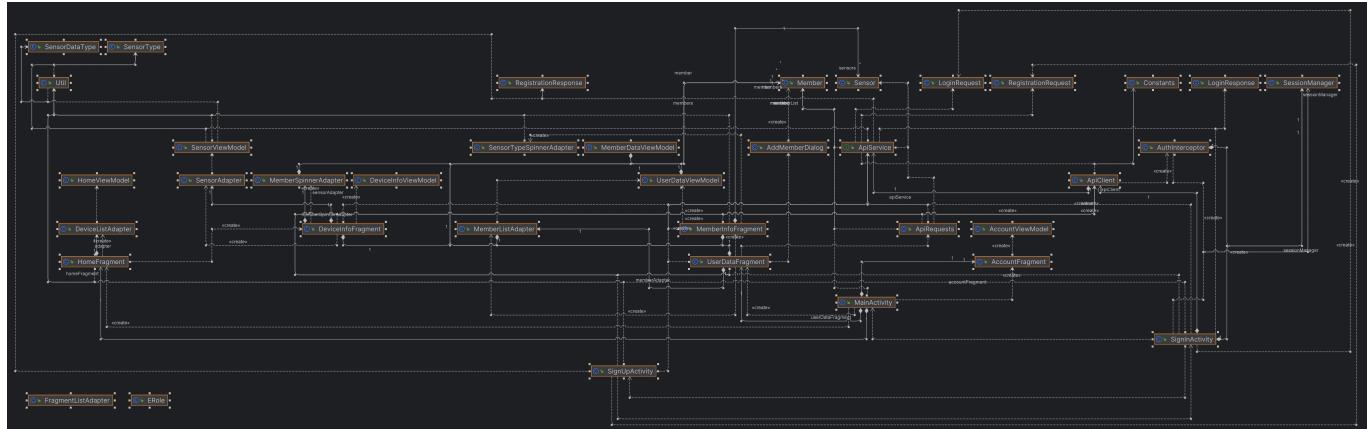
Android Application

Requirements

- IntelliJ IDEA, Android Studio
- Java JDK & JRE (Version \geq 8)
- Android Device(android minimum SDK \geq 4.3.0)

Architecture

Classes Diagram



Demo

API

- Launch the API

```
> & 'C:\Program Files\Java\jdk-17\bin\java.exe' '@C:\Users\aboub\AppData\Local\Temp\cp_1rymr0r39u5t9qglrlmkfb5r09.argfile' 'com.mysignals.api.ApiApplication'
```



- Interface Web screen : <http://localhost:8080/swagger-ui/index.html#/>

localhost:8080/swagger-ui/index.html#/

Swagger
Supported by SMARTBEAR

/api-docs

Explore

OpenAPI definition v0 OAS3

/api-docs

Servers

http://localhost:8080 - Generated server url ▾

sensor-controller ▾

member-controller ▾

auth-controller ▾

test-controller ▾

Schemas

Sensor >

Member >

SignupRequest >

LoginRequest >

Authentication Service

auth-controller

POST /api/auth/signup

POST /api/auth/signin

1. Registration

- Client Request

auth-controller

POST /api/auth/signup

Parameters

No parameters

Cancel**Reset****Request body** required

application/json

```
{
  "username": "kindy",
  "email": "kindy@example.com",
  "role": [
    "mod"
  ],
  "password": "kindy22"
}
```

Execute**Clear****Responses****Curl**

```
curl -X 'POST' \
'http://localhost:8080/api/auth/signup' \
-H 'accept: */*' \
-H 'Content-type: application/json' \
-d '{
  "username": "kindy",
  "email": "kindy@example.com",
  "role": [
    "mod"
  ],
  "password": "kindy22"
}'
```



- API Response

Server response**Code****Details**

200

Response body

```
{
  "message": "User registered successfully!"
}
```

**Download****Response headers**

```
cache-control: no-cache,no-store,max-age=0,must-revalidate
connection: keep-alive
content-type: application/json
date: Tue,15 Aug 2023 15:54:24 GMT
expires: 0
keep-alive: timeout=60
pragma: no-cache
transfer-encoding: chunked
vary: Origin,Access-Control-Request-Method,Access-Control-Request-Headers
x-content-type-options: nosniff
x-frame-options: DENY
x-xss-protection: 0
```

1. Login

- Client Request

POST /api/auth/signin

Parameters

No parameters

Request body required

application/json

```
{
  "username": "kindy",
  "password": "kindy22"
}
```

Execute Clear

Responses

Curl

```
curl -X 'POST' \
  'http://localhost:8080/api/auth/signin' \
  -H 'accept: */*' \
  -H 'Content-type: application/json' \
  -d '{
    "username": "kindy",
    "password": "kindy22"
}'
```

- API Response

Server response

Code	Details
200	<p>Response body</p> <pre>{ "id": 1, "username": "kindy", "email": "kindy@example.com", "roles": ["ROLE_MODERATOR"], "tokenType": "Bearer", "accessToken": "eyJhbGciOiJIUzI1NiJ9.eyJzdWIiOiJraW5keSisImIhdCI6MTY5MjExNTEwMiwiZXhwIjoxNjkyMjAxNTAyfQ.bMh1lu9GqUah96Seyyu91NtkTWhbwPR8zVrpYhIkro" }</pre> <p>Download</p> <p>Response headers</p> <pre>cache-control: no-cache,no-store,max-age=0,must-revalidate connection: keep-alive content-type: application/json date: Tue,15 Aug 2023 15:58:22 GMT expires: 0 keep-alive: timeout=60 pragma: no-cache transfer-encoding: chunked vary: Origin,Access-Control-Request-Method,Access-Control-Request-Headers x-content-type-options: nosniff x-frame-options: DENY x-xss-protection: 0</pre>

How to use the user **AccessToken** to make HTTP Request to the API

The screenshot shows a POST request to `http://localhost:8080/api/members`. The authentication scheme is set to "Bearer". A token is displayed: `eyJhbGciOiJIUzI1NiJ9eyJzdWliOiJraW5keSIsImIhdCI6MTY5MjExNTEwMi`. A note says: "Default authentication scheme is set to "Bearer ". Change this scheme name under "show more".

Member Service

The API endpoint list for member-controller includes:

- GET /api/members/{id}
- PUT /api/members/{id}
- DELETE /api/members/{id}
- GET /api/members
- POST /api/members

1. Create Operation : add a Member (POST)

- Request

The screenshot shows a POST request to `http://localhost:8080/api/members`. The request body is a JSON object:

```
1 {  
2   "name": "DOE",  
3   "surname": "John",  
4   "picture": "https://static.wikia.nocookie.net/twi  
5   "description": "I am John DOE",  
6   "height": 185,  
7   "weight": 80,  
8   "birthday": "2023-08-15"  
9 }
```

- Response

The screenshot shows a successful `201 Created` response with a total time of `143 ms`. The response body is identical to the request body:

```
1 {  
2   "id": 1,  
3   "name": "DOE",  
4   "surname": "John",  
5   "picture": "https://static.wikia.nocookie.net/twistedmetal/images/f/f3/TMBJohnDoe.jpg/revision/latest/thumb/width360/he  
6   "description": "I am John DOE",  
7   "height": 185,  
8   "weight": 80,  
9   "birthday": "2023-08-15T00:00:00.000+00:00"  
10 }
```

1. Read Operation : get a Member or All member ()

The screenshot shows the Postman interface with a successful response for a GET request to `http://localhost:8080/api/members`. The response status is 200 OK with a duration of 41 ms. The request body is displayed as JSON, representing a single member with ID 1, name "DOE", surname "John", and other details.

```
[{"id": 1, "name": "DOE", "surname": "John", "picture": "https://static.wikia.nocookie.net/twistedmetal/images/f/f3/TMBJohnDoe.jpg/revision/latest/thumb/width360height360", "description": "I am John DOE", "height": 185, "weight": 88, "birthday": "2023-08-15T00:00:00.000+00:00"}]
```

3. Delete Operation (HTTP DELETE) : delete a member by id

The screenshot shows the RapidAPI Beta interface. At the top, there's a header bar with a magnifying glass icon and the text "Request 3". Below it is a toolbar with a "DELETE" button, a URL input field containing "http://localhost:8000/api/members/2", and a "Send" button. The main area has tabs for "Description", "Headers", "Query", "Body", "Auth", and "Options", with "Query" being the active tab. Under "Query", there are two rows: "Query Param" and "Value", each with an "Add Query Param" button and an "Add Value" button. The "Headers" tab is selected in the main navigation bar. Below the headers, there are tabs for "Headers", "Text", "JSON Tree", "JSON Text", "Web", "Image", "Hex", and "Raw", with "Raw" being the active tab. The raw response content is displayed in a box with the title "RapidAPI Beta" and the note "This may not accurately reflect the raw HTTP data." The content is as follows:

```
1 HTTP/1.1 204 No Content
2 cache-control: no-cache, no-store, max-age=0, must-revalidate
3 connection: close
4 date: Tue, 15 Aug 2023 16:49:03 GMT
5 expires: 0
6 pragma: no-cache
7 vary: Origin, Access-Control-Request-Method, Access-Control-Request-Headers
8 x-content-type-options: nosniff
9 x-frame-options: DENY
10 x-xss-protection: 1
```

4. Update

The screenshot shows a REST API client interface. On the left, there's a toolbar with buttons for PUT, Headers, Query, Body, Auth, and Options. Below the toolbar, tabs for Text, JSON (which is selected), JSON Tree, Form URL-Encoded, Multipart, and GraphQL are visible. The main area contains a JSON editor with the following code:

```
1 {
2     "name": "DOE",
3     "surname": "Johnn",
4     "picture": "https://static.wikia.nocookie.net/twistedmetal/images/f/f3/TMBJohnDoe.jpg/revision/latest/thumb/width360/he
5     "description": "I am John DOE",
6     "height": 190,
7     "weight": 84,
8     "birthday": "1998-08-15"
9 }
```

To the right, the response details are shown: a status of 201 Created, a response time of 184 ms, and tabs for Info, Request, Response, Headers, Text, JSON Tree, JSON Text (which is selected), and Raw.

Sensor Service

sensor-controller	
PUT	/api/members/{memberId}/sensors/{id}
GET	/api/members/{memberId}/sensors
POST	/api/members/{memberId}/sensors
DELETE	/api/members/{memberId}/sensors
GET	/api/sensors
GET	/api/sensors/{id}
DELETE	/api/sensors/{id}

1. Create

The screenshot shows a POST request to `http://localhost:8080/api/members/1/sensor`. The JSON body contains the following data:

```

1  {
2    "type": "TEMP",
3    "date": "2023-07-13",
4    "unit": "%C",
5    "value": 37.5
6  }

```

The response is a 201 Created status with a response time of 41 ms. The response body is:

```

1  {
2    "id": 1,
3    "type": "TEMP",
4    "date": "2023-07-12",
5    "unit": "%C",
6    "value": "37.5"
7  }

```

2. Read

The screenshot shows a GET request to `http://localhost:8080/api/members/1/sensor`. The response is a 200 OK status with a response time of 46 ms. The response body is:

```

1  [
2    {
3      "id": 1,
4      "type": "TEMP",
5      "date": "2023-07-12",
6      "unit": "%C",
7      "value": "37.5"
8    }
9  ]

```

3. Delete

The screenshot shows a DELETE request to `http://localhost:8080/api/sensors/1`. The response is a 204 No Content status with a response time of 23 ms. The response body is:

RapidAPI Beta
This may not accurately reflect the raw HTTP data.

```

1  HTTP/1.1 204 No Content
2  cache-control: no-cache, no-store, max-age=0, must-revalidate
3  connection: close
4  date: Wed, 16 Aug 2023 19:26:17 GMT
5  expires: 0
6  pragma: no-cache
7  vary: Origin, Access-Control-Request-Method, Access-Control-Request-Headers
8  x-content-type-options: nosniff
9  x-frame-options: DENY
10 x-xss-protection: 0

```

4. Update

The screenshot shows a POST request to `http://localhost:8080/api/members/1/sensc`. The request method is `PUT`, and the URL is `http://localhost:8080/api/members/1/sensc`. The response status is `201 Created` with a time of `36 ms`. The request body is a JSON object:

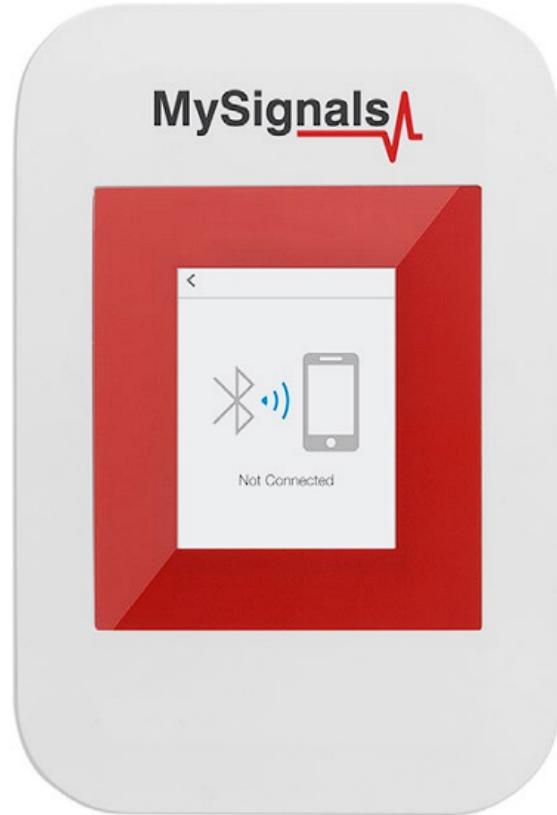
```
1 {  
2   "id": 1,  
3   "type": "TEMP",  
4   "date": "2023-07-11",  
5   "unit": "%C",  
6   "value": "50"  
7 }
```

MySignals Bluetooth Device

- Activate Bluetooth



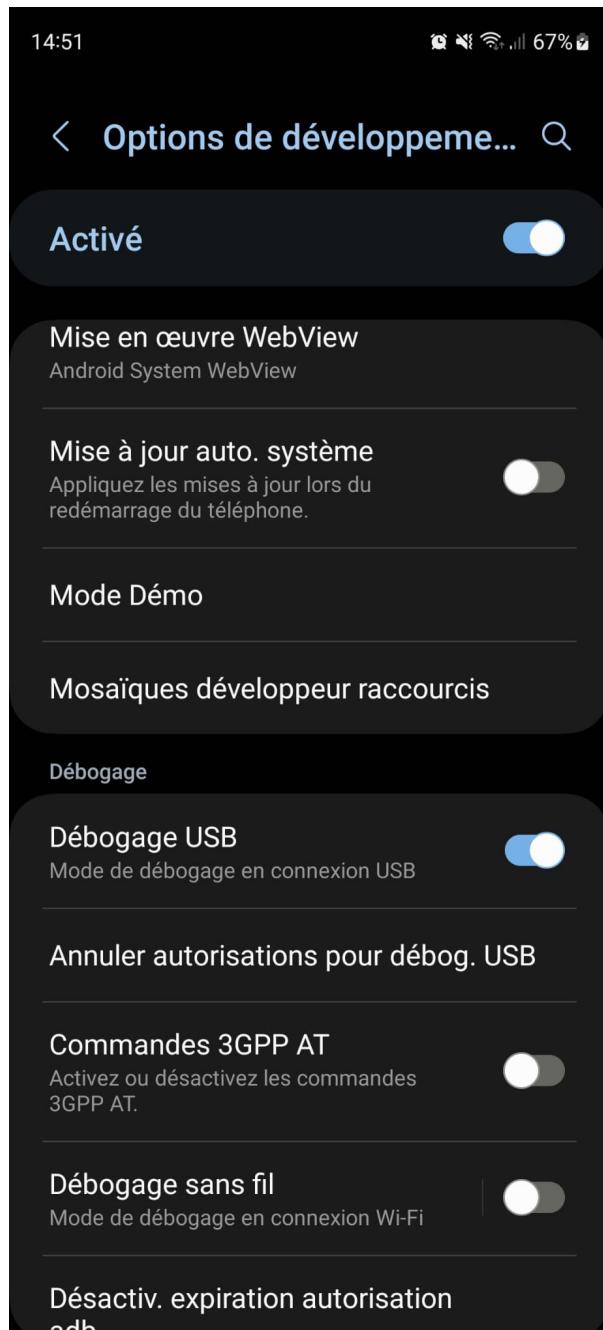
- Start Monitoring



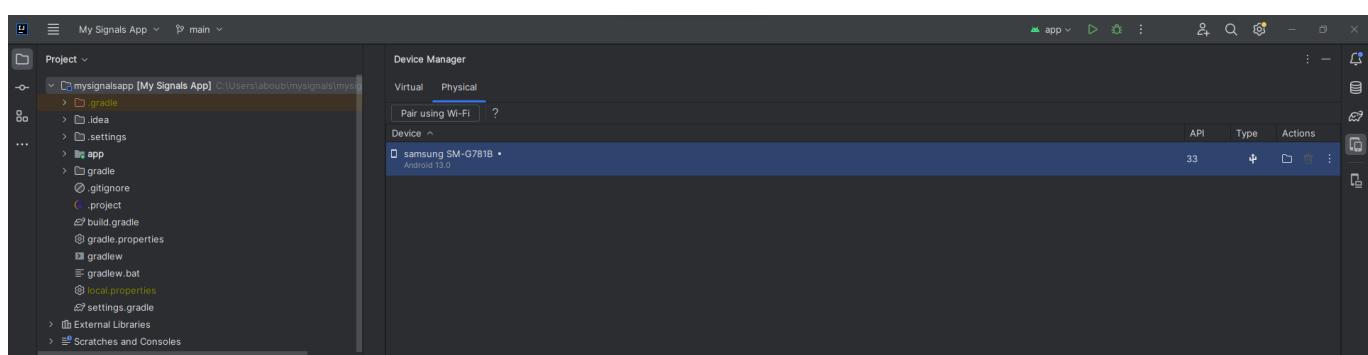
Android App

Installation

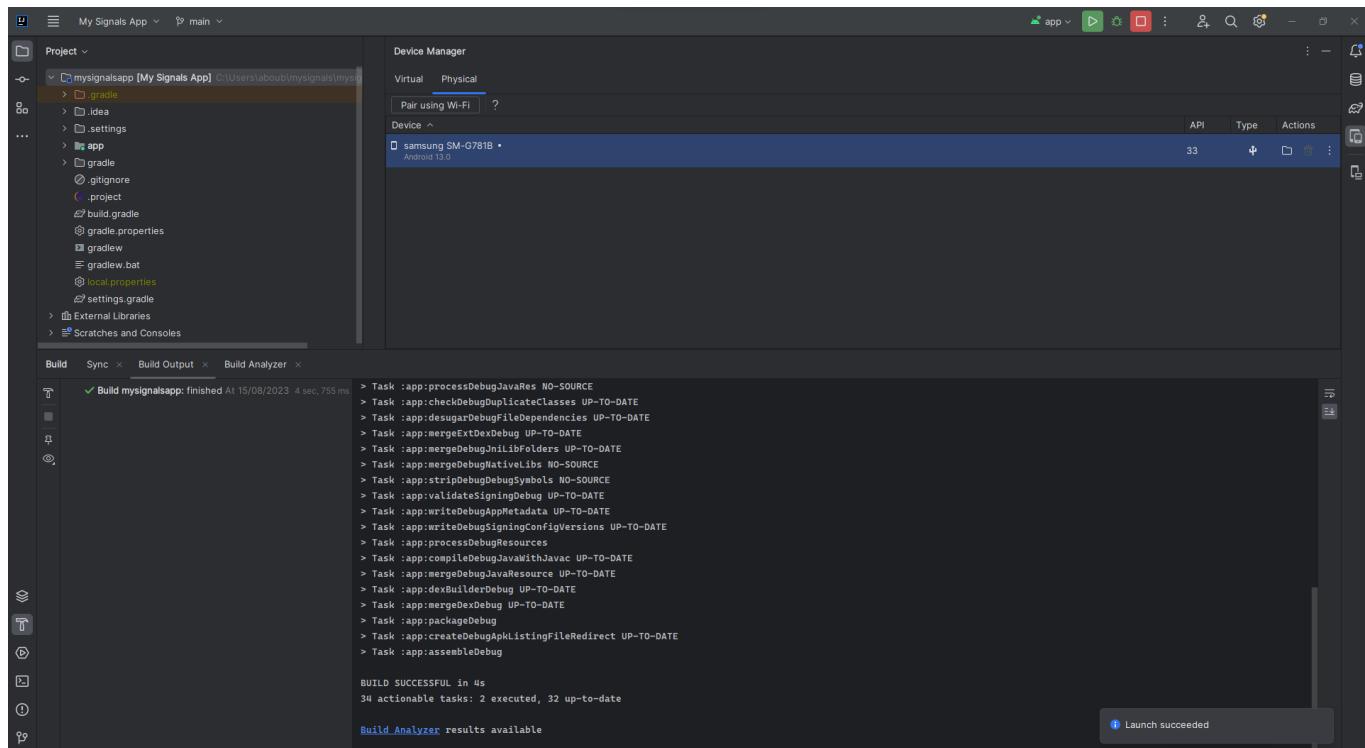
1. Android developpement mode



1. Android device detection in intelliJ



3. My Signals APK Installation



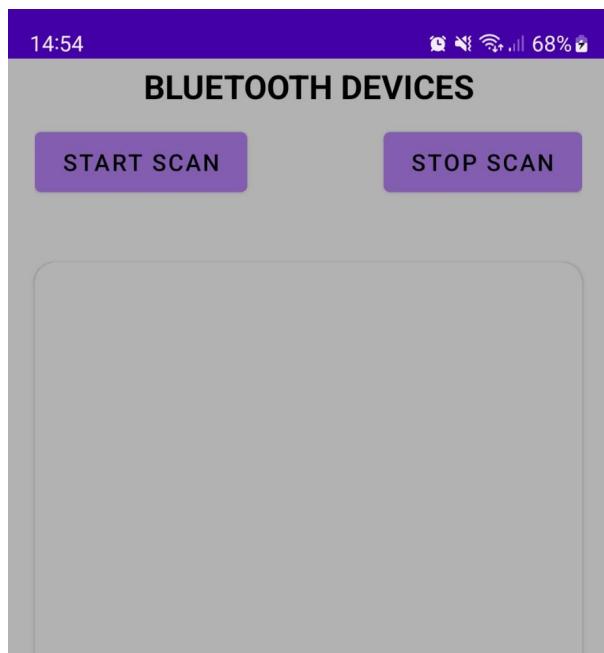
Authentication via android App

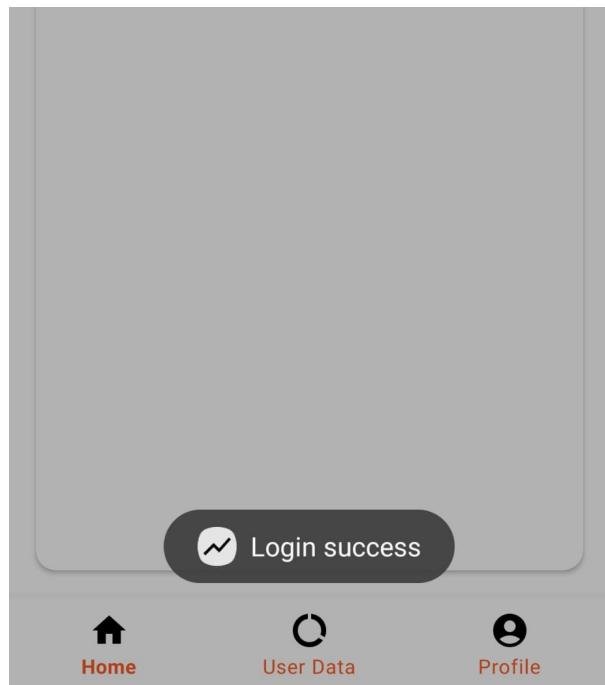


REGISTRATION



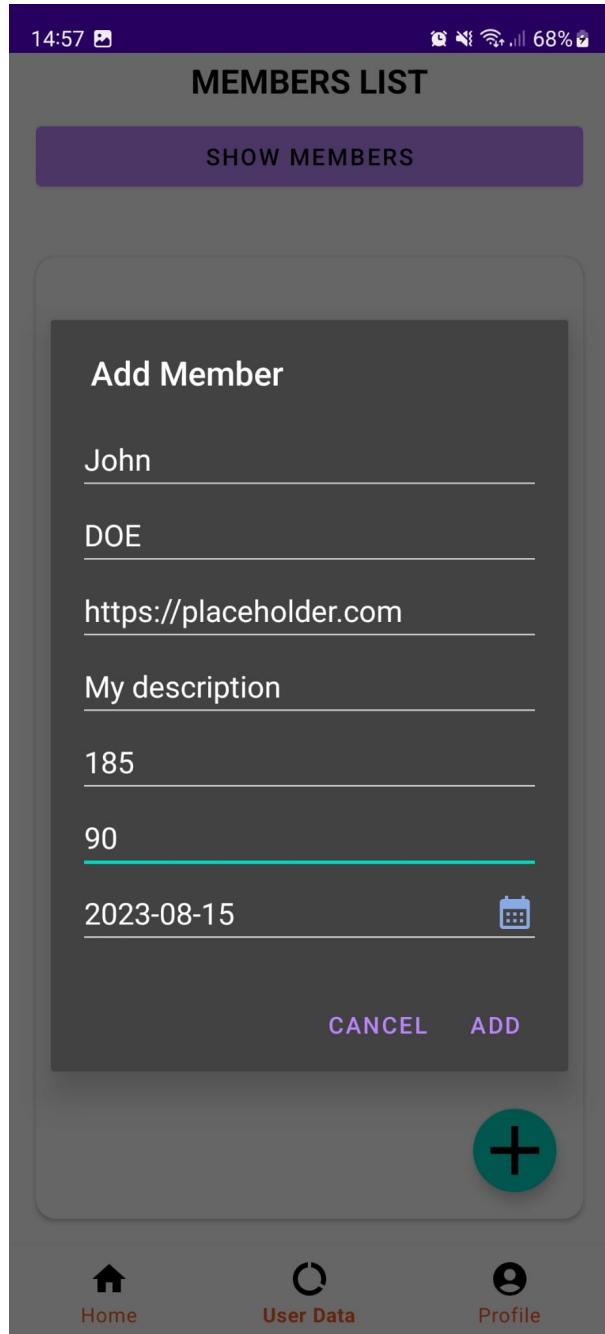
LOGIN



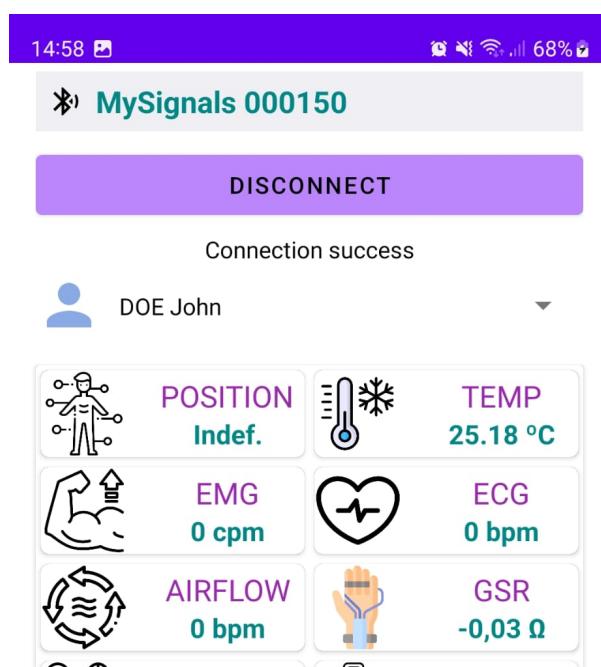
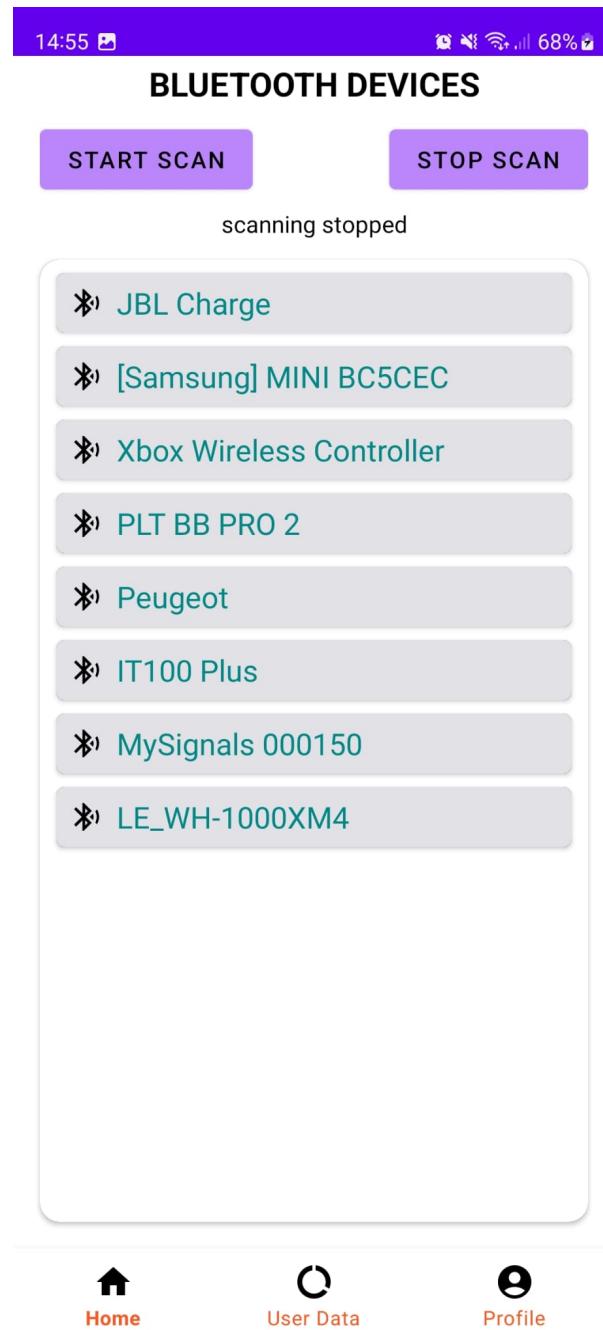


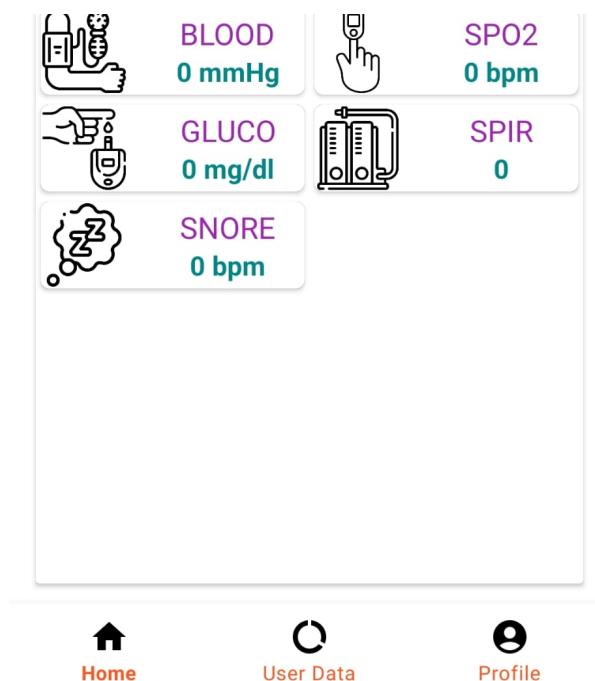
Data Visualization

1. Add a member

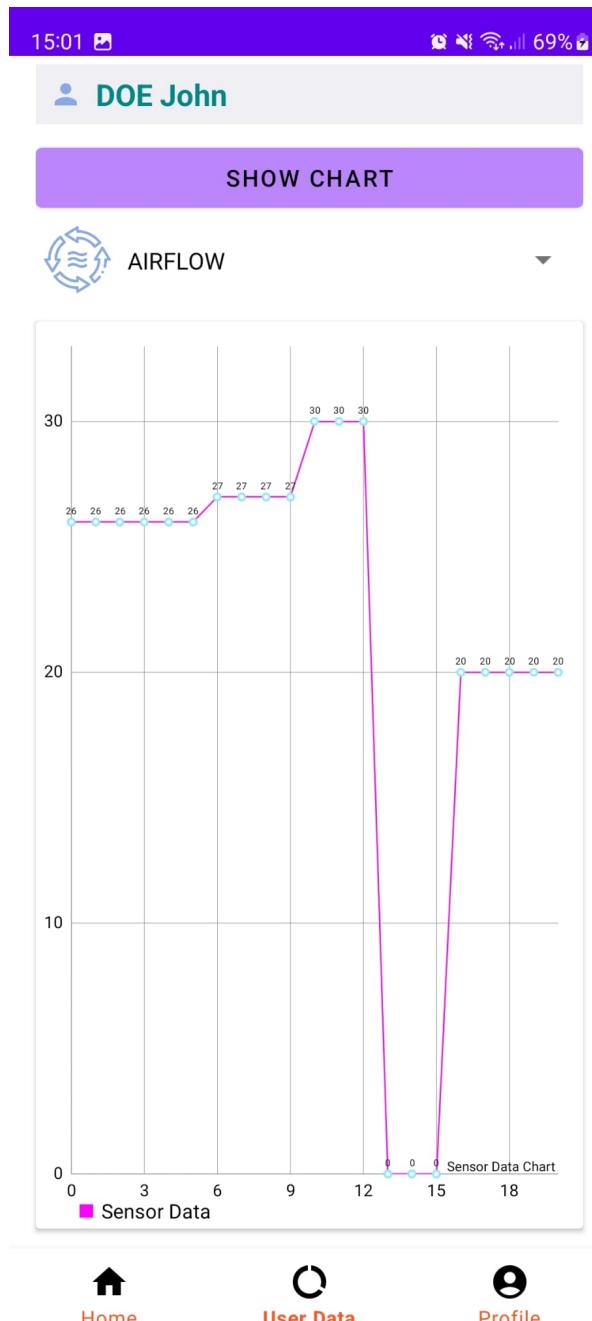


2. Bluetooth Connexion to MySignal Device, Collect and Save sensors data





3. Realtime Data Visualization



Home

User Data

Profile

Improvements

Sources

- Documentation : https://www.generationrobots.com/media/mysignals_technical_guide_sw.pdf
- JWT Authentication : <https://github.com/bezkoder/spring-boot-spring-security-jwt-authentication.git>
- Android/IOS libelium libraries :
 - http://downloads.libelium.com/mysignals/mysignals_android/MySignalsConnectKit.jar.zip
 - http://downloads.libelium.com/mysignals/mysignals_android/MySignalsConnectKitDoc-android.zip
 - http://downloads.libelium.com/mysignals/mysignals_android/MySignalsConnectTest-android.zip