RER – API REST Python

Contexte :

Se familiariser avec les API RESTful Python, définir le lien entre les API et les bases de données et développer une application web permettant de rendre de l'information et des fonctionnalités accessibles via internet avec python (Flask, RESTful).

Problématique(s) :

Comment mettre en place une application RESTFul avec Flask ?

Comment utiliser les méthodes **HTTP (GET, POST, DELETE, PUT) ?**

Comment fonctionne POSTMAN ?

Comment créer de la documentation avec Swagger ?

Mots clés :

API (Application Programming Interface) : C'est une interface qui sert à communiquer avec une application ou entre applications. Elle propose un ensemble de fonctions qui facilitent, via un langage de programmation permettant de lancer des requêtes, l'accès aux services d'une application distante.

C'est une interface logicielle qui permet de « connecter » un logiciel ou un service à un autre logiciel ou service afin d'échanger des données et des fonctionnalités.

REST (Representational State Transfer) : C'est une architecture logicielle definit par l'ensemble de normes et de contraintes pour la création d'un service web afin de faciliter la communication entre applications.

Les 6 contraintes sont :

- La séparation entre client et serveur
- L'absence d'état de sessions (stateless)
- Uniformité de l'interface
- Mise en cache
- L'architecture en couches
- Code à la demande

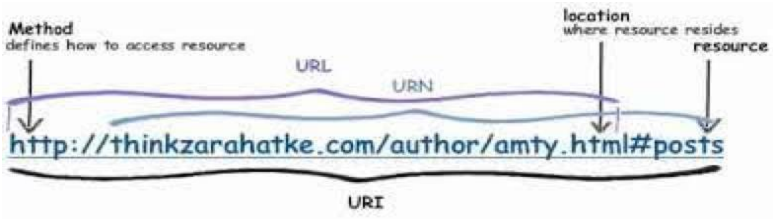
RESTful : est une architecture qui respecte toutes les contraintes de l'architecture REST. Cette architecture s'applique aux services web. Les méthodes utilisées dans Restful sont : GET, POST, PUT, PATCH, DELETE.

FastAPI : C'est un framework web, open source, permettant de créer des API avec python, basé sur des standards comme OpenAPI (SWAGGER) et JSON SCHEMA. Il est rapide, facile à apprendre, robuste, intuitif, moins de bugs, prêt à la production avec documentation interactive automatique.

SOAP (Simple Object Acces Protocol) : C'est un protocole de communication basé sur XML, utilisé pour l'échange de données structurées entre applications.

URI (Uniform Resource Identifier) : C'est une chaine de caractères qui fait référence à une ressource. C'est aussi un chemin. Un URI peut etre un URL ou un URN (Uniform Resource Name).

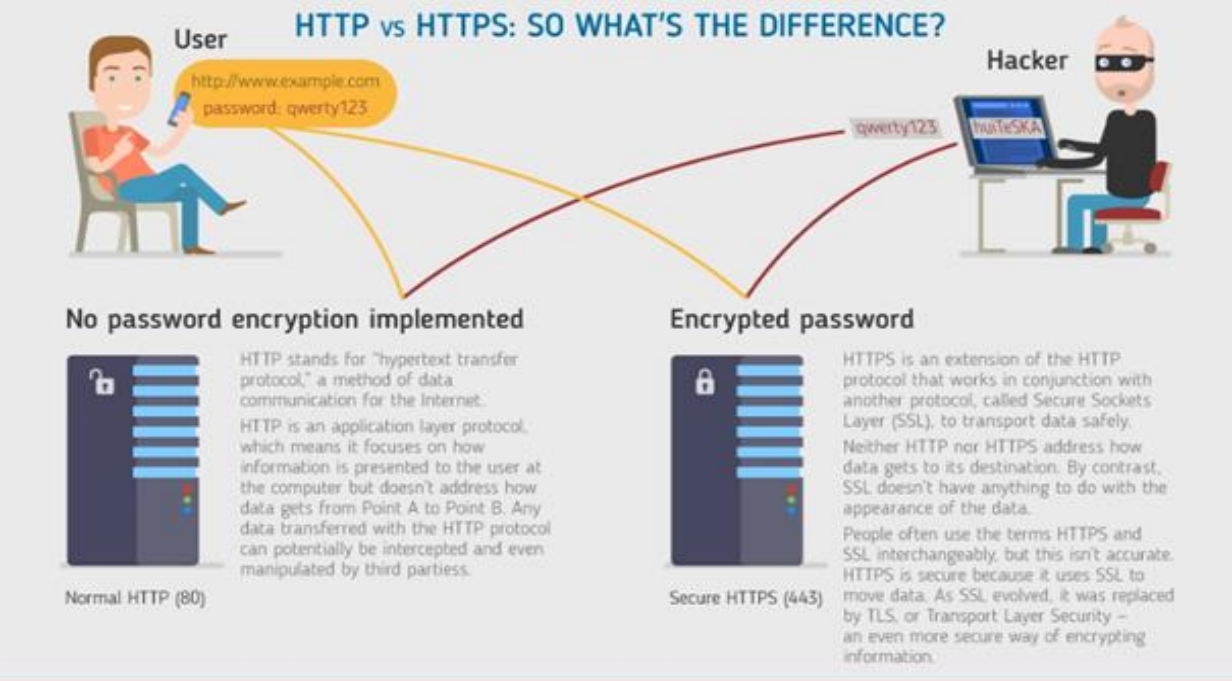
URL (Uniform Resource Locator) : est une information (chaine de caractères) permettant de localiser un élément.



Routage : C'est un processus (ou convention) consistant à appliquer des URLs sur des fonctions (@app.route()). C'est une convention pour déclarer les chemins via un ensemble d'attributs : le nom de la route, URL, la méthode permettant l'invocation ou la réponse de cette route, l'action réalisée via la route.

Endpoint : C'est une extrémité d'un canal de communication. Lorsqu'une API interagit avec un autre système, les points de contacts de cette interaction sont considérés comme des endpoints. C'est le nom de domaine + le chemin de la ressource.

HTTP (Hypertext Transfert Protocol) : Est un protocole de communication client-serveur développé pour le world_wide_web. Elle demande au serveur d'exécuter une commande.



HTTPS (Hypertext Transfert Protocol Secure) : C'est pour augmenter la confidentialité avec une couche d'entité (cryptage asymétrique).

Méthode : Désigne une requête pour un serveur HTTP.

- **GET :** Méthode/fonction qui écrit les données initiées par un client (données du formulaire) et sont envoyées au serveur directement

encoder dans une URL. Il se compose du nom de la page ou du script à charger et on sépare le nom de la page par un “?” suivit par les données du formulaire qui sont séparés par des “&” (chaine dans une chaine de caractère). Méthode non-sécurisée lorsqu’on utilise des données sensibles.

- **POST :** Méthode/Fonction n’utilise pas un URL mais les données sont incluses directement dans la requête HTTP.

- **PUT :** Méthode/Fonction est le plus souvent utilisée pour mettre à jour une ressource existante. Si vous souhaitez mettre à jour une ressource spécifique (qui est fournie avec un URI spécifique), vous pouvez appeler la méthode PUT vers cet URI de ressource avec le corps de la requête contenant la nouvelle version complète de la ressource que vous essayez de mettre à jour.

Postman : C’est une plateforme utilisée pour le développement et l’utilisation d’API. Il facilite et simplifie chaque étape du cycle de vie des API et rationalise la collaboration afin que vous puissiez créer de meilleures API plus rapidement. Cette plateforme permet :

1. Envoyer des requetes REST et SOAP
2. Simuler des endpoints
3. Generer et publier la documentation d’une API
4. Travailler de manière collaborative avec plusieurs espaces de travail
5. Automatiser des tests d’API
6. Surveiller la performance d’une API
7. Permet d’utiliser les verbes suivants (méthodes) : GET, PUT POST, PATCH, DELETE, COPY, HEAD, OPTIONS, LINK, UNLINK, PURGE, LOCK, UNLOCK.

SWAGGER : C’est un ensemble d’outils pour la conception de la documentation et la consommation d’API RESTful. Avec Swagger, on peut :

1. Concevoir une API dans SWAGGER Editor
2. Construire et activer les servers de l’API, SWAGGER CodeGEN

3. Documenter l'API (génération automatique à partir du code).

MARSHMALLOW : Est une bibliothèque Python pour convertir tous les types de données complexes tels que les objets vers et depuis des types de données Python natif. On peut sérialiser ou désérialiser et valider les données d'entrée. Les objets sérialisés peuvent ensuite être rendus dans des formats standards tels que JSON, XML, etc. pour une utilisation dans une API HTTP.

DASH : Est un Framework Python spécialisé dans la création d'applications web de tableaux de bord. Développé au-dessus de Flask.

Hypothèses :

1. Une requête GET d'une API retourne un fichier JSON. (Adrien) **VRAI** (pas forcement un fichier JSON)
2. Pour sécuriser une API, il faut utiliser la méthode POST et non GET. (Osman) **FAUX** (POST est plutôt utilisé pour lire les informations)
3. Les API nécessitent des serveurs. (Solenn) **VRAI**
4. RESt et RESTful se différencie par l'architecture. (Jean Paul) **FAUX**
5. Une API nécessite un URI et un endpoint. (Aude) **VRAI**
6. On peut utiliser une API avec d'autres méthodes que GET et POST. (Tetyana) **VRAI**
7. Pour qu'une API soit REST, elle doit suivre un ensemble de contraintes. (Nicolas) **VRAI**
8. Une API SOAP permet d'utiliser un Swagger tandis qu'une API REST ne le permet pas. (Etienne) **FAUX**
9. La rédaction d'une requête GET est soumise à une écriture spécifique. (Loïc Ier) **VRAI**
10. Pour utiliser l'API REST, il faut d'abord construire un URI(Adeline) **VRAI**

11. Les API permettent de sécuriser les données des applications web. (Axel) **FAUX**
12. A travers l'API, la méthode PUT permet de modifier des données dans la base de données tandis que DELETE la supprime. (Briand) **VRAI**
13. Les échanges de données à travers SOAP sont plus sécurisés que ceux effectués sur les autres API. (Seydou) **FAUX**
14. Une API permet de faciliter la communication entre plusieurs langages. (Loïc II) **VRAI**

Plan d'action :

- Explorer les ressources
- Définir et comprendre les mots clés
- Répondre aux problématiques
- Vérifier les hypothèses
- Comparaison entre API REST et SOAP.

Caractéristiques	API REST	API SOAP
Architecture	Basée sur le web	Basée sur le XML
Transport	Utilise le protocole HTTP	Peut utiliser différents protocoles tels que HTTP, SMTP, TCP, etc.
Format de données	Utilise des formats de données légers tels que JSON ou XML	Utilise uniquement le format XML
Performances	Les performances sont plus rapides car les données sont généralement plus légères	Les performances sont généralement plus lentes en raison de l'utilisation de XML et de l'enveloppe SOAP
Sécurité	Utilise les protocoles de sécurité web tels que HTTPS et TLS	Utilise les protocoles de sécurité web et les fonctionnalités de sécurité SOAP
Scalabilité	Peut être facilement mis à l'échelle en ajoutant des serveurs supplémentaires	Peut être difficile à mettre à l'échelle en raison de son complexe de traitement et de son enveloppe

Niveau de complexité	Relativement simple	Plus complexe en raison de son utilisation d'enveloppes et de règles strictes
Bande passante	Requiert moins de bande passante et moins de ressources	Requiert plus de bande passante et plus de ressources