ADMIN MANUAL

A comprehensive guided explanation of all server functionalities and system processes necessary to make proper use of this unit.

CREDITS

Carmen Areses, Pilar Bourg, Pablo Hervalejo, Carlota Laverón, Alejandra O'Shea

Table of contents:

1. Introduction

1.1. Role

The server is a Spring Boot application which centralizes the communication between patients, doctors and the information contained in a PostgreSQL database.

This manual contains an explanation on how to navigate this unit to ensure proper use of the application. Further details are provided about parameters, the database and errors that may appear in the system.

2. Objectives

2.1. Primary

The main objective of this end of the application is to facilitate a server which when started allows patients and doctors to connect and make use of the available actions of the app. And whenever necessary, to be able to close the application by stopping the server module.

2.2. Secondary

Subsidiary functionalities were included in the architecture of the server unit, no less important, but are complementary to the primary objective.

It was important to implement a save halt to the application; therefore, it was deemed necessary that a password must be introduced, to ensure that the authorised administrator was the only person with the power to stop the server. If this condition does not meet the expected outcome, the administrator will not be allowed to stop the server. Furthermore, the server may evaluate the actual state of the application through some parameters to ensure the optimal functionality of the application is provided.

3. Getting started

You can start the platform in two different ways. The first option is to use Docker, which provides a straightforward setup with all services pre-configured and ready to run. This approach is recommended if you want quick and consistent deployment across different environments without the need to have pre-requisites installed.

Alternatively, you can download the individual repositories and run each component separately. This method offers more flexibility for development and debugging, as it allows you to modify or replace specific modules without affecting the entire platform.

3.1.1. Through Docket Deployment

Docker Deployment provides a fully containerized environment for running the SMA (Spinal Muscular Atrophy) telemedicine system.

In this section how to use and install it is explained below. Nevertheless, for more information refer to the README section in the GitHub repository.

1. Make sure Docker has been installed in your computer through the official website: https://www.docker.com
2. Download the GitHub repository on your computer terminal: https://github.com/pilarbourg/telemedicine-deploy

```
git clone https://github.com/pilarbourg/telemedicine-deploy
```

From this point on, please make sure you have all necessary requirements specified in the README and the certificate in place (Refer to the Certificate Manual)

3. Decompress the zip file and open a terminal or console on your device and navigate to the project directory.
4. Start all services by running the following in your terminal:

```
docker-compose up -d
```

5. The web app should now be accessible locally at https://127.0.0.1

Finally, to shut down all containers run the following in the same terminal: `docker-compose down`

3.1.2. Manual setup

For the clients (doctors and patients) to be able to connect and interact with the web page, the administrator must initialize the server following these instructions:

1- Clone the repository:

   a. Open the terminal.

b. Navigate to the folder where you want to clone the repository.

c. Clone using the instruction: git clone link-to-repository.

- https://github.com/alejandraoshea/sma-server
- https://github.com/alejandraoshea/sma-client

2- Recommended to use GitBash but the project terminal in the coding application (such as Intelligi IDEA, VS Code, etc…) or the computer terminal will allow for initialization.

If in a terminal, go to the "*sma-server"* folder, not necessary if you are already in the coding software terminal with the project opened.

Write: `scripts/start-server.sh`

3- You should now insert the Username and the Password.

Note: You can modify this parameter in lines 15 and 16 of the start-server.sh file.

4- If the username and password are correct, the following should appear on your screen:

```
Username: ${ADMIN_USERNAME}
Password: ${ADMIN_PASSWORD}
Authentication successful.
Starting Telemedicine Server...
Server started. PID: 1072
Logs are being written to /c/Users/User/…/sma-server/scripts/../server.log
```

Highlights must correspond to your system and configurations.

Now the server is running.

Note: If something different appears, there has been an error, please contact us.

3.2. Termination of the server through stop-server.sh

1- To stop the server through the same terminal from which it was started write: `scripts/stop-server.sh`

2- Introduce the username and the password and the following should appear on your screen, confirming the termination of the server:

```
Username: ${ADMIN_USERNAME}
Password: ${ADMIN_PASSWORD}
Authentication successful.
Server stopped.
```

Highlights must correspond to your system and configurations.

3.3. Create an admin user

To create an admin user, you must directly insert it into the database. This ensures that only administrators will have the power to create admins therefore ensuring security. Depending on the system you are using this process could change.

Here is the SQL code to insert a new admin, please change the user and the password:

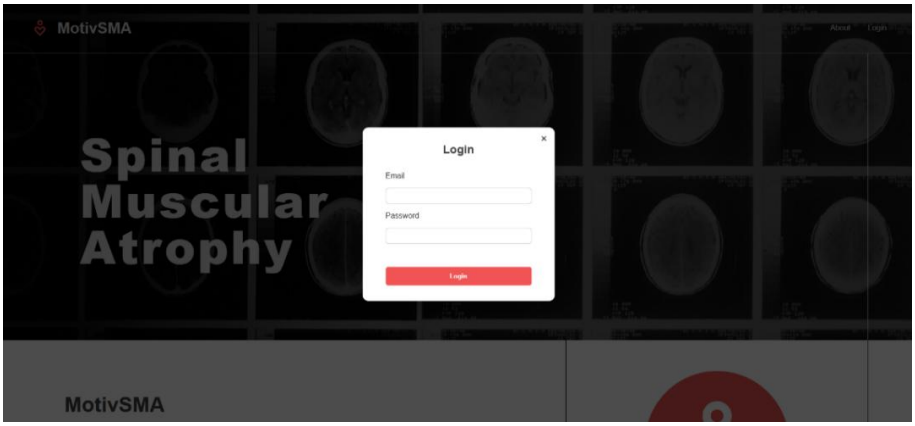
```
INSERT INTO app_users (email, password, role)
VALUES ('admin@example.com', 'password123', 'admin');
```

Highlights must correspond to your admin parameters; these will be used to log into the system.

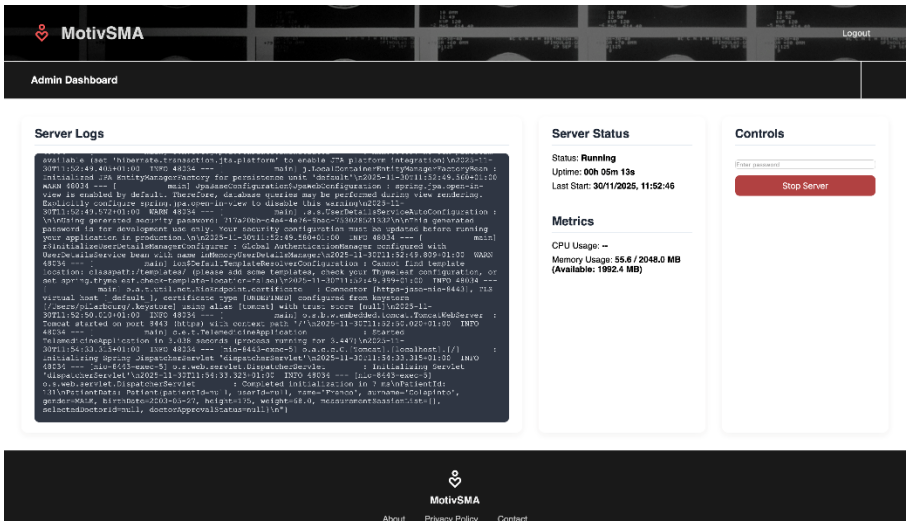
4. Functionalities

4.1. Log-in

You must log-in into the server unit using the *email* and the *password* specified in the database. Whenever these parameters are correct, the user will have access to all actions available to the administrator.



This action allows the user to enter the admin dashboard, which includes the *Status Panel* and a *Stop* button.



4.2. Status panel

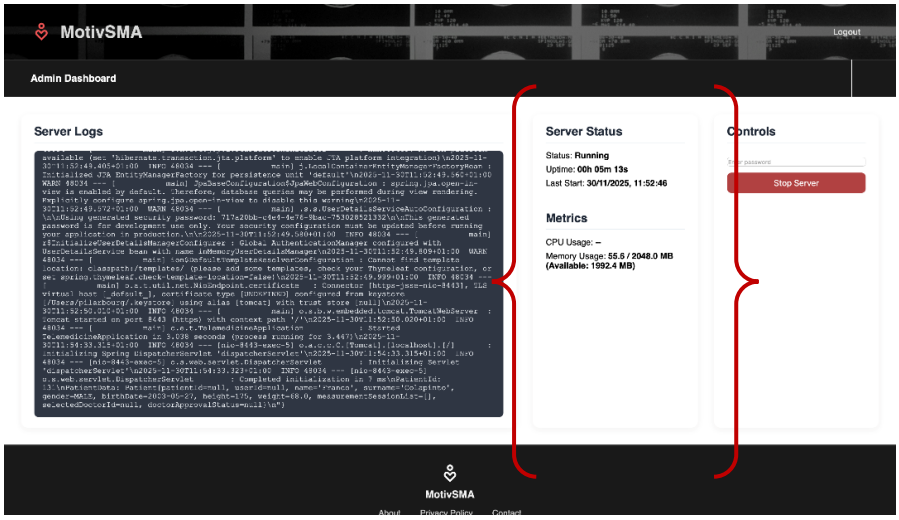
Provides administrator with real-time information about the operational state and status of the back-end system, ensuring visibility into the operational health of the application.

The following parameters are shown to the user:

- Whether the application is running or not. Could show if it has fallen into an error state, which would allow IT services to deal with the problem as soon as possible, without the need for complaints.
- Uptime shows how long the server has been running since it started last.
- Completing this information, it shows the time stamp of the exact moment from when the server started running.

- The memory that is being used from the maximum memory the JVM is allowed to use.
- Current CPU usage by the Java application formatted as a percentage.

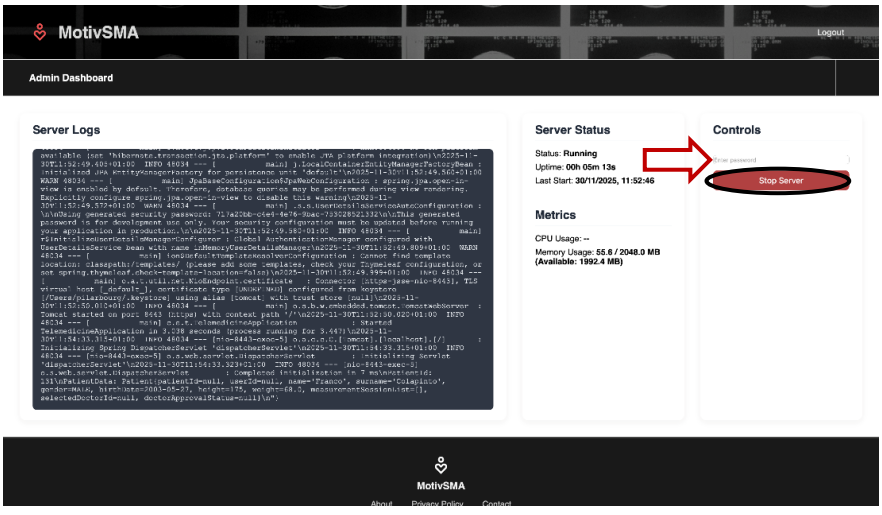
Together, these parameters allow you to assess server health and diagnose potential problems early, ensuring that maintenance decisions are informed.



4.3. Stopping the server through the interface

The server can be stopped in a secure manner. Before commencing the shutdown, the password established during the Sign-in must be introduced, ensuring that only authorized personnel can perform this critical action.

When the password is introduced, the server checks if it is the same one as stablished. If not, an error message will show:



If the password introduced is correct, the server will start its shutdown, disabling all possible connections between clients (doctors and patients).

The password by default is “password”.

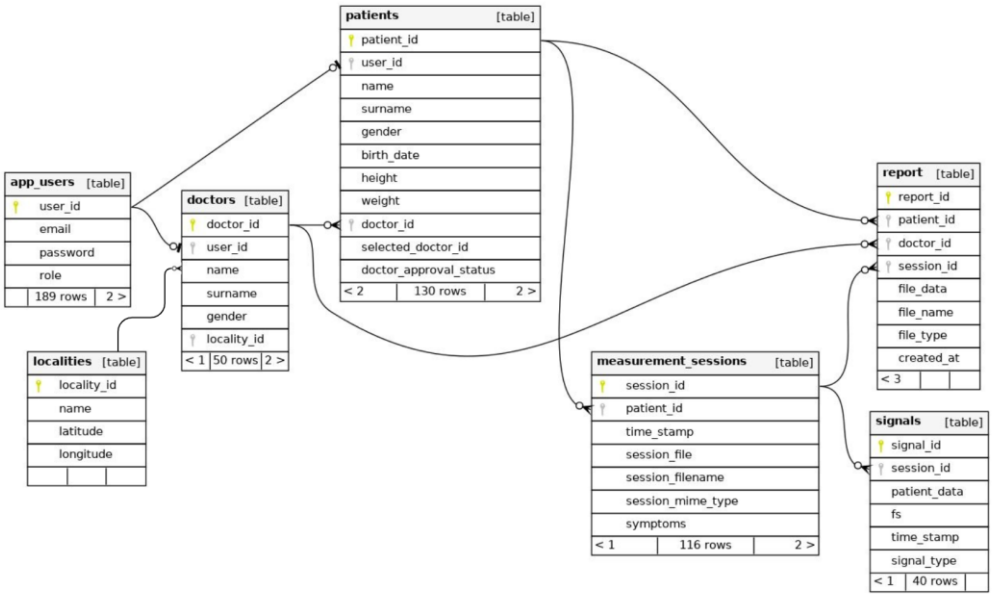
Important! As all the information is stored in a database, data will not be lost when shutting down the server platform.

5. Database

5.1. Database structure

The architecture of the database is designed around three main roles and their interactions: Patient, Doctor and Users and the measurement data generated by the patients:

- All authentication (Log-in/Sign-in) information is stored in the app_users table, which defines login credentials and role (ADMIN, DOCTOR, PATIENT) for each account.
- The patients and doctors extend the user records by storing their corresponding user information such as the name, surname or gender in the database and linking it back to app_users through a shared user_id. A patient can be assigned to a doctor, and doctors may supervise multiple patients (for more information refer to the DOCTOR MANUAL and/or CLIENT MANUAL).
- To support the data recollection by the patients, a measurement_session table which represents individual measurement events performed by a patient at a specific timestamp, and signals, which store the processed physiological data recorded during those sessions (EMG or ECG), were created.
- Each signal entry references both the patient and the session it belongs to, allowing full traceability of all collected data.



5.2. Connecting the database

To connect the database stored in your computer to the server you will need to have the .DUMP file that was given with the documentation, it must follow the exact template on *section 5.1*.

In order to import the database into postgresql, open the terminal where postgresql is downloaded and:

- Create the database: createdb –U postgres name-database

- Import the .dump to that empty database: pg_restore –U postgres –d name-database

During these steps, if you encounter an error mentioning another owner of the database, you will need to reassign the owner to you: REASSIGN OWNED BY previous-owner TO postgres;

Highlights must correspond to your admin parameters; these will be used to log into the system.

In addition, you might choose to use another owner instead of postgres as it is the owner established by default.

Now that the database is created and stored in your computer, you will need to go into *src/main/resources/application-local.yml* and change the following parameters:

```
spring:
  config:
    activate:
      on-profile: local

  datasource:
    url: jdbc:postgresql://localhost:5432/database_name
    username: YOUR_USERNAME
    password: YOUR_PASSWORD
    driver-class-name: org.postgresql.Driver
    hikari:
      schema: public

server:
  port: 8443
  ssl:
    enabled: true
    key-store: /src/main/resources/keystore.p12
    key-store-password: YOUR_KEYSTORE_PASSWORD
    key-store-type: PKCS12
    key-alias: YOUR_KEY_ALIAS

admin:
  username: ADMIN_USERNAME
  password: ADMIN_PASSWORD

operator:
  username: OPERATOR_USERNAME
  password: OPERATOR_PASSWORD

jwt:
  secret: JWT_SECRET
  expiration: 3600000
```

Highlights must correspond to your admin parameters; these will be used to log into the system