**Memoria:**

Realizada por: ***Roberto Arasti Blanco, Guillermo Alcuaz Temiño, Diego Garrido Calvo y Elsa Tolín Carrasco***

*Entrega a fecha:* ***17/1/2021***



índice

[Manuales: 3](#_Toc61797785)

[Instalación 3](#_Toc61797786)

[Uso de la aplicación 3](#_Toc61797787)

[Documentación del protocolo de comunicaciones cliente-servidor 4](#_Toc61797788)

[Arquitectura y diseño del servicio sensor: 8](#_Toc61797789)

[Arquitectura y diseño del cliente: 11](#_Toc61797790)

## 

Manuales:

### Instalación

Una vez descargada la aplicación, a través de la consola se va a la carpeta que contiene la aplicación y ejecutamos uno por uno todos los instaladores de los cuatro componentes de la aplicación. Para ello, introducimos los siguientes comandos en la consola:

* **./components/dms2021core/install.sh**
* **./components/dms2021auth/install.sh**
* **./components/dms2021client/install.sh**
* **./components/dms2021sensor/install.sh**

Para ejecutar la aplicación tras instalarla simplemente tenemos que introducir el comando **dms2021client**.

Si disponemos de Docker y no queremos instalar la aplicación, si no que optamos por solo usarla podemos ejecutar los siguientes comandos:

* **docker-compose-f Docker/config/dev.yml up -d**: Este comando ejecuta la aplicación con Docker.
* **docker cointainer attach dms2021client**: Este segundo comando monta el cliente en el terminal que estamos usando, tras esto pedirá al usuario que se introduzca usuario y contraseña, de esta forma inicia sesión.

Para parar la aplicación y eliminar los containers podemos ejecutar el siguiente comando:

* **docker-compose -f docker/config/dev.yml rm -sfv**

### Uso de la aplicación

Ya con la sesión iniciada, pide al usuario que seleccione una opción del menú de las cuatro opciones que ofrece. Las opciones que ofrece son las siguientes:

* “Crear usuario”: Si el usuario que está logueado en el sistema tiene permisos para hacerlo, tiene la opción de crear un nuevo usuario y guardar a este en la base de datos. Al seleccionar esta opción se pedirá introducir un nuevo usuario y una contraseña.
* “Manejar permisos de usuario”: Esta opción permite conceder y retirar permisos a un usuario. Si se selecciona esta opción, esto llevará a un menú en el que se podrán modificar los permisos del usuario.
* “Cambiar las reglas del sensor”: Si el usuario elige esta opción le llevará a una lista de opciones/ajustes del sistema que se podrán modificar.
* “Mostrar últimos valores monitorizados”: En esta opción se pueden ver las últimas 5 entradas de la base de datos de los valores detectados por el sensor.

De todas estas opciones, hemos implementado las dos primeras.

Documentación del protocolo de comunicaciones cliente-servidor

El API REST es la interfaz encargada de permitir interactuar con los servicios mediante HTTP. En esta comunicación los *endpoints* toman los roles de Cliente y Servidor.

En el Servidor definimos las operaciones que va a poder pedir el Cliente, en el cual definimos la llamada a la función que realizamos en el Servidor a través de HTTP. En la comunicación entre el Cliente y el Servicio de Autenticación las funciones son:

* **Crear usuario**
  + Ruta: /users
  + Método: create\_user
  + Parámetros
    - Username
      * Tipo: String
      * Por dónde se pasa: se encapsula en el formulario de la request
      * Descripción: nombre de usuario que queremos crear
    - Password
      * Tipo: String
      * Por dónde se pasa: se encapsula en el formulario de la request
      * Descripción: contraseña del usuario que queremos crear
    - Session
      * Tipo: String
      * Por dónde se pasa: se encapsula en el formulario de la request
      * Descripción: usuario logueado en el momento
  + Respuestas
    - 200: Usuario creado satisfactoriamente
    - 400: ValueError
    - 401: El usuario no tiene los privilegios necesarios
    - 409: El usuario ya existe
    - 500: HTTPexception (Server error)
  + Descripción: Si el usuario tiene permisos para hacerlo, pedimos al mismo que introduzca un nuevo nombre de usuario junto a una contraseña que no existan ya en la base de datos (con esta función no es posible crear superusuarios\*).
* **Log in**
  + Ruta: /sessions
  + Método: login
  + Parámetros
    - Username
      * Tipo: String
      * Por dónde se pasa: se encapsula en el formulario de la request
      * Descripción: usuario del usuario con el que nos queremos loguear
    - Password
      * Tipo: String
      * Por dónde se pasa: se encapsula en el formulario de la request
      * Descripción: contraseña del usuario con el que nos queremos loguear
  + Respuestas
    - 200: logueado con éxito y devuelve el id de la sesión
    - 401: InvalidCredentialsError
    - 500: HTTPexception (Server error)
  + Descripción: pedimos al usuario que introduzca una combinación usuario-contraseña ya existente.
* **Log out**
  + Ruta: /sessions
  + Método: logout
  + Parámetros
    - Sesión\_id
      * Tipo: String
      * Por donde se pasa: se encapsula en el formulario de la request
      * Descripción: id de la sesión actual devuelto por la función *login*
  + Respuestas:
    - 200: sesión cerrada
    - 401: UnauthorizedError
    - 500: HTTPexception (Server error)
  + Descripción: cerramos la sesión del usuario logueado actualmente.
* **Conceder derechos**
  + Ruta: /users/(usuario al que queremos dar derechos)/rights/(derecho que queremos conceder)
  + Método: grant
  + Parámetros
    - Username
      * Tipo: String
      * Por donde se pasa: se pasa a través de la ruta
      * Descripción: usuario al que queremos conceder los derechos
    - Right\_name
      * Tipo: String
      * Por donde se pasa: se pasa a través de la ruta
      * Descripción: nombre del derecho que queremos conceder
    - Sesión
      * Tipo: String
      * Por donde se pasa: se encapsula en el formulario de la request
      * Descripción: usuario logueado actualmente
  + Respuestas:
    - 200: Permiso concedido
    - 401: sesión no encontrada o permisos insuficientes
    - 404: usuario no encontrado
    - 500: HTTPexception (Server error)
  + Descripción: si el usuario tiene permisos para hacerlo, es capaz de asignar derechos a otros usuarios ya existentes.
* **Revocar derechos**
  + Ruta: /users/(usuario al que queremos dar derechos)/rights/(derecho que queremos quitar)
  + Método: revoke
  + Parámetros
    - Username
      * Tipo: String
      * Por donde se pasa: se pasan a través de la ruta
      * Descripción: usuario al que queremos quitar los derechos
    - Right\_name
      * Tipo: String
      * Por donde se pasa: se pasa a través de la ruta
      * Descripción: nombre del derecho que queremos revocar
    - Sesión
      * Tipo: String
      * Por donde se pasa: se encapsula en el formulario de la request
      * Descripción: usuario logueado actualmente
  + Respuestas
    - 200: derecho revocado
    - 401: sesión no encontrada o permisos insuficientes
    - 404: usuario no encontrado
    - 500: HTTPexception (Server error)
  + Descripción: si el usuario tiene permisos para hacerlo, es capaz de quitar permisos a otros usuarios.

\*Los usuarios con el flag de *superuser* en estado de *true*, no necesitan la comprobación de permisos.

En la comunicación entre el Cliente y el Servicio Sensor las funciones son:

* **Modificar reglas del sensor**
  + Ruta: /rules
  + Método: modify\_rules
  + Parámetros
    - Route
      * Tipo: String
      * Por dónde se pasa: se encapsula en el formulario de la request
      * Descripción: ruta que se comprueba
    - Time
      * Tipo: Int
      * Por dónde se pasa: se encapsula en el formulario de la request
      * Descripción: periodo en ms de comprobación de la ruta
    - Session
      * Tipo: String
      * Por dónde se pasa: se encapsula en el formulario de la request
      * Descripción: usuario logueado en el momento
  + Respuestas
    - 200: Reglas cambiadas satisfactoriamente
    - 401: sesión no encontrada o permisos insuficientes
    - 404: usuario no encontrado
    - 500: HTTPexception (Server error)
  + Descripción: permite modificar una serie de ajustes/reglas del sistema
* **Últimas entradas del sensor**
  + Ruta: /entries
  + Método: latest\_entries
  + Parámetros
    - Session
      * Tipo: String
      * Por dónde se pasa: se encapsula en el formulario de la request
      * Descripción: usuario logueado en el momento
  + Respuestas
    - 200: tipo-List / Lista con las últimas 5 entradas del sensor
    - 401: sesión no encontrada o permisos insuficientes
    - 500: HTTPexception (Server error)
  + Descripción: Devuelve una lista con las últimas 5 entradas del sensor

Cada una de estas funciones se asocia a una operación de CRUD de HTTP y a una URL. Cuando el servicio recibe la llamada a esa operación CRUD con esa URL, ejecuta la función correspondiente.

**Explicación genérica de lo que hace una función de nuestra aplicación**

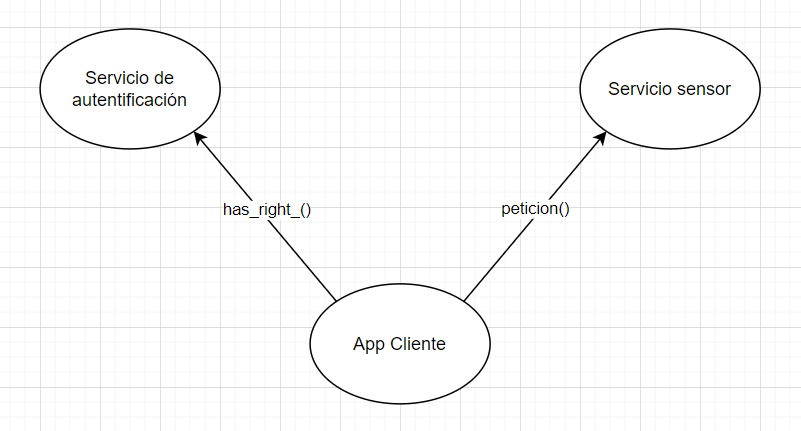
El Cliente envía un request al Servidor y recibe una respuesta de este, para lo cual necesitamos antes establecer la conexión entre el Cliente y el Servidor. También, tenemos que establecer los diferentes componentes del request correctamente, que son:

-La operación CRUD a realizar.  
-La URL con la que se dirige al servicio.  
-El formulario con los parámetros que necesita el servicio del Cliente.  
-Las cabeceras (metadatos).

Tras esto, la función Cliente recibe la respuesta al request y actúa acorde al código HTTP de la respuesta.

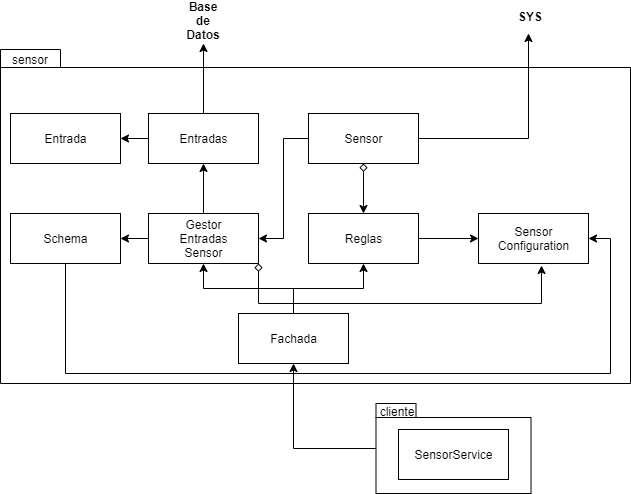
Arquitectura y diseño del servicio sensor:

Añadiriamos una clase en la capa de datos del Cliente para comunicarse con el servicio sensor llamada SensorService() que funcionaria de forma similar a la clase AuthService(). Para ello necesitaríamos obtener los host y port de los sensores por lo que tendríamos que añadir funciones nuevas en la clase ClientConfiguration() para obtenerlos del fichero de configuración.

Al tener dos sensores activos, tendremos dos instancias de SensorService en el cliente, una conectada a cada uno.

Para no necesitar comunicarnos con el Servicio de Autenticación desde el Servicio Sensor, antes de ejecutar las dos últimas opciones del menú, se comprobará si el usuario tiene los derechos correspondientes.

(Somos conscientes de que esta no es la solución más acertada, pero hemos andado bastante escasos de tiempo como para pensar una mejor solución.)



Esta es la estructura de clases que tenemos pensada para el servicio sensor. La forma de comunicarse al servicio sensor desde la clase SensorService del cliente es mediante la clase Fachada que accede a las Reglas del Sensor y al Gestor, que se encarga de las operaciones relacionadas con la Base de Datos

El Sensor va monitorizando la existencia de una ruta (true si existe / false si no existe) determinado por las reglas siguientes:

**Ruta**: La ruta de la que se quiere obtener la existencia

**Periodo**: Periodo entre monitorizaciones

Los valores por defecto de estas reglas se encuentran en SensorConfiguration y se cargan cuando se instancia la clase Reglas (llama desde el constructor a los *gets* de SensorConfiguration). Por ejemplo, por defecto hemos puesto que se detecte la presencia de los USBs ‘KINGSTON’ y ‘USB2’.

El GestorEntradasSensor almacena la configuración y el Schema para poder conectarse a la Base de datos. La clase Entradas se encarga de las operaciones con entradas a nivel de tabla. Instancia la clase Entrada y la almacena en la Base de Datos cuando se llama a su función *create*.   
La clase Entrada define la forma de la tabla en la base de datos

El Schema se encargará de los pasos de inicialización, aquí crearemos la clase base declarativa, el *engine* (utilizando un String de conexión, que sacamos de la configuración) y el mapeado de clases en el ORM, además de un método para obtener nuevas sesiones.

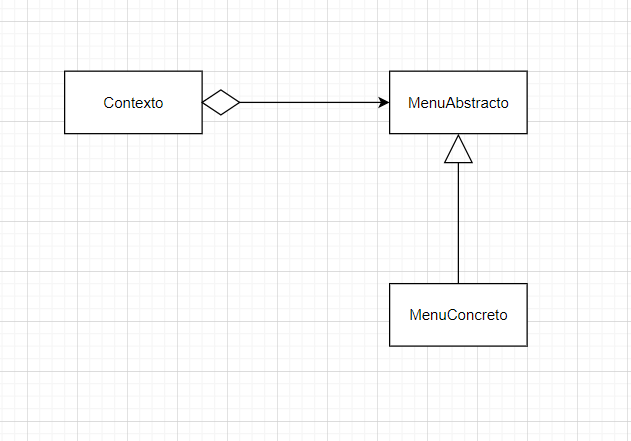
**¿Por qué usamos el PD Fachada?**

Hemos decidido utilizar el patrón de diseño Fachada, que delega las peticiones de fuera a los distintos componentes, para desacoplar el sensor del cliente y evitar así necesitar dos conexiones REST distintas para conectarse con el Cliente, además, así conseguimos que SensorService tenga solo una responsabilidad.

Arquitectura y diseño del cliente:

El programa principal del cliente utiliza las clases AuthService y SensorService para conectarse a los servicios. Para instanciarla, carga la configuración guardada en un archivo en la clase ClientConfiguration, de donde saca el *host* y el *port* de la conexión HTTP que necesita crear. A partir de ahí utiliza las funciones de AuthService y SensorService para realizar las operaciones que quiera el usuario.

Hemos decidido modificar el menú que teníamos anteriormente, el cual estaba realizado a base de una secuencia de *ifs.*



Para implementar el menú hemos utilizado los patrones de diseño Estado y Método Plantilla. El Contexto contiene un Estado interno del que depende su comportamiento y que se puede sustituir, las concreciones de MenuAbstracto hacen las veces de Estado.  
La clase MenuAbstracto define las funciones invariantes de los menús y cada concreción se encarga de implementar las partes variantes.

Contexto permite ejecutar todas las funciones de MenuAbstracto y sus concreciones en bucle las veces que queramos. Desde todos los menús hay una opción de volver que lleva al menú anterior, por ejemplo, en el caso de MenuPrincipal llevaría a Autentificación.

**Jerarquía del menú**

* Autentificación
  + Menú principal
    - Crear usuario
    - Gestionar permisos de usuarios
      * Conceder permisos
      * Revocar permisos
    - Modificar reglas de monitorización
    - Visualizar últimos valores monitorizados

**¿Por qué usamos el PD Estado?**

-Aplicar este patrón nos permite extender con más facilidad el número de opciones del menú  
-Evita el uso de ifs y switches cerrados al cambio  
-Simplifica la comprensión del código

**¿Por qué usamos el PD Método Plantilla?**

-Evitar tener código duplicado (bad smell)