TP1 – Développement Backend Avec Flask

Vous devez implémenter une api basique pour une application permettant de gérer des Todo List.

Cette API sera ultérieurement connectée à un Frontend Angular.

Ce TP ne faisant appel qu'à des fonctions de base, vous n'avez pas le droit d'utiliser **ChatGPT.**

Vous pouvez en revanche librement chercher sur StackOverflow, la documentation Python ou bien celle de Flask et dans vos notes de cours.

N'oubliez pas de tester votre API (extensions Chrome Talend API Tester)

Une fois terminé, déposez votre devoir sur Moodle.

Avant de commencer – A faire ensemble

Créez un fichier python pour représenter la base de données de votre application.

Le contenu de ce fichier est le suivant :

```
'''
Une tâche contient les attributs suivants :
task_id : uuid
name : string
description : string
status : string
created_at : str
updated_at : str

task_db est un index qui se présente sous la forme suivante :
{
   "uuid1" : {
   "task_id" : "uuid1",
   "name": "Nom",
   "description": "Desc",
   "status": "TODO",
   "created_at" : "date de création",
   "updated_at" : "date de mise à jour"
   }
}
'''

task_db = dict()
```

Fonctionnalités à implémenter

A – Avoir une application qui démarre sans crash

B – Faire en sorte que http://localhost:4200 puisse appeler votre backend, avec les méthodes GET / POST / PUT / DELETE et OPTIONS

C – Gestion des tâches :

- **Consulter toutes les tâches**
  - Un utilisateur peut appeler une route qui lui renverra toutes les tâches existantes dans l'application sous forme de liste:
    - **[{task_id : « id », ….}, {task_id : « id2 »,…}]**
  - De manière optionnelle, un utilisateur peut récupérer les tâches par statut, par exemple il peut demander toutes les tâches dont le statut est « done »

- **Consulter une tâche**
  - Grâce à l'id de la tâche, un utilisateur peut récupérer toutes les informations pour une seule tâche sous forme d'objet di
  - Si la tâche n'existe pas, alors il faut renvoyer un message d'erreur avec le code associé

- **Créer une tâche**
  - L'utilisateur peut créer une tâche, le payload doit répondre aux critères suivants :
    - La tâche a un titre
    - La tâche a une description
    - La tâche a un statut. Le statut n'a que **3 valeurs métier** possibles : « TODO », « IN_PROGRESS », « DONE ».
  - Si un élément cité précédemment manque, alors il faut renvoyer un message d'erreur indiquant que la requête n'est pas correcte avec le code associé
  - Si des champs additionnels sont présents, la requête doit échouer
  - Avant d'être enregistrée dans l'index des tâches, un champ nommé **task_id** est généré, ainsi qu'un champ **created_at**
- **Mettre une tâche à jour par remplacement total**
  - L'utilisateur peut demander la mise à jour complète d'une tâche
  - Les valeurs suivantes ne doivent pas être modifiées (et ne font donc pas partie du payload) : task_id, created_at, updated_at
  - Lors de la modification d'une tâche, un champ **updated_at** est soit créé, soit mis à jour

    - Si un champ requis dans dans une tâche n'est pas présent dans cette requête, elle devra échouer
    - Si des champs additionnels sont présents, la requête doit échouer
    - Si la tâche n'existe pas, elle devra échouer également

- **Mettre une tâche à jour par remplacement partiel**
    - L'utilisateur peut demander la mise à jour partielle d'une tâche
    - Les valeurs suivantes ne peuvent pas être modifiées et ne font jamais partie du payload : task_id, created_at, updated_at
    - Un champ requis ne peut pas devenir vide après modification
    - La tâche est modifiée
    - Si la tâche n'existe pas, la requête doit échouer

- **Supprimer une tâche**
    - L'utilisateur peut supprimer une tâche. Si la tâche existe, il faut la supprimer de l'index des tâches.
    - Renvoyer à l'utilisateur un message vide et le code de réussite correspondant

D – Accès Admin

- Un utilisateur peut envoyer une demande à la route admin
- Comme il n'est pas connecté, il reçoit toujours une erreur indiquant qu'il n'a pas l'autorisation de demander cette ressource. Cette erreur renvoie une page d'erreur et non pas un message formaté

Également évalué :

L'absence de ces caractéristiques pourra vous coûter des points :

A – Mettre à disposition du développeur les informations sur le projet et un fichier permettant d'installer les dépendances requirements.txt comprenant les deps à installer

B – Respecter les principes d'architecture vus jusqu'à présent

C – Respecter des principes de codage et de sécurité du code (noms de fonctions et variables lisibles et informatifs, utilisation de fonctions built-in de python si possible, versionnage des fichiers)

D – Construction des endpoints et des réponses en respectant le principe REST de ressource uniforme et des codes http

Quelques indications :

- Pour générer les id dans l'application, vous utiliserez **uuid4** qui vient du module **uuid**
- Pour générer les dates de création et de modification, vous utiliserez **datetime.now()** qui vient du module **datetime**
- **A part pour la route /admin,** les erreurs ne renvoient pas vers une page par défaut, mais bien un objet json.
- Comme nous n'avons pas vu la validation d'objet par le métier, vous pouvez valider les règles métier dans les DTO