1 Introduction

Dans ce TP, vous allez découvrir comment définir, créer et appeler une *API* [Wika] afin d'en comprendre l'intérêt, les avantages et les limites. Vous allez en particulier découvrir l'usage de la spécification *OpenAPI* [Swae] au travers des outils mis en place par *Swagger* [Swag].

Pour rappel, vous devez rendre un rapport individuel à la fin du semestre. Vous devez y mettre toutes les informations qui vous semblent pertinentes. Notez que les réponses aux questions encadrées en gris seraient à considérer comme un minimum à faire apparaitre dans ce rapport.

2 Découverte d'OpenAPI [Swae]

Le principe fondamental d'OpenAPI est de définir une API au travers d'un fichier de description (YAML [Wikb]). Ce simple fichier doit permettre :

- à un humain de comprendre les points d'accès à l'API (les serveurs, les chemins disponibles avec ses paramètres, les valeurs de retour attendues etc).
- à un développeur de créer une structure de code pour le serveur de cette API, notamment au travers d'un générateur de code automatique (nous allons utiliser Swagger Codegen [Swad]).
- à un client de l'API de créer une structure de code permettant d'interroger cette API simplement.

Vous avez un exemple de base proposé en Figure 1. Cet exemple sera la base de travail pour ce TP.

```yaml
openapi: 3.0.0
info:
  title: TimeSeries API IoT
  description: Optional multiline or single-line description in [CommonMark](http://commonmark.org/help/) or HTML.
  version: 0.42
servers:
  - url: http://localhost:8080/v1
    description: Optional server description, e.g. Internal staging server for testing
  - url: http://api.example.com/v1
    description: Optional server description, e.g. Main (production) server
paths:
  /mean/{sensorId}:
    get:
      summary: Calculer la moyenne d'un capteur entre deux dates
      description: Optional extended description in CommonMark or HTML.
      parameters:
        - in: path
          name: sensorId
          schema:
            type: string
          required: true
          description: String Id of the sensor to get
        - in: query
          name: startDate
          schema:
            type: integer
          description: Integer/timestamp of the start date
        - in: query
          name: endDate
          schema:
            type: integer
          description: Integer/timestamp of the end date
      responses:
        '200':    # status code
          description: A JSON array of the mean of values between start and end dates
          content:
            application/json:
              schema:
                type: array
                items:
                  type: integer
```

Figure 1 – Exemple d'*OpenAPI* permettant de définir une API avec un chemin (*/mean/{sensorId}*) prenant 3 paramètres, l'un dans le chemin et deux de forme "query". Cet exemple est basé sur les sources disponibles dans la documentation d'OpenAPI, notamment cette source [Swaa].

Je vous invite à explorer la documentation d'OpenAPI notamment pour :
- Comprendre la structure basique d'un fichier YAML OpenAPI [Swaa].
- Comprendre les principes de "path" et "operation" [Swaf].
- Comprendre comment déclarer les paramètres des différents chemins [Swab].
- Comprendre la façon de décrire les réponses des requêtes [Swac].

3 Prise en main de Swagger Codegen [Swad]

Bien, vous êtes maintenant incollables sur OpenAPI (Pas encore ? Bah retournez lire la doc alors !), nous allons pouvoir voir comment fonctionne la génération automatique de code avec Swagger Codegen [Swad].

Vous allez déployer un environnement de développement se basant sur Swagger Codegen et Docker :

1. Vous pouvez utiliser la commande basée sur le Docker préparé par Swagger :

   ```
   docker run --rm -v $PWD:/local swaggerapi/swagger-codegen-cli-v3:3.0.14 generate -i
   /local/timeseries_iot.yaml -l python-flask -DpackageName=TimeSeriesIoT -o /local/out/python-ts-iot
   ```

   pour :
   - générer un code à partir du fichier "`timeseries_iot.yaml`" (qui contiendrait par exemple... le code proposé en Figure 1).
   - basé sur le framework "`Python-Flask`" [Ron]. (Mais vous pouvez aussi choisir parmi les options suivantes : aspnetcore, csharp, csharp-dotnet2, dynamic-html, html, html2, java, jaxrs-cxf-client, jaxrs-cxf, inflector, jaxrs-cxf-cdi, jaxrs-spec, jaxrs-jersey, jaxrs-di, jaxrs-resteasy-eap, jaxrs-resteasy, micronaut, spring, nodejs-server, openapi, openapi-yaml, kotlin-client, kotlin-server, php, python, python-flask, scala, scala-akka-http-server, swift3, swift4, typescript-angular, javascript...)
   - vers le dossier "`out/python-ts-iot`".
   - en le nommant "`TimeSeriesIoT`".
2. Vous avez donc maintenant, dans le dossier "`out/python-ts-iot`", une structure de fichiers permettant de déployer une API basée sur votre définition OpenAPI et s'appuyant sur le framework Python-Flask. (Non, n'applaudissez pas encore, c'est encore plus beau après...)

   Nous allons donc maintenant déployer cette API avec... Docker ! (Bah non, pourquoi vous fuyez maintenant ?)
   (a) Vous devriez voir un fichier "`Dockerfile`" dans le dossier "`out/python-ts-iot`" généré. Vous allez modifier ce fichier en lui ajoutant une ligne permettant d'intégrer dans votre API une interface de test et de documentation. Faites en sorte que le fichier "`Dockerfile`" ressemble à celui proposé en Figure 2. (Normalement, vous avez simplement à ajouter la ligne "`RUN pip3 install connexion[swagger-ui]`" au fichier généré automatiquement).
   (b) À partir de la Figure 3, créez un fichier "`docker-compose.yml`" dans le dossier pour faciliter la création du Docker et son ajout à notre réseau virtuel "`iot-labs`".
   (c) Créez l'image Docker avec la commande `docker-compose build`
   (d) Lancez le service Docker avec la commande `docker-compose up -d`
   (e) Ouvrez votre navigateur à l'adresse `http://localhost:8080/v1/ui` pour voir cette belle interface et jouer avec votre API déployée ! Elle devrait ressembler à la Figure 4.

```
1  FROM python:3.6-alpine
2
3  RUN mkdir -p /usr/src/app
4  WORKDIR /usr/src/app
5
6  COPY requirements.txt /usr/src/app/
7
8  RUN pip3 install --no-cache-dir -r requirements.txt
9  RUN pip3 install connexion[swagger-ui]
10
11 COPY . /usr/src/app
12
13 EXPOSE 8080
14
15 ENTRYPOINT ["python3"]
16
17 CMD ["-m", "TimeSeriesIoT"]
```

FIGURE 2 – Fichier *Dockerfile* permettant d'intégrer dans un Docker un code Python en y installant les dépendances et lançant le programme au lancement du Docker.

```yaml
version: '3.7'
services:

    api:
        build: .
        image: "myapi"
        hostname: "myapi"
        ports:
            - "8080:8080"
        networks:
            - iot-labs
        labels:
            NAME: "myapi"

networks:
    iot-labs:
        external: true
```

FIGURE 3 – Fichier *docker-compose.yml* permettant de définir un service Docker basé sur un *build* et attaché à notre réseau virtuel "*iot-labs*".

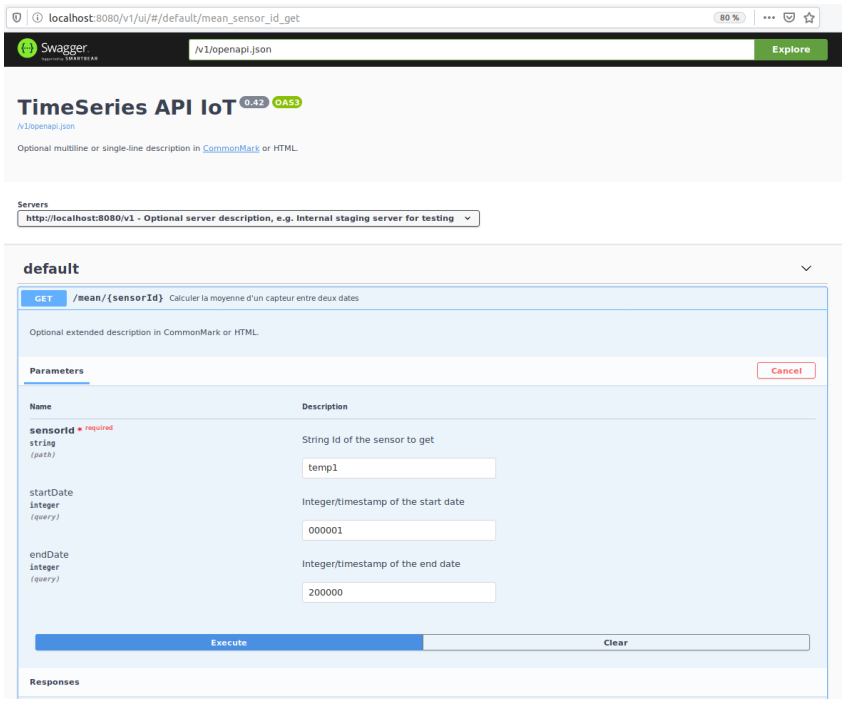

FIGURE 4 – Interface *Swagger* permettant de tester l'API déployée.

4 How to "do some magic" ? – Remplir le code de Swagger Codegen

C'est bien beau tout ça, j'ai bien une structure de code qui me permet d'appeler une API en lui passant des paramètres mais... ça me renvoie toujours un `"do some magic!"`

Vous avez vraiment cru que ça allait tout générer pour vous ? (Parfois je me dis... non, je vais le garder pour moi...).

Il est temps de bosser un peu !

Vous allez maintenant modifier le code généré pour... lui faire faire quelque chose :

1. Ouvrez le fichier `"controllers/default_controller.py"`, c'est lui qui contient les fonctions qui sont appelées quand vous requêtez bien votre API. Vous devriez voir une fonction `"mean_sensor_id_get"` (Hum... c'est étrange ça, dans le YAML en Figure 1, on a déclaré un "path" `/mean/sensorId` dans lequel on a mis une méthode `get`, est-ce que ça pourrait être lié à ce nom de fonction ? !). Vous devriez voir que cette fonction finit par... `return 'do some magic!'` !
2. Modifiez cette ligne de retour par `return 'Sensor : ' + sensor_id + ' [start:end] : [' + str(start_date) + ':' + str(end_date) + ']'`
3. Reconstruisez votre image Docker : `docker-compose build`

4. Recréez votre conteneur : `docker-compose up -d -force-recreate`
5. Allez tester à nouveau votre API :
   — À partir de l'interface `http://localhost:8080/v1/ui`
   — À partir de Postman [Pos] que vous aurez préalablement installé. Vous devriez avoir quelque chose comme présenté en Figure 5.

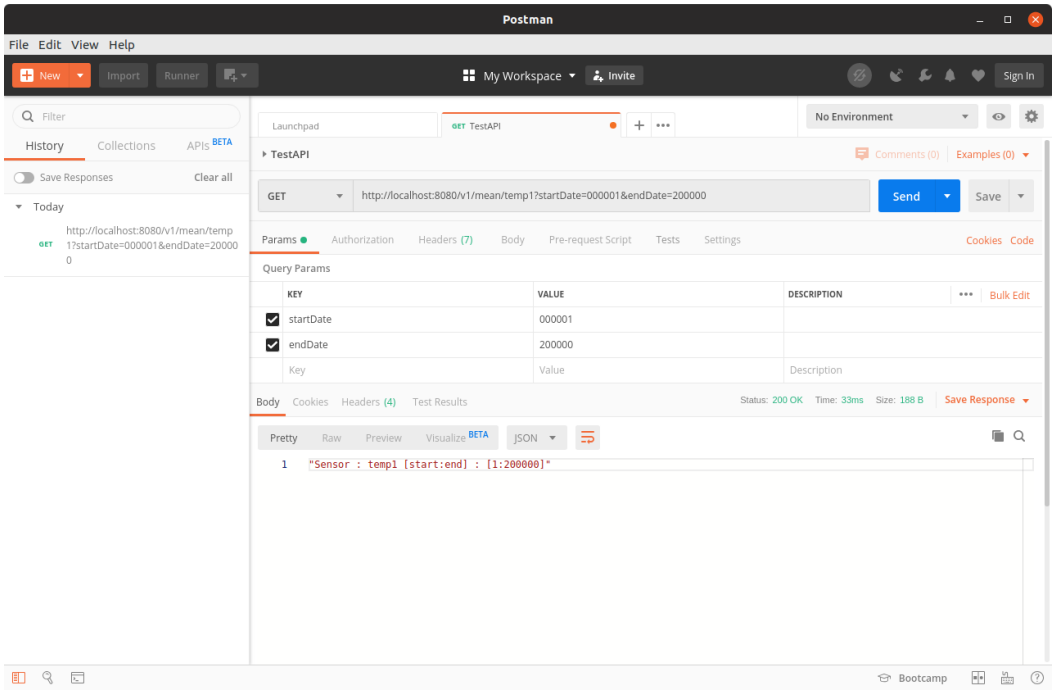

FIGURE 5 – Interface *Postman* [Pos] permettant de tester l'API déployée.

5 Et maintenant... on fait le lien avec le TP2 ?

Vous pouvez maintenant récupérer les codes créés durant le TP2 pour :

1. Ajouter dans la fonction `"mean_sensor_id_get"` le code permettant de requêter votre base de données *MongoDB* et calculer la moyenne des valeurs d'un capteur entre deux dates fixées.
2. Modifier votre définition OpenAPI et le code généré pour ajouter un chemin permettant de récupérer la valeur minimale d'un capteur entre deux dates fixées à partir des données de votre base de données *MongoDB*.
3. Modifier votre définition OpenAPI et le code généré pour ajouter un chemin permettant de récupérer la dernière valeur connue d'un capteur à partir des données de votre base de données *MongoDB*.

6 Optionnel – Et si on faisait le lien avec Nifi ?

Votre API est maintenant fonctionnelle. Mais pourquoi ?

1. Ajouter un *flow Nifi* permettant de requêter votre API toutes les 10 minutes afin de vérifier si la moyenne d'un capteur de puissance ne dépasse pas un seuil spécifique. Si ce seuil est dépassé, utiliser un processeur Nifi pour vous envoyer un mail avec la valeur de la moyenne.

Références

[Pos] POSTMAN. *The Collaboration Platform for API Development*. URL : `https://www.getpostman.com/`.

[Ron] ARMIN RONACHER. *Python Flask*. URL : `https://palletsprojects.com/p/flask/`.

[Swaa] SWAGGER. *Basic Structure*. URL : `https://swagger.io/docs/specification/basic-structure/`.

[Swab] Swagger. *Describing Parameters*. URL : https://swagger.io/docs/specification/describing-parameters/.

[Swac] Swagger. *Describing Responses*. URL : https://swagger.io/docs/specification/describing-responses/.

[Swad] Swagger. *Git repository of SwaggerCodeGen*. URL : https://github.com/swagger-api/swagger-codegen.

[Swae] Swagger. *OpenAPI Specification (OAS)*. URL : https://swagger.io/specification/.

[Swaf] Swagger. *Paths and Operations*. URL : https://swagger.io/docs/specification/paths-and-operations/.

[Swag] Swagger. *Swagger – API Development for Everyone*. URL : https://swagger.io.

[Wika] Wikipedia. *Interface de programmation (API)*. URL : https://fr.wikipedia.org/wiki/Interface_de_programmation.

[Wikb] Wikipedia. *Yet Another Markup Language (YAML)*. URL : https://fr.wikipedia.org/wiki/YAML.