Tarea 3: API IoT

Diego Carrillo
Profesor: Carlos Garcia

1. Implementación de API

Esta API, fue implementada con el uso de python mediante las librerías de Flask, SQLite y SQLAlchemy

- Flask: Permite la creación de la API de manera rápida y sencilla.
- SQLite: Permite crear la base de datos en python de manera rápida y sencilla.
- SQLAlchemy: Permite el uso de sentencias SQL de manera sencillas gracias el uso de python.

Para esta aplicación se solicitó levantar la API en un servidor Cloud, lo cual no ha sido realizado, pero estará en el repositorio de github con las respectivas instrucciones de uso.

En el siguiente informe, se explicará el código creado, los ejemplos realizados y cual es el trasfondo de esta aplicación.

2. Codigo

La estructura de esta aplicación es similar a un MVC (Modelo Vista Controlador), los archivos se describirán a grandes rasgos a continuación.

- app.py : Están todas las rutas de la aplicación y es donde se declara la aplicación.
- models.py: se definen las clases utilizadas para la base de datos, como son inicializados y manipulados al momento de hacer llamadas de datos.
- seed.py: Aquí se crean las tablas en la base de datos y es lo que permite crear el archivo de bases de datos.

3. Cliente API

Al ingresar a la aplicación se ve de la siguiente manera:

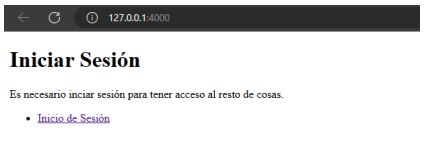


Figura 1. Pagina de inicio

Debido a que dentro de las solicitudes de la actividad, está que debe existir un administrador que pueda crear toda la información y manipularla, será necesario iniciar sesión como administrador

Para poder ingresar como administrador, sera necesario saber las credenciales para entrar, las cuales son las siguientes:

- **Usuario:** admin
- **Contraseña:** admin

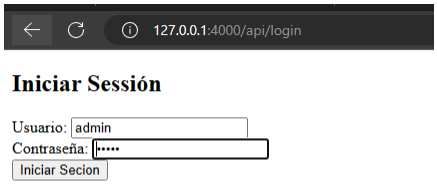


Figura 2. Inicio de Sesión

Una vez iniciado sesión, se podrá acceder a 3 opciones. Las cuales consisten en crear una compañía, localización o sensores, a los cuales se les insertará la información mediante un JSON explicado más adelante.

Figura 3. Pagina de inicio

A continuación, se muestran las imágenes para poder crear una Compañía, localización y Sensor. Lo cual consiste en un formulario simple que agregará la información a la base de datos.

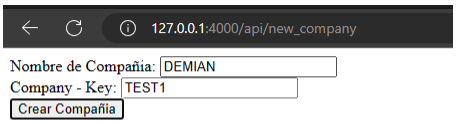


Figura 4. Crear Compañía

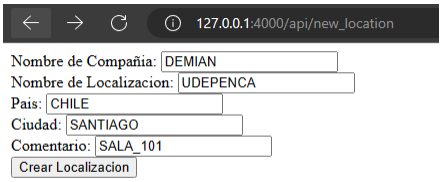


Figura 5. Crear Localización

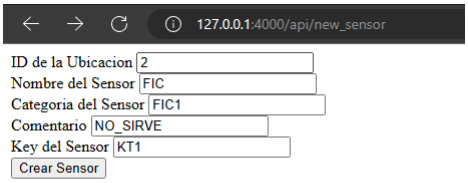


Figura 6. Creador de Sensor

4. Base de datos

Aquí, se mostrarán la base de datos creadas por SQLite mediante una extensión que permite la visualización en visual Studio Code. Además, en las fotos a continuación, se verán evidenciados los resultados mostrados en las imágenes anteriores.

SQL ▾ 1 / 1

id	company_name	company_api_key
1	Company A	api_key_123
2	DEMIAN	TEST1

Figura 7. Tabla Company SQL

SQL ▾ 1 / 1 1 - 2 of 2

id	company_id	location_name	location_country	location_city	location_meta
1	1	Location A	Country A	City A	NULL
2	2	UDEPENCA	CHILE	SANTIAGO	SALA_101

Figura 8. Tabla Location SQL

SQL ▾ 1 / 1 1 - 6 of 6

id	location_id	sensor_name	sensor_category	sensor_meta	sensor_api_key
1	1	Sensor A	Category A	Meta A	api_key_sensor_123
2	1	PRUEBA	PRUEBA	PRUEBA	123
3	1	Panaderia	nada	prueba2	0000
4	1	ARQUI	ARQUI	ARQ	111
5	1	SOFT	SOFT	SOFT	2
6	2	FIC	FIC1	NO_SIRVE	KT1

Figura 9. Tabla Sensor SQL

5. Endpoints

Para poder realizar las peticiones, dentro de esta aplicación se crearon varias funciones asociadas a distintas rutas, que mediante el uso de métodos GET y POST logra acceder a la información o guardarla en el caso de ser necesario.

Las pruebas para esta actividad, fueron realizadas con **insomnia**, aplicación la cual, nos permite hacer peticiones de la misma manera que postman.

5.1. Inserción de sensor data

Primero, tenemos la inserción de sensor_data, la cual debe seguir una cierta estructura. Esta se realiza mediante una petición post y con el uso de sensor_api_key como mecanismo validador. Retornando finalmente un aviso de Status HTTP 201. Esto se verá en la siguiente imagen.

```python
@app.route('/api/v1/sensor_data', methods=['POST']) #SOLICITADO
@basic_auth.required
def insert_sensor_data(): # Estructura JSON   {"api_key":<sensor_api_key>, "json_data":[{…}, {….}] }
    request_data = request.get_json()

    if 'api_key' not in request_data:
        return jsonify({"error": "No se ha enviado la api_key"}), 400
    if 'json_data' not in request_data:
        return jsonify({"error": "No se ha enviado el json_data"}), 400

    sensor_api_key = request_data['api_key']
    sensor = Sensor.query.filter_by(sensor_api_key=sensor_api_key).first()
    if sensor is None:
        return jsonify({"error": "API key inválido"}), 400

    json_data = request_data['json_data']
    if not isinstance(json_data, list):
        return jsonify({"error": "json_data debe ser una lista de diccionarios"}), 400

    for data in json_data:
        if 'timestamp' not in data:
            return jsonify({"error": "No se ha enviado el timestamp"}), 400
        timestamp = datetime.fromtimestamp(data['timestamp'])
        variable_data = {key:value for key, value in data.items() if key != 'timestamp'}

        sensor_data = SensorData(
            sensor_id=sensor.id,
            timestamp=timestamp,
        )

        sensor_data.set_data(variable_data)
        db.session.add(sensor_data)
    db.session.commit()
    return jsonify({"message": "Datos del sensor guardados correctamente"}), 200
```

Figura 10. Inserción Sensor Data

Para poner a prueba esta inserción de datos en un sensor, utilizaremos el sensor creado anteriormente.

Los valores que identificamos son los siguientes:

- **id:** 6
- **location_id:** 2
- **sensor_name:** FIC
- **sensor_category:** FIC1
- **sensor_api_key:** KT1

Para poder enviar información a este sensor, se envía a la siguiente ruta:

- http://127.0.0.1:4000/api/v1/sensor_data

Y lo que fue enviado mediante el método post, fue mediante la aplicación insomnia con el siguiente contenido JSON.

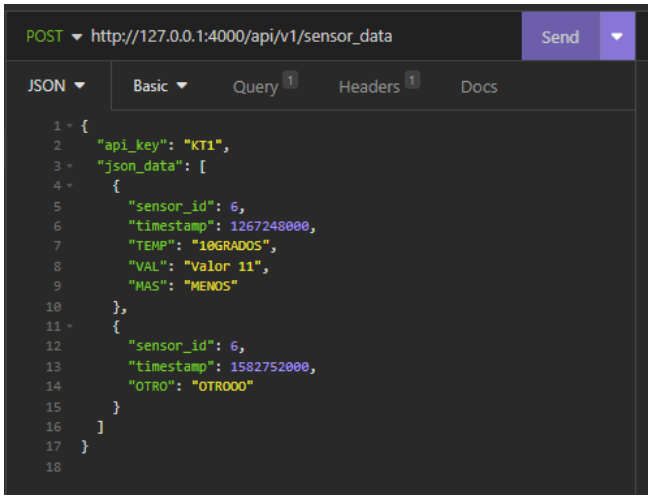


Figura 11. POST - .JSON

Como se puede observar, dentro de la información se envían 2 peticiones asociadas al mismo sensor, pero para un caso se envía con dos variables adjuntas, mientras que para otro caso se envían 3 variables adjuntas. Esto permite que el usuario cree la cantidad de variables que desee y con la información que estime conveniente.

Ante esta solicitud, se obtiene como respuesta un estado 200, que indica que la información fue recibida correctamente. Ahora, como se solicita que exista una autentificación al

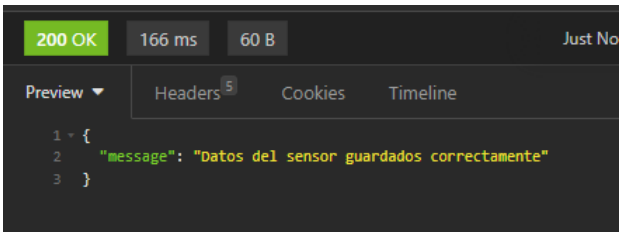


Figura 12. POST - Respuesta

momento de realizar la petición, es necesario ingresar con ciertas credenciales, ya que no bastará con tan solo iniciar sesión dentro de la API. Para esto, se habilita mediante flash un método llamado **Basic Auth** que nos permite enviar un usuario y contraseña al momento de hacer la petición.

Esto le da seguridad a las peticiones y, permite, que no cualquier persona pueda mandar peticiones a la aplicación, además que la API posee una key para su funcionamiento. Las credenciales para habilitar esta "Auth" son las mismas que para un inicio de sesión.

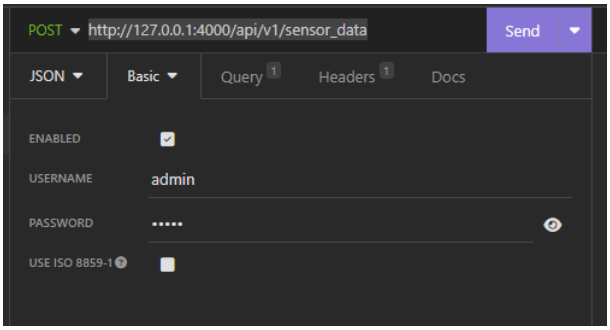
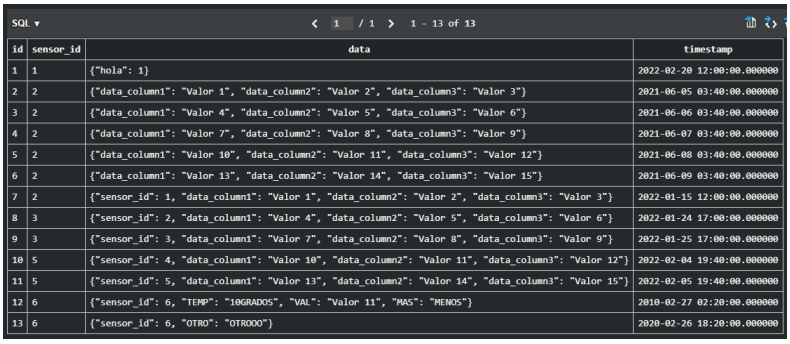


Figura 13. Basic Auth - .JSON

Para poder corroborar que esta petición fue ingresada, se adjunta una captura de la base de datos.



SQL ▾ 1 / 1 1 - 13 of 13

id	sensor_id	data	timestamp
1	1	{"hola": 1}	2022-02-20 12:00:00.000000
2	2	{"data_column1": "Valor 1", "data_column2": "Valor 2", "data_column3": "Valor 3"}	2021-06-05 03:40:00.000000
3	2	{"data_column1": "Valor 4", "data_column2": "Valor 5", "data_column3": "Valor 6"}	2021-06-06 03:40:00.000000
4	2	{"data_column1": "Valor 7", "data_column2": "Valor 8", "data_column3": "Valor 9"}	2021-06-07 03:40:00.000000
5	2	{"data_column1": "Valor 10", "data_column2": "Valor 11", "data_column3": "Valor 12"}	2021-06-08 03:40:00.000000
6	2	{"data_column1": "Valor 13", "data_column2": "Valor 14", "data_column3": "Valor 15"}	2021-06-09 03:40:00.000000
7	2	{"sensor_id": 1, "data_column1": "Valor 1", "data_column2": "Valor 2", "data_column3": "Valor 3"}	2022-01-15 12:00:00.000000
8	3	{"sensor_id": 2, "data_column1": "Valor 4", "data_column2": "Valor 5", "data_column3": "Valor 6"}	2022-01-24 17:00:00.000000
9	3	{"sensor_id": 3, "data_column1": "Valor 7", "data_column2": "Valor 8", "data_column3": "Valor 9"}	2022-01-25 17:00:00.000000
10	5	{"sensor_id": 4, "data_column1": "Valor 10", "data_column2": "Valor 11", "data_column3": "Valor 12"}	2022-02-04 19:40:00.000000
11	5	{"sensor_id": 5, "data_column1": "Valor 13", "data_column2": "Valor 14", "data_column3": "Valor 15"}	2022-02-05 19:40:00.000000
12	6	{"sensor_id": 6, "TEMP": "10GRADOS", "VAL": "Valor 11", "MAS": "MENOS"}	2010-02-27 02:20:00.000000
13	6	{"sensor_id": 6, "OTRO": "OTROOO"}	2020-02-26 18:20:00.000000

Figura 14. Base de datos con el registro

5.2. Consulta de sensor data

Luego, esta la consulta de datos, la cual nos permitirá obtener la información de un sensor en un tiempo que sea especificado por el cliente. Básicamente, ingresamos el rango de fechas que queremos analizar, la clave asociada a una compañía junto con el id del sensor que se pregunta. Esto es enviado mediante un JSON con el método GET que nos permite visualizar los resultados buscados.

```python
@app.route('/api/v1/sensor_data', methods=['GET'])  #SOLICITADO
@basic_auth.required
def get_sensor_data():

    request_data = request.get_json()
    company_api_key = request_data.get('company_api_key')
    from_timestamp = request_data.get('from')
    to_timestamp = request_data.get('to')
    sensor_ids = request_data.get('sensor_ids')

    if not company_api_key:
        return jsonify({"error": "No se ha enviado la company_api_key"}), 400

    company = Company.query.filter_by(company_api_key=company_api_key).first()
    if company is None:
        return jsonify({"error": "API key inválido"}), 400

    sensor_data = SensorData.query.filter(
        SensorData.sensor_id.in_(sensor_ids),
        SensorData.timestamp >= from_timestamp,
        SensorData.timestamp <= to_timestamp
    ).all()

    if not sensor_data:
        return jsonify({"error": "No hay datos de sensores para mostrar"}), 404

    response_data = {
        "company_name": company.company_name,
        "company_api_key": company.company_api_key,
        "sensor_data": [data.serialize() for data in sensor_data]
    }
    print(request_data)
    return jsonify(response_data), 200
```

Figura 15. Consulta Sensor Data

Para poder poner a prueba esta consulta, mediante insomnia accedemos a la ruta:

- http://127.0.0.1:4000/api/v1/sensor_data

y se realiza la siguiente petición:

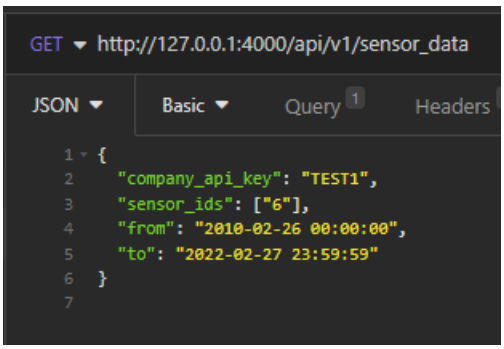


Figura 16. GET - .JSON

En donde se indica la llave de acceso de la compañía, y se busca según los ID de sensores que se quieren identificar, además, se puede filtrar según un periodo de fechas datos en formato EPOCH.

Esto nos entregará la sensor_data que cumpla con lo solicitado, como podrá verse en la siguiente imagen. Como se puede ver en la imagen, entregas como resultado los

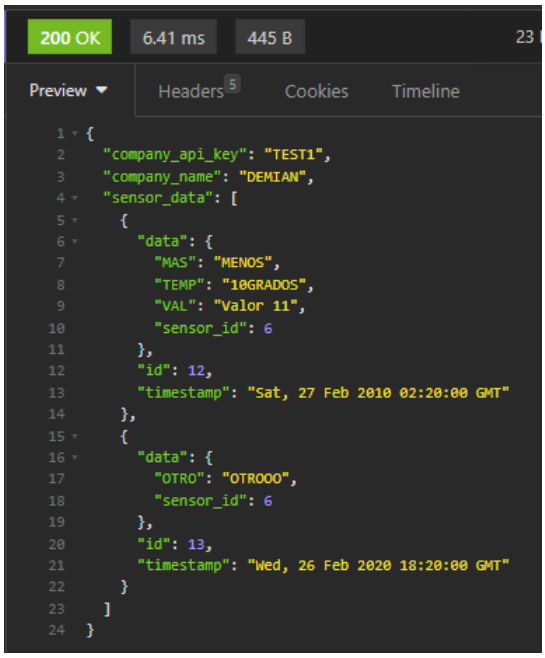


Figura 17. GET - Respuesta

valores que fueron ingresados en el comienzo del informe. Demostrando así que cumple con los filtros.

6. ANEXO

El presente informe evidencia de manera general el funcionamiento de esta API creada en python con el uso de Flask, SQLite y SQLAlchemy.

Si se desea entender correctamente el funcionamiento de este proyecto es recomendable dirigirse al siguiente repositorio.

Repositorio de Github - DCvrro

Una vez en este repositorio tendrá acceso a la base de datos con los valores de prueba presentes en este informe y una posible actualización para que esto sea usado de manera publica en un servidor gratis.

Si bien, esta aplicación debió de ser levantada desde un comienzo en un servidor. Esto no fue realizado debido a los problemas que conlleva hacer esto. Ya que, se requiere de ingresar tarjetas bancarias para poder obtener estos servicios gratuitos lo que hace más difícil obtenerlos. Además, la tarjeta de débito que posee el estudiante fue rechazada por la pagina, llevándolo a tomar esta decisión.

Por otro lado, lo que se entrega en este trabajo es lo que interpreto el estudiante al leer la actividad.