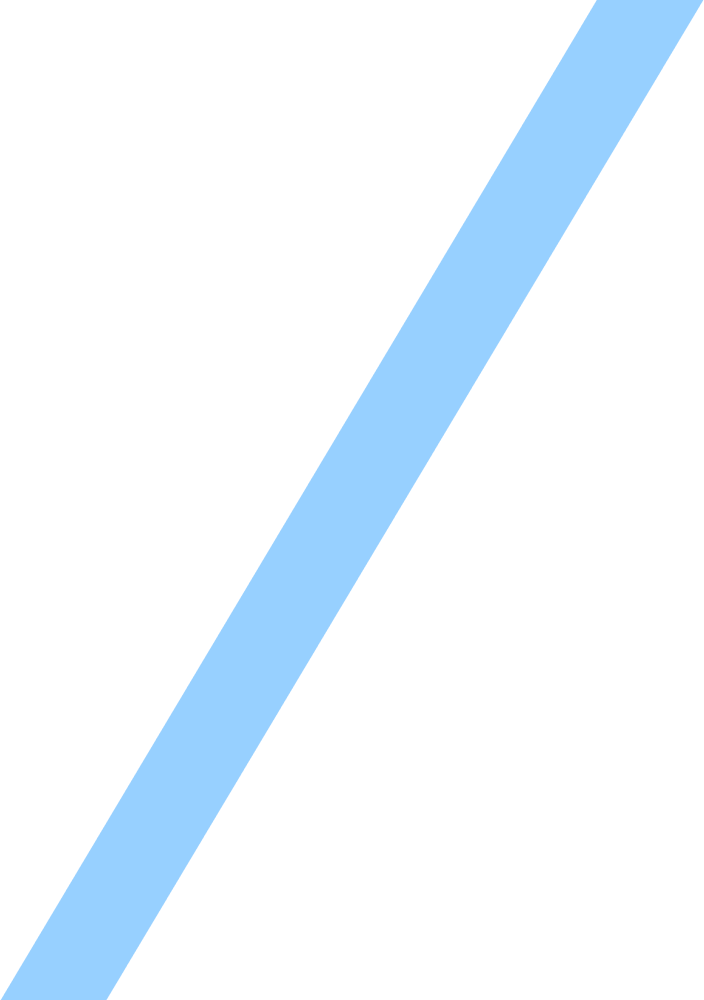
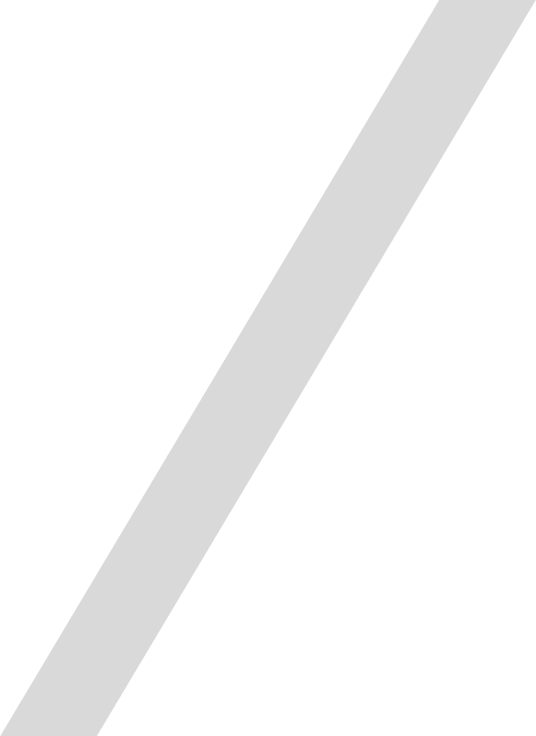
|  |
| --- |
| SMART LAP |

|  |
| --- |
| MEMORIA  Desarrollo de Aplicaciones Distribuidas  2019-2020 |

|  |  |
| --- | --- |
| GUSTAVO ARIEL HONORES MURRAY  PEDRO MIGUEL VIEGAS PEÑALOSA  LOURDES MILLÁN CARRETERO |  |



Entregable 1:

(carpeta Iteración 2)

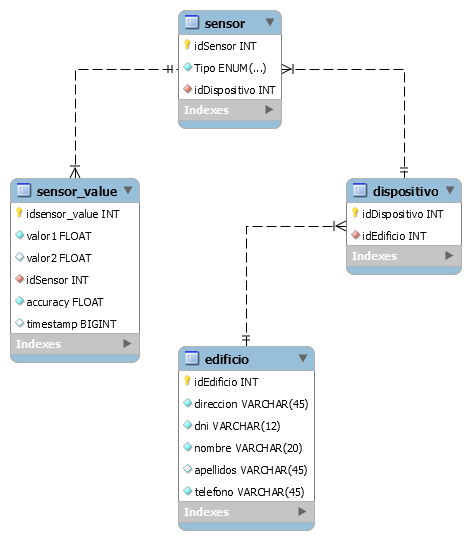
Para la entrega de la primera iteración hemos hecho una breve descripción de lo que va a consistir nuestro proyecto.

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.

Entregable 2 (17 de abril):

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:

(NUEVA BASE DE DATOS)

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

* getpisodondehacecalor: nos dirá si en una planta de un edificio la temperatura es superior a 28 grados. (URL: /lap/dispositivo/pisocalor/:IdEdificio/:location)
* getpisodondehacefrio: función que nos permite saber si la temperatura de una planta de un edificio es menor de 26 grados. (URL: lap/dispositivo/pisofrio/:IdEdificio/:location)
* getpisodondehacecalor: nos dará aquellas plantas de un edificio en las que la temperatura sea superior a 28 grados. (URL: /lap/dispositivo/pisocalor/:IdEdificio/:location)
* getpisosdondehacefrio: función que nos permite saber en qué plantas de un edificio hace menos de 26 grados. (URL: /lap/dispositivo/edificiofrio/:IdEdificio)
* getpisoscalor: nos permitirá saber en qué plantas de un edificio la temperatura es superior a 28 grados. (URL: /lap/dispositivo/edificiocalor/:IdEdificio)
* getValueByDispositivo: nos devolverá todos los datos de un dispositivo. (URL: /lap/dispositivo/values/:idDispositivo)
* getValueBySensorAndTimestamp: obtendremos la información de un sensor en un instante de tiempo concreto. (URL: /lap/dispositivo/values/:idSensor/:timestamp)
* getValueBySensor: obtendremos los datos de un sensor en concreto. (URL: /lap/sensor/values/:idSensor")
* getValueByPiso: nos devolverá los daros de una planta de un edificio. (URL: /lap/dispositivo/values/:IdEdificio/:location)
* getValueByEdificio: nos dará los datos de un edificio concreto. (URL: /lap/dispositivo/value/:IdEdificio)
* deleteEdficio: borra un edificio. (URL: /lap/sensor/borrarEdificio/:idEdificio)
* putvalueSensor: crea un nuevo sensor. (URL: /lap/sensor/values)
* putEdificio: crea un nuevo edificio. (URL: /lap/edificio)

* 17 de abril: esquema de API Rest, API Rest funcional y conexión con BBDD. Una vez modelada la BBDD, procederemos a implementar todos los métodos necesarios de la API Rest (GET, PUT, POST o DELETE). En principio, la mayoría de entidades tendrán asociado uno o más de estos métodos, los cuales serán necesarios para procesar peticiones tanto del módulo hardware como de una hipotético cliente que necesite interaccionar con nuestro sistema de información. En este entregable se pide elaborar un documento en el que se recojan los diferentes puntos de acceso a las funciones de la API Rest (URLs), el tipo de método (GET, PUT, POST, DELETE, ...), los parámetros en caso de tenerlos (nombre, posición y breve explicación sobre qué se espera que se pase en este parámetro), cuerpo de la petición (en el caso de los métodos PUT, POST, DELETE, será necesario un cuerpo, por lo que se pide que se explique el contenido de este cuerpo y la descripción del JSON que el cliente debe proporcionar) y, por último, la respuesta que este método dará al cliente. Además de este documento explicativo, se pedirá que cada grupo suba el código fuente del servicio REST que deberá estar conectado con la BBDD (esto último veremos cómo implementarlo el próximo miércoles). Por último, seberán documentarse las pruebas que demuestren que el servicio funciona correctamente. La manera más sencilla de hacer esto es realizando capturas de pantalla de POSTMAN ( <https://www.postman.com/downloads/> ) donde aparezcan los parámetros de la petición y el resultado de la misma. Con esto ya tendríamos cerrado toda la capa de servidor, a excepción de MQTT.

Entregable 3 (24 de abril):

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.

* 24 de abril: mensajes y canales MQTT. Aún no hemos empezado con esta parte, pero ya hemos adelantado que MQTT es un protocolo que permiten 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 4 (15 de mayo):

Hemos definido e implementado las herramientas necesarias para comunicar el módulo con el servidor.

* 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.

Entregable 5 (29 de abril):

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.

* 29 de mayo: entregable final. En este último entregable, con el servidor preparado desde hace semanas y las herramientas de comunicación entre el módulo y el servidor desarrolladas desde el día 15 de mayo, se pide que se integren los sensores en la solución. Aquí cada grupo deberá desarrollar el subsistema que permita leer los datos desde los sensores para enviarlos al servidor a través de la API establecida (REST o MQTT, la primera preferiblemente) y, por supuesto, hacer que los actuadores puedan controlarse desde el servidor mediante el parseo de los mensajes MQTT que se reciban por el canal concreto. Con esto cerraríamos el proyecto y habríamos desarrollado una solución altamente escalable y distribuida mediante el uso de protocolos de comunicación, plataformas y hardware muy presentes en el mercado actual.