Universidad de Sevilla
Escuela Técnica Superior de Ingeniería Informática
Departamento de Lenguajes y Sistemas informáticos
Profesor: Luis Miguel Soria Morillo

Memoria segundo entregable Hcleaning

Fecha: 26/05/2021

Miembros del equipo: Joaquín Almarcha Conejero

Índice

1. Descripción del proyecto

Controlar la contaminación en varias áreas hoy en día puede ser un gran reto, existen multitudes de empresas que se dedican a la desinfección de distintas salas, estudios, hospitales, oficinas etc.

Las salas blancas, son un espacio cerrado, pensado para mantener unos niveles de contaminación mínimos o casi nulos. Sin embargo, uno de los grandes retos que se enfrenta hoy en día la tecnología, es eliminar cualquier agente infeccioso de todas estas salas, evitando de esta manera, la reinfección de pacientes que se someten a algunas intervenciones quirúrgicas.

El objetivo del proyecto de la evaluación continua es el control y la prevención de la aparición de hongos (concretamente el hongo aspergillus). Este hongo, se encuentra en suspensión en el ambiente, por ello, mi propuesta es realizar o diseñar un sistema de control aplicado a dichas salas blancas, el cual controla, la aparición de hongos aspergillus.

Adicionalmente, se controlarán a los usuarios que accedan a las salas blancas y se creará un registro de las operaciones planificadas.

A continuación, se realiza una descripción de los pasos que se han ido desarrollando con el objetivo de la creación de dicha herramienta.

2. Hardware empleado en la solución

Se adjunta los elementos Hardware que se están empleando en el proyecto

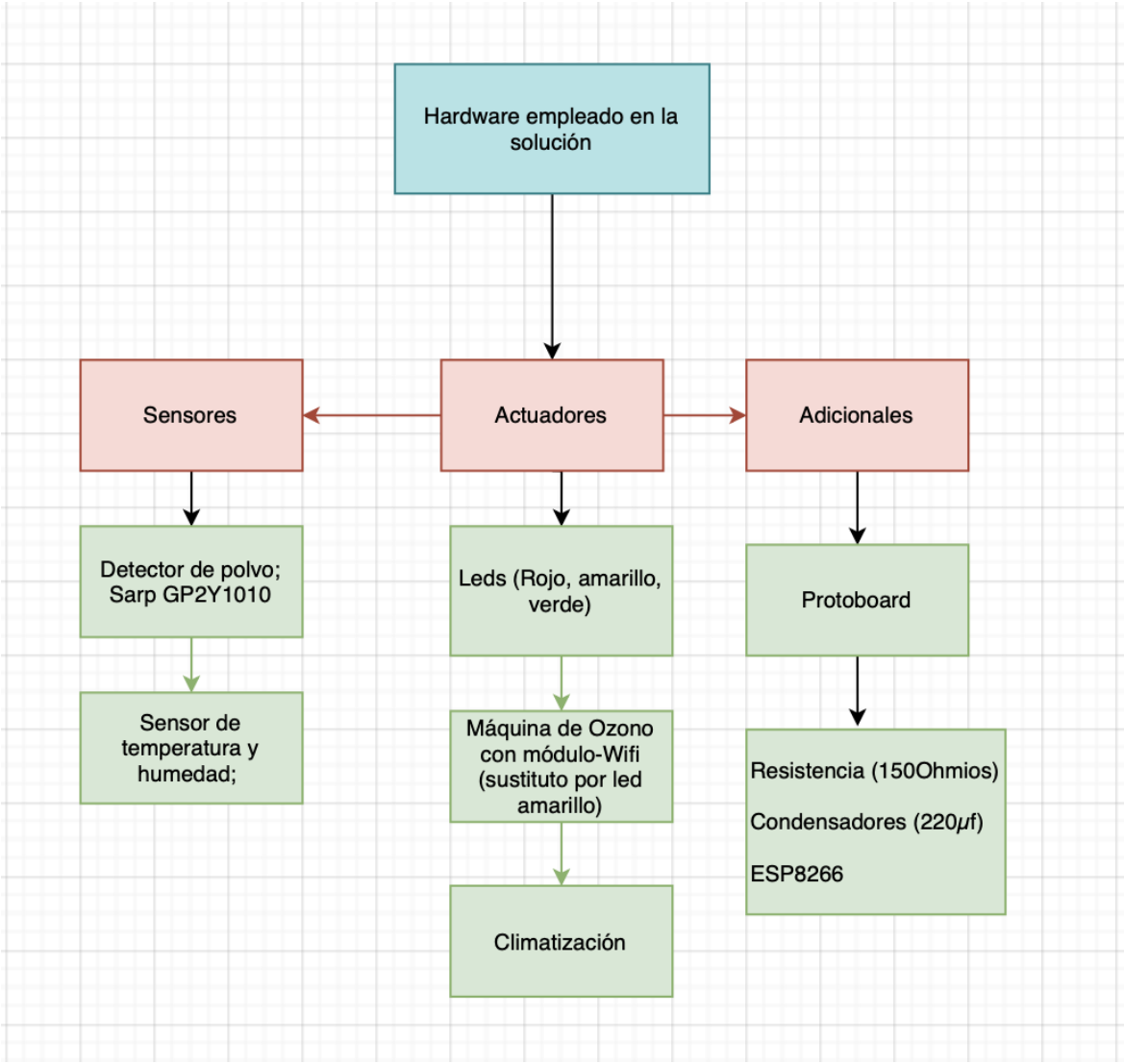
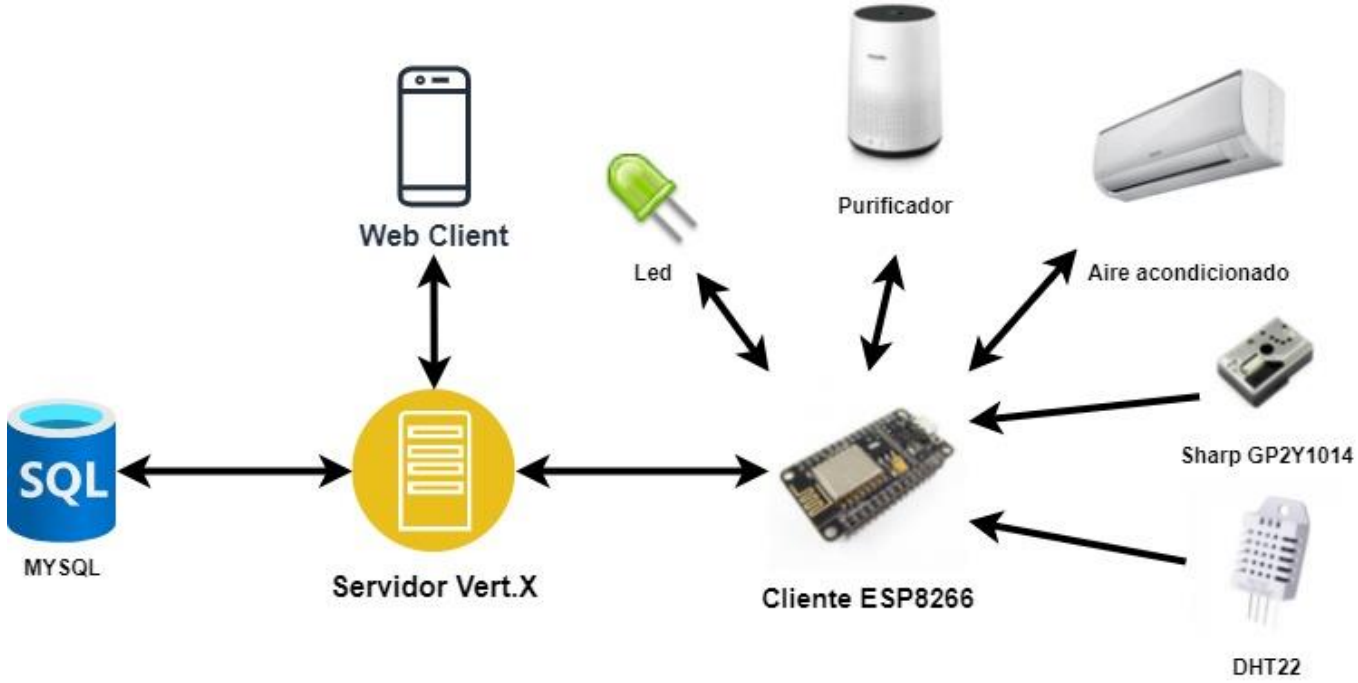


Figura 1: Elementos Hardware en la solución

3. Diagrama de conexión de los componentes

A continuación, se adjunta el diagrama de conexión de los componentes

Figura 2: Diagrama de conexión de los componentes



4. Modelo de la base de datos

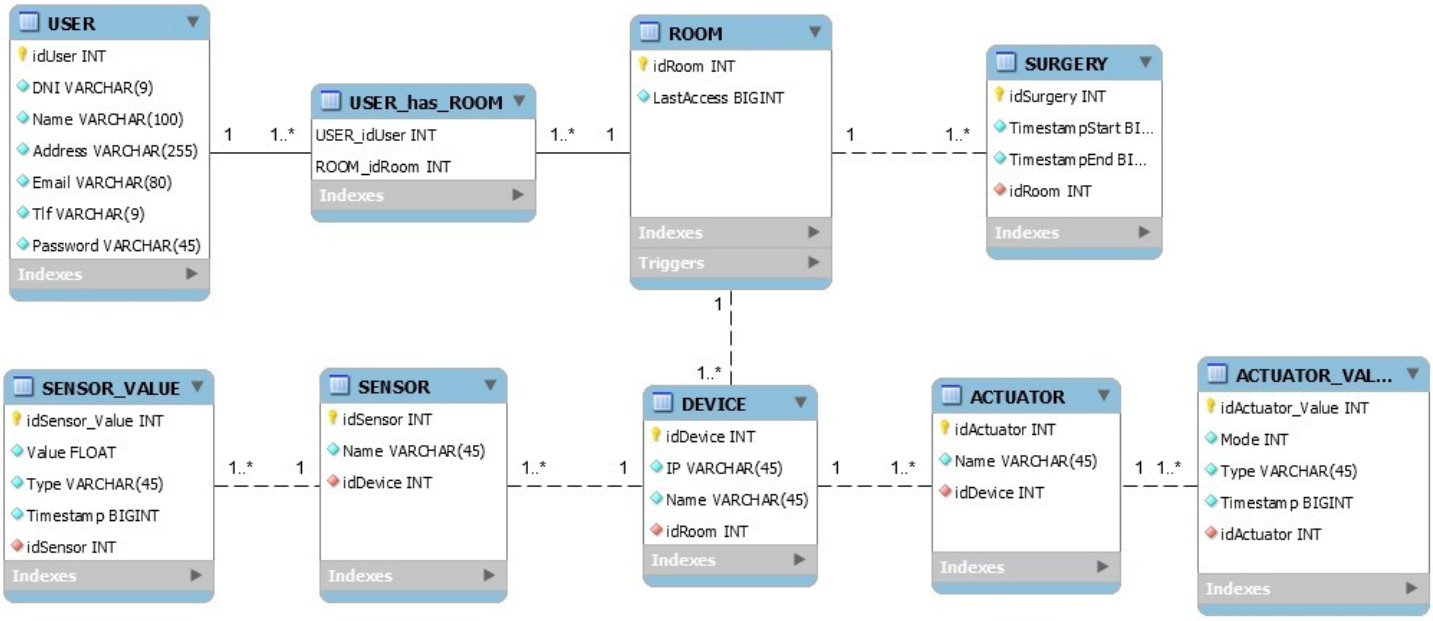


Figura 3: Modelo de la base de datos

La base de datos almacenará información sobre los usuarios de la API Rest, las operaciones programadas. Las salas se controlarán con sensores y actuadores y la información de cada uno de estos dispositivos.

Existe una clase DEVICE que controla los dispositivos (actuadores y sensores). Estos dispositivos serán colocados en salas blancas. Se controla el acceso de los usuarios “USER” a estas salas y se llevará un control sobre las operaciones “SURGERY” programadas para tener las salas totalmente listas.

5. Descripción de los métodos de la API Rest

El diseño de una API REST, se trata de una creación de una interfaz empleando unas reglas muy bien definidas con el objetivo de interactuar con un sistema y obtener su información.

El método REST se basa en la separación de su API en recursos lógicos, donde estos se manipulan mediante peticiones HTTP con **métodos GET, POST, PUT y DELETE.**

Métodos GET

Para realizar las consultas a las tablas se ha realizado un método GET para cada tabla que filtra por "id".
Casos específicos:

- SURGERY: Se dispone de métodos GET que filtran por "fecha" y "idRoom" para obtener la información de todas las operaciones programadas para una determinada sala.

- USER: Se dispone de un método GET para obtener los usuarios por el "DNI", facilitando de esta manera la labor de identificación de un usuario.

- DEVICE, SENSOR, ACTUATOR: Los dispositivos disponen métodos GET que filtran por "idSensor" obteniendo de esta manera la información acerca del sensor/actuador a consultar

.

Métodos POST

Se han implementado métodos POST para la inserción de datos. Esta inserción de datos se realiza para unos casos determinados:

- Añadir un nuevo usuario
- Añadir un nuevo dispositivo
- Añadir valores de un sensor
- Añadir valores de un actuador
- Añadir una nueva operación
- Iniciar sesión en la aplicación

Métodos PUT

Se han implementado métodos PUT para la actualización de los datos existentes.
Los datos de las siguientes tablas pueden ser actualizados:

- SURGERY
- USER
- DEVICE

Métodos DELETE

Se ha implementado un método DELETE para la gestión de las operaciones.

**IMPLEMENTACIÓN*

Entidad Device

- Peticiones GET

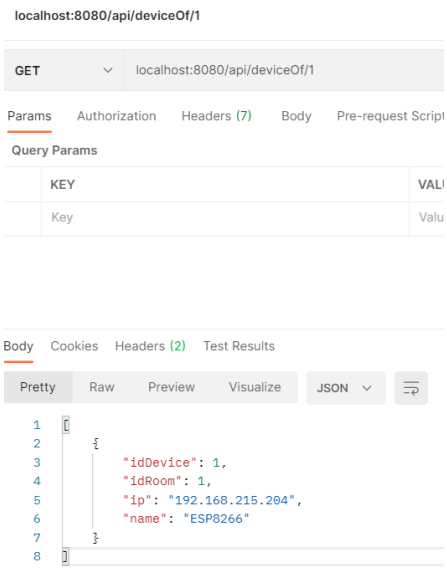
Función: getDevice

Url: ("/api/device/:idDevice")

Descripción: Devuelve la información de un dispositivo filtrado por su ID.

Cuerpo de la respuesta:


```
{
"idDevice": 1,
"idRoom": 1,
"ip": "192.168.215.204",
"name": "ESP8266"
}
```

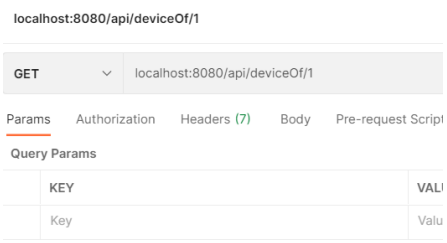
Función: getDeviceRoom

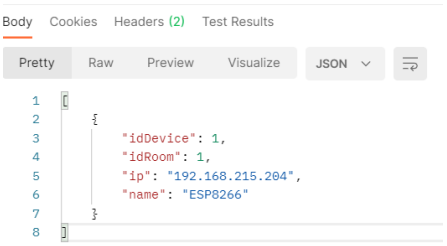
Url: ("/api/deviceOf/:idRoom")

Descripción: Devuelve la información de un dispositivo filtrado por salas.

Cuerpo de la respuesta:

```
    {
"idDevice": 1,
"idRoom": 1,
"ip": "192.168.215.204",
"name": "ESP8266"
    }
```



- Peticiones POST

Función: postDevice

Url:("/api/device/new")

Descripción: Inserta un nuevo dispositivo.

Entidad Sensor

- Peticiones GET

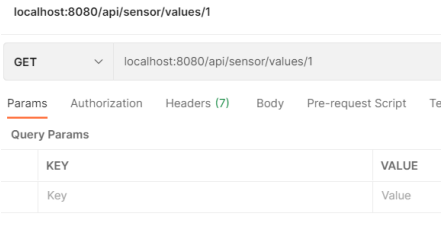
Función: getSensorValues

Url: ("/api/sensor/values/:idSensor")

Descripción: Devuelve la informacion de un sensor.

Cuerpo de la respuesta:


```
    {
"idSensor_Value": 1,
"idSensor": 1,
"value": 0.0,
"type": "Dust",
"timestamp": 1621929600
}
```

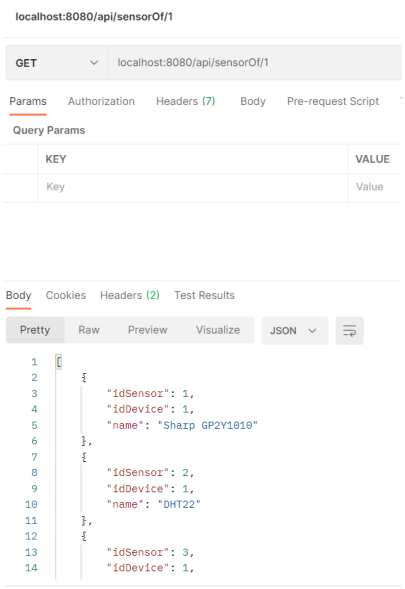
Función: getSensorDevice

Url: ("/api/sensorOf/:idDevice")

Descripción: Devuelve la información de todos los sensores asociados a un dispositivo.

Cuerpo de la respuesta:


```
    {
    "idSensor": 1,
    "idDevice": 1,
    "name": "Sharp GP2Y1010"
  },
  {
    "idSensor": 2,
    "idDevice": 1,
    "name": "DHT22"
  },
  {
    "idSensor": 3,
    "idDevice": 1,
    "name": "DHT22"
```

- Peticiones POST

Función: postSensorValues

Url:("/api/sensor/values/:idSensor")

Descripción: Inserta los valores de un sensor

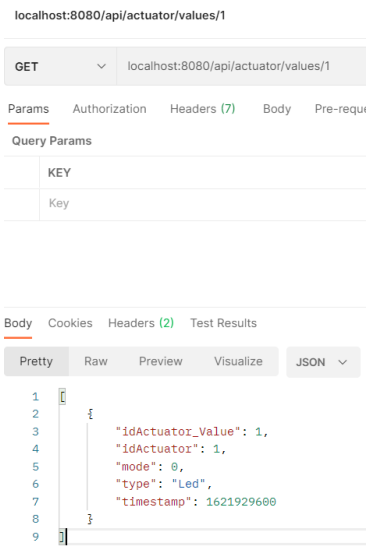
<u>Entidad Actuador</u>

- Peticiones GET

Función: getActuatorValues

Url: ("/api/actuator/values/:idActuator")

Descripción: Devuelve la información de un actuador.



Cuerpo de la respuesta:


```
{
"idActuator_Value": 1,
"idActuator": 1,
"mode": 0,
"type": "Led",
"timestamp": 1621929600
}
```

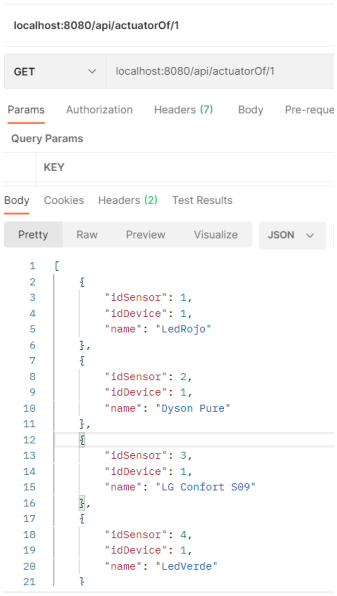

Función: getActuatorDevice

Url: ("/api/actuatorOf/:idDevice")

Descripción: Devuelve la información de todos los acctuadores asociados a un dispositivo.

Cuerpo de la respuesta:


```
{
"idSensor": 1,
"idDevice": 1,
"name": "LedRojo"
},
{
"idSensor": 2,
"idDevice": 1,
"name": "Dyson Pure"
},
{
"idSensor": 3,
"idDevice": 1,
"name": "LG Confort S09"
},
{
```

```
    "idSensor": 4,
    "idDevice": 1,
    "name": "LedVerde"
  }
```


Peticiones POST

Función: postActuatorValues

Url:("/api/actuator/values/:idActuator")

Descripción: Inserta los valores de un actuador

<u>Entidad Surgery</u>

- Peticiones GET

Función: getSurgery

Url1: ("/api/surgery/:idSurgery")

Url2: ("/api/surgery/:idSurgery/:timestampStart")

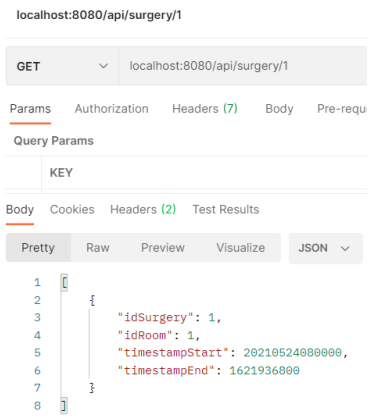
Url3: ("/api/surgery/:idSurgery/:timestampStart/:idRoom")

Url4: ("/api/surgery/:idSurgery/:idRoom")

Descripción: Devuelve la información de las operaciones con sus respectivos filtros.

Cuerpo de la respuesta:


```
  {
"idSurgery": 1,
"idRoom": 1,
"timestampStart": 20210524080000,
"timestampEnd": 1621936800
  }
```

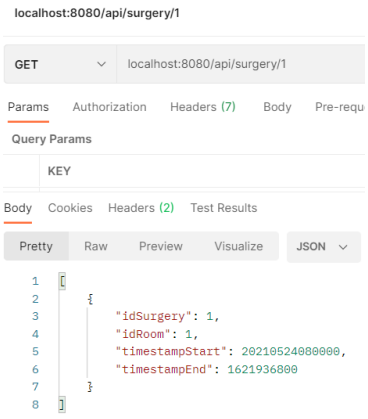
Función: getScheduleSurgery

Url: ("/api/scheduleSurgery/:timestampStart")

Descripción: Devuelve la información de las operaciones programadas para un dia especifico.

Cuerpo de la respuesta:


```
  {
"idSurgery": 1,
"idRoom": 1,
"timestampStart": 20210524080000,
"timestampEnd": 1621936800
  }
```

Peticiones POST

Función: postSurgery

Url:(" /api/surgery/new")

Descripción: Inserta una nueva operación

Peticiones PUT

Función: putSurgery

Url:("/api/surgery/:idSurgery")

Descripción: Actualiza los valores de una operación

Peticiones DELETE

Función: deleteSurgery

Url:("/api/surgery/:idSurgery")

Descripción: Elimina una operación

```
        router.get("/api/scheduleSurgery/:timestampStart").handler(this::getScheduleSurgery);

                /*
                 *
                 *                                      POST
                 *
                 *
                                                */
                router.post("/api/user/new").handler(this::postUser);
                router.post("/api/device/new").handler(this::postDevice);


        router.post("/api/sensor/values/:idSensor").handler(this::postSensorValues);

        router.post("/api/actuator/values/:idActuator").handler(this::postActuatorValues);

                router.post("/api/surgery/new").handler(this::postSurgery);
                router.post("/api/login").handler(this::postLogin);

                /*
                 *
                 *                                      PUT
                 *
                 *
                                                */
                router.put("/api/user/:idUser").handler(this::putUser);

        router.put("/api/surgery/:idSurgery").handler(this::putSurgery);
```

```
        /*
         *
         *                                    DELETE
         *
         *
                                          */

    router.delete("/api/surgery/:idSurgery").handler(this::deleteSurgery
);
```

6. Descripción de los mensajes MQTT

Se trata de un protocolo basado en TCP/IP como base de la comunicación. En el caso de la MQTT, hay que tener en cuenta que cada conexión se mantiene abierta siendo reutilizada en cada comunicación.

Su funcionamiento se basa en un servicio de mensajería push con patrón publicador/suscriptor (pub-sub). Varias fuentes de información afirman que este tipo de infraestructura utiliza una conexión cliente con un servidor central conocido como broker.

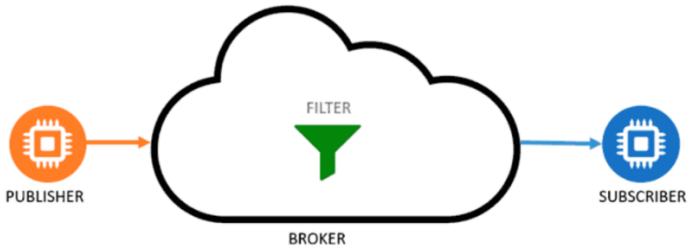


Figura 4: Diagrama descripción mensajes MQTT

En el presente proyecto, nuestra estructura de los mensajes MQTT, se representa mediante el siguiente diagrama.

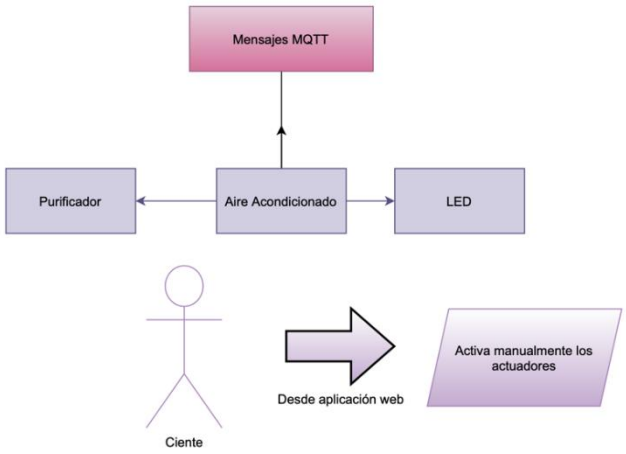


Figura 5: Diagrama descripción mensajes MQTT (proyecto)

En resumen, el cliente, desde la aplicación web, active de manera manual los actuadores.