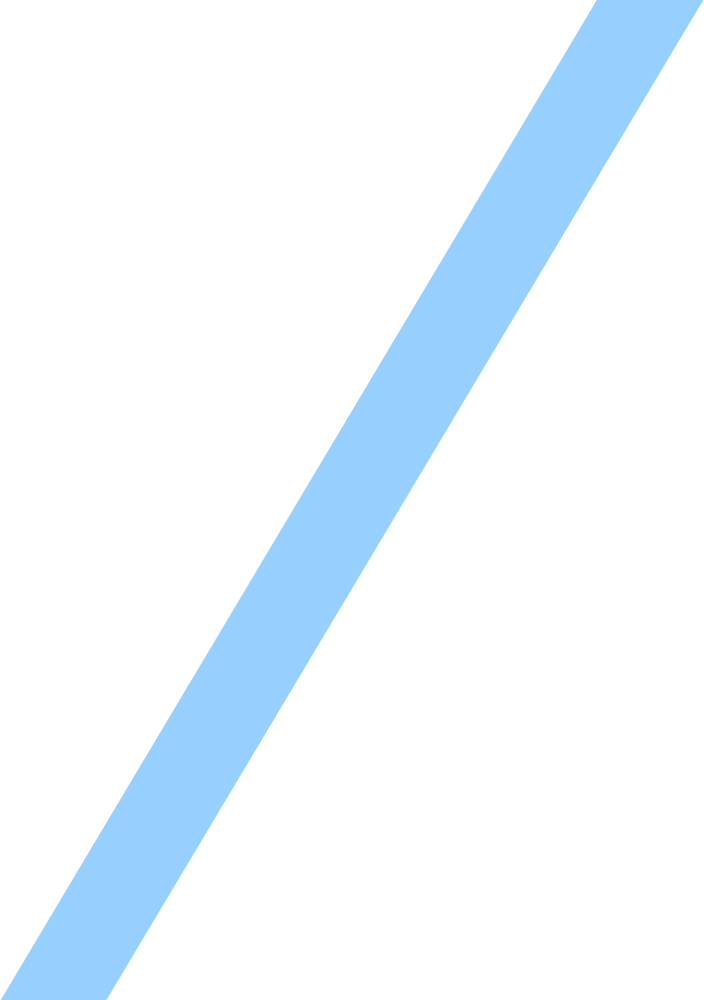
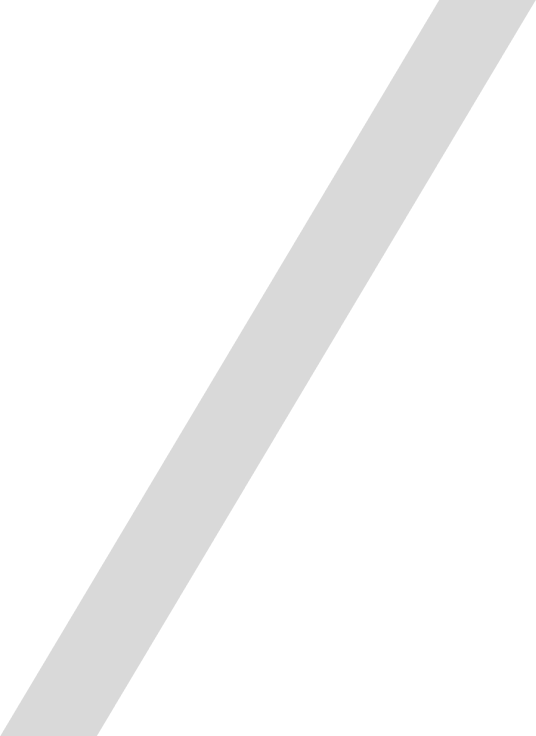
|  |
| --- |
|  |
|  |

|  |  |
| --- | --- |
|  |  |



Entregable 1:

(carpeta Iteración 2)

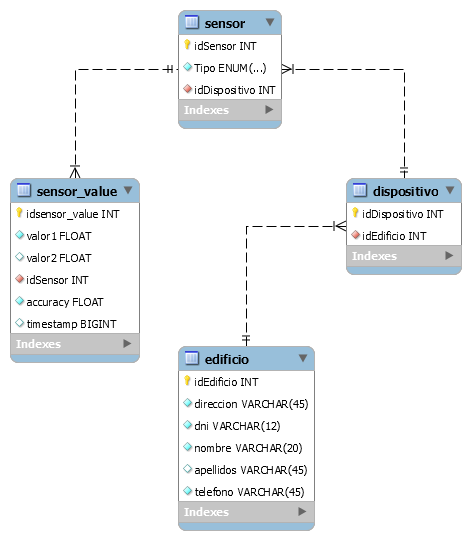
Para la entrega de la primera iteración hemos hecho una breve descripción de lo que va a consistir nuestro proyecto.

En definitiva, el objetivo de este proyecto es realizar un aparato capaz de hacer lecturas de una serie de datos en todas las plantas de un edificio. De esta forma podremos controlar algunos aspectos del edificio de forma remota, como son la temperatura, el CO2 y la humedad.

Entregable 2 (27 de marzo):

(carpeta Iteración 2)

Para este, hemos realizado el modelado de la base de datos de nuestro proyecto. Es decir, un modelo UML con las distintas entidades y sus atributos:

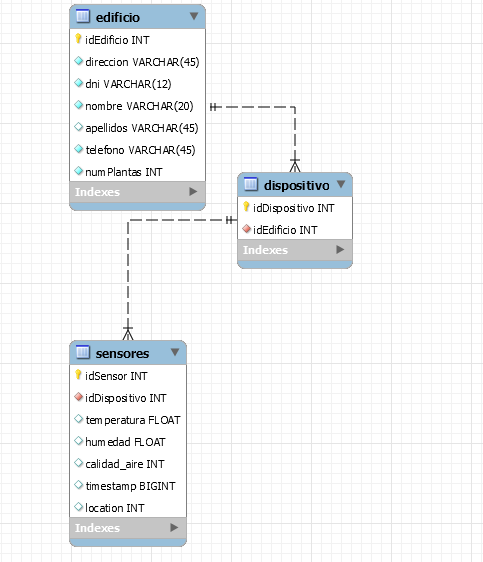


Además del modelo UML, hemos entregado un volcado de la base de datos con las entidades identificadas y valores de ejemplo (Dump20200324.sql).

Entregable 3 (17 de abril):

(carpeta iteración 3)

Hemos realizado la API Rest funcional y conexión con BBDD. Para este entregable, mientras implementábamos los métodos de la API Rest, nos vimos obligados a modificar la base de datos que hicimos en un principio (BBDDv300.sql):



Para procesar las peticiones, hemos implementado algunos métodos de la API REST:

* getpisodondehacecalor: si le pasamos un edificio y la planta, nos dirá si en esa planta del edificio especificado, la temperatura es superior a 28 grados.

(URL: /lap/dispositivo/pisocalor/:IdEdificio/:location)

* getpisodondehacefrio: pasándole un edificio y una planta, nos permitirá saber si la temperatura de la misma es menor de 26 grados.

(URL: lap/dispositivo/pisofrio/:IdEdificio/:location)

* getpisosdondehacefrio: si introducimos un edificio, nos devolverá aquellas plantas en la que las que la temperatura sea inferior a 26 grados.

(URL: /lap/dispositivo/edificiofrio/:IdEdificio)

* getpisoscalor: pasándole un edificio como parámetro, obtendremos las plantas en las que la temperatura sea superior a 28 grados.

(URL: /lap/dispositivo/edificiocalor/:IdEdificio)

* getValueByDispositivo: introduciendo el id de un dispositivo, nos devolverá toda la información del mismo.

(URL: /lap/dispositivo/values/:idDispositivo)

* getValueBySensorAndTimestamp: introduciendo el id de un sensor y un timestamp, obtendremos la información del sensor en ese instante de tiempo.

(URL: /lap/dispositivo/values/:idSensor/:timestamp)

* getValueBySensor: pasando el id de un sensor, nos devolverá toda la información del sensor especificado.

(URL: /lap/sensor/values/:idSensor")

* getValueByPiso: introduciendo un edificio y una planta, nos devolverá todos los datos almacenados de esa planta.

(URL: /lap/dispositivo/values/:IdEdificio/:location)

* getValueByEdificio: introduciendo el id de un edificio, nos dará los datos del mismo.

(URL: /lap/dispositivo/value/:IdEdificio)

* deleteEdficio: le pasamos el edificio que deseamos eliminar.

(URL: /lap/sensor/borrarEdificio/:idEdificio)

* putvalueSensor: introducimos el sensor que deseamos crear.

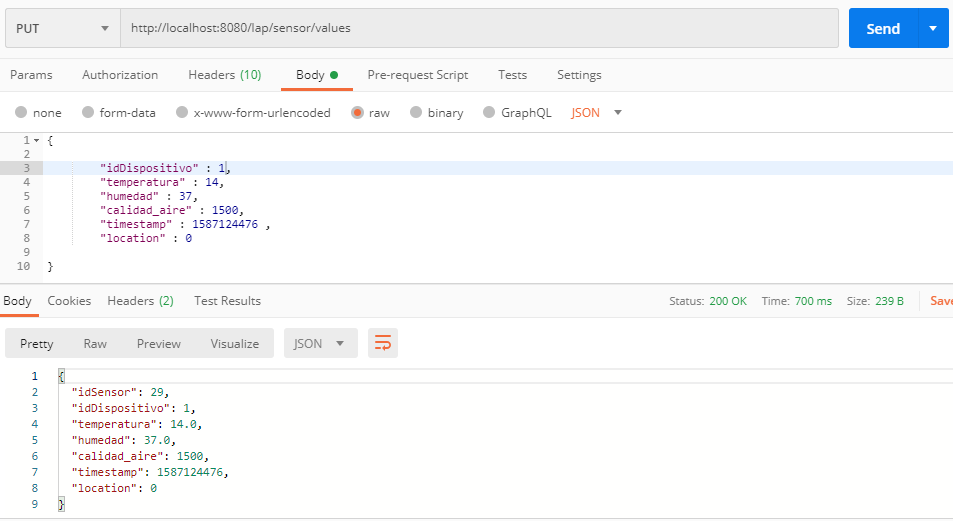
(URL: /lap/sensor/values)

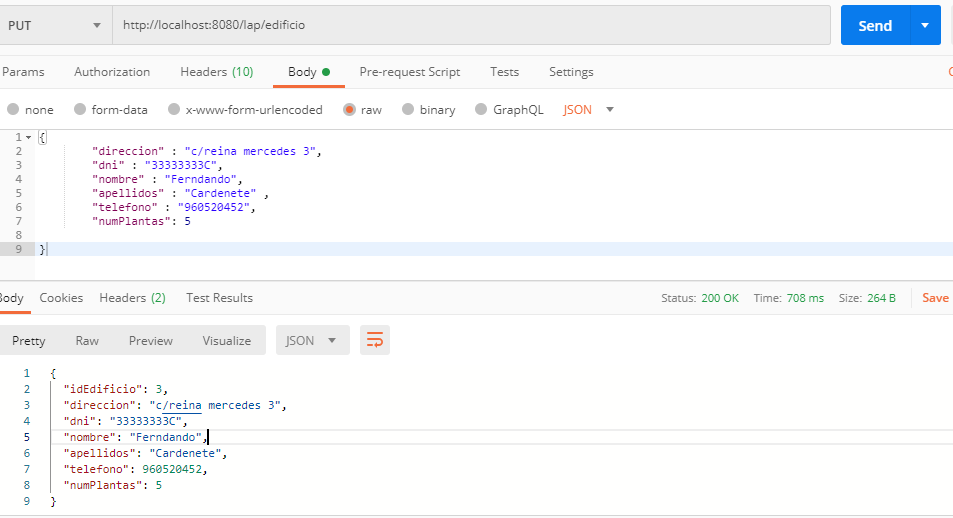
* putEdificio: introducimos el edificio que deseamos crear.

(URL: /lap/edificio)

Por último, hemos implementado el código para que el servicio REST esté conectado con la BBDD.

Algunas pruebas para demostrar que el servicio funciona:





Entregable 4 (24 de abril):

(carpeta iteración 5)

Mensajes y canales MQTT. Lo que conseguimos es que haya una comunicación bidireccional entre el cliente y el servidor una vez el primero ha establecido contacto con el servidor.

* El protocolo MQTT es mucho más sencillo que REST, puesto que se basa en canales y mensajes. Los mensajes, que suelen ser objetos JSON, se publican en un canal concreto. Previamente los clientes se han suscrito a este canal y el mensaje publicado será recibido por todos los clientes. De esta forma, si un cliente publica un mensaje, todos los clientes suscritos a este canal (y el servidor también podría suscribirse como cliente) recibirán la comunicación. Esto permite lo que se denomina comunicación broadband. A lo largo de la próxima semana veremos cómo desplegar un servidor MQTT y cómo crear clientes software que permitan enviar y recibir mensajes de estos canales. El 24 de abril se pide, por tanto, que desplegueis todos los canales MQTT necesarios para que el correcto funcionamiento del proyecto, así como que describáis tanto el nombre y el propósito de estos canales, como el contenido de los mensajes que se publicarán en ellos.

Entregable 5 (15 de mayo):

(carpeta iteración 5)

Hemos definido e implementado las herramientas necesarias para comunicar el módulo con el servidor, siendo estas las peticiones rest de get y put.

* 15 de mayo: entrega de un prototipo hardware basado en ESP32 o ESP8266 que permita realizar peticiones REST y enviar y recibir mensajes MQTT a través de la plataforma software desplegada anteriormente. En este entregable se pide, por primera vez, la integración del componente hardware en el proyecto. A este nivel no es necesario que los sensores y actuadores funcionen correctamente, simplemente se trata de definir e implementar las herramientas necesarias para comunicar el módulo con el servidor. Para ello será necesario emplear los métodos REST y MQTT que anteriormente se han desplegado. Previamente veremos cómo implementar un cliente REST en un módulo ESPXX y cómo suscribirnos, enviar y recibir mensajes MQTT desde este mismo módulo. Con esto lo dejaremos todo preparado para el último entregable. El 15 de mayo se pedirá que cada grupo suba al repositorio el software desarrollado para el ESP.

Última iteración (17 de junio):

(carpeta última iteración)

Por último, hemos recopilado todo lo que hemos ido realizando a lo largo del proyecto para la memoria. Para este entregable se pedía la integración de los sensores en la solución, pero nosotros ya los teníamos integrados en el entregable anterior. Nuestro dispositivo es capaz de leer los datos desde los sensores y enviarlos al servidor a través de la API y que los actuadores se puedan controlar desde el servidor.

En la carpeta Última iteración, podemos encontrar la carpeta comprimida codigoVertex.zip en la se encuentra todo el proyecto.