Project Report - Application of Big Data

Introduction

This report describes the IT project made by **ARBEY** Louis, **BILLAUDEAU** Thomas and **CRETINON** Pierre-Louis as part of the "Application of big data" course. The aim of the project was to create an API to access a database in real time.

We worked on a dataset named ***Base Sirene des entreprises et de leurs établissements (SIREN, SIRET)***. This dataset is a list of all the companies in France. It contains lot of information about the companies such as the siret number, the address ...

Our project aims to allow the users to access the data of the dataset in real time using an API. The API will be able to **fetch**, **delete**, **add** or **modify** a siret number.

We will now see how the API works and how we managed to create it.

Steps of the project

In order to realize this project, we followed some steps that we will describe now. For the different technologies that we have used, we will see them in a following part.

- Get the dataset
- Insert it into a database (PostgreSQL)
- Create the Rest API

Technologies we have used

During this project we decided to use several technologies, for the coding part, but also the organisational one. We will list the different technologies bellow and argue why we chose it:

The first step was to insert the dataset in the **PostgreSQL** database.

> We decided to use a PostgreSQL database because it is one of the type of Database we didn't work on a lot in our recent projects.

We decided to use **Visual Studio Code** for the coding part of the project.

> This IDE is the one we all use the most in our team. It is also full of great extensions to help me us for a lot of things, like for example *Thunder Client* who allows us to test API requests inside VSCode, or also *Console Ninja* who displays `console.log` output and `runtime errors` directly inside the code

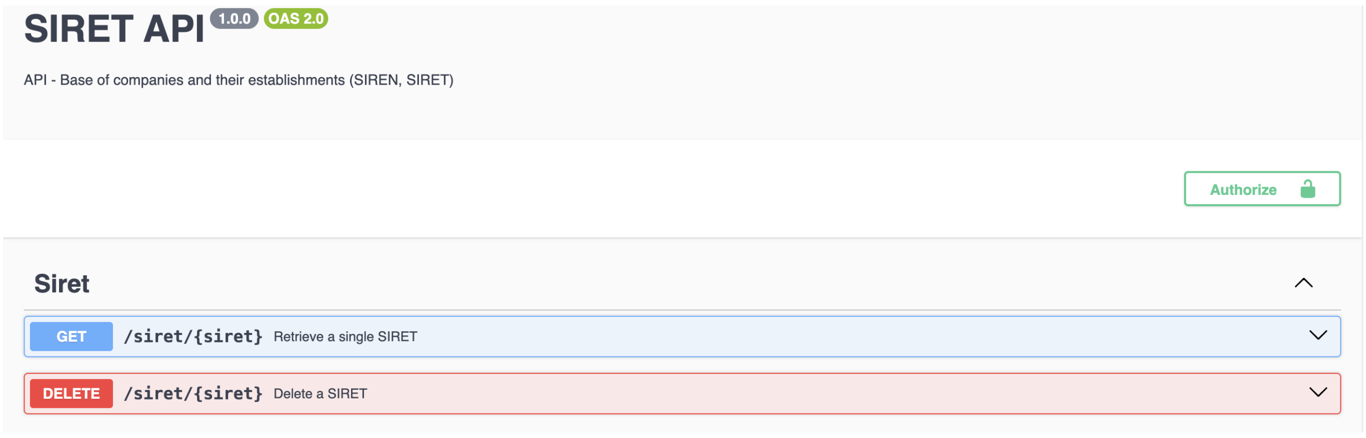
The project will run in **Node.js**.

> We had to choose between Python and Node to develop this API as it is the two language we have experience in API's. But we have chosen Node as we work a lot on Python and we know how Js language is used a lot in general.

To generate some clean and usefull doc for the API we used **Swagger**

> Swagger is an open-source framework for designing, building, and documenting RESTful APIs. We decided to use this because Thomas already knew about it and we saw there a great way to generate clean doc for our API

Swagger interface for documentation



SIRET API 1.0.0 OAS 2.0

API - Base of companies and their establishments (SIREN, SIRET)

GET `/siret/{siret}` Retrieve a single SIRET

DELETE `/siret/{siret}` Delete a SIRET

For our workflow and code storage we used **Github**

> We don't have a lot to say there, but are really used to work with Github as it is a great way to keep track on all our projects, and also being able to revert changes on our code.

How did we used Generative AI

1. We used the extension ***Codeium*** to generate JSDoc for the JS functions. As JSDoc can be time consuming to do this extension allows to do that in a much faster way where we just have to verify that the doc is fine.

2. We also used ***Copilot*** to help us write the code for the functions to add/modify the data in the database. We did that because as the dataset has a lot of columns, it was very dull to do.

3. We also used Copilot to get hints on the structure of the **unit tests** for our API endpoints. However, we made modifications to these tests to ensure they fit our specific workflow and requirements.

How did we do ?

The PostgreSQL database

To setup our **PostgreSQL** database we just did a simple installation who is detailled later in the documentation.

We ran it on one computer and used Ngrok to access the database from other computers from the team. We just had to use temporary url generated by ngrok and add it into the `.env` file.

Then we could create a Table using the Query builder from PGAdmin, but we could just use the SQL terminal also. We used PGAdmin just to have more tools to visualize the data and verify that we had the good numbers of rows inserted for example.

```
CREATE TABLE siret_dataset (
    siren VARCHAR(255),
    nic VARCHAR(255),
    siret VARCHAR(255),
    statutDiffusionEtablissement VARCHAR(255),
    dateCreationEtablissement VARCHAR(255),
    trancheEffectifsEtablissement VARCHAR(255),
    anneeEffectifsEtablissement VARCHAR(255),
    activitePrincipaleRegistreMetiersEtablissement VARCHAR(255),
    dateDernierTraitementEtablissement VARCHAR(255),
    etablissementSiege VARCHAR(255),
    nombrePeriodesEtablissement INTEGER,
    complementAdresseEtablissement VARCHAR(255),
    numeroVoieEtablissement VARCHAR(255),
    indiceRepetitionEtablissement VARCHAR(255),
    typeVoieEtablissement VARCHAR(255),
    libelleVoieEtablissement VARCHAR(255),
    codePostalEtablissement VARCHAR(255),
    libelleCommuneEtablissement VARCHAR(255),
    libelleCommuneEtrangerEtablissement VARCHAR(255),
    distributionSpecialeEtablissement VARCHAR(255),
    codeCommuneEtablissement VARCHAR(255),
    codeCedexEtablissement VARCHAR(255),
    libelleCedexEtablissement VARCHAR(255),
    codePaysEtrangerEtablissement VARCHAR(255),
    libellePaysEtrangerEtablissement VARCHAR(255),
    complementAdresse2Etablissement VARCHAR(255),
    numeroVoie2Etablissement VARCHAR(255),
    indiceRepetition2Etablissement VARCHAR(255),
    typeVoie2Etablissement VARCHAR(255),
    libelleVoie2Etablissement VARCHAR(255),
    codePostal2Etablissement VARCHAR(255),
    libelleCommune2Etablissement VARCHAR(255),
    libelleCommuneEtranger2Etablissement VARCHAR(255),
    distributionSpeciale2Etablissement VARCHAR(255),
    codeCommune2Etablissement VARCHAR(255),
    codeCedex2Etablissement VARCHAR(255),
    libelleCedex2Etablissement VARCHAR(255),
    codePaysEtranger2Etablissement VARCHAR(255),
    libellePaysEtranger2Etablissement VARCHAR(255),
    dateDebut VARCHAR(255),
    etatAdministratifEtablissement VARCHAR(255),
    enseigne1Etablissement VARCHAR(255),
    enseigne2Etablissement VARCHAR(255),
    enseigne3Etablissement VARCHAR(255),
    denominationUsuelleEtablissement VARCHAR(255),
    activitePrincipaleEtablissement VARCHAR(255),
    nomenclatureActivitePrincipaleEtablissement VARCHAR(255),
    caractereEmployeurEtablissement VARCHAR(255)
);
```

To add the data into the database we had to use a command line as the dataset was very big and using PGAdmin for example wasn't possible.

Here is the command

```
COPY siret_dataset FROM [CSV File Url] WITH (FORMAT csv, HEADER);
```

We now have a working PostgreSQL database ready to be used !

With this method, the time for the API to respond was very long (10-12 seconds). In order to accelerate it, we created an **index** on the siret column. The command in order to do it is the following :

```
CREATE INDEX idx_siret ON dataset_first_version(siret)
```

Now, the response time is very quick (Less than 2 seconds !)

The Rest API

Our API is working as follows:

`server.js`: This is the entry point of the API. It sets up the Express server, connects middleware, defines the base route, and starts listening for requests.

`/controllers/siretController.js`: This file contains the controller functions for the SIRET routes. It handles the logic for each route, such as fetching, adding, or deleting SIRET records from the database.

`/models/siretModel.js`: This module interacts directly with the PostgreSQL database. It contains functions that execute SQL queries for the operations initiated by the controller like retrieving or modifying data.

`/routes/siretRoutes.js`: This file defines the routes for the API that relate to SIRET operations. It imports the controller functions and binds them to specific HTTP methods and paths.

`/utils/logger.js`: This file generates logs into the `log.txt` file.

`/tests/siret.test.js`: This file contains unit tests for the API endpoints.

We used an `.env` file to store our private database informations to connect to it.

All the code is available at the Github Repository

Documentation

How to install and run the Database

MacOS (Using HomeBrew 🍺)

Install PostgreSQL database

```
brew install postgresql
```

Start/Stop the database

```
brew services start/stop postgresql
```

To add an admin user

```
psql postgres
```

```
CREATE ROLE newUser WITH LOGIN PASSWORD ‘password’;
ALTER ROLE newUser CREATEDB;
```

Windows (Using Docker 🐳)

1. Install Docker
2. Download PostgreSQL Image

```
docker pull postgres
```

3. Create a Container

```
docker run --name postgres -e POSTGRES_PASSWORD=postgres -p 5432:5432 -d
postgres
```

Verify container is running

```
docker ps
```

4. Connect to the PostgreSQL Database

```
docker exec -it postgres psql -U postgres
```

5. Add a user

```
CREATE ROLE newUser WITH LOGIN PASSWORD 'password';
ALTER ROLE newUser CREATEDB;
```

How to install and run the API

1. Clone the Github Repository

```
git clone https://github.com/TBillaudeau/SIREN-SIRET-API
```

2. Access the folder

```
cd SIREN-SIRET-API
```

3. Install the dependencies

```
npm install
```

4. Create `.env` file
   We need to create our own environements file to store the database information. Just create a file named `.env` as follows:

```
DB_USER=<username>
DB_HOST=localhost
DB_NAME=<db-name>
DB_PASSWORD=<password>
DB_PORT=5432
```

5. Test the application

```
npm test
```

6. Run the application

```
npm start
```

The API is now running and ready to be used. Because we used Swagger the app will open the swagger interface on the port 3000 and allows you to test all the endpoints. But you can also use normal requests, like CURL to try the API.

Examples:

```
curl -X 'GET' \
  'http://localhost:3000/siret/39889833800023' \
  -H 'accept: */*'
```

> This request will return the data of the siret number 39889833800023

```
curl -X 'DELETE' \
  'http://localhost:3000/siret/39889833800023' \
  -H 'accept: */*'
```

> This request will delete the siret number 39889833800023

```
curl -X 'POST' \
  'http://localhost:3000/siret?
siret=39889833800023&libellevoieetablissement=Avenue%20de%20la%20r%C3%A9pu
blique&libellecommuneetablissement=Villejuif' \
  -H 'accept: */*' \
  -d ''
```

> This request will add the siret number 39889833800023 with the libellevoieetablissement "Avenue de la république" and the libellecommuneetablissement "Villejuif"

```
curl -X 'PUT' \
  'http://localhost:3000/siret?
siret=39889833800023&libellevoieetablissement=Avenue%20de%20la%20r%C3%A9pu
blique&libellecommuneetablissement=Villejuif' \
  -H 'accept: */*' \
  -d ''
```

> This request will modify the siret number 39889833800023 with the libellevoieetablissement "Avenue de la république" and the libellecommuneetablissement "Villejuif"

References

Here is a list of blogs we used to help us during this project.

- Install PostgreSQL using docker🐳
- Install PostgreSQL using Homebrew🍺
- Getting started on Swagger
- Create a REST api with Node.js
- How to use Thunder Client, an alternative to Postman
- How to create supertest

> A project made with a lot of ❤️ by **ARBEY** Louis, **BILLAUDEAU** Thomas and **CRETINON** Pierre-Louis