



65, rue des Grands Moulins, 75013 PARIS

Création d'un back end permettant l'accès à des données avec une API REST

BOUZARD ARTHUR & AUFRRET FABIENNE

Institut National des Langues et Civilisations Orientales (I.N.A.L.C.O)

Master 2 TAL IM- Février 2021

Sommaire

CREATION D'UN BACK END PERMETTANT L'ACCES A DES DONNEES AVEC UNE API REST	1
1. CREATION ET TESTS EN LOCAL	1
2. DEPLOIEMENT SUR HEROKU	7
2.1 PREPARATION ET UPLOAD	7
2.2 TESTS SUR HEROKU.....	9
3. MODES D'EMPLOI	11
3.1 INSTALLATION	11
3.2 UTILISATION	12
4. CONCLUSION & PERSPECTIVES	25

1. *Création et tests en local*

Nous avons tout d'abord développé notre application en local.

Nous avons utilisé Flask et SQLAlchemy pour créer une base de données sous PostgreSQL avec deux tables (Users et Data) qui seront remplies en lisant les fichiers users.json et les données FR.tsv.

Nous avons fait en sorte que le programme se connecte à cette base de données dont l'Uri est spécifié dans le fichier config.py (SQLALCHEMY_DATABASE_URI = 'postgresql+psycopg2://tp:tp@localhost/geoname').

Le fichier api.py est ce qui servira de point d'entrée pour notre application. Cela indiquera à notre serveur Gunicorn comment interagir avec l'application. Nous y avons indiqué les « routes » des requêtes.

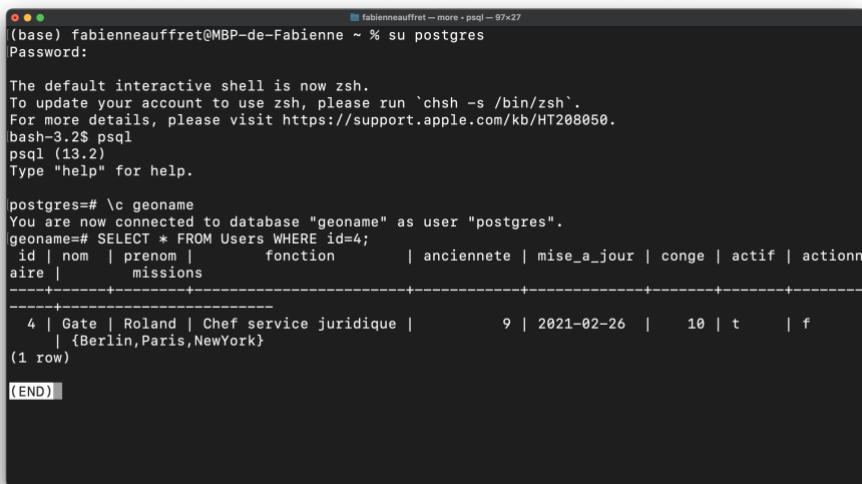
C'est le fichier models.py qui contient la structure des tables et les méthodes qui leur seront appliquées (lire, effacer, éditer, ajouter).

Le programme construit ensuite les tables users et data si elles n'existent pas dans la base de données puis les peuple avec les données fournies.

Nous avons ensuite écrit les fichiers DataService.py et UsersService.py (dans un répertoire services) qui traitent les requêtes des utilisateurs en utilisant les méthodes définies dans chaque classe Users ou Data.

Nous avons ajouté les méthodes peu à peu, les testant au fur et à mesure (en regardant d'abord dans la base en ligne de commande, puis avec Postman).

Nous avons fait des requêtes en local en ligne de commande (on se connecte avec le bon utilisateur, puis on lance Postgres – commande psql- puis on se connecte à la table\c nom_de_la_table et on peut faire des requêtes) pour vérifier que notre base de données avait été bien initialisée.



```
(base) fabienneauffret@MBP-de-Fabienne ~ % su postgres
Password:
The default interactive shell is now zsh.
To update your account to use zsh, please run `chsh -s /bin/zsh`.
For more details, please visit https://support.apple.com/kb/HT208050.
bash-3.2$ psql
psql (13.2)
Type "help" for help.

postgres=# \c geoname
You are now connected to database "geoname" as user "postgres".
geoname=# SELECT * FROM Users WHERE id=4;
 id | nom    | prenom   | fonction          | anciennete | mise_a_jour | conge | actif | actionnaire | missions
----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
  4 | Gate   | Roland   | Chef service juridique | 9 | 2021-02-26 | 10 | t   | f
   | {Berlin,Paris,NewYork}
(1 row)

(END)
```

Nous avons traité divers cas d'erreurs (mauvais champ, données absentes, etc.) et renvoyé un message indiquant à l'utilisateur le statut de la requête traitée.)

Nous avons aussi proposé une requête avec regex (sachant que pour alchemy, les _ remplacent le point).

Nous avons testé toutes les sortes de requêtes et nos messages d'erreurs avec Postman en local :

The screenshot shows the Postman application interface. On the left, the 'Team Workspace' sidebar lists various collections, environments, and monitors. In the center, a specific collection named 'TP' is selected, containing several API endpoints. One endpoint, 'GET / Get data by field and regex', is highlighted. The 'Body' tab of the request panel shows a GET request to 'localhost:5000/data/admin4/84_'. The response panel displays a JSON object with numerous fields, including 'admin4', 'longitude', 'asciname', 'id', 'alternate_names', 'cc2', 'admin1', 'geonameid', 'country_code', 'feature_class', 'admin2', 'elevation', 'feature_code', 'admin3', 'modification_date', 'latitude', 'dem', 'name', 'timezone', and 'population'. The status bar at the bottom indicates a 200 OK status with a time of 240 ms and a size of 469.88 KB.

Et ici avec le champ « professeur » au lieu de « actionnaire » dans le body de la requête :

The screenshot shows the Postman application interface. On the left, the sidebar displays a 'Team Workspace' with various collections, environments, and mock servers. The main workspace shows a 'TP / Put user' collection under 'PUT'. The request URL is 'localhost:5000/users'. The 'Body' tab is selected, showing a JSON payload:

```
1 {
2     "nom": "Samuel",
3     "prenom": "Jackson",
4     "fonction": "Chef service matériel informatique",
5     "anciennete": 20,
6     "conge": 14,
7     "actif": true,
8     "professeur": false,
9     "missions": [
10         "Paris"
11     ]
12 }
```

Below the request, the response status is shown as 'Status: 400 BAD REQUEST' with a size of '186 B'. The response body is:

```
1 {
2     "error": "Wrong fields"
3 }
```

Nous avons ensuite implémenté Flask-JWT dans AuthService.py pour limiter les connexions (sans créer de nouveaux champs password et uuid dans la table user, ce qui avait été la première piste suivie, avec l'utilisation de tokens), en adaptant les choix de ce tutoriel : <https://pythonhosted.org/Flask-JWT/>.

Désormais il faut s'identifier dans Postman pour se connecter à l'api. Il faut envoyer un body sur l'url du serveur /auth, contenant les champs username et password correctement renseignés pour recevoir un token qu'il faut coller dans l'onglet 'Authorization' de postman :

The screenshot shows the Postman interface with a collection named 'TP'. A GET request is selected with the URL 'localhost:5000/data/admin4/2,*'. The 'Authorization' tab is active, displaying a 'Bearer Token' field containing a long string of characters: 'eyJhbGciOiJIUzI1NiJ9.eyJzdWIiOiJlZG1tY2EwMjAxMjIyNjQxMSIsImF1ZGUiOiJ0ZXN0ZWdlbmRvZWQifQ.8e4gjmblmugXcdVp0018y2a7cuChfC-taK-wTsEJR1'. Below the token, a note says: 'The authorization header will be automatically generated when you send the request. Learn more about authorization'.

The screenshot shows the Postman interface with a collection named 'TP'. A POST request is selected with the URL 'localhost:5000/auth'. The 'Body' tab is active, showing a JSON object with two fields: 'username' and 'password', both set to 'admin2021'. The response status is 200 OK, and the response body is a JSON object with an 'access_token' field containing a long string of characters: 'eyJhbGciOiJIUzI1NiJ9.eyJzdWIiOiJlZG1tY2EwMjAxMjIyNjQxMSIsImF1ZGUiOiJ0ZXN0ZWdlbmRvZWQifQ.8e4gjmblmugXcdVp0018y2a7cuChfC-taK-wTsEJR1'.

Nous récupérons ce token, et si nous ne copions pas le token dans l'en tête de la requête nous avons un message d'erreur :

The screenshot shows the Postman application interface. On the left, the sidebar includes sections for Team Workspace, APIs, Environments, Mock Servers, Monitors, and History. The main workspace displays a collection named 'TP' containing a 'POST Login' request. This request is set to 'GET' method, targeting 'localhost:5000/users/1'. In the 'Authorization' tab, the 'Type' is set to 'Bearer Token' and the 'Token' field is empty. Below the request, the response body is shown in JSON format:

```
1  ...
2  ...
3  ...
4  ...
5  ...
```

The response status is 401 UNAUTHORIZED. The error message in the response body is: "description": "Request does not contain an access token", "error": "Authorization Required", "status_code": 401".

Avec le token on obtient l'affichage suivant :

The screenshot shows the Postman application interface. On the left, the sidebar displays 'Team Workspace' with a collection named 'TP'. Under 'TP', there is a 'POST Login' section containing several methods: 'Get user by id', 'Get all users', 'Get user field by id', 'Get user by field and regex', 'PUT user', 'Edit user by id', 'Delete user', 'Get data by id', 'Get data field by id', 'Get data by field and regex', 'PUT Put data', 'Edit data', 'Delete data', 'POST Login Heroku', and 'GET Test heroku'. The main workspace shows a 'GET / Get user by id' request to 'localhost:5000/users/1'. The 'Authorization' tab is selected, showing a 'Bearer Token' input field with a placeholder 'Token' and a note: 'The authorization header will be automatically generated when you send the request. Learn more about authorization'. The 'Body' tab is selected, displaying a JSON response with the following content:

```
1  "fonction": "TEEEEEST22",
2  "id": 1,
3  "missions": [
4    "Bruxelle",
5    "Paris",
6    "Pakistan"
7  ],
8  "prenom": "Thomas",
9  "mise_a_jour": "2021-02-26",
10  "nom": "GIRERO",
11  "actif": false,
12  "actionnaire": true,
13  "anciennete": 10,
14  "conge": 15
```

The status bar at the bottom indicates a 200 OK response with 300 ms and 435 B.

2. Déploiement sur Heroku

2.1 Préparation et upload

Nous avons créé un fichier requirements.txt listant modules python nécessaires à Heroku et avons généré un fichier pipenv avec la commande ‘pipenv install –r requirements.txt’ utilisé par Heroku pour installer les librairies python.

Puis un fichier Procfile, qui définit les processus/commandes qui doivent être lancés pour que Heroku puisse faire tourner notre application, où nous avons mis la commande web pour indiquer à Heroku de lancer gunicorn (wsgi http serveur pour notre application).

Ensuite, nous avons ajouté une base PostgreSQL dont nous avons récupéré les variables (mot de passe, hôte, nom de la db). La limite pour un compte gratuit étant de 10,000 lignes et notre fichier de lieux en faisant 148539, nous l'avons d'abord réduit à 8000 lignes (cf. l'application ici, par exemple avec cette requête :

https://tp1web.herokuapp.com/data/asciiname/Bi_ * qui renvoie des lieux dont le nom ASCII commence par Bi).

Par la suite, nous avons connecté le programme à une base de données externe (stockage Amazon WS gratuit pour une année, cf. <https://aws.amazon.com/fr/products/databases/> et <https://aws.amazon.com/fr/getting-started/hands-on/create-connect-postgresql-db/>) qui permet d'exploiter les fichiers de données en entier, voir ici : <https://inappco.herokuapp.com/data/56991>.

Dans les deux cas, nous avons stocké les identifiants de connexion dans une variable locale (utilisée dans le fichier config.py) :

The screenshot shows the Heroku dashboard interface. At the top, there's a navigation bar with various links like 'Dashboard', 'Gestion des plans...', 'Paramètres', 'Importés depuis Fl...'. Below it, the main header says 'tp1techniqueweb > tp1web' with tabs for 'Overview', 'Resources', 'Deploy', 'Metrics', 'Activity', 'Access', and 'Settings'. Under 'Overview', the 'App Information' section shows the app name 'tp1web', region 'Europe', stack 'heroku-20', framework 'Python', slug size '98.3 MB of 500 MB', GitHub repo 'totoine/TotoB1', and Heroku git URL 'https://git.heroku.com/tp1web.git'. In the 'Config Vars' section, there's a note: 'Config vars change the way your app behaves. In addition to creating your own, some add-ons come with their own.' Below this, there's a table with four rows: 'DATABASE_URL' with value 'postgres://txmgdpzvuzw1sq:f76858a9d62a415d658;', 'URL' with value '//txmgdpzvuzw1sq:f76858a9d62a415d658;', 'SK' with value 'g\snclzTfI\''v8CwIzL\''URMlA'', and 'KEY' with a 'VALUE' input field and an 'Add' button. At the bottom, there's a 'Buildpacks' section with a 'Docker.dmg' entry and a 'Tout afficher' button.

Ce qui s'écrit dans le config.py sous la forme :

```

config.py -- techniques web
EXPLORER TECHNIQUES WEB
> MBONING_eval_SA_serveur_M2_INALCO-2021
> proj2
> projet
  > api_run.py
  > Techniques web_2020 - 2021-cours2_files
> TP
  > python-getting-started
> Twel1
  > __pycache__
  > .vscode
  > Data
    FR.tsv
  README.md
  users.json
  myapp
    > __pycache__
    > services
      > __pycache__
        AuthService.py
        DataService.py
        UserService.py
        __init__.py
        api.py
        models.py
      .env
    config.py
    equip.txt
    geoname.dump
    Pipfile
    Procfile
    README.md
> OUTLINE
> TIMELINE

```

```

config.py
1 import os
2 #User: rdp@ip/database
3
4 DATABASE_URI = os.environ['URI']
5 SQLALCHEMY_DATABASE_URI = "postgresql+psycopg2://{}@{}/{}".format(DATABASE_URI)
6 SQLALCHEMY_TRACK_MODIFICATIONS = False
7 #Hash Key
8
9 SECRET_KEY = os.environ['SK']
10
11 JWT_AUTH_HEADER_PREFIX = 'BEARER'

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

No problems have been detected in the workspace.

Heroku a le SSL (encryption) activé par défaut, ce qui garantit une certaine sécurité.

Une fois ces préparations faites nous avons connecté notre git à Heroku et pushé l'application pour la tester. La commande heroku logs --tail permet de suivre les logs et de voir que les requêtes passent bien :

```

2021-02-26T21:53:28.147026+00:00 app[api]: Deploy a65e3307 by user tofrine@gmail.com
2021-02-26T21:53:28.147026+00:00 app[api]: Release v24 created by user tofrine@gmail.com
2021-02-26T21:53:30.480740+00:00 heroku[web.1]: Starting process with command `unicorn`
2021-02-26T21:53:30.540227+00:00 app[web.1]: [2021-02-26 21:53:30 +0000] [8] [INFO] Worker exiting (pid: 8)
2021-02-26T21:53:30.540239+00:00 app[web.1]: [2021-02-26 21:53:30 +0000] [4] [INFO] Handling signal: term
2021-02-26T21:53:30.541293+00:00 app[web.1]: [2021-02-26 21:53:30 +0000] [9] [INFO] Worker exiting (pid: 9)
2021-02-26T21:53:30.640866+00:00 app[web.1]: [2021-02-26 21:53:30 +0000] [4] [INFO] Shutting down: Master
2021-02-26T21:53:34.732538+00:00 heroku[web.1]: Process exited with status 0
2021-02-26T21:53:34.732538+00:00 heroku[web.1]: Starting process with command `unicorn`
2021-02-26T21:53:37.323679+00:00 app[web.1]: [2021-02-26 21:53:37 +0000] [4] [INFO] Starting unicorn 20.0.4
2021-02-26T21:53:37.323679+00:00 app[web.1]: [2021-02-26 21:53:37 +0000] [4] [INFO] Listening at: http://0.0.0.0:17213 (4)
2021-02-26T21:53:37.325201+00:00 app[web.1]: [2021-02-26 21:53:37 +0000] [4] [INFO] Using worker: sync
2021-02-26T21:53:37.330023+00:00 app[web.1]: [2021-02-26 21:53:37 +0000] [8] [INFO] Booting worker with pid: 8
2021-02-26T21:53:37.373415+00:00 app[web.1]: [2021-02-26 21:53:37 +0000] [9] [INFO] Booting worker with pid: 9
2021-02-26T21:53:39.710770+00:00 heroku[web.1]: State changed from starting to up
2021-02-26T21:53:39.240975+00:00 app[web.1]: WARNING:root:Database initialized!
2021-02-26T21:53:39.245249+00:00 app[web.1]: WARNING:root:Database initialized!
2021-02-26T21:53:39.245249+00:00 app[web.1]: Build succeeded
2021-02-26T21:57:47.405229+00:00 app[web.1]: at=info method=GET path="/data/ascinname/Y_*" host=tpiweb.herokuapp.com request_id=d88cc0d1-a65a-4780-8887-0d4d80ff0e83 fwd="170.132.287.239" dyno=web.1 connect=1ms service=11ms status=200 bytes=47190 protocol=https
2021-02-26T21:57:47.398751+00:00 app[web.1]: 10.9.53.46 - - [26/Feb/2021:21:57:47 +0000] "GET /data/ascinname/Y_* HTTP/1.1" 200 41550 "-" "Mozilla/5.0 (Macintosh; Intel Mac OS X 11_2_1) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/88.0.4324.182 Safari/537.36"
2021-02-26T22:00:14.443105+00:00 heroku[router]: at=info method=GET path="/users/6" host=tpiweb.herokuapp.com request_id=3e670 def=40b4-44eb-884a-5b18a43724bc7 fwd="176.132.207.239" dyno=web.1 connect=1ms service=10ms status=404 bytes=187 protocol=https
2021-02-26T22:00:14.445030+00:00 app[web.1]: 10.74.57.236 - - [26/Feb/2021:22:00:14 +0000] "GET /users/6 HTTP/1.1" 404 28 "-" "Mozilla/5.0 (Macintosh; Intel Mac OS X 11_2_1) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/88.0.4324.182 Safari/537.36"
2021-02-26T22:00:20.404615+00:00 heroku[router]: at=info method=GET path="/users/6" host=tpiweb.herokuapp.com request_id=d7db725e9-34d4-422c-a1e4-c8238e67cd9c fwd="176.132.207.239" dyno=web.1 connect=0ms service=9ms status=200 bytes=378 protocol=https
2021-02-26T22:00:20.406507+00:00 app[web.1]: 10.74.57.236 - - [26/Feb/2021:22:00:20 +0000] "GET /users/6 HTTP/1.1" 200 225 "-" "Mozilla/5.0 (Macintosh; Intel Mac OS X 11_2_1) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/88.0.4324.182 Safari/537.36"
2021-02-26T22:10:21.819317+00:00 heroku[router]: at=info method=GET path="/data/id/5488" host=tpiweb.herokuapp.com request_id=76910e02-0d9e-abef-abef-5b05e4136a1b fwd="176.132.207.239" dyno=web.1 connect=1ms service=3ms status=200 bytes=189 protocol=https
2021-02-26T22:10:21.816959+00:00 app[web.1]: 10.9.54.145 - - [26/Feb/2021:22:10:21 +0000] "GET /data/id/5488 HTTP/1.1" 200 37 "-" "Mozilla/5.0 (Macintosh; Intel Mac OS X 11_2_1) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/88.0.4324.182 Safari/537.36"

```

2.2 Tests sur Heroku

Comme en local nous avons d'abord vérifié la création de la base et son initialisation correcte en faisant des requêtes en ligne de commande, en utilisant : **psql -h hostname -d databasename -U username** :

```
(base) fabienneauffret@MBP-de-Fabienne Tweb1 % psql -h ec2-54-73-68-39.eu-west-1.compute.amazonaws.com -d dcrvk1f903a127 -U zxmgdpzvuzwlxq
Password for user zxmgdpzvuzwlxq:
SSL connection (protocol: TLSv1.3, cipher: TLS_AES_256_GCM_SHA384, bits: 256, compression: off)
Type "help" for help.

dcrvk1f903a127=> SELECT * FROM Data where id=80012
dcrvk1f903a127-> ;
 id | geonameid | name | asciname | alternate_names | latitude | longitude | feature_class | feature_code | country_code | cc2 | admin1 | admin2 | adm
in3 | admin4 | population | elevation | dem | timezone | modification_date
-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
(0 rows)

dcrvk1f903a127=>
dcrvk1f903a127=> SELECT * FROM Data where id=8012
dcrvk1f903a127-> exit
Use \q to quit.
dcrvk1f903a127-> clear
dcrvk1f903a127-> \q
(base) fabienneauffret@MBP-de-Fabienne Tweb1 % psql -h ec2-54-73-68-39.eu-west-1.compute.amazonaws.com -d dcrvk1f903a127 -U zxmgdpzvuzwlxq
Password for user zxmgdpzvuzwlxq:
SSL connection (protocol: TLSv1.3, cipher: TLS_AES_256_GCM_SHA384, bits: 256, compression: off)
Type "help" for help.

dcrvk1f903a127=> SELECT * FROM Data where id=8102;
 id | geonameid | name | asciname | alternate_names | latitude | longitude | feature_class | feature_code | country_code | c
c2 | admin1 | admin2 | admin3 | admin4 | population | elevation | dem | timezone | modification_date
-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
8102 | 2975208 | Capo Selolla | Capo Selolla | Capo Selolla,Capo la Sellola | 42.45842 | 9.00521 | T | MT | FR | 
 | 96 | 2B | 2B3 | 2B023 | 0 | 1402 | Europe/Paris | 2019-02-21
(1 row)

(END)
```

Nous avons utilisé Postman (comme en local) pour effectuer des tests (nous avons repris le fichier des « templates de test » utilisés pour les tests en local) sur les deux tables :

Note : en cas d'erreur d'authentification, sur heroku le message d'erreur est moins détaillé qu'en local, il faut lire les logs pour savoir qu'il s'agit d'un souci d'authentification :

The screenshot shows the Postman application interface. The left sidebar contains sections for Workspaces, Reports, and Explore. Under 'My Workspace', there is a 'Collections' section with a 'TP' collection expanded, showing various API endpoints like 'GET Logos', 'GET User by id', etc. The main workspace shows a 'TP / Test heroku' collection with a single 'GET' request to 'https://herappi.herokuapp.com/data/1'. The 'Params' tab is selected, showing a 'Query Params' table with one row: 'Key' (id) and 'Value' (1). The 'Body' tab is selected, showing the response body as a JSON object with a single key-value pair: 'message': 'Internal Server Error'. The status bar at the bottom indicates 'Status: 500 INTERNAL SERVER ERROR Time: 96 ms Size: 229 B'.

3. Modes d'emploi

3.1 Installation

Prérequis : une machine avec Python et Postman (<https://www.postman.com/downloads/>) installés.

En ligne de commande :

- Installation des modules python mentionnés dans le fichier requirements.txt (pip install <nom du module>)
- Installation de PostgreSQL et création d'une base de données locale

(cf. <https://www.postgresqltutorial.com/install-postgresql-linux/>)

- Lancement du serveur de base de données (Postgres, avec la commande ‘service postgresql start’)
- Crédation de la base de données
- Crédation d'un utilisateur et de son mot de passe (Usr et pwd)
- Modification du fichier de connexion : remplacer SQLALCHEMY_DATABASE_URI = f"postgresql+psycopg2:{DATABASE_URI}" par SQLALCHEMY_DATABASE_URI = postgresql+psycopg2://usr:pwd@localhost/nom_table'
- Remplir la variable SECRET_KEY du fichier config.py par une clé que vous générerez en lançant le script keygen.py
- Lancer python run.py. Vous devriez avoir l'affichage suivant : (Le remplissage de la base de données créée en local peut prendre entre 1 et 3 minutes selon la vitesse de votre machine et retarder l'affichage) :

WARNING:root:Database initialized!

* Serving Flask app "myapp" (lazy loading)

* Environment: production

WARNING: This is a development server. Do not use it in a production deployment.

Use a production WSGI server instead.

* Debug mode: on

INFO: werkzeug: * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)

INFO: werkzeug: * Restarting with fsevents reloader

WARNING:root:Database initialized!

WARNING:werkzeug: * Debugger is active!

INFO:werkzeug: * Debugger PIN: 333-455-098

```
webapplingo -- python -m run.py -- 161x36
Last login: Fri Feb 26 18:13:27 on ttys000
(base) Fabienneuffré:~/Documents/webapplingo % python run.py
INFO:werkzeug: * Serving Flask app "myapp" (lazy loading)
INFO:werkzeug: * Environment: production
WARNING:root:Database initialized!
WARNING:root:This is a development server. Do not use it in a production deployment.
Use a production WSGI server instead.
* Debug mode: on
INFO: werkzeug: * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
INFO: werkzeug: * Restarting with fsevents reloader
WARNING:root:Database initialized!
WARNING:werkzeug: * Debugger is active!
INFO:werkzeug: * Debugger PIN: 333-455-098
INFO:werkzeug:127.0.0.1 - [26/Feb/2021 21:02:34] "GET /data/9 HTTP/1.1" 200 -
INFO:werkzeug:127.0.0.1 - [26/Feb/2021 21:02:34] "POST /users/1 HTTP/1.1" 200 -
INFO:werkzeug:127.0.0.1 - [26/Feb/2021 21:02:34] "PUT /users/1 HTTP/1.1" 200 -
INFO:werkzeug:127.0.0.1 - [26/Feb/2021 21:02:43] "GET /data/admin/3 HTTP/1.1" 200 -
INFO:werkzeug:127.0.0.1 - [26/Feb/2021 21:02:43] "PUT /data/admin/3 HTTP/1.1" 200 -
INFO:werkzeug:127.0.0.1 - [26/Feb/2021 21:02:43] "GET /data/admin/9 HTTP/1.1" 200 -
INFO:werkzeug:127.0.0.1 - [26/Feb/2021 21:02:43] "PUT /data/admin/9 HTTP/1.1" 200 -
INFO:werkzeug:127.0.0.1 - [26/Feb/2021 21:02:43] "GET /data/admin/g HTTP/1.1" 200 -
INFO:werkzeug:127.0.0.1 - [26/Feb/2021 21:02:43] "PUT /data/admin/g HTTP/1.1" 200 -
INFO:werkzeug:127.0.0.1 - [26/Feb/2021 21:02:43] "GET /data/admin/p HTTP/1.1" 200 -
INFO:werkzeug:127.0.0.1 - [26/Feb/2021 21:02:43] "PUT /data/admin/p HTTP/1.1" 200 -
INFO:werkzeug:127.0.0.1 - [26/Feb/2021 21:02:43] "GET /data/admin/v HTTP/1.1" 200 -
INFO:werkzeug:127.0.0.1 - [26/Feb/2021 21:02:43] "PUT /data/admin/v HTTP/1.1" 200 -
INFO:werkzeug:127.0.0.1 - [26/Feb/2021 21:02:43] "GET /data/asciiname/y HTTP/1.1" 200 -
INFO:werkzeug:127.0.0.1 - [26/Feb/2021 21:02:43] "PUT /data/asciiname/y HTTP/1.1" 200 -
INFO:werkzeug:127.0.0.1 - [26/Feb/2021 21:02:43] "GET /data/9 HTTP/1.1" 200 -
INFO:werkzeug:127.0.0.1 - [26/Feb/2021 22:06:24] "GET /data/9 HTTP/1.1" 200 -
INFO:werkzeug:127.0.0.1 - [27/Feb/2021 12:06:39] "PUT /users HTTP/1.1" 200 -
```

Vous pouvez néanmoins utiliser le fichier *geoname.dump* pour peupler rapidement la nouvelle base de données à partir du dump des données non modifiées. Pour cela utilisez la commande ' pg_restore -d nom_de_votre_db geoname.dump' .

3.2 *Utilisation*

Nous avons décidé de sécuriser notre API en utilisant le module Flask-JWT. Le cahier des charges demandant simplement que les utilisateurs devaient se "connecter" et non "s'identifier" et la table des utilisateurs ne possédant pas de champ mot de passe, nous avons opté pour la mise en place d'un mot de passe administrateur.

Vous pouvez ouvrir Postman, importer le fichier *TP.postman_collection.json*. Ces templates sont là pour vous aider à former des requêtes avec Postman.

L'onglet de sortie « pretty » donne un affichage plus lisible.

Les utilisateurs doivent aller à l'adresse de l'api suivie de \auth et entrer le login et mot de passe nécessaires pour pouvoir faire des requêtes :

The screenshot shows the Postman application interface. On the left, the 'Team Workspace' sidebar lists various collections, APIs, environments, mock servers, monitors, and history. The main workspace displays a 'TP / Get data by field and regex' collection. A specific GET request is selected, targeting the URL `localhost:5000/data/admin4/2.*`. The 'Authorization' tab in the request settings is active, showing a dropdown set to 'Bearer Token'. Below this, a note states: 'The authorization header will be automatically generated when you send the request.' To the right of the dropdown is a 'Token' input field containing a long, encoded string: `eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJleHAiOiE2MTQ1MzI4ODUsImhdCI6MTYxNDUzMjU4NSwibmJmIjoxNjE0NTMyNTg1LCJpZGVudGljOeSi6MX0.0thPRdFzvHh4ekhiS4gwdJO-NMZduGdLKVDsji-L_w`. The 'Send' button is visible at the top right of the request panel. The bottom section of the workspace is labeled 'Response'.

The screenshot shows the Postman application interface. On the left, the sidebar includes 'Home', 'Workspaces' (selected), 'Reports', and 'Explore'. The 'Workspaces' section shows 'Team Workspace' with a collection named 'TP' containing a 'Login' endpoint. The main workspace displays a POST request to 'localhost:5000/auth'. The 'Body' tab is selected, showing a JSON payload:

```
1   {
2     "username": "admin",
3     "password": "extim2021"
4 }
```

The response status is 200 OK, 22 ms, 339 B. The response body is a JSON token:

```
1   {
2     "access_token": "eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.
eyJleHAiOjE2MTQ1MzEwODE2a7cuChfC-tak-wTIE3RI".
8mJgjmbh0mqXcdVP0010by2a7cuChfC-tak-wTIE3RI"
```

Ils obtiennent un token qu'il faudra recopier dans les autorisations (type : bearer) pour pouvoir faire les requêtes :

The screenshot shows the Postman application interface. In the top navigation bar, there are tabs for Home, Workspaces, Reports, and Explore. The main workspace is titled "Team Workspace". A search bar at the top right says "Search Postman". Below the search bar are several action buttons: [DELETE] (red), [DELETE] (red), [DELETE] (red), [POST] (blue), [GET] (green), [GET] (green), [GET] (green), [Send] (blue), and [Save] (grey). The status bar indicates "No Environment".

The left sidebar contains sections for Collections, APIs, Environments, Mock Servers, and Monitors. Under the "APIs" section, there is a "Login" collection with various requests like "Get user by id", "Get all users", etc.

The central workspace shows a "TP / Get user by id" request. The method is "GET" and the URL is "localhost:5000/users/1". The "Authorization" tab is selected, showing "Bearer Token" as the type. A note says: "The authorization header will be automatically generated when you send the request. Learn more about authorization". The "Token" field contains a long JWT token: eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJleHAiOjE2MTI...

The "Body" tab is selected, showing a JSON response:

```
1
2   "fonction": "TEEEEEEST22",
3   "id": 1,
4   "missions": [
5     "Bruxelle",
6     "Paris",
7     "Pakistan"
8   ],
9   "prenom": "Thomas",
10  "mise_a_jour": "2021-02-26",
11  "nom": "GIRERD",
12  "actif": false,
13  "actionnaire": true,
14  "ancienne": 10,
15  "conge": 15
```

The status bar at the bottom shows "200 OK 300 ms 435 B Save Response".

Sans celui-ci, on a :

The screenshot shows the Postman application interface. On the left, the sidebar displays 'Team Workspace' with various collections, APIs, environments, mock servers, monitors, and history. The main workspace shows a collection named 'TP' containing a 'Login' endpoint (POST) and several 'Get user by id' endpoints (GET). A specific 'Get user by id' request is selected, showing a 'GET' method to 'localhost:5000/users/1'. The 'Authorization' tab is active, set to 'Bearer Token' with a placeholder 'Token'. The 'Body' tab shows a JSON response with a status code of 401 UNAUTHORIZED, indicating an unauthorized access token. The bottom status bar shows '401 UNAUTHORIZED 11 ms 327 B'.

Si on est sur une application déployée, il est possible que les messages d'erreur soient plus succincts, mais les logs server side indiquent l'erreur complète : :

The screenshot shows the Postman application interface. On the left, the sidebar displays a 'Team Workspace' with various collections and environments. The main workspace shows a 'TP / Get data by id' collection. A 'GET' request is selected, pointing to the URL <https://tp1web.herokuapp.com/data/9>. The 'Authorization' tab is active, showing a 'Bearer Token' input field with a placeholder 'Token' and a long token value. Below the request details, the 'Body' tab is selected, showing the response body as a JSON object with a single key 'message' containing the value 'Internal Server Error'. The status bar at the bottom indicates a '500 INTERNAL SERVER ERROR' response with a duration of 132 ms and a size of 229 B.

Ci-dessous, vous affichez tous les utilisateurs :

The screenshot shows the Postman application interface. On the left, there's a sidebar with 'Team Workspace' containing various API endpoints like 'Get user by id', 'Get all users', etc. The main area shows a collection named 'TP' with a single item: 'TP / Get all users'. This item is a GET request to 'localhost:5000/users/all'. The 'Params' tab shows 'Query Params' with a 'Key' column ('Key') and a 'Value' column ('Value'). The 'Body' tab is selected, showing a JSON response with multiple user objects. Each user object has properties like 'conge', 'actionnaire', 'nom', 'fonction', 'actif', 'ancienne', 'id', 'prenom', 'mise_a_jour', and 'missions'. The 'Responses' tab indicates a successful 200 OK status with a response size of 2.17 KB.

```
1 [ 2   { 3     "conge": 14, 4     "actionnaire": false, 5     "nom": "Berger", 6     "fonction": "Chef service matériel informatique", 7     "actif": true, 8     "ancienne": 20, 9     "id": 1, 10    "prenom": "Francine", 11    "mise_a_jour": "2021-02-26", 12    "missions": [ 13      "Paris" 14    ] 15  }, 16  { 17    "conge": 5, 18    "actionnaire": false, 19    "nom": "Takem", 20    "fonction": "Cuisinier en chef", 21    "actif": true, 22    "ancienne": 3, 23    "id": 3, 24    "prenom": "Georges", 25    "mise_a_jour": "2021-02-26", 26    "missions": [ 27      "Londre", 28      "Tunis" 29    ] 30  }, 31  { 32    "conge": 10, 33    "actionnaire": false, 34    "nom": "Gate", 35    "fonction": "Chef service juridique", 36    "actif": true, 37    "ancienne": 9, 38    "id": 9, 39    "prenom": "Roland", 40    "mise_a_jour": "2021-02-26", 41    "missions": [ 42      "Berlin", 43      "..."] 44  }]
```

Vous pouvez ajouter un utilisateur (en utilisant la requête « put ») :

Cliquez sur l'onglet intitulé Body. Sélectionnez ensuite le bouton brut et choisissez le format JSON.

Entrez les champs nécessaires avec les infos que vous voulez ajouter dans la partie « body » et appuyez sur « Send ».

The screenshot shows the Postman application interface. On the left, the sidebar displays a 'Team Workspace' with various collections and environments. The main area shows a 'PUT / Put user' request to 'localhost:5000/users'. The 'Body' tab is selected, showing a JSON payload:

```
1 {
2     "nom": "Samuel",
3     "prenom": "Jackson",
4     "fonction": "Chef service matériel informatique",
5     "anciennete": 20,
6     "conge": 14,
7     "actif": true,
8     "actionnaire": false,
9     "missions": [
10         "Paris"
11     ]
12 }
```

Below the body, the response status is shown as 200 OK with a time of 402 ms and a size of 168 B. The response body is also displayed in JSON format:

```
1 {
2     "status": "ok"
3 }
```

Exemple avec la table data :

The screenshot shows the Postman application interface. On the left, the sidebar displays 'Team Workspace' with various collections, APIs, environments, mock servers, monitors, and history. The main workspace shows a 'TP / Put data' collection under 'PUT' method. The URL is set to 'localhost:5000/data'. The 'Body' tab is selected, showing a JSON payload:

```
1 {"geonameid": 12213976,
2 "name": "TEST",
3 "asciiname": "Villarenger",
4 "alternatename": "Villarenger",
5 "latitude": 45.38833,
6 "longitude": 6.4434,
7 "feature_class": "P",
8 "feature_code": "PPL",
9 "country_code": "FR",
10 "cc2": "FR",
11 "admin1": 84,
12 "admin2": 73,
13 "admin3": 731,
14 "admin4": 73257,
15 "population": 0,
16 "elevation": 0,
17 "dem": 1399,
18 "timezone": "Europe/Paris"
20 }
```

The response section shows a status of 200 OK with 737 ms and 168 B. The response body is:

```
1 {"status": "ok"
2
3 }
```

Avec « Post » vous pouvez éditer des données, il faut également cliquer sur l'onglet body (dès que l'on souhaite modifier ou ajouter des données) et y mettre les champs et informations que l'on veut changer :

The screenshot shows the Postman application window. On the left, the sidebar displays a 'Team Workspace' with various collections, APIs, environments, mock servers, monitors, and history. In the main area, a 'TP / Edit data' collection is selected. A POST request is being edited for the endpoint 'localhost:5000/data/1'. The 'Body' tab is active, showing a JSON payload:

```
1 {  
2     "geonameid": 122113976,  
3     "name": "Villazenger",  
4     "asciiname": "Villazenger",  
5     "alternatenames": "Villazenger",  
6     "latitude": 45.38833,  
7     "longitude": 6.4434,  
8     "feature_class": "P",  
9     "feature_code": "PPL",  
10    "country_code": "FR",  
11    "cc2": "FR",  
12    "admin1": 84,  
13    "admin2": 73,  
14    "admin3": 731,  
15    "admin4": 73257,  
16    "population": 0,  
17    "elevation": 0,  
18    "dem": 1399,  
19    "timezone": "Europe/Paris"  
20}
```

Below the body, the response section shows a successful 200 OK status with a response time of 218 ms and a response size of 168 B. The response body is also JSON:

```
1 {  
2     "status": "ok"  
3 }
```

Pour effacer un lieu ou un user, il faut choisir une requête « del » :

The screenshot shows the Postman application interface. On the left, the sidebar includes sections for Home, Workspaces, Reports, Explore, Collections, APIs, Environments, Mock Servers, Monitors, and History. The main workspace displays a collection named 'TP' containing various API endpoints. A specific endpoint, 'Delete user', is selected. The request details pane shows a 'DELETE' method directed at 'localhost:5000/data/14857'. The 'Params' tab is active, showing a single parameter 'Key' with value 'Value'. Below the request details, the response pane shows a JSON object with three lines of data: 1, 2, and 3. The status bar at the bottom indicates a successful response: 200 OK, 143 ms, 168 B.

Postman vous indique les erreurs (ici on tente à nouveau d'effacer le lieu/il n'existe plus) :

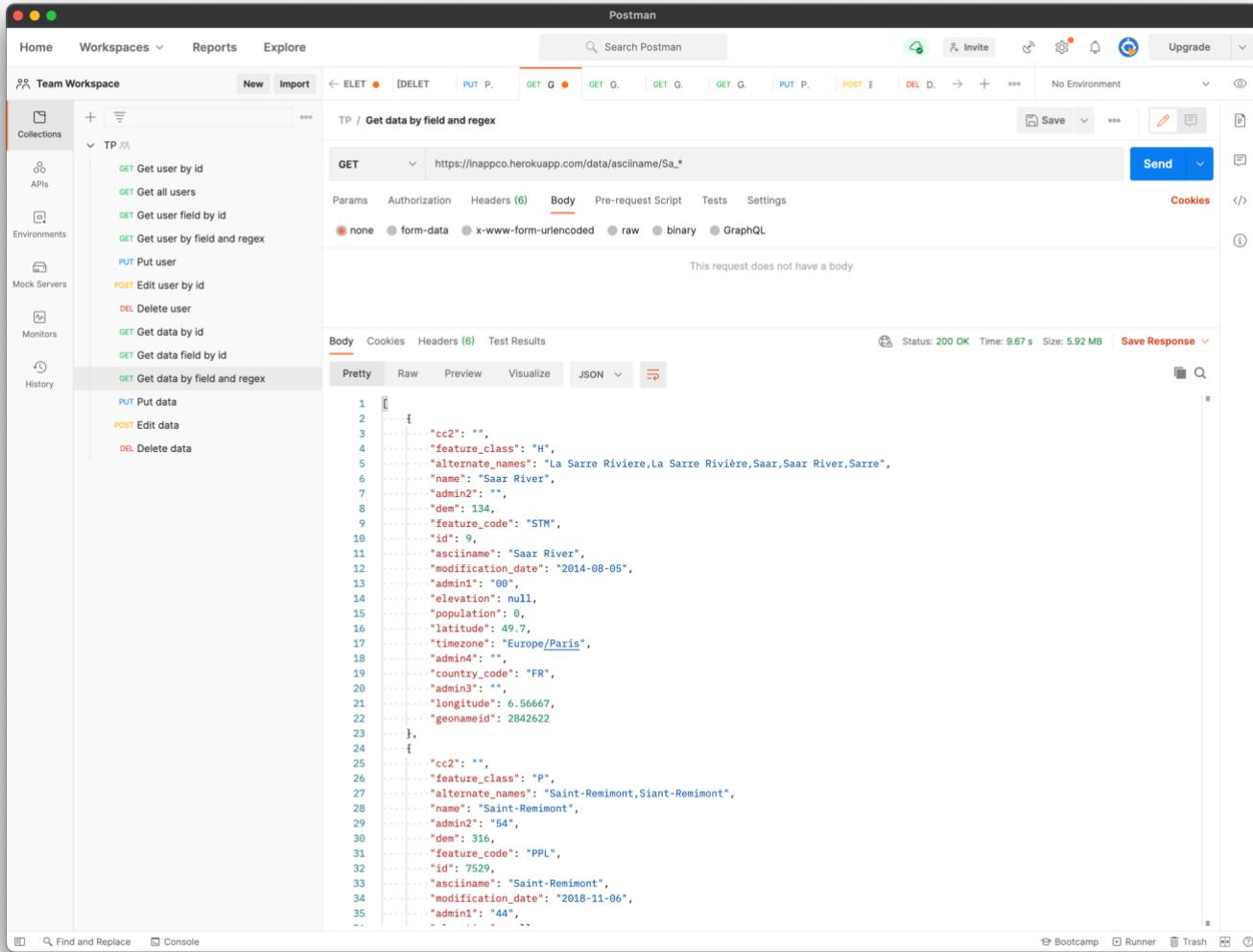
The screenshot shows the Postman application interface. On the left, the sidebar displays 'Team Workspace' with various collections, APIs, environments, mock servers, monitors, and history. The main workspace shows a 'TP / Delete data' collection under 'TP'. A DELETE request is being prepared to 'localhost:5000/data/14857'. The 'Params' tab is selected, showing a single parameter 'Key' with value 'Value'. The 'Body' tab shows a JSON response with three lines of code:

```
1 {"error": "Location not found"}  
2  
3
```

The status bar at the bottom indicates a 404 NOT FOUND response with 409 ms and 190 B.

On peut aussi filtrer les résultats d'une requête avec une regex, il suffit de mettre dans l'URL le champ qu'on veut filtrer et terminer la requête par une regex (champ/regex). Attention, dans la syntaxe postgresql, un caractère est désigné par « _ » (underscore) et non par un point comme souvent.

Deux exemples (toutes les requêtes allant chercher des informations dans la base sont des requêtes « get ») :



The screenshot shows the Postman application interface. On the left, the sidebar lists collections, APIs, environments, mock servers, monitors, and history. A collection named "TP / Get data by field and regex" is selected. Inside the collection, there are several requests:

- GET Get user by id
- GET Get all users
- GET Get user field by id
- GET Get user by field and regex
- PUT Put user
- POST Edit user by id
- DELETE Delete user
- GET Get data by id
- GET Get data field by id
- GET Get data by field and regex

The "GET Get data by field and regex" request is highlighted. Its details are shown in the main panel:

- Method: GET
- URL: https://inappco.herokuapp.com/data/asciiname/Sa_.*
- Params: None
- Headers: (6)
- Body: (none)
- Pre-request Script: None
- Tests: None
- Settings: None

The response status is 200 OK, time 9.67 s, size 5.92 MB. The response body is displayed in JSON format:

```
1 [ { 2   "cc2": "", 3   "feature_class": "H", 4   "alternate_names": "La Sarre Riviere,La Sarre Rivière,Saar,Saar River,Sarre", 5   "name": "Saar River", 6   "admin2": "", 7   "dem": 134, 8   "feature_code": "STM", 9   "id": 9, 10  "asciiname": "Saar River", 11  "modification_date": "2014-08-05", 12  "admin1": "00", 13  "elevation": null, 14  "population": 0, 15  "latitude": 49.7, 16  "timezone": "Europe/Paris", 17  "admin3": "", 18  "country_code": "FR", 19  "admin4": "", 20  "longitude": 6.56667, 21  "geonameid": 2842622, 22  }, 23  ], 24  { 25   "cc2": "", 26   "feature_class": "P", 27   "alternate_names": "Saint-Remimont,Saint-Remimont", 28   "name": "Saint-Remimont", 29   "admin2": "54", 30   "dem": 316, 31   "feature_code": "PPL", 32   "id": 7529, 33   "asciiname": "Saint-Remimont", 34   "modification_date": "2018-11-06", 35   "admin1": "44", 36 } ]
```

The screenshot shows the Postman application interface. On the left, there's a sidebar with sections for Team Workspace, Collections, APIs, Environments, Mock Servers, Monitors, and History. The main area displays a collection named "TP / Get data by field and regex". Under this collection, there are several requests listed: "Get user by id", "Get all users", "Get user field by id", "Get user by field and regex", "PUT Put user", "POST Edit user by id", "DEL Delete user", "GET Get data by id", "GET Get data field by id", and "GET Get data by field and regex". The "Get data by field and regex" request is selected and expanded. It shows a GET method pointing to the URL https://lappco.herokuapp.com/data/admin4/8.*. The "Body" tab is selected, showing the response body as a JSON array:

```

1  [
2   {
3     "cc2": "",
4     "feature_class": "P",
5     "alternate_names": "Pairac,Pairac d'Amostier,Pejra-le-Shato,Peyrat,Peyrat-le-Chateau,Peyrat-le-Chateau,pei la lei sha tuo,
6       Neipa-ne-Baro,佩拉勒沙托",
7     "name": "Peyrat-le-Chateau",
8     "admin": "87",
9     "dem": 453,
10    "feature_code": "PPL",
11    "id": 22,
12    "asciiname": "Peyrat-le-Chateau",
13    "modification_date": "2020-06-10",
14    "admin2": "75",
15    "elevation": null,
16    "population": 1148,
17    "latitude": 45.81376,
18    "timezone": "Europe/Paris",
19    "admin3": "87117",
20    "country_code": "FR",
21    "admin4": "872",
22    "longitude": 1.7726,
23    "geonameid": 2967103
24  },
25  {
26    "cc2": "",
27    "feature_class": "P",
28    "alternate_names": "Domency,Domancy-sur-le-Vault",
29    "name": "Domancy-sur-le-Vault",
30    "admin": "89",
31    "dem": 203,
32    "feature_code": "PPL",
33    "id": 26,
34    "asciiname": "Domancy-sur-le-Vault",
35    "modification_date": "2016-02-18",
36    "-----"
37  }
]

```

The "Test Results" tab shows a status of 200 OK, time 10.02 s, and size 5.13 MB. The "Body" tab has sub-options for Pretty, Raw, Preview, Visualize, and JSON.

4. Conclusion & perspectives

Voilà un back-end qui n'attend plus qu'un frontend avec authentification des utilisateurs autorisés à se connecter à la table « data » via des formulaires html servis par nginx (qui n'est donc pas encore utile au niveau du back-end).