

Tarea 3

Arquitecturas Emergentes

Christian Bastias, Joaquín Neira
`christian.bastiasj@mail.udp.cl`, `joaquin.neira@mail.udp.cl`

Índice general

1. Introducción	2
2. Base de datos	3
3. Consultas a la API	6
4. Repositorio en Github y enlace a API	12
5. Conclusión	13

1. Introducción

La tarea tiene por objetivo la construcción de una API que presente los métodos asociados a la arquitectura REST, situada en el contexto de una interfaz para un servidor IoT.

Para esto, se implementa una base de datos relacional con múltiples tablas que provean la estructura y datos necesarios para la ejecución de los métodos de la API.

El stack utilizado es JavaScript, acompañado de NodeJS y ExpressJS.

2. Base de datos

Se diseña una base de datos que contenga las tablas, **Admin**, **Company**, **Location**, **Sensor** y **Sensor_data**, que contienen los siguientes datos:

Para la tabla **Admin**:

- Username (string)
- Password (string)

Para la tabla **Company**:

- ID (int)
- company_name (string)
- company_api_key (string)

Para la tabla **Location**:

- location_id (int) (*Primary Key*)
- company_id (int) (*Foreign Key*, referenciando a ID de **Company**)
- location_name (string)
- location_city (string)
- location_meta (string)

Para la tabla **Sensor**:

- location_id (int) (*Foreign Key*, referenciando a location_id de **Location**)
- sensor_id (int) (int)
- sensor_name (string)
- sensor_category (string)
- sensor_meta (string)
- sensor_api_key (string)

Para la tabla **Sensor_Data**:

- value (int)
- timestamp (string)
- sensor_id (*Foreign Key*, referenciando a sensor_id de **Sensor**)

Se inicializan estas tablas con los siguientes datos:

id	username	password
1	user	4da49c16db42ca04538d629ef0533fe8
2	admin	a66abb5684c45962d887564f08346e8d

ID	company_name	company_api_key
1	Company2	67890
2	Company1	12345

Figura 2.1: Se crean dos compañías, con su respectiva `company_api_key`

location_id	company_id	location_name	location_country	location_city	location_meta
1	1	Location2	Country1	City1	Meta2
2	2	Location3	Country2	City2	Meta3
3	2	Location4	Country2	City2	Meta4
4	1	Location1	Country1	City1	Meta1

Figura 2.2: Se crean cuatro *Locations*. Aquellas con ID igual a 1 y 2 pertenecen a la compañía con ID 1 y aquellas con ID igual a 3 y 4 pertenecer a la compañía con ID 2.

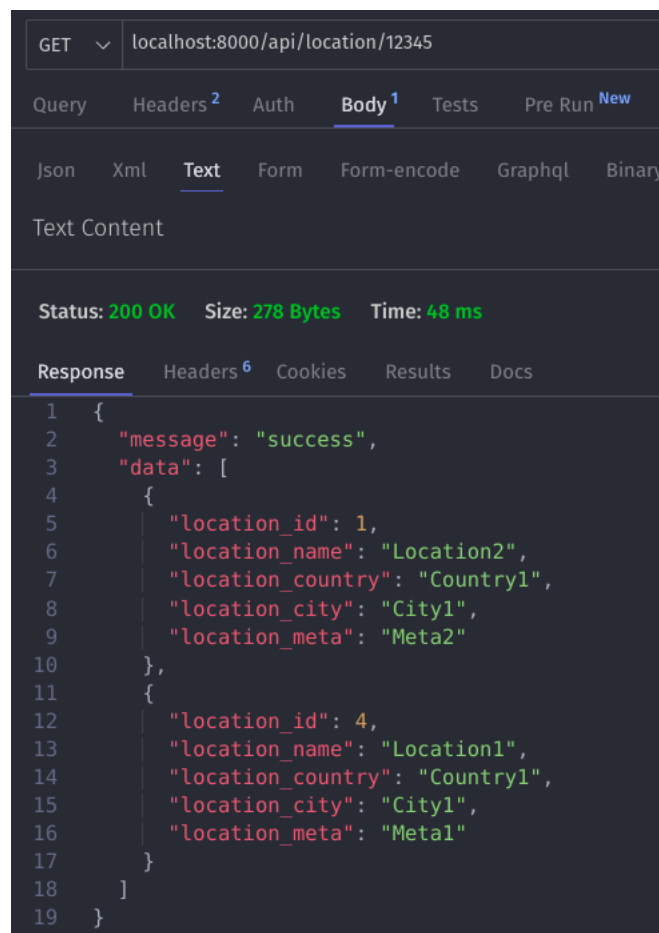
SQL ▼			< 1 / 2 > 1 - 50 of 60	
value	timestamp	sensor_id		
21	1970-01-05 12:00:00.000	1		
20	1970-01-06 12:00:00.000	1		
19	1970-01-07 12:00:00.000	1		
19	1970-01-08 12:00:00.000	1		
20	1970-01-09 12:00:00.000	1		
19	1970-01-10 12:00:00.000	1		
25	1970-01-01 12:00:00.000	1		
17	1970-01-12 12:00:00.000	1		
24	1970-01-02 12:00:00.000	1		
15	1970-01-14 12:00:00.000	1		
14	1970-01-15 12:00:00.000	1		
25	1970-01-01 12:00:00.000	2		
24	1970-01-02 12:00:00.000	2		
23	1970-01-03 12:00:00.000	2		

Figura 2.3: Datos insertados en Sensor_Data. Se generaron 15 registros para cuatro sensores diferentes.

3. Consultas a la API

En base a los datos anteriores, se generan los siguientes *endpoints* para las tablas **Location** y **Company**:
Para los *endpoints* de **Location**:

- Método GET (todos los datos pertenecientes a Locations respectivos de una compañía):



The screenshot shows a REST client interface with the following details:

- Method:** GET
- URL:** localhost:8000/api/location/12345
- Response Status:** 200 OK
- Response Size:** 278 Bytes
- Response Time:** 48 ms
- Response Body (JSON):**

```
1 {
2   "message": "success",
3   "data": [
4     {
5       "location_id": 1,
6       "location_name": "Location2",
7       "location_country": "Country1",
8       "location_city": "City1",
9       "location_meta": "Meta2"
10    },
11    {
12      "location_id": 4,
13      "location_name": "Location1",
14      "location_country": "Country1",
15      "location_city": "City1",
16      "location_meta": "Meta1"
17    }
18  ]
19 }
```

- Método GET (todos los datos de una Location especificada):

```

GET localhost:8000/api/location/12345/1
Query Headers2 Auth Body Tests Pre
Json Xml Text Form Form-encode GraphQL
Text Content

Status: 200 OK Size: 152 Bytes Time: 16 ms

Response Headers6 Cookies Results Docs
1 {
2   "message": "success",
3   "data": {
4     "location_id": 1,
5     "location_name": "Location2",
6     "location_country": "Country1",
7     "location_city": "City1",
8     "location_meta": "Meta2"
9   }
10  }

```

- Método PUT (insertar datos en la tabla Location para una compañía especificada):

```

PUT localhost:8000/api/location/1/12345
Query Headers2 Auth Body1 Tests Pre Run New
Json Xml Text Form Form-encode GraphQL Binary
1 {
2   "location_name": "Location1111",
3   "location_country": "Country1111"
4 }

Status: 200 OK Size: 106 Bytes Time: 24 ms

1 {
2   "message": "success",
3   "data": {
4     "location_name": "Location1111",
5     "location_country": "Country1111"
6   },
7   "changes": 1
8 }

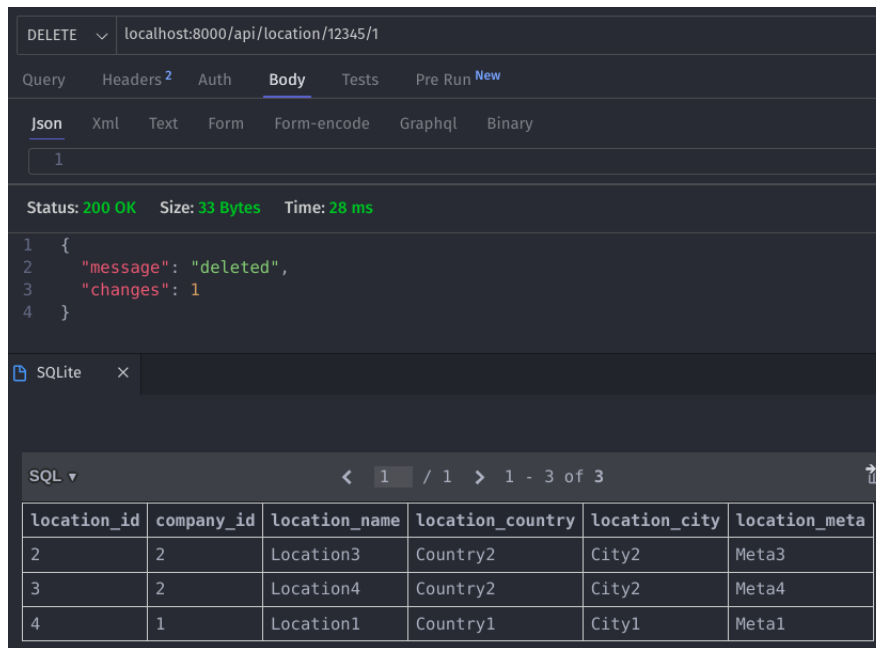
```

SQLite

SQL < 1 / 1 > 1 - 4 of 4

location_id	company_id	location_name	location_country	location_city	location_meta
1	1	Location1111	Country1111	City1	Meta2
2	2	Location3	Country2	City2	Meta3
3	2	Location4	Country2	City2	Meta4
4	1	Location1	Country1	City1	Meta1

- Método DELETE (borrar un registro de la tabla **Location** en particular para una compañía específica). En este ejemplo, se borra el registro de Location con id igual 1 para la compañía con `api_key` igual a 12345:



DELETE localhost:8000/api/location/12345/1

Query Headers² Auth **Body** Tests Pre Run ^{New}

Json Xml Text Form Form-encode GraphQL Binary

1

Status: 200 OK Size: 33 Bytes Time: 28 ms

```
1 {
2   "message": "deleted",
3   "changes": 1
4 }
```

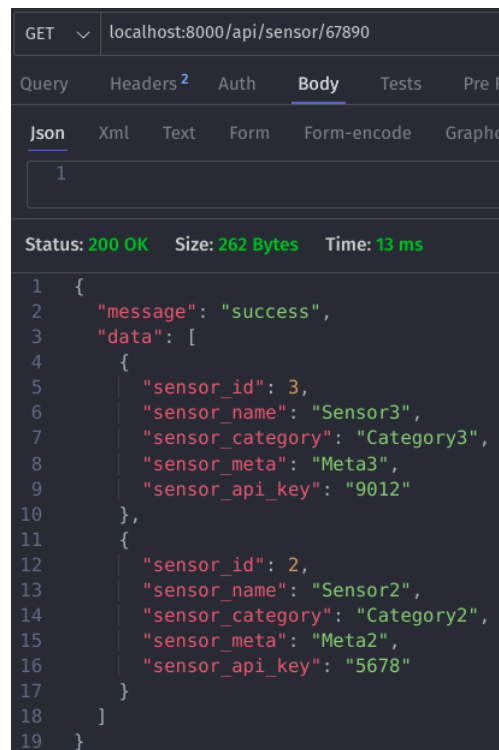
SQLite ×

SQL ▾ < 1 / 1 > 1 - 3 of 3

location_id	company_id	location_name	location_country	location_city	location_meta
2	2	Location3	Country2	City2	Meta3
3	2	Location4	Country2	City2	Meta4
4	1	Location1	Country1	City1	Meta1

Para los *endpoints* de **Sensor**:

- Método GET (todos los datos pertenecientes a Sensor respectivos de una compañía):



GET localhost:8000/api/sensor/67890

Query Headers² Auth **Body** Tests Pre Run

Json Xml Text Form Form-encode GraphQL

1

Status: 200 OK Size: 262 Bytes Time: 13 ms

```
1 {
2   "message": "success",
3   "data": [
4     {
5       "sensor_id": 3,
6       "sensor_name": "Sensor3",
7       "sensor_category": "Category3",
8       "sensor_meta": "Meta3",
9       "sensor_api_key": "9012"
10    },
11    {
12      "sensor_id": 2,
13      "sensor_name": "Sensor2",
14      "sensor_category": "Category2",
15      "sensor_meta": "Meta2",
16      "sensor_api_key": "5678"
17    }
18  ]
19 }
```

- Método GET (datos pertenecientes a un Sensor en particular, respectivo a una compañía):

```

GET localhost:8000/api/sensor/67890/3
Query Headers 2 Auth Body Tests Pre
Json Xml Text Form Form-encode Graph
1
Status: 200 OK Size: 146 Bytes Time: 3 ms
1 {
2   "message": "success",
3   "data": [
4     {
5       "sensor_id": 3,
6       "sensor_name": "Sensor3",
7       "sensor_category": "Category3",
8       "sensor_meta": "Meta3",
9       "sensor_api_key": "9012"
10    }
11  ]
12 }

```

- Método PUT (actualizar datos de Sensor de una compañía en particular): respectivo a una compañía):

```

PUT localhost:8000/api/sensor/3/67890
Query Headers 2 Auth Body 1 Tests Pre Run New
Json Xml Text Form Form-encode GraphQL Binary
1 {
2   "sensor_name" : "Sensor3333",
3   "sensor_category" : "Category3333"
4 }
Status: 200 OK Size: 153 Bytes Time: 34 ms
1 {
2   "message": "success",
3   "data": {
4     "sensor_name": "Sensor3333",
5     "sensor_category": "Category3333"
6   },
7   "req": {
8     "sensor_id": "3",
9     "company_api_key": "67890"
10  },
11  "changes": 1
12 }

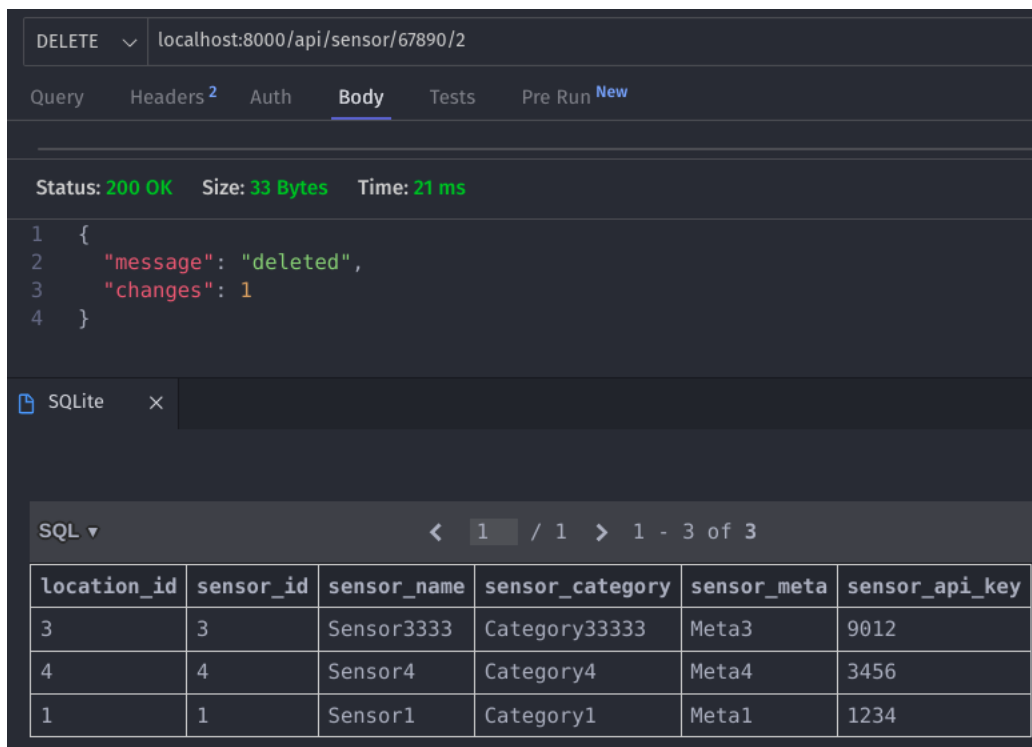
```

SQLite

SQL < 1 / 1 > 1 - 4 of 4

location_id	sensor_id	sensor_name	sensor_category	sensor_meta	sensor_api_key
3	3	Sensor3333	Category33333	Meta3	9012
4	4	Sensor4	Category4	Meta4	3456
2	2	Sensor2	Category2	Meta2	5678
1	1	Sensor1	Category1	Meta1	1234

- Método DELETE (borrar el registro de un Sensor de una compañía en particular) En este ejemplo, se borra el registro de Sensor con id igual 2 para la compañía con api_key igual a 67890:



DELETE localhost:8000/api/sensor/67890/2

Query Headers 2 Auth Body Tests Pre Run New

Status: 200 OK Size: 33 Bytes Time: 21 ms

```
1 {
2   "message": "deleted",
3   "changes": 1
4 }
```

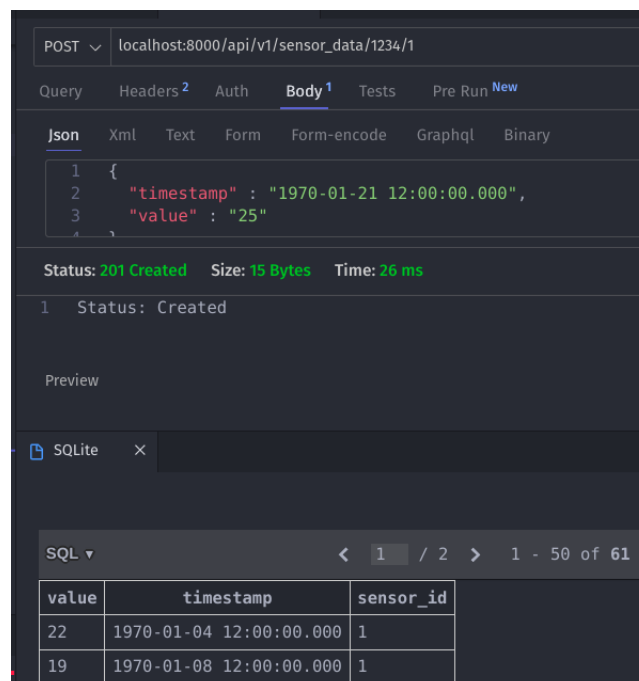
SQLite

SQL 1 / 1 1 - 3 of 3

location_id	sensor_id	sensor_name	sensor_category	sensor_meta	sensor_api_key
3	3	Sensor3333	Category33333	Meta3	9012
4	4	Sensor4	Category4	Meta4	3456
1	1	Sensor1	Category1	Meta1	1234

Para los *endpoints* de Sensor_Data:

- Método POST (Se añade el valor 25 con la fecha "1970-01-21 12:00:00.000" al sensor con api_key igual a 1234):



POST localhost:8000/api/v1/sensor_data/1234/1

Query Headers 2 Auth Body 1 Tests Pre Run New

Json Xml Text Form Form-encode Graphql Binary

```
1 {
2   "timestamp": "1970-01-21 12:00:00.000",
3   "value": "25"
4 }
```

Status: 201 Created Size: 15 Bytes Time: 26 ms

1 Status: Created

Preview

SQLite

SQL 1 / 2 1 - 50 of 61

value	timestamp	sensor_id
22	1970-01-04 12:00:00.000	1
19	1970-01-08 12:00:00.000	1

- Método GET (se extraen los valores registrados dada una company_api_key, en un lapso de tiempo, de un conjunto de sensores:

The screenshot displays a REST client interface with the following details:

- Method:** GET
- URL:** localhost:8000/api/v1/sensor_data/12345
- Body:** A JSON object with the following structure:

```
1 {
2   "to" : "1970-01-09 12:00:00.000",
3   "from": " 1970-01-05 12:00:00.00",
4   "sensor_id": ["1","2"]
5 }
```
- Status:** 200 OK
- Size:** 1.18 KB
- Time:** 13 ms
- Response:** A JSON object with the following structure:

```
1 {
2   "message": "success",
3   "data": [
4     [
5       {
6         "value": 19,
7         "timestamp": "1970-01-07 12:00:00.000",
8         "sensor_id": 1
9       },
10      {
11        "value": 19,
12        "timestamp": "1970-01-08 12:00:00.000",
13        "sensor_id": 1
14      },
15      {
16        "value": 20,
17        "timestamp": "1970-01-06 12:00:00.000",
18        "sensor_id": 1
19      },
20      {
21        "value": 20,
22        "timestamp": "1970-01-09 12:00:00.000",
23        "sensor_id": 1

```

4. Repositorio en Github y enlace a API

La base de datos y la API diseñada se encuentran disponibles en el siguiente repositorio de Github: // Enlace a código y API: [API RESTful para Arquitecturas Emergentes](#).

5. Conclusión

En esta tarea se cumple el objetivo de diseñar una API con arquitectura RESTful para servidores IoT.