MotivSMA: Telemedicine Platform Documentation

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

Index

SMA patients monitoring

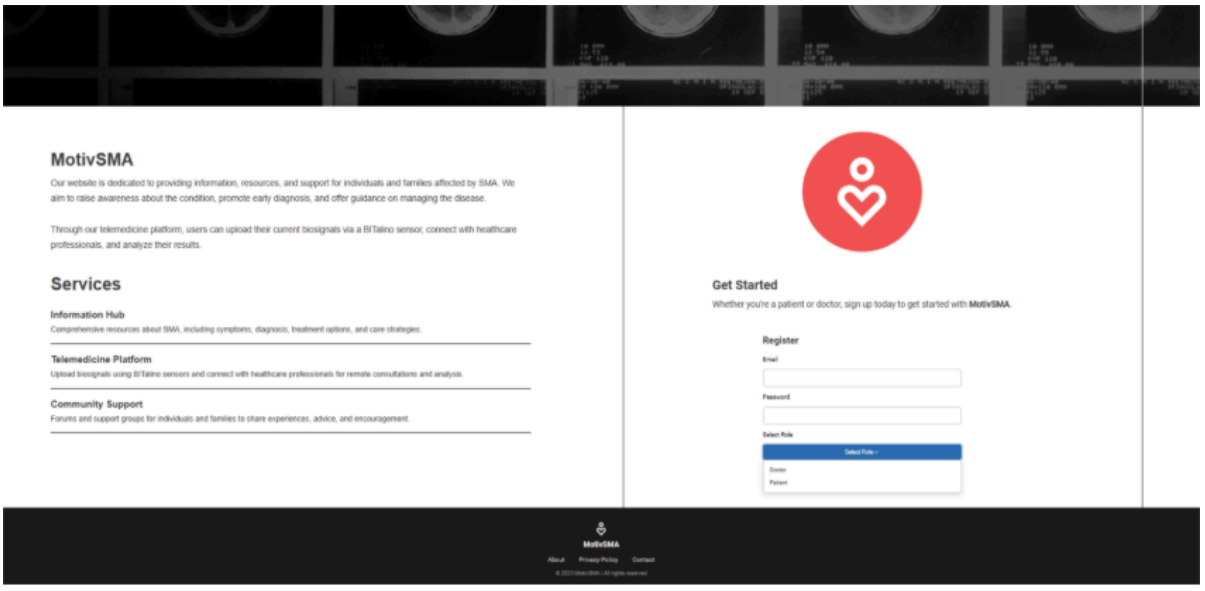
Overall Summary

This document gives an extensive explanation about a telemedicine application for SMA (Spinal Muscular Atrophy) patient monitoring. The application, named MotivSMA, is designed to support patients with that condition and to help the doctors achieve a better monitoring analysis and diagnosis.

MotivSMA is designed to monitor patients remotely, integrating bio-signal acquisition through a BITalino device, web applications, and a secure established server.

It provides two main applications where the patients will register their symptoms and biomedical signals, specifically electromyography (EMG) and electrocardiography (ECG) through a BITalino. Afterwards, the patient's information, its credentials, symptoms, and bio-signals will be sent to a doctor to fulfil the monitoring process.

The following information presents the main system architecture, installation instructions, system requirements, functionality regarding each application and communication system protocol.



Objectives

Main Objectives

MotivSMA aims to integrate the combination of web sources provided to the administrators, patients and doctors.

It is also focused on allowing:

1. User registration, either patients or doctors.

   Admins are not allowed to register for security purposes.

2. Ability to allow login options to registered users (patients, doctors and admin).
3. Update doctors' and patients' profile.
4. Request from a patient to a specific doctor.
5. Ability to upload and send recorded ECG/EMG and register symptoms from the patient's dashboard.
6. Management of clinically monitored sessions.
7. Bio-signals (ECG/EMG) recording through a BITalino device.
8. Allow doctors to supervise the patient's evolution SMA stages.
9. Generate PDF records for the clinicians to provide analysis and follow-up monitoring.

Secondary objectives

1. Provide intuitive interfaces for users, either patients or doctors.
2. Confidential and secure communication between the hospital and its patients, managing efficient data access control.
3. Storage and analysis of biomedical records saved in the hospital.
4. Easier real-time monitoring of patients from their homes, maintaining comfort for them.
5. Reduce the number of hospital citations.

Technical Requirements

Hardware

1. Conventional BITalino board device or (r)evolution version.
2. PC that enables Bluetooth connection, compatible with BlueCove.

Software

1. Current available Git version
2. JDK 22
3. Java 19+
4. Maven 3.8+
5. OS: Windows/Linux/macOS
6. Web framework: implementation of SpringBoot, CSS, HTML
7. Cipher and encryption implementations: BCrypt, HTTPS and JWT
8. PostgreSQL 15 +
9. HTTPS current version certificate

Managed Parameters

· **User credentials**: Each user must authenticate using their email (unique identifier) and password (encrypted and stored in the database) are required to access any part of the application.

· **Patient-Doctor Assignment Workflow**: the system manages the relationship between patients and doctors through a controlled request flow where:

- Patients request to be assigned to a doctor
- Doctor reviews the pending request
- Doctors accepts or rejects the request

· **Measurement Session Data**: Every medical measurement session must be stored with a unique ID, a timestamp (date and time of recording) and both symptoms and bio-signals logged. The user will be forced to log symptoms first and then the signals. For the latter, the order when uploading the files is not relevant as long as the user selected the right signal type when uploading.

· **Symptom Reporting:** Patients can select the symptoms to describe their current condition. These will inform about the state of the patient. We can find some general symptoms, motor and neuromuscular symptoms, speech and swallowing symptoms; respiratory, cardiac and neurological symptoms. The list of symptoms presented to the user is: fever, muscle weakness, slurred speech, breathing difficulty, tremors, swallowing difficulty, skeletal deformities, muscle atrophy, fasciculations, cardiac abnormalities, respiratory failure, decreased motor activity, joint contractures, abdominal distention, hand tremors, fatigue, poor feeding, speech delay, sleep disturbances, autonomic dysfunction, facial weakness, postural instability, head control difficulty, loss of motor skills.

· **Bio-Signal Data Upload**: Patients must upload the physiological signals recorded with a Bitalino as a .txt file. The latter will contain the ECG or EMG raw data and a sampling rate. The signals will be processed and stored in the database. In addition, a .csv file will be generated containing the patient's information as well as both signals already processed as raw data along with the signal's type.

Security

1. **HTTPS/TLS: Secure Transport Layer**

All communication between the client and the Spring Boot server is protected using HTTPS over TLS (Transport Layer Security).
This provides:

- Encryption
- Integrity: Ensures the transmitted data cannot be modified in transit
- Authentication: Confirms the client is communicating with the legitimate server thanks to the digital certificate configured in Spring Boot

2. **JWT (JSON Web Tokens): Application-Level Authentication**

After a user successfully logs into the system, the backend issues a JWT token, which contains the user's identity and roles (patient or doctor). This token is used not only for authentication but also to safely identify the user in endpoints without exposing raw IDs in plain text.

JWT provides:

- Stateless authentication: The server does not need to store session data. Each request contains all the information required to authenticate the user.
- Digital signature: Ensures the token cannot be forged or tampered with.
- Role-based access control: Each request to a protected endpoint must include the token, allowing the server to validate automatically who the user is and whether they are allowed to access that resource (doctor specific or patient-specific operations).
- All protected API endpoints require a valid JWT token in the HTTPS header: `Authorization: Bearer <token>`.

This ensures that only authenticated and authorized users can access sensitive actions, such as viewing medical data, uploading biosignal files, or managing patient–doctor relationships.

3. **BCrypt: Secure Password Hashing for the Database**

User passwords are never stored in plain text.
Before being stored in the database, each password is processed using the BCrypt hashing algorithm, which provides:

- Salted hashes: Automatically adds a random salt (piece of data) to each password to prevent rainbow-table attacks (password-cracking technique to reverse cryptographic hashes and reveal plaintext passwords) making every stored password unique.
- Slow computational cost: Makes brute-force attempts significantly more difficult.
- Adaptive security: The hashing cost can be increased as hardware becomes faster.

Actions Available in the Project Application

Patient

a. Register/log in
b. See his/her profile
c. Update his/her profile
    i. Name
    ii. Surname
    iii. Gender
    iv. Birthdate
    v. Weight (kg)
    vi. Height (cm)
d. See doctor's localities in a map
e. Requests a doctor
f. Start a measurement session
    i. Registers symptoms.
    ii. Download the SMA Server App to record signals
    iii. Uploads signals as files .txt
g. Download PDF reports generated by the patient where the patient can see all the information about a session including the patient's profile information, timestamp of the session, symptoms, both signals painted and the doctor's notes.
h. Log out.

Doctor

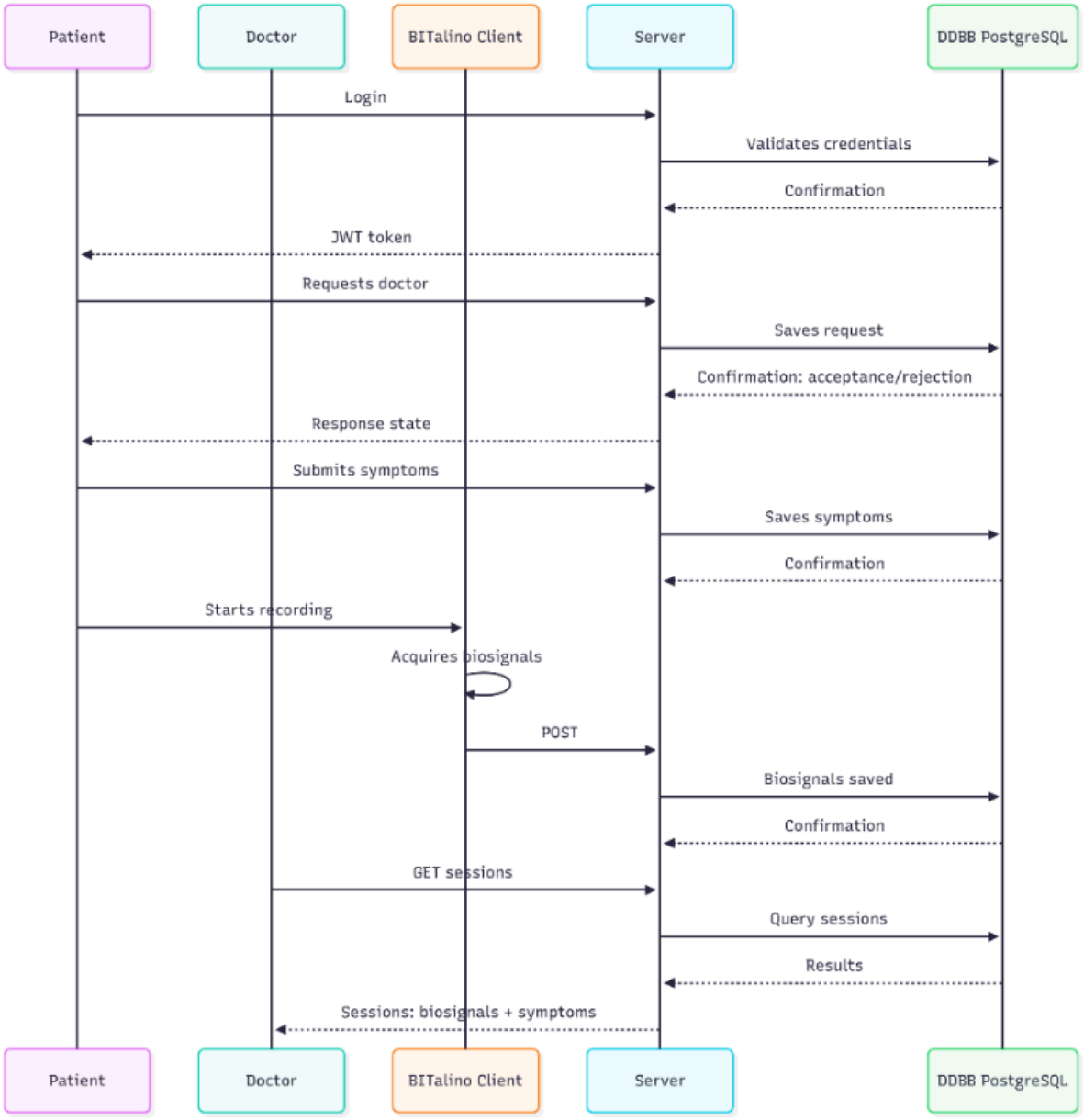
a. Register/log in
b. See his/her profile
c. Update his/her profile
d. See requests of patients along with their profile information such as name, surname or birthdate among others
e. Accept/Reject patient assignment requests
f. See assigned patients and their profile information
g. Access to all measurement sessions logged by each patient
    i. See all the details of a specific session: Timestamp, Symptoms recorded, both bio-signals
h. Generate a report (PDF file) with all the information present in each measurement session, along with an optional clinical comment (doctor's notes).
i. Access all reports generated.
j. Download a previously generated report. Allows the doctor to see all session reports which have been emitted by him/her.
k. Log out

Admin

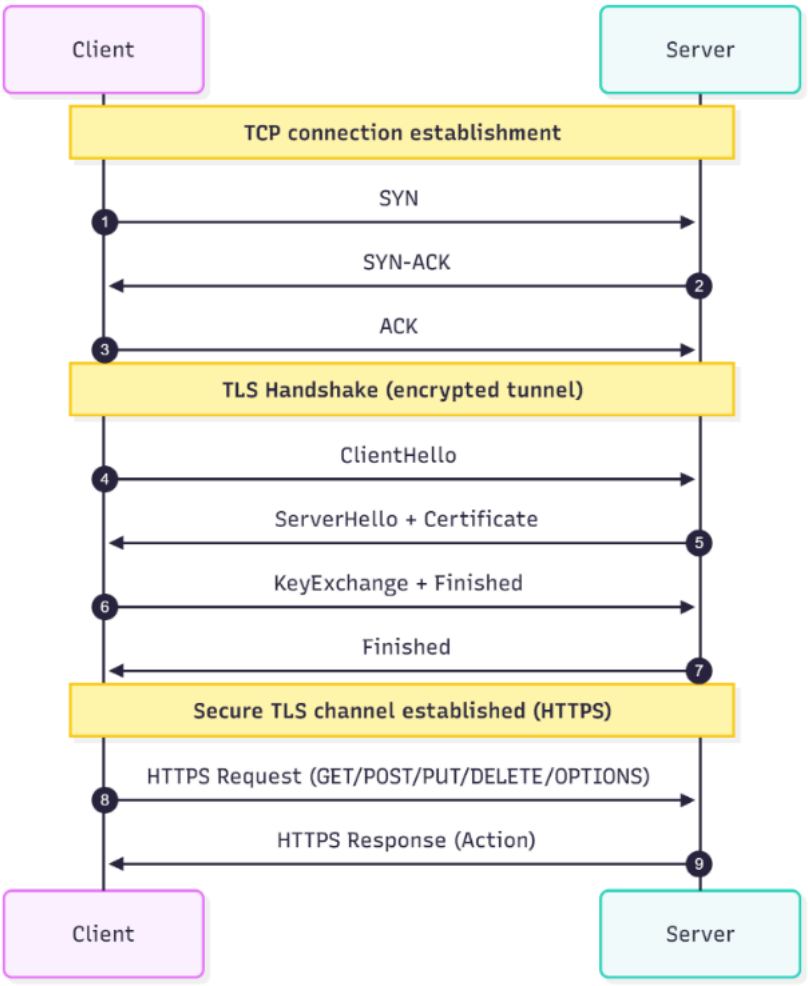
a. Log in
b. See CPU state
c. Visualize memory usage
d. Stop the hospital server, this must be
e. Management of DD.BB.: backup and restore it manually
f. Log out

It is necessary to mention that admin accounts are created manually for security purposes. Therefore, they do not have direct register access, as it happens with other user types that might select directly their role from the main web page.

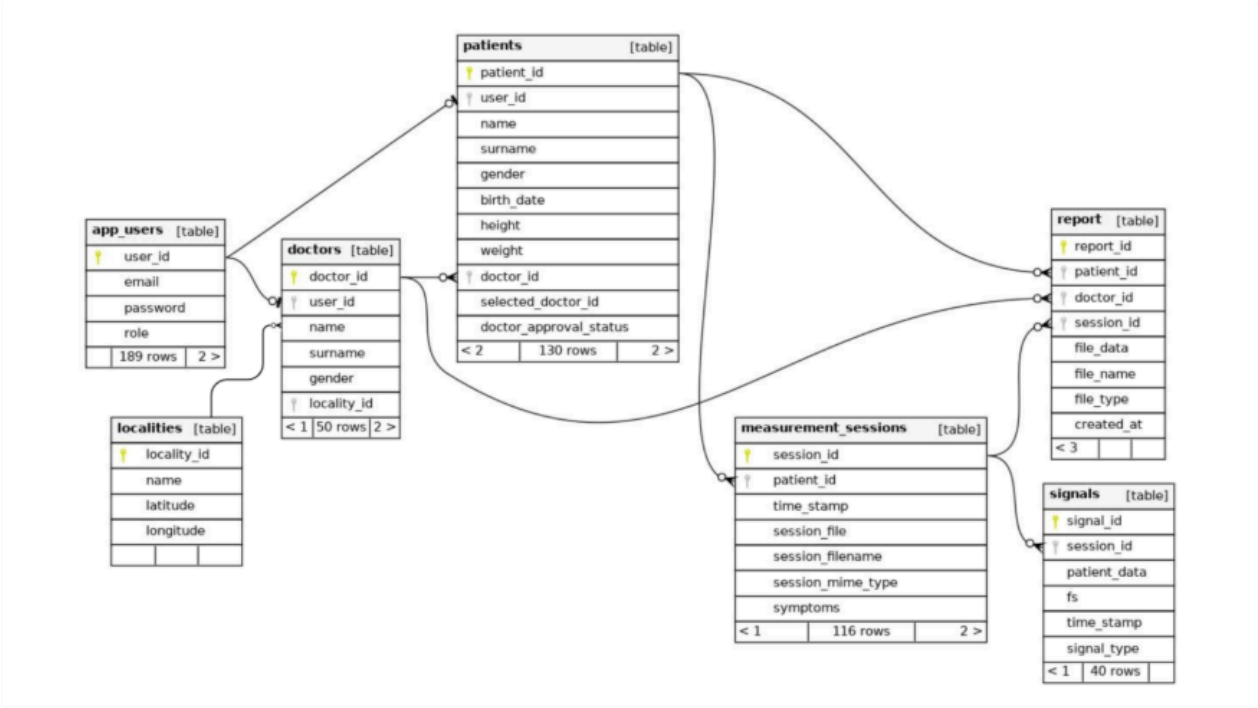
System Workflow



Communication Protocol

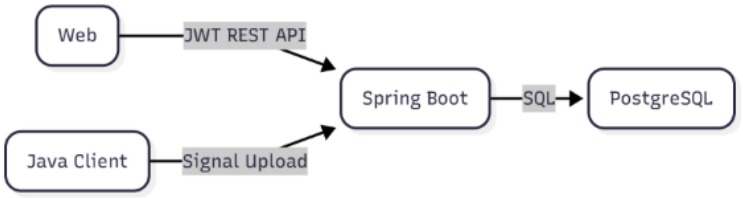


Database Schema



Developer Documentation

Main System Architecture



System components

1. Web frontend (HTML/CSS/JavaScript)

- Provides the full user interface for the register, login and general navigation for both patients and doctors.
- Offers dedicated dashboards where patients can upload data and doctors can review it.
- Includes interactive views for accessing medical sessions, viewing measurements, managing personal information, and handling doctor-patient assignments.
- Communicates securely with the backend using HTTPS and REST API calls.

2. Web Backend (Spring Boot Java application with Embedded Tomcat Server / REST APIs)

- Implements all server-side logic, data processing, and communication with the database.
- Uses a structured REST API architecture, exposing controllers that handle different functionalities:
    - Admin Controller - allows server management
    - Authentication Controller – verifies user credentials, issues JWT tokens, and manages access control.
    - Patient Controller:
        - Provide role-specific endpoints to access profiles, dashboards, and linked users
        - Receive physiological data (ECG/EMG files), sampling rates, and patient-reported symptoms.
        - Manages creation, retrieval, and timestamping of measurement sessions.
    - Doctor Controller:
        - Provide role-specific endpoints to access profiles, dashboards, and linked users
        - Handles patient session’s data and reports generation
- Ensures that all personal and medical data are encrypted, access-restricted, and properly authenticated

- Applies security mechanisms including HTTPS/TLS, JWT authentication, and BCrypt password hashing.
- Handles data storage, patient–doctor linking, and permissions for viewing or modifying medical information.

3. BITalino Client (Java Desktop Application)

- Connects to the BITalino hardware via Bluetooth using BlueCove
- Configures acquisition parameters for ECG/EMG signals (channels, sampling rate, etc.).
- Reads data frames sent by the device, converts them into file-based byte streams, and stores them locally.
- Allows the patient to download the captured bio-signal files and metadata (sampling rate, timestamps) to the computer to later upload it to the website (through the frontend) for medical evaluation by the assigned doctor.

4. BITalino device board (Biomedical Hardware)

- Establishes a Bluetooth connection with the client application before acquisition begins.
- Captures analog biomedical signals (ECG or EMG) and digitizes them using the device's internal ADC (Analog-to-Digital Converter).
- Packages the sampled data into Bluetooth-encoded frames and streams them continuously to the connected client.

Installation Process

To set up and run the application, two options are available.

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.

A/ Through Docker 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 as a .zip file: 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`

B/ Manual setup

1. Install the JDK, Java Development Kit, 22 version.
2. Configure the environment variables such that the system can detect Java by:
    a. Setting the JAVA_HOME variable.
    b. Adding the Java bin directory to the system PATH.
3. Install the latest available and current Git version.
4. Install Maven: latest Apache Maven version should be installed.
5. Verification of Maven installation: introducing the command `mvn -v`.
6. Repository needs to be cloned: executing the following command:

```
git clone https://github.com/alejandraoshea/sma-server
```

`git clone https://github.com/alejandraoshea/sma-client`

7. Navigate to the Project directory: *cd <project-directory>*
8. Build the Project
9. Execute on the project terminal: `mvn clean install`
10. Run the application: `mvn spring-boot:run` execution command
11. Certificate installation (refer to the *Certificate Installation Manual*)
12. Configure the application.yml and application-local.yml files as mentioned in the README.md document and in the manuals.

Backend Structure

1. **Controllers:**

Responsible for handling incoming HTTP requests and exposing the API endpoints used by the frontend and the BITalino client. Main controllers include:

- **Authentication Controller** – manages login, JWT token generation, and user verification.
- **Administrator Controller** – handles administrative tasks such as user management and role assignments.
- **Patient Controller** – provides patient-specific operations such as uploading signals, viewing sessions, and managing profile data.
- **Doctor Controller** – allows doctors to review assigned patients, access measurement sessions, and create medical reports.

2. **Services (Business Logic Layer):**

Encapsulate the application's core logic.

- Process domain operations (patient–doctor linking, session creation, report creation, etc.).
- Validate data, manage workflows, and coordinate communication between controllers and repositories.
- Ensure clean separation between the API layer and data persistence.
- Pdf generation for the reports

3. **Repository Layer (Data Persistence Layer)**

Responsible for interacting directly with the database using Spring Data JDBC.

· Repositories

- PatientRepository – manage patient profiles and related sessions, store and retrieve bio-signal sessions.
- DoctorRepository – manage doctors and assigned patients and their sessions as well as store doctor-written medical reports.

It also contains custom mappers to convert rows from a SQL query result into Java objects, in this case, into User objects (UserRowMapper) or into Patients objects (PatientRowMapper).

4. **Domain Model (Entities and Enums)**

Represents the data structure stored in the database and used across the system. Includes:
Entities:

- User, Patient, Doctor, Role
- MeasurementSession, Report, Signal,
- Locality

Enums:

- DoctorApprovalStatus, Gender, SymptomType, SignalType

These entities form the core of the data model, defining how medical information, users, sessions, and permissions relate to each other.

5. **Security Layer**

Provides a safe and controlled access environment for all backend resources.

- JWT Authentication – ensures stateless, token-based authentication through digitally signed tokens and secures all protected endpoints via `Authorization: Bearer <token>` headers.
- BCrypt Password Hashing – hashes and salts passwords before storing them in the database. It also ensures that stored credentials cannot be reversed or matched through rainbow tables.
- CORS Configuration – defines which frontend origins are allowed to access the API and prevents unauthorized cross-domain requests.

6. **Configuration layer**

Contains application-wide configuration classes that define runtime behaviour.

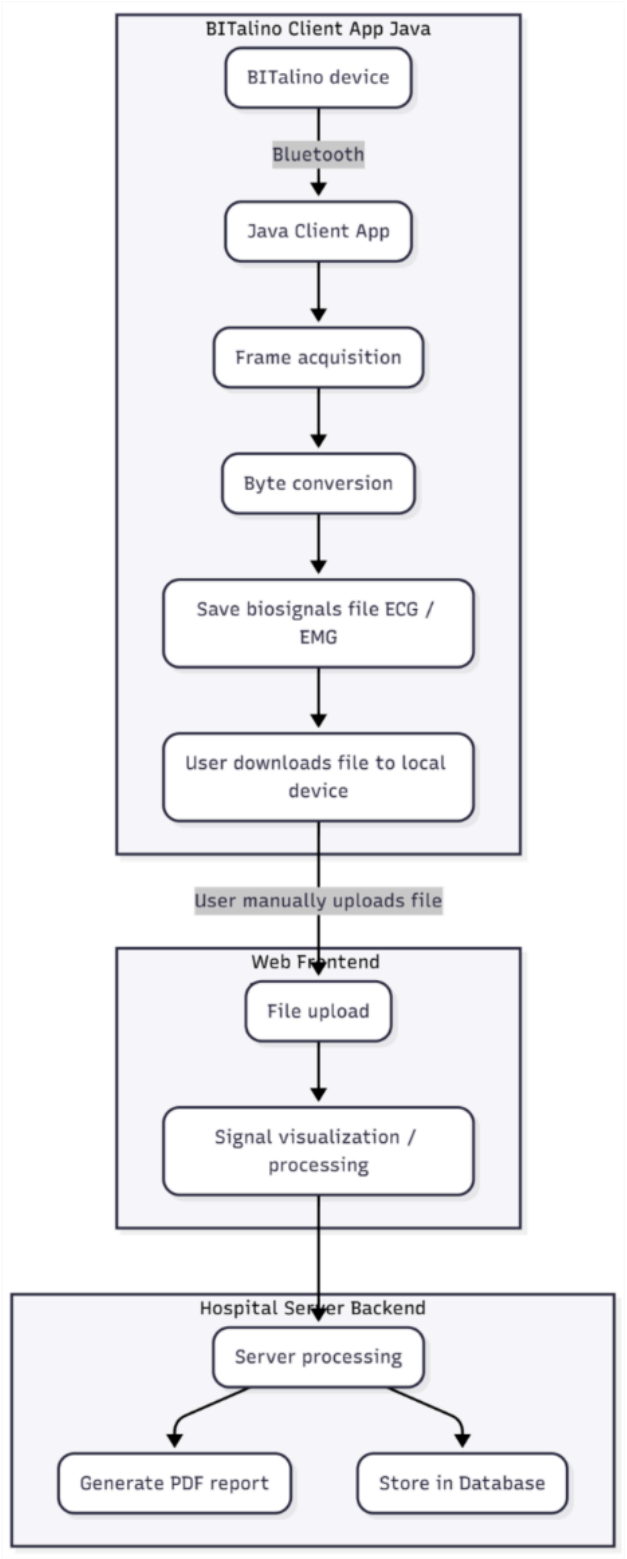
- SSL/TLS Configuration – sets the HTTPS certificate for encrypted communication and ensures all medical data is transmitted securely.
- CORS Settings – specifies allowed origins, headers, and HTTP methods and exposed headers.
- WebConfig – handles additional Spring Boot customizations, such as static resources or message converters.

Signals workflow

The workflow is the following:

1. Download SMA-Server application .msi
2. BITalino acquisition
3. Record bio-signals
    a. Save the signals in a file .txt
    b. Upload the .txt files containing the recorded signals to the web frontend client in “log signals” (once the user has started a session and logged a set of symptoms)
    c. Signals will be processed and saved in the database

BITalino Client workflow



System’s Operational Parameters

System Requirements and Configuration

Software Requirements

- Java 19+
- Maven 3.8+
- PostgreSQL database (given as a .dump)

Security / Networking

- HTTPS enabled using a keystore.p12 PKCS#12 certificate for secure communication
- Backend runs with SSL on port 8443 for development and testing

Port Usage

- 8443: Spring Boot backend with HTTPS (development mode).
- 5432: PostgreSQL database

Browser Configuration

- Google Chrome: import and trust the HTTPS certificate for local development