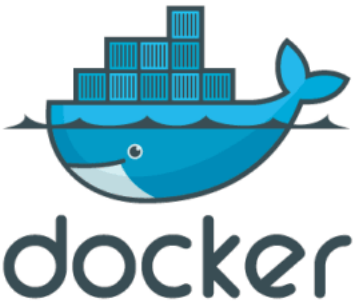
IA Data Entrepreneurship Program

B-IADATA-xxx

Back I

First Steps in Back-End Development



1.0

Flask est un micro framework pour le web, developpé en python.
Dans ce premier atelier sur le développement back-end, nous allons mettre en place un mini serveur Flask capable de réagir à des requêtes HTTP et de communiquer avec une base données pour éditer ou explorer des données.

Environnement technique

- Editeur de code : VsCode / atom / sublimetext / etc.
- Langage de programation : Python 3.x
- Libs Python : Flask, Flask-SQLAlchemy, Flask-Migrate, psycopg2, python-dotenv
- Gestion de BDDs : PostgreSQL, Adminer (ou PHPMyAdmin)
- Contenarization : Docker, Docker-Compose
- CURL

I. Prélude

Python doit être installé sur la machine pour cet exercice

Rien de plus simple pour mettre en place un mini-serveur back-end en Flask. Dans ce prélude nous allons :

1. Installer flask et coder un mini-serveur
2. Exécuter le mini-serveur en local
3. Accéder au mini serveur depuis un navigateur
4. Accéder un mini-serveur depuis une invite de commande

I.1. INSTALLATION DE FLASK

Dans une invite de commandes, utilisez la commande `pip` pour installer la lib Flask

```
Terminal
B-IADATA-xxx> mkdir 00-prelude
B-IADATA-xxx> cd 00-prelude
B-IADATA-xxx/00-prelude> pip install flask
```

Créez un fichier python `app.py` avec le code suivant :

```python
# app.py
from flask import Flask
app = Flask(__name__)

@app.route('/')
def hello_world():
    return 'Hello World!'

if __name__ == '__main__':
  app.run()
```

I.2 EXECUTION DU MINI-SERVEUR FLASK

Dans une invite de commandes, exécuter votre mini-serveur flask :

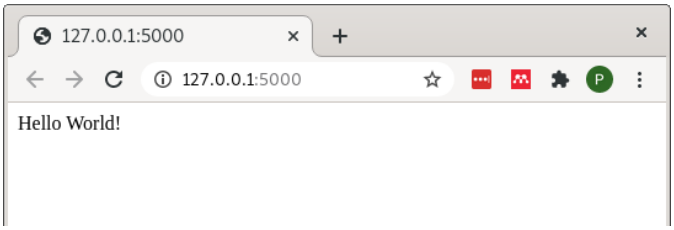
```
Terminal
B-IADATA-xxx/00-prelude> python app.py


* Serving Flask app "app" (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: off
* Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
```

Félicitations, vous venez de créer un back-end fonctionnel !

Laissez votre serveur *allumé*

I.3 Accès depuis une page web

Ouvrez un navigateur, puis, saisissez l'url : `http://127.0.0.1:5000`. Vous devriez obtenir le résultat suivant :



Le navigateur affiche le résultat de la seule et unique requête prise en charge par votre back-end (définie dans le fichier `app.py`).

Vous pouvez envoyer cette requête depuis un client encore plus basique.

Amusez-vous et modifiez le fichier `app.py` pour afficher plus de contenu `HTML`

I.4 Acces depuis une invite de commande

Téléchargez l'utilitaire `CURL` compatible avec votre `OS` (windows, linux ou mac). `CURL` vous permet d'envoyer des requêtes HTTP à un serveur web quelconque depuis une invite de commande.

Maintenant, dans un autre terminal exécutez la commande suivante :

```
Terminal
B-IADATA-xxx/00-prelude> curl 127.0.0.1:5000/
```

Vous devrez obtenir le résultat de la fonction `hello_world()` de votre fichier `app.py` dans votre terminal.

Vous pouvez rendre votre back-end encore plus *accessible* :

1. depuis n'importe quelle ordinateur de votre réseau local en remplaçant **app.run()** par `app.run(host=0.0.0.0)` dans le fichier `app.py`. Essayez avec votre téléphone portable.

2. de manière sécurisée, de n'importe où dans le monde et gratuitement grâce à ngrok. Essayez avec votre équipe…

EXERCICE 1

Pour le premier vrai exercice, nous allons packager notre back-end flask. Nous allons :

1. Reformatez le contenu du dossier `00-prelude` dans le dossier `01-helloworld_back-end` en respectant l'arborescence traditionnelle d'un projet flask
2. Ajouter une mini authentification
3. Utiliser la fonction `render_template` pour gérer les vues.

Tout d'abord, rendez-vous à cette page pour un petit point documentation (jusqu'à la section **Message Flashing**... au moins).

EXERCICE 1.1. REFORMATTAGE PACKAGE-FRIENDLY

Nous allons transformer le code de la section précédente pour qu'il respecte l'arborescence suivante :

Terminal

```
B-IADATA-xxx> cd 01-helloworld_back-end
B-IADATA-xxx/01-helloworld_back-end> tree -distfirst -charset=ANSI

.
|-- application
|   |-- static
|   |   `-- style.css
|   |-- templates
|   |   `-- hello.html
|   |-- __init__.py
|   `-- views.py
|-- config.py
|-- requirements.txt
`-- run.py
```

Dans cette organisation :
- le fichier `run.py` devient le fichier qui permet d'exécuter votre back-end
- la définition des routes est reportée dans le fichier `views.py`
- le style de la page est défini dans le fichier `style.css`
- le contenu de la page est défini dans un des fichiers `html` du dossier `templates`
- les paramètres du serveur sont définis dans le fichier `config.py`
- les dépendances (python) sont définies dans le fichier `requirements.txt`

Définissez les fichiers `run.py`, `__init__.py` et `views.py` pour reproduire le serveur dans le même état que l'exercice précédent – tous les tests précédents (navigateur er `curl`) doivent fonctionner.

Terminal

```
B-IADATA-xxx> cd 01-helloworld_back-end
B-IADATA-xxx/01-helloworld_back-end> python run.py
```

Exercice 1.2. Mini Authentification

En vous inspirant du code de la section Sessions, implémentez une mini-authentification et vérifier qu'elle fonctionne correctement.

Exercice 1.3. Utilisation de la fonction *render_template*

Déplacez le contenu de votre vue dans le fichier `hello.html` et utilisez la fonction render_template dans le fichier `views.py` pour afficher votre page.

Jouez avec le *Design* et créer des *API*s simples :

- Utilisez une feuille de style CSS (`style.css`) définie par le *designer* pour afficher votre vue (plus d'infos sur l'intégration d'une feuille css ici)

- Ajouter une route dans le fichier `views.py` pour renvoyer des données (quelconque) au format JSON une fois la connexion validée (i.e. Login et un mot de passe reconnus)

Exercice 2

Dans ce deuxième exercice, nous faisons un pas de plus vers l'isolation de notre mini serveur flask en lui donnant tous les éléments nécessaires pour le rendre autonome.

Nous allons le faire de deux manières différentes:

1. En installant un environnement python dédié pour notre back-end et en exécutant notre mini serveur dans cet environnement.
2. En utilisant docker pour transformer notre back-end en un container virtuel (autonome)

On vérifiera dans les deux cas que les requêtes fonctionnent toujours.
Le dossier final sera organiser comme suit :

Exercice 2.1. Utilisation d'un environnement virtuel

Recopiez entièrement le dossier `01-helloworld_back-end` dans `02-containarizing_your_back-end`, puis, désinstaller complètement flask.

Après la désinstallation, l'exécution du back-end ne devrait plus être possible.

Installez ensuite les libraries virtualenv et virtualenvwrapper pour pouvoir créer environement virtuel dédié au mini-serveur.

Terminal

```
B-IADATA-xxx> cp 01-helloworld_back-end 02-containarizing_your_back-end -r
B-IADATA-xxx> cd 02-containarizing_your_back-end
B-IADATA-xxx/02-containarizing_your_back-end> pip uninstall flask
B-IADATA-xxx/02-containarizing_your_back-end> pip install -user virtualenv
virtualenvwrapper
```

Créez un environnement virtuel nommé `back_1`, et installez-y toutes les dépendances nécessaires depuis le fichier `requirements.txt`.

Exécutez ensuite le serveur depuis l'environnement virtuel et vérifiez que tout fonctionne normalement.

Terminal

```
/02-containarizing_your_back-end> mkvirtualenv back_1 -python=python3
...
(back_1) /02-containarizing_your_back-end> pip install -r requirements.txt
(back_1) /02-containarizing_your_back-end> pip run.py
```

EXERCICE 2.2. DOCKERISATION

En utilisant l'outil docker, on se rapproche encore plus de la mise en production.
Le serveur sera transformé en un container que l'on pourra déployer presque immédiatement en ligne.

Rendez-vous sur la documentation officielle de docker pour installer docker. Revenez après avoir lu la doc.

Pour créer le container, il faut se souvenir de toutes les opérations nécessaires pour l'exécution du mini-serveur, depuis l'installation de python jusqu'à l'exécution de la commande qui lance le mini-serveur.

Toutes ces instructions sont résumées dans un fichier spécial appelé `Dockerfile` et il suffira de l'exécuter via la commande `docker run`.

En vous inspirant de cet exemple, créez un `Dockerfile` à la racine du projet pour exécuter le mini-server et exécutez-le avec une commande `docker run`. Vérifiez que toutes les requêtes fonctionnent bien.

A la fin de l'exercice, l'arborescence pourra être la suivante :

Terminal

```
B-IADATA-xxx> cd 02-containarizing_your_back-end
02-containarizing_your_back-end/> tree -F -charset=ANSI -dirsfirst

.
|-- application/
|   |-- static/
|   |   `-- style.css
|   |-- templates/
|   |   `-- hello.html
|   |-- __init__.py
|   `-- views.py
|-- config.py
|-- Dockerfile
|-- requirements.txt
`-- run.py
```

EXERCICE 3

Vous avez les bases pour créer des containers et les exécuter via une commande `docker run`.

Nous allons rajouter un niveau d'organisation supplémentaire dans cet exercice, en agrégeant (ou en *composant*) les dockers entre eux, pour créer un back-end plus riche.

Rendez-vous sur la documentation officielle de docker-compose pour comprendre l'utilisation de la commande.

En vous inspirant de cet exemple :

- créer un ficher `docker-compose.yml` qui vous permet d'instancier une base de données `postgresql` et une interface web `adminer` permettant de la gérer.
- exécuter la commande `docker-compose up` pour exposer l'interface de gestion de la base données via votre navigateur (`http://localhost:8080` sur *chrome* par exemple)

```
Terminal
B-IADATA-xxx> cd 03-connect_to_db
B-IADATA-xxx/03-connect_to_db> docker-compose up
```

Recopiez le dossier de l'exercice précédent dans `03-connect_to_db/web`, puis modifiez le fichier `docker-compose.yaml` pour rajouter votre back-end sur le port `5000`.

La commande `docker-compose up` doit maintenant vous donner accès à la fois à l'interface de gestion de votre BDD et à votre back-end.

```
Terminal
B-IADATA-xxx/03-connect_to_db> cp ../../02-containarizing_your_back-end/* web/ -r
B-IADATA-xxx/03-connect_to_db> docker-compose up
```

Vérifiez que tout fonctionne normalement.

EXERCICE 4

Dans cet exercice, nous allons connecter notre back-end avec une base de données `postgresql` pour effectuer des opérations CRUD:

- Create
- Read
- Update
- Delete

sur des objets de type '*Produit*' avec les attributs suivants :

- Nom (Exemple : "chemise") : str
- Origine (Exemple : "chine") : str
- Prix (Exemple : 12) : real

Pour cela, nous allons rajouter un fichier `models.py` et un fichier `manage.py` dans le dossier de travail :

- `models.py` va contenir la définition du (des) modèle(s) de données (i.e. l'objet *Produit*)
- `manage.py` sera utilisé pour manipuler la base de données.

Il faudra se familiariser avec les extensions Flask-Migrate et Flask-SQLAlchemy pour réussir cet exercice. L'arborescence finale pourra être la suivante :

Terminal

```
B-IADATA-xxx> cd 04-app_bdd_crud
B-IADATA-xxx/03-connect_to_db> tree -dirsfirst -F -charset=ANSI

.
|-- db/
|   |-- data/
|   `-- Dockerfile
|-- web/
|   |-- application/
|   |   |-- static/
|   |   |   `-- style.css
|   |   |-- templates/
|   |   |   `-- hello.html
|   |   |-- __init__.py
|   |   |-- models.py
|   |   `-- views.py
|   |-- config.py
|   |-- Dockerfile
|   |-- manage.py
|   |-- requirements.txt
|   `-- run.py
`-- docker-compose.yml
```

`models.py` ET `manage.py`

En vous inspirant de la documentation de Flask-SQLAlchemy et de Flask-Migrate, ajoutez les fichiers `models.py` et `manage.py` de sorte que l'objet *Produit* soit instancié dans la base de données au lancement du mini-serveur (toujours via la commande `docker-compose up`)

Vous pourrez vérifier grâce à l'interface de gestion de base de données de votre choix.

CRUD

Rajoutez 4 nouvelles requêtes au niveau du back-end dans le fichier `views.py`.

- /create : crée un nouveau produit
- /read : affiche tous les produits crées
- /update : met à jour un produit
- /delete : supprime un produit de la base de données

et modifiez le fichier `hello.html` pour pouvoir les exécuter depuis une page web.

EXERCICE 5

La boucle est bouclée! Ecrivez maintenant un workflow à vous qui part du front-end (développé en équipe) vers le back-end, et qui manipule des données de votre choix.

Seule condition : vous devez en être fiers.