

### **Tarea N°3: Iot API Development**

#### **1. Descripción general.**

Esta tarea está orientada al desarrollo de una API Rest para una plataforma de IoT del cual usted es el ingeniero responsable.

El escenario es el siguiente:

- Los dispositivos IoT envían paquetes de datos en estructura JSON cada 1 a 30 segundos al servidor API.
- El servidor API debe guardar estos datos y permitir que el usuario consulte o filtre la dicha información.
- Debe ser una API ligera, que implemente criterios de seguridad.
- Usar para la solución una base de datos SQLite.

#### **2. Modelo de datos.**

Debe existir un usuario Administrador (admin) el cual tiene los siguientes privilegios:

- Puede crear una compañía (cliente)
- Puede crear lugares (Location)
- Una compañía tiene muchos lugares (Location)
- Puede crear sensores
- Un lugar tiene muchos sensores
- Un sensor tiene muchas variables (sensor\_data)

##### **Modelo Entidad – Relación**

Admin

Company

Location

Sensor

Sensor Data

El estudiante debe establecer las relaciones adecuadas a cada caso.

A continuación, el detalle de las tablas/campos:

Admin:

- Username (string)
- Password (string)

Company:

- ID (int)
- company\_name (string)
- company\_api\_key (string) (generada cuando se crea la entidad. Se utilizará para las consultas tipo /GET)

Location:

- company\_id (int)
- location\_name (string)
- location\_country (string)
- location\_city (string)
- location\_meta (string)

Sensor:

- location\_id (int)
- sensor\_id (int)
- sensor\_name (string)
- sensor\_category (string)
- sensor\_meta (string)
- sensor\_api\_key (string) (generada cuando se crea la entidad. Se utilizará para insertar datos desde un dispositivo IoT)

Sensor Data:

- Las que sean configuradas para el sensor respectivo.

JSON de datos enviados por un sensor:

```
{"api_key":<sensor_api_key>, "json_data":[{...}, {...}] }
```

Los atributos corresponden a los atributos de del sensor definidos en sensor\_data. El sensor\_api\_key nos dice a cuál sensor\_id corresponden los datos. Si el api key está errado o no tiene sensor asociado, debe retornarse un error 400. Como puede observar, puede enviar un arreglo de objetos, es decir, se pueden mandar mas de una medición por petición.

### 3. API Endpoints

- Endpoints REST estándar para cada modelo, excepto admin y compañía.
- Muestra todo, muestra uno, edita, elimina. (GET, GET, PUT, DELETE).
- Todos los métodos API requieren el `company_api_key`, exceptuando la inserción en `sensor_data` (el cuál usará el `sensor_api_key`, tal como fue descrito).
- La inserción de `sensor_data` debe tener la siguiente estructura:
  - POST `/api/v1/sensor_data`
  - Insertar datos ocupando el `sensor_api_key` como mecanismo de autorización.
  - Debe retornar Status HTTP 201 (created).
- La consulta de `sensor_data` debe tener la siguiente estructura:
  - GET `/api/v1/sensor_data`
  - Parámetros requeridos:
    - La autorización será posible mediante el uso de un Header HTTP o puede tener un parámetro en la URL `&company_api_key=`
    - `from = < marca de tiempo en formato EPOCH >`
    - `to = < marca de tiempo en formato EPOCH >`
    - `sensor_id = [2,3,4,5,10,220]` (Arreglo de `sensor_id` para los cuales se consultan los `sensor_data`)

### 4. Condiciones de desarrollo

- Lenguajes permitidos: Python + Flask, Javascript (Node + Express JS), Ruby on Rails.
- Debe realizar el deploy en una capa gratuita de AWS (o del servicio cloud de su preferencia).
- Integrantes: 1 o 2 personas.
- Realizar un informe sencillo reportando los endpoints implementados, con su respectivo verbo http y captura de pantalla de prueba con Postman.
- Para las pruebas del endpoint de consulta de `sensor_data`, debe crear AL MENOS 2 sensores con 2 variables cada uno.
- Así mismo, debe reportar la dirección IP y el puerto donde funciona su solución, así como la dirección del repositorio GIT del código fuente.