Implementación Técnica del Proyecto

Autor: Joaquín Gómez

Título del proyecto: ASISBIOM (Asistencia Biométrica)

Fecha de creación: 05/08/2024 (DD/MM/AAAA)

Contenidos

Prefacio

Este documento fue creado con la intención de detallar la implementación técnica del software del proyecto. Está dirigido a personas que tengan conocimientos acerca del tema o estén interesados en ver cómo se desarrolló la aplicación.

Si desea ver un documento más general o no conoce de qué trata el proyecto, le recomiendo que revise la **documentación oficial**.

Sin más que agregar, muchas gracias a todos los que hicieron posible el desarrollo de este proyecto, y a mis compañeros, Constanzo y Máximo.

Estructura del proyecto

Todo el proyecto se puede encontrar en el repositorio de Github. Consta de tres aplicaciones que van a funcionar de forma independiente, sin embargo una de ellas estará ligada al funcionamiento del sensor, ya que será la interfaz que permitirá comunicarse con el mismo.

Tecnologías utilizadas

Para la comunicación entre el sensor y las demás aplicaciones utilizamos el protocolo MQTT, que nos va a permitir una comunicación segura entre el sensor (que consta de un *Espressif ESP32-Wroom-32*) y el software, el cuál podrá ser accedido mediante un servidor.

Los datos de la aplicación son almacenados en una base de datos MySQL. Para el backend se utilizó Spring Boot (Java 17), con el apoyo de las siguientes librerías:

- → **spring-boot-starter-data-jpa**
- → **spring-integration-mqtt**
- → **spring-boot-starter-security**
- → **mysql-connector-j**
- → **spring-boot-configuration-processor**
- → **lombok**
- → **jjwt-api**
- → **jjwt-impl**
- → **jjwt-jackson**

(Se omitieron algunas librerías ya que no representan demasiada importancia).

Para el Broker MQTT utilizamos *mosquitto*, y lo configuramos para escuchar el puerto 1887/TCP y 8084/websocket. En el puerto 1887/TCP se configura la entrada y salida de mensajes para el Backend y para el microcontrolador. La implementación del puerto 8084/websocket es utilizada por el Front-End para mostrar los datos en tiempo real.

Aplicaciones que conforman el proyecto

La estructura de las aplicaciones es la siguiente:

- → **asisbiom_backend**: contiene el servidor backend, la función principal es servir una API-REST para que las demás aplicaciones accedan a los datos. Puedes ver los puntos de entrada del API-REST en la sección Interfaz de datos**.**
- → **asisbiom_frontend**: se trata de la interfaz visual de la pantalla que utilizamos para comunicar el sensor con el usuario. Contiene la funcionalidad que permite interactuar con el sensor, y lo hace mediante el broker MQTT en el puerto 8084/websocket.
- → **asisbiom_admin_front**: contiene la aplicación administrativa, el panel de control que permite al equipo directivo, secretarios, preceptores y profesores, a acceder a

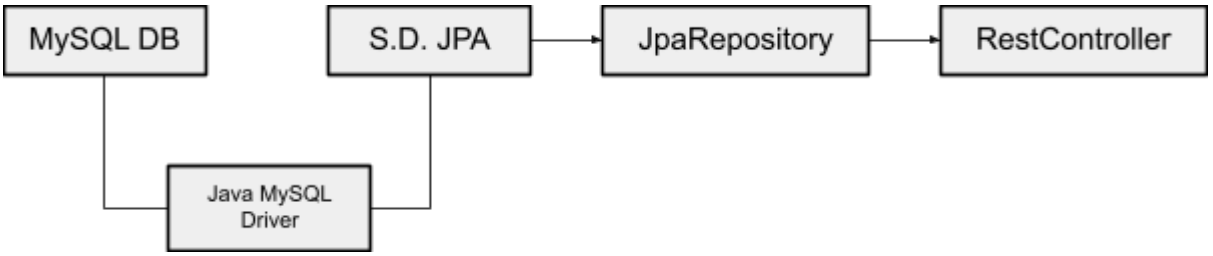
los datos, administrar los alumnos, estadísticas y demás funciones que se implementen. Esta aplicación web puede usarse en cualquier dispositivo.

Estructura del Backend

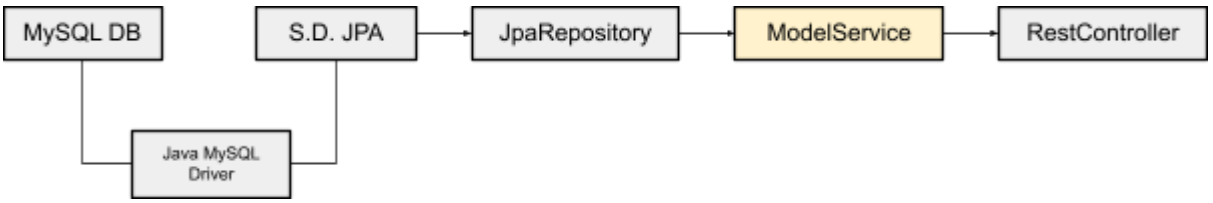
¿Cómo funciona el backend? El backend tiene una estructura sencilla, cuenta con los siguientes directorios, cabe aclarar que cada carpeta que represente un modelo dentro de la aplicación va a tener un controlador, un repositorio y posiblemente un servicio. El controlador sirve para exponer endpoints en el API-REST, el repositorio es una interfaz con la base de datos y el servicio contiene funciones útiles para facilitar el manejo de los datos o las acciones que se deban hacer.

→ **alumnos** – Contiene el controlador y servicios referidos a los alumnos (asistir, retirar, etc)
→ **asistencias** – Contiene archivos referidos a los registros de asistencias
→ **config** – Configuraciones web y de seguridad
→ **conteoasistencias** – Contiene archivos representativos del conteo de asistencias y estadísticas
→ **converter** – Archivos útiles para convertir valores
→ **cursos** – Archivos y modelo referidos a los cursos
→ **docentes** – Archivos y modelo referidos a docentes
→ **document** – Archivos referidos a documentos autogenerados (no implementado)
→ **filter** – Filtros para las peticiones (procesamiento de tokens, cookies, etc)
→ **general** – Archivos de uso general que no tienen lugar en otros directorios
→ **horarios** – Archivos y modelo referidos a horarios de entrada y salida
→ **materias** – Archivos y modelo referidos a materias y horarios de materias
→ **mqtt** – Archivos referidos a la conexión e implementación de mensajería por MQTT
→ **notas** – Archivos y modelo referidos a las notas para los alumnos
→ **notification** – Archivos y modelo referido a notificaciones en la aplicación de administración
→ **pdf** – Archivos y modelo cargar PDFs
→ **schedule** – Procesamiento de acciones automáticas programadas
→ **stats** – Estadísticas y extracción de datos
→ **user** – Archivos y modelo de usuarios. Registro y autenticación.

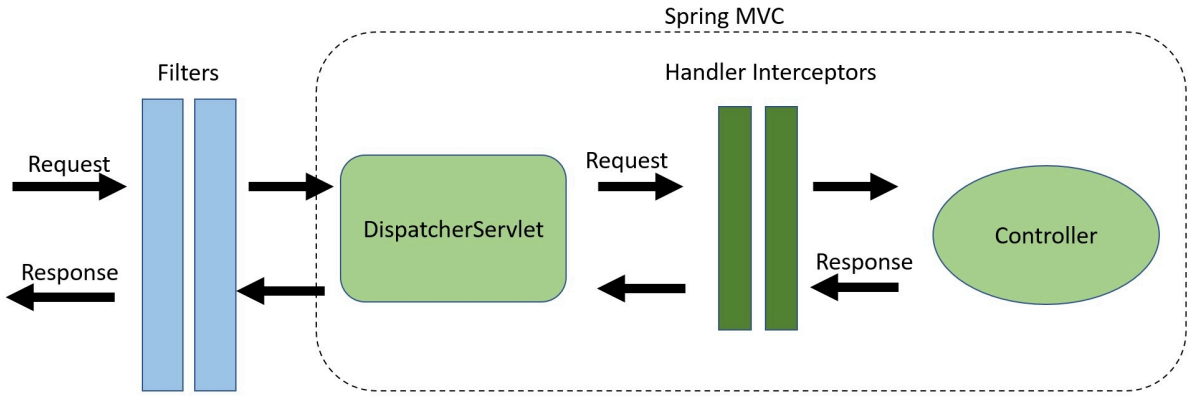
La implementación básica en cada controlador es la siguiente:



Es posible que para algunos métodos exista entre **JpaRepository** y **RestController** un método extra llamado **service**. Ya se mencionó que una clase *service* nos sirve para hacer acciones repetitivas o redundantes, lo cual simplifica los métodos en el controlador.



Un ejemplo claro en el código fuente es **StatsService.java** que se encuentra en el directorio */stats*. Para la parte de autenticación el programa utiliza filtros en las peticiones, que verifican la existencia de una **JWT** (JSON Web Token) para aquellas peticiones en las que es necesario procesar algún tipo de política de seguridad.



(fuente: https://velog.velcdn.com/images/ddangle/post/bf30bf45-7627-4e13-a219-84baf0372c87/image.png)

En la imagen se puede ver que "Handler Interceptors" representaría estos filtros anteriormente mencionados. Dentro de la carpeta */filters* se encuentran los filtros para las JSON Web Tokens, que deberán ser enviadas desde el Front-End para poder acceder a los datos que sean privados o requieran un permiso o rol. Si el Front-End no tiene la JWT, no cargará las páginas que requieran acceso. De intentar ser accedidas se mostrará un mensaje de error.

Resumen

Para resumir, la estructura del proyecto consta de tres aplicaciones, la principal (*asisbiom_backend*) se encarga de procesar solicitudes, procesar datos, e interactuar con la base de datos. Las otras dos aplicaciones son interfaces visuales que permiten la interacción con el sensor (*asisbiom_frontend*) y la administración (*asisbiom_admin_front*). Para comunicar las aplicaciones con el microcontrolador utilizamos un Broker MQTT llamado *mosquitto*, utilizando dos puertos para permitir la comunicación con todas las aplicaciones. (En entorno de desarrollo utilicé Docker para tener un contenedor con la base de datos).

Código del microcontrolador

Dentro de la carpeta **ESP32-MQTT-FINGERPRINT** se encuentra el código del microcontrolador. Se utilizó la librería de Arduino para simplificar la interacción, ya que a pesar de poder haber usado el propio RTOS (Real-Time Operating System) que nos brinda *Espressif*, se tomó la decisión de utilizar Arduino porque facilitaba el trabajo y no requería de mucho esfuerzo para lograr el mismo objetivo. Sin embargo no se descarta que sea una mejor alternativa, ya que se aprovecharía mejor la capacidad del microcontrolador si se usa el SDK que brinda el propio productor del chip.
De todas formas, Arduino resultó sencillo para las acciones básicas que debe hacer el microcontrolador junto con el sensor:

- → **Detectar si hay una huella dactilar en el sensor**
- → **Confirmar peticiones** (por ejemplo, al retirar un alumno, se debe confirmar que haya un miembro del personal de la escuela y éste debe colocar su huella dactilar)

→ **Manejar los procesos de forma síncrona** (ya que no se puede registrar y autenticar a una persona al mismo tiempo)

Protocolo Diseñado

En general, estos procesos constan de intercepción de mensajes (lo que causará una interrupción en el flujo normal de trabajo del sensor), verificación del protocolo o del mensaje (para ver si es correcto o inválido) y procesamiento o cumplimiento de la petición, para luego otorgar una respuesta válida. A continuación se define el protocolo que se creó para el propósito mencionado:

(MqttUtils.h)

```
// Estructura de 128 BITS (16 bytes)
// Se utiliza para la comunicación en toda la aplicación

typedef struct {
    uint32_t message_id;
    uint32_t sensor_id;
    uint32_t action;
    uint32_t student_id;
} mqtt_message;

// tipos de acción y respuesta (0x00 hasta 0xE9)
#define MQTT_ACTION_AUTH                            0x00
#define MQTT_ACTION_REGISTER                        0x01
#define MQTT_ACTION_CONFIRM                         0x02
#define MQTT_ACTION_PING                            0x03
#define MQTT_ACTION_PUT_FINGER                      0x04
#define MQTT_ACTION_REMOVE_FINGER                   0x05

// confirmaciones
#define MQTT_ACTION_COMPLETED                       0x06
#define MQTT_ACTION_CONFIRMATION_COMPLETE           0x07
#define MQTT_ACTION_REGISTER_COMPLETE               0x08

// mensajes de error (ACCION + 0xF0, de 0xF0 en adelante, no se necesitan más)
#define MQTT_ERROR_AUTH_FAILED                      0xF0
#define MQTT_ERROR_REGISTER_FAILED                        0xF1
#define MQTT_ERROR_CONFIRM_FAILED                   0xF2
```

El protocolo, verbalmente, consta de la conexión con el broker MQTT, y luego de la comunicación lado a lado de las partes que lo componen. Como MQTT se basa en suscripciones y publicaciones, los mensajes de todos los clientes conectados al broker se envían a todos los clientes. Esto sería un problema si solo se quiere enviar o recibir un mensaje a un solo dispositivo, por ende se implementa dentro de la estructura un entero de 32 bits llamado *sensor_id*. *message_id* sirve para verificar la concurrencia de los mensajes, sin embargo no tiene ningún propósito más que para verificar si los mensajes son correctos en orden cuando una comunicación entre dos partes es continua, por ejemplo, el registro de personal o una confirmación. *student_id* y *action* son los campos más importantes, ya que definen la acción a tomar (las cuales se definen abajo y tienen como prefijo “MQTT_ACTION_[acción]”) y un identificador para el estudiante o persona que haya colocado su huella o requiera de dicha acción.

Conexión a WIFI

Se utilizan algunas librerías que facilitan la intercepción de mensajes del broker (PubSubClient.h), otra para la comunicación serial entre el sensor de huella dactilar (Adafruit_Fingerprint.h) y una para conexión a internet en caso de fallar la conexión predeterminada.

```
// Credenciales del A.P. creado por el dispositivo (WifiManager.h)
#define WIFI_SSID_DEFAULT       "AP_SSID";
#define WIFI_PWD_DEFAULT        "AP_PWD";

// Credenciales por defecto
#define DEFAULT_WIFI_SSID       "DEFAULT_WIFI_SSID"
#define DEFAULT_WIFI_PWD        "DEFAULT_WIFI_PWD"
```

No hay nada más acerca del microcontrolador que sea necesario destacar (del lado del software).

Desafíos en la creación del software

Mucha parte del trabajo fue modelar los datos y crear modelos, funciones y controladores que ejecuten operaciones **CRUD** (Create, Read, Update, Delete), además de crear una lógica de seguridad e implementar la interfaz, lo cual fue una cuestión de tiempo y trabajo repetitivo. Lo que considero resultó más desafiante fue la comunicación entre el sensor y las demás partes, ya que requieren de sincronización y validación de ambos lados. Sin embargo no se descartan algunas deficiencias conocidas, de las que voy a hablar.

Deficiencias y puntos débiles

En lo que al sistema se refiere, se conocen los siguientes puntos débiles:

1. **El dispositivo de lectura no permite una memoria externa**, por lo tanto, a la hora de extraer huellas y guardarlas en su memoria interna carecemos de la posibilidad de centralizar los datos en un solo lugar (ya sea un disco, base de datos, etc.). Esto causa que no puedan existir dos sensores con la misma información, algo que en el futuro es reemplazable si se implementa un sistema con sensores biométricos que posean esta característica.
2. **La configuración de los dispositivos es manual**, por lo que si deseamos conectarnos a una red tenemos que configurar el microcontrolador o utilizar el Access Point que nos provee. Además, la ID del dispositivo debe ser modificada en el propio código del mismo. En un caso real de implementación se podría usar la dirección MAC o la IP, pero se decidió simplificar para poder implementarlo más rápido (por el momento). En resumen, la única forma de identificar un sensor es mediante la ID que se le asigna al cargar el código dentro del microcontrolador.
3. **Errores en comunicación serial**, que crean un sistema deficiente ya que cualquier interrupción o interferencia en las señales enviadas entre el microcontrolador y el sensor de huellas digitales podría causar un error.

4. **Requiere una red para funcionar**, y quizás la instalación o manejo de la misma junto con el despliegue de la aplicación resulte una tarea complicada para personas no-técnicas o sin conocimientos del tema. (Véase Distribución)

Interfaz de datos

El API-REST tiene los siguientes puntos de entrada que son necesarios para el funcionamiento de las aplicaciones

→ `/auth/v1/register`
→ `/auth/v1/user`
→ `/auth/v1/jwt-credentials-check`

→ `/api/sensor`
→ `/api/sensor/send-message`
→ `/api/sensor/parse`
→ `/api/sensor/confirm-retirar/{sensor_id}/{alumno_id}`
→ `/api/sensor/{id}`
→ `/api/sensor/last-message/{id}`
→ `/api/sensor/get-messages/{id}`

→ `/api/user`
→ `/api/user/{email}`
→ `/api/user/by-id/{id}`
→ `/api/user/editrole/{email}`

→ `/api/pdf`
→ `/api/pdf/{name}`

→ `/api/notification/{receiver_username}`

→ `/api/nota`
→ `/api/nota/{id_alumno}`

→ `/api/materia`
→ `/api/materia/{id_curso}`

→ `/api/horario`
→ `/api/horario/{curso}/{div}`
→ `/api/horario/hoy`
→ `/api/horario/hoy/{curso}/{div}`

→ /api/estadistica
→ /api/estadistica/{curso}
→ /api/estadistica/descarga/{cursoId}
→ /api/estadistica/cantidades/personal
→ /api/estadistica/cantidades/personal-presentes
→ /api/estadistica/cantidades/alumnos
→ /api/estadistica/cantidades/alumnos-presentes
→ /api/estadistica/_initialize
→ /api/estadistica/editar/{id}

→ /api/curso
→ /api/curso/{curso}/{division}
→ /api/curso/turno
→ /api/curso/_initialize

→ /api/asistencia/{curso}/{div}

→ /api/alumno
→ /api/alumno/{id}
→ /api/alumno/{curso}/{div}
→ /api/alumno/stats
→ /api/alumno/stats/{alumno_id}
→ /api/alumno/search
→ /api/alumno/list-cursos
→ /api/alumno/latestid
→ /api/alumno/documento/{dni}
→ /api/alumno/remover/{idAlumno}
→ /api/alumno/tardanza/{id}
→ /api/alumno/retirar/{idAlumno}
→ /api/alumno/registrar
→ /api/alumno/asistir/{id}

→ /api/retiro
→ /api/retiro/{id}
→ /api/retiro/editar/{id}

Modelos y Base de Datos

A continuación, los modelos u objetos de cada tipo, los cuales representan una tabla en la base de datos.

InnerConteoAsistencia

id
nombreCompleto
tardanza
retirado
presente
diasHabiles
inasistencias1
inasistencias2
inasistencias3

Sensor
- id
- ip
- ubicacion

Materia
- id
- nombre

Curso
- id
- division
- curso
- turno

User
- id
- role
- email
- phone
- nombreCompleto
- dni

Alumno
- id
- curso
- nombreCompleto
- telefono
- correoElectronico
- dni

Retiro
- id
- razon
- fecha
- alumno
- profesor

Nota
- id
- alumno
- nivel_urgencia
- asunto
- fecha
- vencimiento
- contenido

Horario

```
id
curso
horarioEntrada
horarioSalida
clase
dia
valorInasistencia
```

Asistencia

```
id
alumno
asistencia
horarioEntrada
horarioRetiro
fecha
tardanza
retirado
razonRetiro
 dia
enabled
```

ConteoAsistencia

```
id
alumno
tardanzas
retiros
diasHabiles
inasistencias1
inasistencias2
inasistencias3
inasistenciasEnRacha
racha
```

Se omitieron algunos modelos como los **DTOs** (Data Transfer Object) o relaciones.
Para consultar la especificación podés descargar el documento llamado **rest_api_especificacion.pdf** dentro de la carpeta **/docs**. Podés consultar el repositorio de *Github*: https://github.com/Joagz/eetp612-asisbiom/tree/dev/docs/pdf.

Distribución

Por el momento el software mencionado está en desarrollo. Para poder utilizarlo podés seguir la guía detallada en el repositorio. No se espera que un usuario común ejecute el programa de momento, sin embargo, no se descarta que en un futuro se lance una distribución del mismo.