

ADRAR\o/ **PÔLE NUMÉRIQUE**

- > **INFRASTRUCTURES** SYSTÈMES & RÉSEAUX
- > **CYBERSÉCURITÉ** INFRASTRUCTURES & APPLICATIONS
- > **DEVOPS / SCRIPTING** & AUTOMATISATION
- > **DEVELOPPEMENT** WEB & MOBILE
- > **TRANSFORMATION NUMERIQUE DES ENTREPRISES**

www.adrar-numerique.com

API est un acronyme signifiant *Application Programming Interface*.

Le concept des APIs et de rendre accessible toute ou partie de données gratuitement ou non, avec des restrictions, permettant de lier des services entre eux.

C'est ce qui permet à de nombreux services d'exister et co-exister ! Prenons par exemple les comparateurs en ligne d'articles/vols/hôtels... Ces comparateurs font des appels API aux différents services visés pour récupérer des informations à transmettre à leurs utilisateurs.

Dans nos exemples, l'API va nous permettre de réaliser plusieurs actions sur un cours, un chapitre...

Les méthodes HTTP

Symfony

Vous connaissez déjà certaines méthodes HTTP, revoyons cela et abordons-en de nouvelles:

- **GET:** permet de récupérer des informations par rapport à l'URI. Cette requête doit respecter deux principes:
 - être *safe*, elle ne doit pas affecter les données du serveur
 - être *idempotent*, elle doit toujours faire la même chose
- **POST:** permet de créer une ressource au travers du contenu de la requête
- **PUT:** permet de remplacer les informations d'une ressource existante au travers du contenu de la requête
- **DELETE:** permet de supprimer une ou plusieurs ressources par rapport à l'URI
- ...

Les codes de *statut* Symfony

Comme pour les méthodes HTTP, vous connaissez déjà des **status code**:

- **1xx (informations)**: permet d'informer le client de l'état de la demande (requête reçue, traitement en cours, ...)
- **2xx (succès)**: tout va bien
- **3xx (redirections)**: redirection à venir (exception pour le code 304 qui signifie contenu inchangé dû au cache)
- **4xx (erreurs client)**: requête erronée, traitement impossible
- **5xx (erreurs serveur)**: problème empêchant le traitement de la requête

Les différents types d'API

Symfony

Il existe 4 gros types d'API

- publiques: API connues de tous et peut être utilisée par tout développeur ou acteur tiers
- partenaires: disponibles pour les développeurs externes. Monétisation interdite
- internes: permet de connecter des services au sein même d'une entreprise
- composites: traite des tâches lourdes et complexes. Fonctionnent au travers de plusieurs API. Permet d'améliorer la vitesse et la performance

Nomenclature Symfony

Nous allons nous baser sur le niveau 2 du modèle de maturité de Richardson

```
POST /{entites}
```

=> pour la création

```
GET /{entites}/{identifiant}
```

=> pour la récupération

```
PUT /{entites}/{identifiant}
```

=> pour la mise à jour

```
DELETE /{entites}/{identifiant}
```

=> pour la suppression

NB: les éléments entre accolades "{}" DOIVENT être remplacés par le nom de votre entité, ceux entre crochets "[]" PEUVENT être remplacés

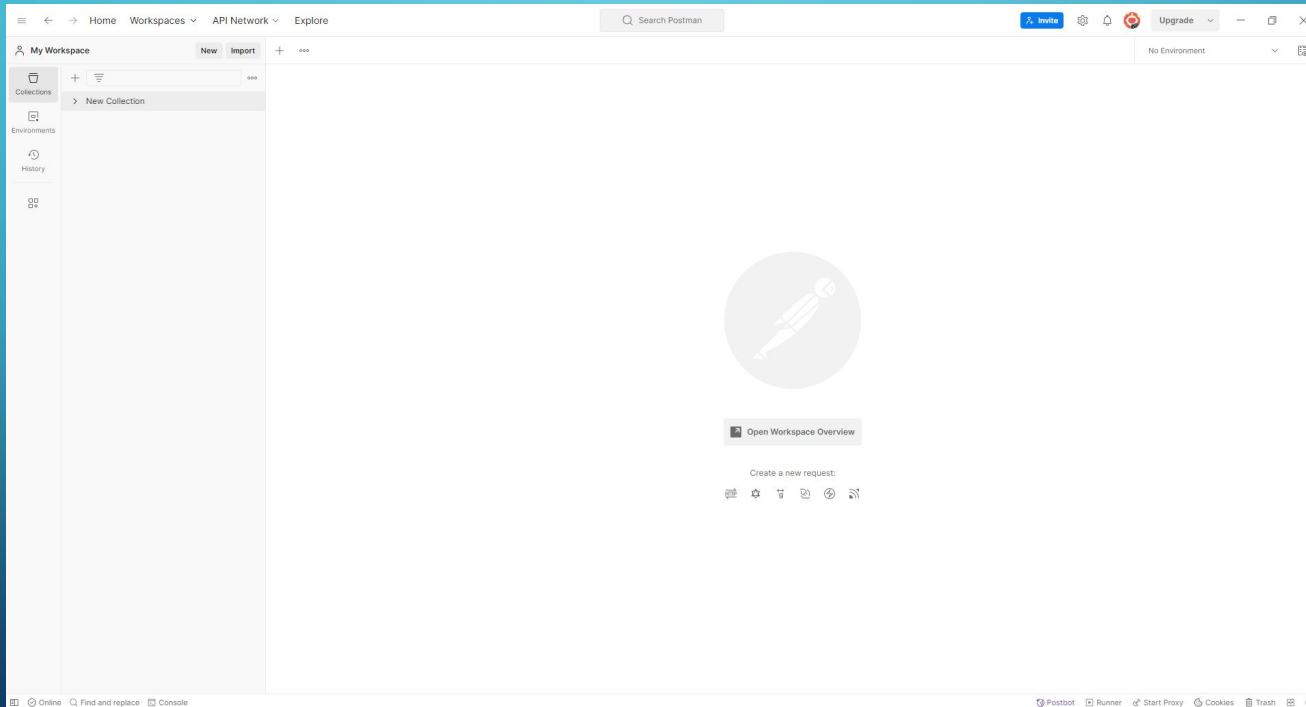
Nomenclature Symfony

En nous basant sur ce modèle, il faudra nous assurer de retourner le bon code de statut (*code status*):

- 200 (ok): tout s'est bien déroulé
- 201 (created): création réussie, le contenu de la nouvelle ressource est renvoyé en réponse de façon optionnelle, et une redirection est effectuée avec l'URL de la nouvelle ressource
- 204 (no content): création réussie mais la ressource n'est pas renvoyée dans la réponse, généralement lors d'une suppression
- 304 (not modified): contenu non modifié depuis la dernière mise en cache
- 400 (bad request): la demande n'a pas pu être traitée correctement
- 401 (unauthorized): authentification échouée
- 403 (forbidden): accès non autorisé
- 404 (not found): ressource non trouvée
- 405 (method not allowed): méthode HTTP non traitée par l'API
- 406 (not acceptable): ne peux pas générer la chose demandée dans le Accept (en en-tête)
- 500 (server error): le serveur rencontre un problème

Postman (1/2) Symfony

Pour effectuer nos tests de requêtes, nous allons utiliser l'application Postman.
Une fois connecté·e, vous devriez tomber sur une fenêtre similaire à ceci:

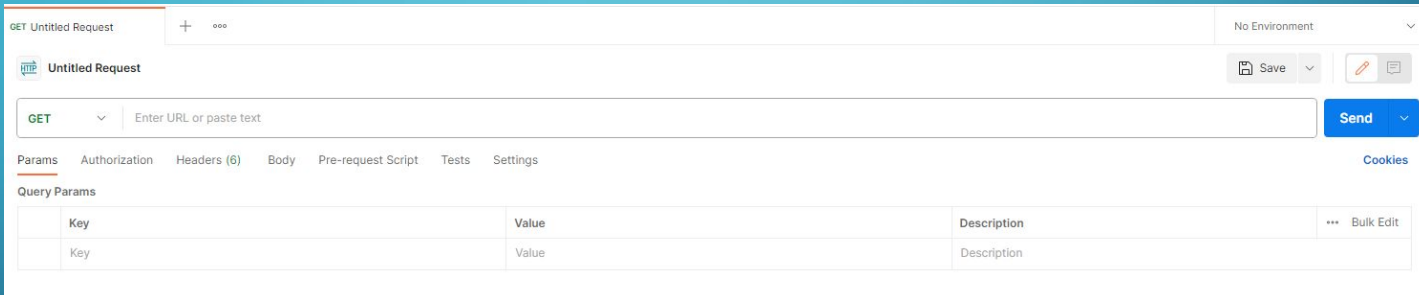


Postman (2/2)

Symfony

Pour effectuer un appel, il suffit de:

- cliquer sur le "+"
- choisir la méthode HTTP désirée
- saisir le lien sur lequel effectuer la requête
- saisir les paramètres/en-têtes... pour des requêtes plus poussées



The screenshot shows the Postman application interface. At the top, there's a header with "GET Untitled Request" and a "+" button. Below this, there's a section for "Untitled Request" with a "Save" button and a "Send" button. The main area is divided into tabs: "Params", "Authorization", "Headers (6)", "Body", "Pre-request Script", "Tests", and "Settings". The "Params" tab is selected, showing a table for "Query Params".

| Key | Value | Description |
|-----|-------|-------------|
| Key | Value | Description |

Créez un nouveau projet pour tester sans le drapeau "--webapp"

```
$ symfony new api_adrar
```

Ajoutez le bundle Maker et orm fixtures permettant la génération des fichiers et la génération d'enregistrements (en mode dev)

```
$ composer require symfony/maker-bundle --dev && composer require orm-fixtures --dev && composer require fzaninotto/faker --dev
```

Puis ajoutez Doctrine pour faire les liens entre entités et tables SQL

```
$ composer require orm
```

Pensez à configurer le .env et créer la BDD

Dépendance API Platform Symfony

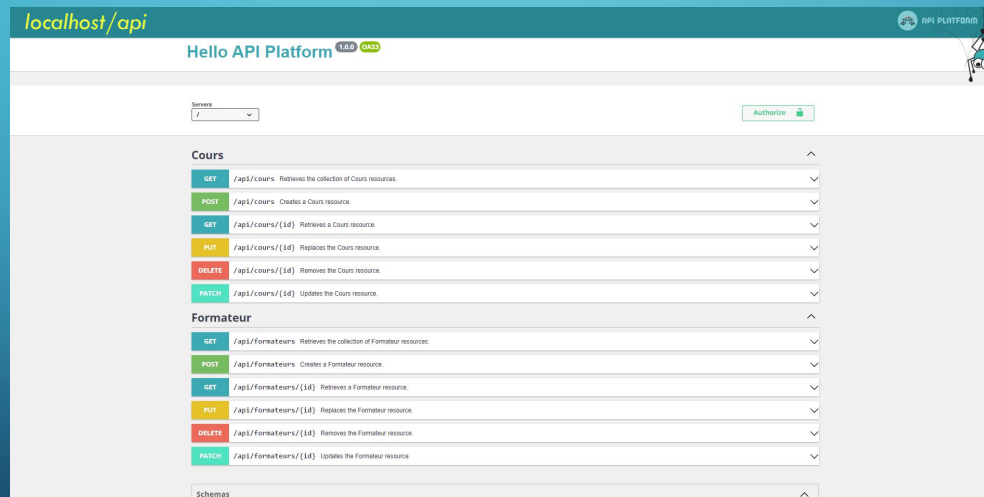
API Platform

Ajoutez maintenant la dépendance pour installer l'API Platform qui va nous faciliter grandement la tâche !

```
$ symfony composer req api
```

Rendez-vous sur `/api` pour voir la page

Lors de la génération de vos entités, la CLI vous demande si vous désirez intégrer cette entité dans l'API Platform pour le cours j'ai dit oui, dans votre cas.



NB: la dépendance API permet de construire un environnement entièrement fonctionnel implémentant déjà les méthodes, je vous montre comment cela fonctionne, mais de votre côté, il faudra plutôt utiliser la méthode classique

Cas pratique Symfony

Générez vos entités, pour l'exemple j'en ai généré deux **Formateur** et **Cours**
Générez des données grâce aux Fixtures et le PHPFaker pour alimenter ces tables

Méthode classique

Puisque nous désirons générer l'entièreté de nos routes, nous allons les faire nous mêmes.
Créez un contrôleur (soit un par entité, soit un générique).

```
#[Route('/api/v2')]
class AppController extends AbstractController
{
    #[Route('/formateurs', name: 'app_list_formateurs', methods: ['GET'])]
    public function getFormateursList(FormateurRepository $formateurRepository,
    SerializerInterface $serializer): JsonResponse
    {
        $formateurs = $formateurRepository->findAll();
        $jsonFormateurs = $serializer->serialize($formateurs, 'json');
        return new JsonResponse($jsonFormateurs, Response::HTTP_OK, ['Content-Type' =>
        'application/json'], true);
    }
}
```

AppController.php

Méthode classique

Sur le code précédent on retrouve plusieurs principes:

- l'annotation de route au dessus de la classe me permet de dire que toutes les routes à l'intérieur commenceront par cette URI suivie de l'URI de la méthode en question
- sur chaque méthode où je définis une route, on peut également restreindre la méthode qui y aura accès. Dans le code d'avant, je restreins cette route sur la méthode HTTP GET
- je sérialise ma chaîne de caractères en **JSON**
- je précise que ma méthode devra retourner le type **JsonResponse**
- dans le retour, j'instancie ce type et lui renseigne:
 - les informations qu'il devra retourner
 - le code HTTP renvoyé (important pour que le navigateur ou l'application sache comment traiter l'information)
 - les en-têtes
 - et si le format est du **JSON** (pour permettre son interprétation)
- ...

JsonResponse (3/3) Symfony

Voilà le résultat du rendu récupéré avec Postman:

```

GET http://api.marceau-rodrigues.fr/api/v2/formateurs Send
Params Auth Headers (6) Body Pre-req. Tests Settings Cookies
Body 200 OK 172 ms 1.05 KB Save as Example
Pretty Raw Preview Visualize JSON
1 {
2   "id": 21,
3   "nom": "Rousset",
4   "prenom": "Brigitte",
5   "email": "jeannine50@tele2.fr",
6   "mdp": "C1FpM2_6gTt6/Uo",
7   "cours": [
8     "/api/cours/7",
9     "/api/cours/10",
10    "/api/cours/13",
11    "/api/cours/16"
12  ]
13 },
14 {
15   "id": 22,
16   "nom": "Barbe",
17   "prenom": "Bernadette",
18   "email": "eleonore93@tele2.fr",
19   "mdp": "6n7NM010{[",
20   "cours": [
21     "/api/cours/8",
22     "/api/cours/11",
23     "/api/cours/14",
24     "/api/cours/17"
25  ]
26 },
27 {
28   "id": 23,
29   "nom": "Fauze",
30   "prenom": "Vincen",
31   "email": "caulier.marco@noos.fr",
32   "mdp": "oSeVSxB",
33   "cours": [
34     "/api/cours/9",
35     "/api/cours/12",
36     "/api/cours/15",
37     "/api/cours/18"
38  ]
39 }
  
```

Méthode classique

Utilisation d'une donnée provenant d'une API Symfony

Dans **VOTRE** projet (pas celui qui crée l'API), installez cette dépendance qui va vous permettre de faire les appels

```
$ composer require symfony/http-client
```

Ensuite, testez en utilisant ce code sur une route (en l'adaptant):

```
#[Route('/', name: 'app_home')]
public function index(HttpClientInterface $client): Response {
    $response = $client->request(
        'GET',
        'http://api.marceau-rodrigues.fr/api/v2/formateurs'
    );

    return $this->render('fake/index.html.twig', [
        'controller_name' => 'AppController',
    ]);
}
```

AppController.php

NB: Ici, je viens faire appel à la liste de mes formateurs en faisant un appel Curl via l'interface cliente HTTP.