# TP – Découverte d'API Platform

### Objectif : créer les premières entités du projet "Visites Tuteurs"

## Contexte

Vous allez démarrer le projet de fin d'année, qui consistera à réaliser une application de gestion des visites d'entreprise pour les tuteurs qui suivent les étudiants en stage ou en alternance.

Dans ce TP, vous allez construire la partie back-end de cette application, à l'aide de Symfony et API Platform.
Vous apprendrez à créer une API REST qui permettra ensuite à une interface front (Twig ) de communiquer avec les données.

## Prérequis

Avant de commencer :

- Docker Desktop est lancé
- Vos conteneurs Symfony fonctionnent avec le **nouveau DockerFile fourni**

```
docker compose down --remove-orphans (si vos conteneurs sont déjà en route)
docker compose build --no-cache app
docker compose up -d
```

- Vous savez accéder au conteneur PHP avec :

```
docker exec -it symfony_app bash
```

## Étape 1 — Travailler avec API Platform

1. Dans le conteneur PHP, placez-vous dans le répertoire `/var/www/apptuteur` :
2. Installez Doctrine et de quoi faire des entités

```
composer require symfony/orm-pack
composer require doctrine/doctrine-migrations-bundle
composer require –dev symfony/maker-bundle
```

3. Installez API Platform :

```
symfony composer require api
```

4. Démarrez le serveur Symfony :

```
symfony server:start
```

5. Ouvrez dans votre navigateur :
   - http://localhost:8000 → Page d'accueil Symfony
   - http://localhost:8000/api → Interface Swagger d'API Platform

Si vous voyez Swagger avec la mention "No resources defined", c'est normal : aucune entité n'a encore été créée.

Le service `db` contient une base MySQL accessible depuis Symfony grâce à la variable `DATABASE_URL` dans le fichier `.env`.

## Étape 2 — Configurer la base de données

1. Ouvre le fichier `.env` du projet `apptuteur`
   et vérifie la ligne :

   ```
   DATABASE_URL="mysql://user:password@db:3306/symfony_db?charset=utf8mb4"
   ```

   > ⚠ Si le mot de passe MySQL de ton Docker diffère, adapte-le.
   > (Dans le TP Docker précédent, l'utilisateur et le mot de passe étaient souvent `user/password`)

2. Crée la base dans le conteneur (mais nous l'avons déjà créée au TP précédent) :

   ```
   symfony console doctrine:database:create
   ```

3. Plus tard, lorsque vous créerez vos entités (Tuteur, Visite), Doctrine pourra ensuite générer les tables automatiquement grâce à :

   ```
   symfony console make:migration
   symfony console doctrine:migrations:migrate
   ```

4. Vous pouvez visualiser la base et les tables dans PhpMyAdmin à l'adresse :
   http://localhost:8081
   - utilisateur : `user`
   - mot de passe : `password`
   - base : `symfony_db`

"Quand vous créez vos entités dans API Platform, Symfony utilise Doctrine pour créer automatiquement les tables correspondantes dans la base MySQL du conteneur `db`.
Vous pouvez vérifier ces tables dans PhpMyAdmin.
C'est cette même base que votre futur front (Twig) interrogera via Doctrine."

## Étape 3 — Créer l'entité `Tuteur`

Dans le conteneur PHP :

```
1. symfony console make:entity
```

Répondez aux questions suivantes :

```
Class name of the entity to create or update: Tuteur
Mark this class as an API Platform resource (expose a CRUD API for it) (yes/no): yes
New property name: nom (string)
New property name: prenom (string)
New property name: entreprise (string)
New property name: email (string)
New property name: telephone (string)
```

2. appliquez les migrations :

```
symfony console make:migration
symfony console doctrine:migrations:migrate
```

3. Rechargez la page http://localhost:8000/api.
   ➜ Une ressource `Tuteur` apparaît automatiquement.

4. Dans Swagger :

   Testez **GET** `/api/tuteurs` (la liste est vide).

   Testez **POST** `/api/tuteurs` et ajoutez un tuteur :

   ```
   {
     "nom": "Johnson",
     "prenom": "Paul",
     "entreprise": "Acme",
     "email": "paul.johnson@acme.com",
     "telephone": "0600000001"
   }
   ```

   Puis testez à nouveau **GET** `/api/tuteurs`.

# Étape 4 — Créer l'entité `Visite`

1. Dans le conteneur PHP :

   ```
   symfony console make:entity
   ```

2. Répondez :

   ```
   Class name of the entity: Visite
   Mark this class as an API Platform resource (yes/no): yes
   New property name: date (datetime_immutable)
   New property name: commentaire (string)
   New property name: tuteur (relation)
   ```

   Lorsqu'on vous demande le type de relation :

   ```
   Relation type? [ManyToOne, OneToMany, ManyToMany, OneToOne]:
   > ManyToOne
   Target Entity:
   > Tuteur
   ```

3. Exécutez les migrations :

```
symfony console make:migration
symfony console doctrine:migrations:migrate
```

4. Rechargez la page Swagger : vous devez voir les deux ressources :
   - `/api/tuteurs`
   - `/api/visites`

# Étape 5 — Tester les routes dans Swagger

Créez une **Visite** en cliquant sur "Try it out" puis sur **POST /api/visites** :
Exemple de données :

```
{
  "date": "2025-03-14T09:30:00+00:00",
  "commentaire": "Visite de suivi très positive",
  "tuteur": "/api/tuteurs/1"
}
```

1. Faites un **GET /api/visites** pour voir toutes les visites.
   Vous pouvez aussi faire un **GET /api/tuteurs/1** pour vérifier que le tuteur a bien une relation avec ses visites.

# Étape 6 — Ajouter un peu de validation

1. Ouvrez le fichier `src/Entity/Visite.php`.
   Ajoutez les contraintes suivantes au-dessus des propriétés :

```
use Symfony\Component\Validator\Constraints as Assert;

#[Assert\NotBlank]
private ?\DateTimeImmutable $date = null;

#[Assert\Length(min: 5)]
private ?string $commentaire = null;
```

2. Dans Swagger, essayez de créer une visite **sans commentaire**.
   ➜ Observez la réponse renvoyée par API Platform : une **erreur de validation** au format JSON.

# Étape 7 — Discussion

Répondez à ces trois questions :

1. Quelle est la différence entre un **backend** et un **frontend** ?
2. Pourquoi utiliser une **API** dans un projet web ?
3. Quelle sera la prochaine étape dans ce projet (selon vous) ?