Publier des messages (en POST)

Amina MARIE

Solutions d’échange inter-applicatif

Ce module présente différentes méthodes d'échange de données entre systèmes (interne ou publics). Il se focalise particulièrement sur l'utilisation d'APIs en présentant leur fonctionnement et les méthodes courantes pour communiquer avec elles (récupérer ou envoyer des données).

- Partie 1 – Exposer ses données via une API REST sécurisée
- Partie 2 – Requêter des données et construire un mini ETL
- Partie 3 – Utiliser des API Push
- Partie 4 – Publier des données avec un POST

Projet fil rouge – API de gestion de personnages de manga

Objectif global : Construire une API complète de gestion de personnages fictifs (type Olive et Tom) avec des fonctionnalités :

- Exposition sécurisée (GET /personnages)
- Scripts de requêtage
- Simulation d'événements (push)
- Envoi de données à l'API (POST /scores)
- Chaque partie du cours alimente et améliore ce projet en y ajoutant une brique fonctionnelle.

Plan de la 3ème partie

Objectifs pédagogiques

- Comprendre le fonctionnement d’un endpoint POST
- Envoyer des données à une API via body ou formulaire
- Traiter ces données côté serveur avec FastAPI ou en JavaScript
- Réutiliser les données collectées précédemment et les renvoyer
- Intégrer proprement la transformation + publication dans un flux automatisé

Définition d’un endpoint POST

Définition

Un **endpoint POST** permet à un client d'**envoyer des données** à une API.

Contrairement au GET, qui sert à **lire** des données, le POST est utilisé pour :

- Créer une ressource
- Soumettre un formulaire
- Déclencher un traitement (ex : envoyer un message, enregistrer une info)

Comparaison rapide

Type	Usage principal	Données envoyées ?	Exemple
GET	Lire des données	❌ Non	/api/users
POST	Envoyer/créer	✅ Oui (JSON, Form)	/api/users + {nom, email}

Comment recevoir des données en POST

En python

Corps JSON

```python
from fastapi import FastAPI
from pydantic import BaseModel

app = FastAPI()

class Personnage(BaseModel):
    nom: str
    score: int

@app.post("/ajouter")
async def ajouter_personnage(p: Personnage):
    return {"message": f"{p.nom} reçu avec score {p.score}"}
```

Tu envoies depuis Postman, curl ou JS

```json
{
  "nom": "Sakura",
  "score": 73
}
```

En python

Données formulaire (type HTML classique)

```python
from fastapi import Form

@app.post("/formulaire")
async def formulaire(nom: str = Form(...), score: int = Form(...)):
    return {"message": f"{nom} reçu via formulaire"}
```

A utiliser si les données viennent d’un form HTML classique

En JavaScript (fetch)

Envoyer un JSON

```
fetch("http://localhost:8000/ajouter", {
  method: "POST",
  headers: {
    "Content-Type": "application/json"
  },
  body: JSON.stringify({ nom: "Itachi", score: 99 })
})
.then(res => res.json())
.then(data => console.log(data));
```

Envoyer en FormData

```
const formData = new FormData();
formData.append("nom", "Naruto");
formData.append("score", 87);

fetch("http://localhost:8000/formulaire", {
  method: "POST",
  body: formData
})
.then(res => res.json())
.then(data => console.log(data));
```

Exercices

Objectifs

Créer un **nouvel endpoint POST** dans ton API pour recevoir des données transformées.

Puis, depuis ton script Python de la Partie 2, **transformer** les données collectées, puis les **renvoyer en POST** vers cet endpoint.

Exercice

1. Créer un endpoint POST dans l'API FastAPI

- /traitement
- Accepte un JSON avec nom + score
- Calcule un champ "niveau" selon le score
- Retourne une version enrichie

2. Reprendre les données collectées dans l'exercice 2

- Lire le fichier JSON contenant les personnages
- Pour chaque personnage :
    - Ajouter un champ "score_doublé" ou "niveau"
    - Les renvoyer vers /traitement (POST)

3. Vérifier la réponse de l'API

- Afficher un résumé dans la console : personnage + niveau calculé

Exercice

Exemple de résultat

Avant :

```
[
  {"nom": "Shikamaru", "score": 82},
  {"nom": "Hinata", "score": 65}
]
```

Après :

```
{"nom": "Shikamaru", "score": 82, "niveau": "expert"}
```

Annexe - dépannage

Problème	Cause fréquente	Solution
422 Unprocessable Entity	JSON mal formé ou mauvais champ	Vérifie nom des clés, types
500 Internal Error	Mauvais accès fichier ou champ absent	Utilise .get() et try/except
TypeError: object is not subscriptable	Tu traites un objet comme une liste	Utilise .dict() sur le modèle Pydantic

Annexe - ressources

- FastAPI POST docs
- Form handling in FastAPI
- JS fetch POST

Merci