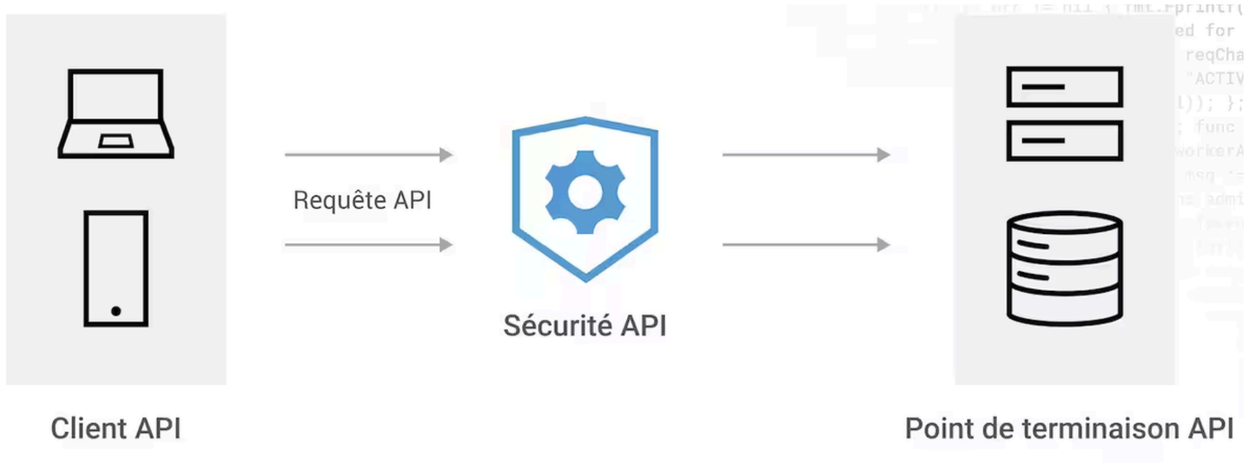
CRÉATION API/DASHBOARD AVEC PYTHON

SOMMAIRE

1. Définition des concepts
2. Création d’une API avec python (flask, fastapi, ...)
3. Création d’un dashboard avec streamlit

C'est quoi une API ?

Les **API (Application Programming Interfaces)** permettent aux **applications de communiquer entre elles** et d'**échanger des données de manière structurée**.
Python est un langage de programmation **populaire pour la création d'API** en raison de sa **simplicité**, de sa **richesse en bibliothèques** et de sa **facilité d'utilisation**.
Une API est une interface de communication entre deux systèmes informatiques. Elle permet à une application (ex: site web, app mobile, modèle ML) de demander ou envoyer des données à un autre système. Les donnés échangés le sont en format texte structuré (JSON).



Exemples:

- Une application web qui interroge une base de données pour récupérer les données sur les clients
- Une app mobile qui récupère des prévisions météo via une API météo
- Un modèle ML qui expose ses prédictions via une API

C'est quoi une API ?

Voici un aperçu des principaux concepts et des bibliothèques couramment utilisées pour créer des API en Python:

1. **Flask**: c'est un framework web léger et facile à utiliser qui permet de créer des API rapidement.
2. **Django**: est un framework web complet qui fournit un ensemble de fonctionnalités pour développer des applications web, y compris la création d'API.
3. **FastAPI:** c'est un framework moderne et performant pour la création d'API en utilisant Python. Il offre une vitesse élevée grâce à l'utilisation de fonctionnalités asynchrones et prend en charge la validation des données, la génération automatique de la documentation, l'injection de dépendances et bien plus encore.

Ces exemples offrent une introduction à la création d'API en utilisant différentes bibliothèques Python. Cependant, il existe de nombreuses autres bibliothèques et outils pour créer des API en Python, chacun offrant des fonctionnalités et des avantages spécifiques.

Méthodes de requêtes HTTP

Lorsque vous créez une API, vous pouvez définir des endpoints qui répondent à des requêtes HTTP spécifiques, tels que GET, POST, DELETE, etc. Ces endpoints sont associés à des fonctions ou des classes qui effectuent des actions spécifiques en fonction des données reçues.

GET: La méthode GET est utilisée pour **récupérer des données à partir d'une ressource spécifiée**. Lorsqu'une requête GET est effectuée sur une route d'API, le serveur renvoie les données demandées au client. Il s'agit d'une opération "lecture seule" qui ne modifie pas les données côté serveur.

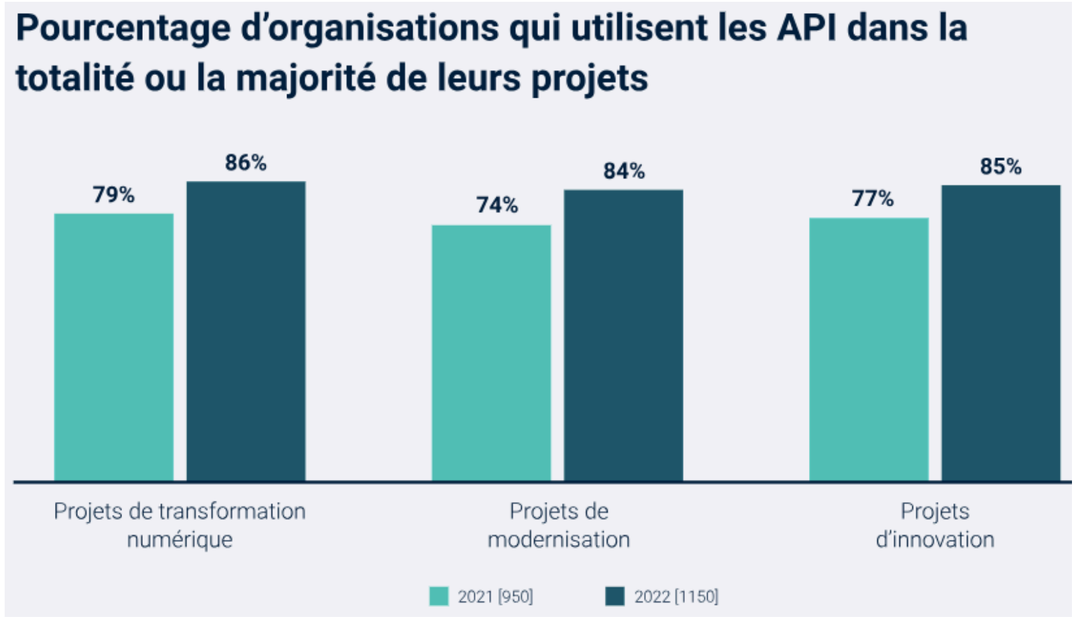
POST: La méthode POST est utilisée pour **envoyer des données au serveur** afin de créer une nouvelle ressource. Lorsqu'une requête POST est effectuée sur une route d'API, les données fournies dans le corps de la requête sont utilisées pour créer une nouvelle ressource sur le serveur. Par exemple, lors de la création d'un nouvel utilisateur, les données d'identification peuvent être envoyées via une requête POST.

DELETE: La méthode DELETE est **utilisée pour supprimer une ressource spécifiée**. Lorsqu'une requête DELETE est effectuée sur une route d'API avec un identifiant ou une référence de ressource, le serveur supprime cette ressource de manière permanente. Par exemple, une requête DELETE peut être utilisée pour supprimer un utilisateur spécifique de la base de données.

UPDATE (ou PUT/PATCH): La méthode UPDATE est utilisée pour **mettre à jour une ressource spécifiée avec de nouvelles données**. Parfois, les termes PUT ou PATCH sont également utilisés pour cette opération. Lorsqu'une requête UPDATE est effectuée sur une route d'API avec un identifiant ou une référence de ressource, le serveur met à jour cette ressource avec les données fournies dans le corps de la requête. Par exemple, une requête UPDATE peut être utilisée pour modifier les informations d'un utilisateur existant.

Quelques exemples d'entreprises utilisant les APIs

- **Facebook:** Fournit des APIs pour l'authentification des utilisateurs, la récupération de données des profils et des publications, et la publication de contenu sur la plateforme.
- **Twitter:** Offre des APIs pour la récupération de tweets, la publication de tweets, la gestion des abonnés, etc.
- **Amazon:** Propose des APIs pour l'accès aux produits, la recherche d'articles, la gestion des commandes et des paiements.
- **Shopify:** Fournit des APIs pour la gestion des produits, des commandes, des clients, etc.
- **Uber:** Offre des APIs pour la réservation de trajets, la récupération de données de localisation, la gestion des comptes utilisateurs.
- **FedEx:** Propose des APIs pour le suivi des colis, la création d'étiquettes d'expédition, la tarification des services.
- **Airbnb:** Fournit des APIs pour la recherche et la réservation d'hébergements, la gestion des calendriers et des prix.



FASTAPI

Pourquoi FastApi ?

FastAPI est un framework web Python qui permet de créer rapidement des API de haute performance.

Voici la liste des principaux avantages de FastAPI:

- **Haute performance:** La performance est supérieure à Django et Flask et est même comparable aux performances de NodeJS et GO
- **Facile à utiliser:** Créer des API 2 à 3 fois plus rapidement

1. **Typage fort et validation des données:** FastAPI utilise les types d'annotations de Python 3.7+ pour définir les types des paramètres de requête, les types de retour et les modèles de données. Cela permet de détecter les erreurs de type à la fois lors de la compilation et à l'exécution, facilitant ainsi le développement et la maintenance de l'API. De plus, FastAPI effectue une validation automatique des données entrantes, garantissant ainsi leur conformité avant leur utilisation.

- **Génération automatique de la documentation:** FastAPI génère automatiquement une documentation interactive en format Swagger UI et ReDoc. Cela signifie que vous pouvez visualiser et tester votre API à l'aide d'une interface web interactive
- **Large communauté et support actif:** FastAPI bénéficie d'une communauté en croissance rapide et d'un support actif de la part de ses développeurs.
- **Intégration transparente avec d'autres bibliothèques:** FastAPI est conçu pour être facilement intégré avec d'autres bibliothèques Python populaires, telles que SQLAlchemy pour l'interaction avec les bases de données, Pydantic pour la validation des données et la sérialisation/désérialisation, et bien d'autres.

FastApi: premiers pas !

1- Créer et accéder à un dossier pour notre projet

```
$ mkdir fastapi-tuto
$ cd fastapi-tuto
```

2- Créer un environnement virtuel avec python ou avec anaconda (Python >= 3.10)

À partir du dossier créé, vous pouvez maintenant créer l'environnement virtuel qui sera attaché au projet:

```
$ python3 -m venv env
```

Ici on lance python3 avec l'option module venv. **env** est le nom que l'on donne à notre environnement virtuel.
Une fois l'environnement virtuel créé vous pouvez l'activer:

- MacOS / Linux: $ source env/bin/activate
- Windows: $ env\Scripts\activate.bat
- À noter que pour désactiver l'environnement virtuel vous devez exécuter: deactivate

3- Installation de FastAPI et uvicorn

```
$pip install fastapi
$pip install uvicorn
```

FastApi: Exemple avec GET

1- Créer un fichier nommé 'main.py' (ou comme vous souhaitez)

- Commencez par créer un fichier Python et importez les modules FastAPI et uvicorn (pour le serveur web):

```
from fastapi import FastAPI
import uvicorn
```

2- Configuration de l'application FastAPI:

- Créez une instance de l'application FastAPI:

```
app = FastAPI()
```

3- Définition des routes et des fonctions de gestion:

- Utilisez les décorateurs fournis par FastAPI pour définir les routes de votre API et les fonctions de gestion associées. Par exemple (une fonction qui return un message => 'Hello World'):

```
@app.get("/")
def root():
  return {"message": "Hello World"}
```

4- Lancement du serveur FastAPI:

- Ajoutez le code suivant à la fin de votre fichier pour démarrer le serveur FastAPI (host et port sont optionnels):

```
if __name__ == '__main__':
   uvicorn.run(app, host='0.0.0.0', port=8000). # ou 127.0.0.1
```

5- Exécution de l'API:

- Dans votre terminal, accédez au répertoire où se trouve votre fichier Python et exécutez-le:

```
python main.py
```

FastApi: Exemple avec GET, option d'exécution avec unicorn

1- Créer un fichier nommé 'main.py' (ou comme vous souhaitez)

- Copier le code suivant

```
from fastapi import FastAPI
import uvicorn

app = FassAPI()

@app.get("/")
def root():
  return {"message": "Hello World"}
```

2- Execution de l'API: Dans votre terminal, accédez au répertoire où se trouve votre fichier Python et exécutez-le

```
uvicorn main:app --reload
```

3- Ouvrez votre navigateur et copiez le lien fourni avec la commande ci-dessus

Notes:
La commande uvicorn main:app fait référence à:
main: le fichier main.py (le "module" Python).
app: l'objet créé à l'intérieur de main.py avec la ligne app = FastAPI().
--**reload** (optionnel): fait redémarrer le serveur après des changements de code. A n'utiliser que pour le développement.
Vous pouvez ajouter d'autre paramètres comme le host (--host xxxx), le port (--port xxxx), ...

FastApi: Fonction asynchrone (async) !

Dans FastAPI, le mot-clé **async** est utilisé pour déclarer des **fonctions asynchrones**. Les fonctions asynchrones sont des fonctions qui **peuvent être exécutées de manière asynchrone**, ce qui **signifie qu'elles peuvent effectuer des opérations longues ou bloquantes sans bloquer l'exécution du programme**.

L'utilisation de fonctions asynchrones avec FastAPI permet d'améliorer la scalabilité et les performances de votre application en évitant les blocages inutiles lors de l'exécution d'opérations longues. Cela vous permet de créer des applications rapides et réactives, capables de gérer efficacement des charges de travail élevées.

Ajouter dans le fichier main.py (précédemment créé), la fonction suivante (calcul du prix de vente total d'un article):

```
@app.get("/pricer/{price}")
async def compute_total_price(price: float, tax: float):
  item = {"price": price, "tax": tax, "price_with_tax": price * (1 + tax / 100 )}
  return item
```

Si vous ouvrez dans votre navigateur une URL comme: http://127.0.0.1:8000/pricer/10000
...sans ajouter le paramètre requis taxt, vous verrez une erreur., vous verrez une erreur.

Comme tax est un paramètre obligatoire, vous devez le définir dans l'URL:

http://127.0.0.1:8000/pricer/10000?tax=18

... cela fonctionnerait:


```
{
"price": 10000,
 "tax": 18,
"price_with_tax": 11800
}
```


.... Vous pouvez tester votre API via docs/ ou redoc/ (cf slide suivante)

FastApi: Métadonnées et URLs de documentation

Dans FastAPI, les chemins **/docs** et **/redoc** sont utilisés pour fournir une documentation interactive pour votre API. Ces chemins spécifiques génèrent des pages HTML qui permettent aux utilisateurs d'explorer et de tester les endpoints de votre API.

- **/docs:** Lorsque vous accédez à /docs dans votre application FastAPI, vous êtes redirigé vers une page de documentation interactive générée automatiquement. Cette page utilise Swagger UI, qui affiche tous les endpoints de votre API, les modèles de données associés et les paramètres acceptés.
- /redoc: De manière similaire, l'accès à /redoc dans votre application FastAPI vous redirige vers une page de documentation interactive, mais cette fois-ci générée à l'aide de ReDoc.

Vous pouvez configurer les deux interfaces utilisateur de documentation:

- **Swagger UI:** servie à **/docs**.
  - Vous pouvez définir son URL à l'aide du paramètre docs_url.
  - Vous pouvez la désactiver en définissant docs_url=None.
- **ReDoc:** servi à **/redoc**.
  - Vous pouvez définir son URL à l'aide du paramètre redoc_url.
  - Vous pouvez le désactiver en définissant redoc_url=None.

```
from fastapi import FastAPI

app = FastAPI(docs_url="/documentation", redoc_url=None)

@app.get("/pricer/{price}")
async def compute_total_price(price: float, tax: float):
 item = {"price": price, "tax": tax, "price_with_tax": price * (1 + tax / 100 )}
 return item
```

Exemple, pour configurer l'interface Swagger afin qu'elle soit servie dans /documentation et désactiver ReDoc.

FastApi: Exemple avec POST

1- Créer un fichier nommé 'myapp.py' (ou comme vous souhaitez)

- Commencez par créer un fichier Python, copiez le code suivant et lancez l'API

```
from fastapi import FastAPI
from pydantic import BaseModel

class Item(BaseModel):
  name: str
  description: str | None = None
  price: float
  tax: float | None = None


app = FastAPI()


# TODO: une fonction qui ajoute le prix TTC (toute taxe comprise) en fonction de price et de tax fournis ensuite met à jour les données d'entrée.

@app.post("/items/")
async def create_item(item: Item):
  item_dict = item.dict()
  if item.tax:
    price_with_tax = item.price + item.tax
    item_dict.update({"price_with_tax": price_with_tax})
   return item_dict
```

Exposer une API pour la tester

- Local: via uvicorn ou flask run
- Public (test rapide):
  - ngrok pour créer un tunnel sécurisé, exposition rapide d'une API locale
  - Render ou Railway pour le déploiement gratuit
  - Serveurs cloud: AWS EC2, AWS (Lambda + API Gateway), Google Compute Engine, Azure Virtual Machines, Heroku

Tester d'API:

- Lancer l'API localement via le navigateur
  - Si vous utilisez FastAPI, vous avez une documentation interactive automatique (par defaut: http://127.0.0.1:8000/docs)
- Tester avec **curl** via ligne de commande
  - Tester une route GET: curl http://localhost:8000/
- Tester avec Postman (GUI populaire)

DASHBOARD AVEC STREAMLIT

Pourquoi streamlit ?

Streamlit

Streamlit est un framework open-source qui permet de créer facilement des applications Web interactives en utilisant du code Python. Il est conçu spécifiquement pour les data scientists, les data analysts, les développeurs, etc. qui souhaitent partager rapidement et facilement leurs analyses, leurs modèles de machine learning ou leurs visualisations de données. Streamlit offre plusieurs avantages pour le développement d'applications Web interactives en utilisant Python:

1. **Simplicité d'utilisation:** Streamlit est conçu pour être simple et intuitif.
2. **Intégration aisée:** Streamlit s'intègre facilement avec d'autres bibliothèques Python populaires telles que Pandas, Matplotlib, Plotly et bien d'autres.
3. **Mise à jour en temps réel:** L'un des avantages clés de Streamlit est qu'il met à jour automatiquement votre application à chaque fois que vous modifiez le code. Vous pouvez simplement enregistrer vos modifications et votre application se rafraîchira automatiquement dans le navigateur.
4. **Ajout de fonctionnalités avancées:** Streamlit offre une grande variété d'éléments interactifs et de widgets pour rendre votre application plus dynamique. Vous pouvez ajouter des graphiques interactifs, des sélecteurs de données, des curseurs, des cartes, etc
5. **Déploiement facile:** Streamlit facilite le déploiement des applications.

Streamlit: premier pas !

1- **Installation**: installer Streamlit en exécutant la commande suivante dans votre terminal:

```
pip install streamlit
```

2- **Création d'une application Streamlit:** Commencez par créer un nouveau fichier Python (dashboard_tuto.py). Importez le module streamlit et commencez à écrire votre code.

```
import streamlit as st

st.title("Mon application Streamlit")
st.write("Bienvenue sur mon application !")
```

3- **Exécution de l'application:** Enregistrez votre fichier Python et exécutez-le dans votre terminal à l'aide de la commande suivante. Cela démarrera le serveur Streamlit et ouvrira automatiquement votre application dans votre navigateur par défaut.

```
streamlit run dashboard_tuto.py
```

Ajout de widgets et graphiques interactifs en

TP: Dashboard pour le suivi des performances marketing

Streamlit

1. **Données**: les fichiers impressions, clics et achats utilisés lors du TP sur Tableau (ici)
2. **Créer une API** qui charge les données et les fusionne (merge des trois tables)
3. **Mettre en place le dashboard** avec streamlit (le choix des graphiques est libre) qui fait appel à l'API pour obtenir les données.
4. **Déployer votre dashboard** dans streamlit cloud (vous pouvez consulter la documentation officielle)
5. **Envoyer** le **lien de votre dashboard ainsi que celui du repository github contenant le code par mail** au plus tard **vendredi 07/07/2023 à 18h (GMT)** avec l'objet '**TP-streamlit**'

Références

https://dev.to/ericlecodeur/introduction-a-fastapi-python-5mf

https://fastapi.tiangolo.com/

https://flask.palletsprojects.com/en/2.3.x/

https://www.softwareag.com/fr_fr/resources/api/guide/api-management.html?utm_source=google&utm_medium=cpc&utm_campaign=aim_api-intg&utm_region=hq&utm_subcampaign=stg-1&utm_content=stg-1_guide_practical-guide-to-api-mgmt&gclid=CjwKCAjwkLCkBhA9EiwAka9QRhZiaoyfnqH_dcxGbm1PClwGll7zRM4DjYMRC0l_KzltUDraLhL2rBoCx70QAvD_BwE

https://docs.streamlit.io/

https://docs.streamlit.io/streamlit-community-cloud

https://medium.com/codex/streamlit-fastapi-%EF%B8%8F-the-ingredients-you-need-for-your-next-data-science-recipe-ffbeb5f76a92