**Memoria:**

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

*Entrega a fecha:* ***20/4/2020***



índice

[Introducción 3](#_Toc59356179)

[Manuales: 4](#_Toc59356180)

[Instalación 4](#_Toc59356181)

[Uso de la aplicación 4](#_Toc59356182)

[Documentación del protocolo de comunicaciones cliente-servidor 5](#_Toc59356183)

[Arquitectura y diseño del servicio sensor: 6](#_Toc59356184)

[Arquitectura y diseño del cliente: 7](#_Toc59356185)

## Introducción

En esta primera parte de la entrega, debido a una gestión del tiempo poco optima, únicamente hemos desarrollado y añadido las funciones de creación de usuario y gestión de privilegio de usuario asi que hemos decidido explicar en la memoria, el diseño de la parte no implementada esperando recibir feedback para la realización de la segunda practica de la mejor manera posible.

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:

* “create user”: Si el usuario que está logeado 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.
* “manage user rights”: 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.
* “change sensor rules”: Si el usuario elige esta opción le llevará a una lista de opciones/ajustes del sistema que se podrán modificar.
* “show latest monitored values”: En esta opción se puede ver el último registro 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 Servidor las funciones son:

* **Crear usuario**
  + Ruta: /users
  + Método: create\_user
  + Parámetros
    - Self
    - 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
    - Self
    - 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
    - Self
    - 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
    - Self
    - 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
    - Self
    - 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.

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:

En cuanto a las reglas, hemos decidido tenerlas en una sola clase, ya que esto simplifica la comunicación con el servicio sensor y la configuración

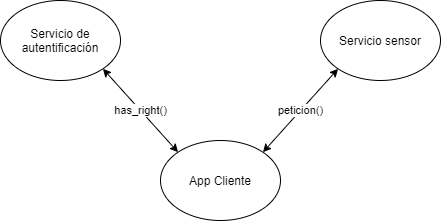
En la capa de Lógica, tendríamos la clase Sensor(), que iría recogiendo y almacenando periódicamente los datos dependiendo de los datos de las reglas.

En la misma capa tendríamos una clase Reglas() cuyos atributos determinarían las reglas para que la clase Sensor() pueda acceder a ellas.

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.

En la capa de Presentación tendríamos definida la clase Reading() que haría de API.

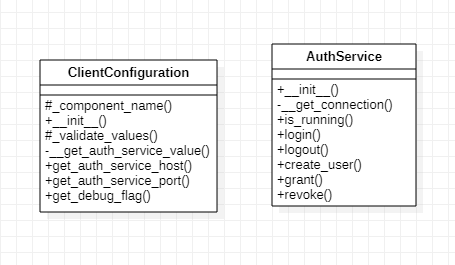
En la capa de datos, crearemos el schema para realizar la conexión entre el servicio sensor y la base de datos. Al mismo tiempo definiríamos la tabla donde guardaríamos las lecturas del sensor llamadas *readings.*

**

Para comprobar si el usuario tiene los derechos para modificar las reglas o ver los últimos valores del sensor, llamará antes al servicio de autentificación para comprobar los derechos antes de realizar la petición al servicio sensor.

El main() se encarga de gestionar las peticiones.

Arquitectura y diseño del cliente:



El programa principal del cliente utiliza la clase AuthService() 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() para realizar las operaciones que quiera el usuario.