Symfony

API, Microservices et Evénements

Maxime Puys

22 mars 2025

Plan

Rappels API et Microservices

Qu'est-ce qu'une architecture d'application Web ?

Composants de l'architecture :

- Browser, Front-End, Back-End, BDD, API, etc

Architecture :

- Cadre qui détermine comment les composants d'une application communiquent et interagissent.
- Elle façonne le développement, la maintenance et l'évolutivité des applications.

Sites Web vs. Services

Site Web

- Contenu navigable par un navigateur web
- Utilisation par un utilisateur final (humain)
- Interface conviviale : images, formulaires, liens, etc
- **Exemples :** Sites d'actualités, portails d'entreprise, boutiques en ligne, blogs, réseaux sociaux.

Service Web

- Fournit des fonctionnalités et des données à d'autres applications
- Pas d'accès direct pour l'utilisateur final
- Interface via une API ou une bibliothèque de méthodes
- **Exemples :** Paiement en ligne, cartographie, météo, etc

Architecture Monolithique

Définition

- Unité unifiée où tous les composants de l'application sont étroitement liés et fonctionnent comme un seul service.

Caractéristiques

- Base de code unique pour toutes les fonctionnalités.
- Une seule base de données pour tous les besoins de l'application.
- Déployée comme un seul bloc, souvent rendant l'installation initiale plus facile.

Avantages de l'Architecture Monolithique

- Simplicité dans le développement et le déploiement
- Débogage et test plus faciles grâce à une base de code unique
- Scalabilité verticale simple par exécution de multiples copies sur des serveurs plus grands

Inconvénients de l'Architecture Monolithique

- Défis de scalabilité
- Difficulté d'adopter de nouvelles technologies
- Temps de démarrage plus long
- Impact d'un changement affectant tout le système, augmentant le risque

Architecture Monolithique

A monolithic application puts all its functionality into a single process...

... and scales by replicating the monolith on multiple servers

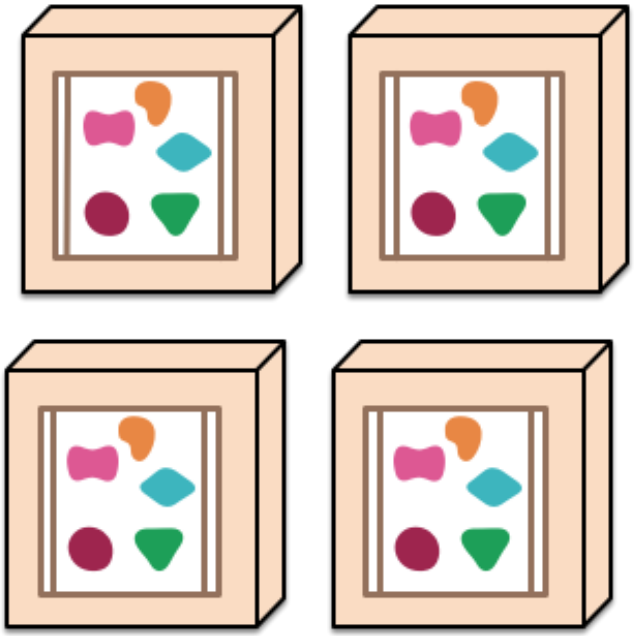


Figure – https://martinfowler.com/articles/microservices.html

Architecture Microservices - Vue d'ensemble

Définition

- L'architecture microservices décompose une application monolithique en une collection de petites unités indépendantes, chacune avec une fonctionnalité.

Caractéristiques

- Chaque microservice est axé sur une fonctionnalité spécifique.
- Les services sont déployés indépendamment.
- Différent langages/frameworks peuvent être utilisées dans différents services.

Avantages des Microservices

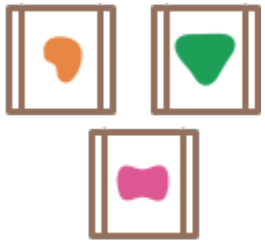
- Scalabilité améliorée
- Flexibilité dans l'utilisation de différentes technologies
- Cycles de déploiement plus rapides

Inconvénients des Microservices

- Complexité dans la gestion de multiples services
- Surcharge dans la communication entre les services
- Défis dans le maintien de la cohérence des données

Architecture Microservices

A microservices architecture puts each element of functionality into a separate service...



... and scales by distributing these services across servers, replicating as needed.

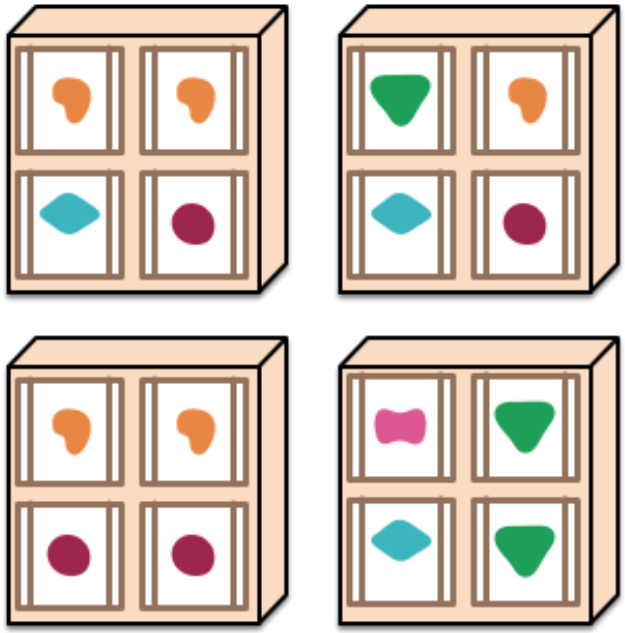


Figure – https://martinfowler.com/articles/microservices.html

API (Interface de Programmation d'Applications)

Définition

- Les API permettent à différents composants logiciels de communiquer entre eux.

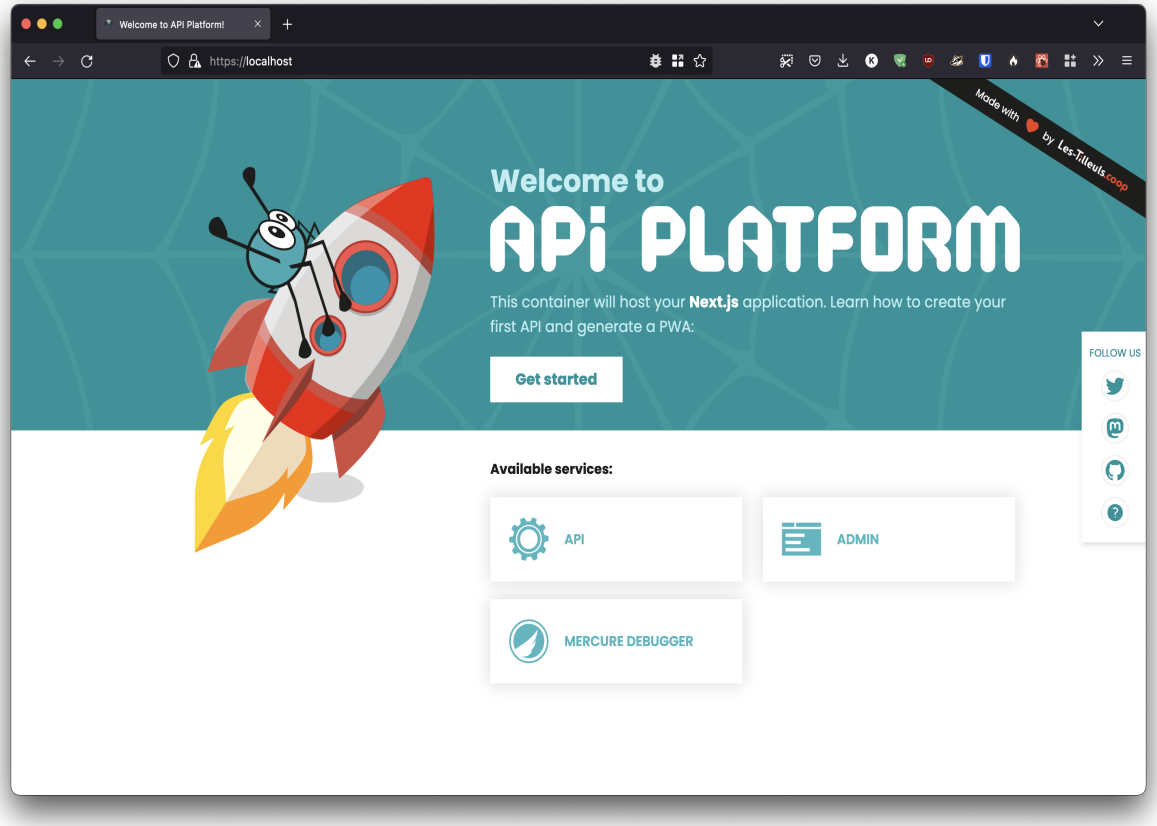
Exemples d'API

- **API RESTful :** Utilisent des requêtes HTTP pour accéder aux ressources.
- **API SOAP :** Utilisent des messages XML pour les demandes de service.
- **API GRPC/Protobuf :** Définition des méthodes appelées à distance et serialisation de données.

API en Symfony

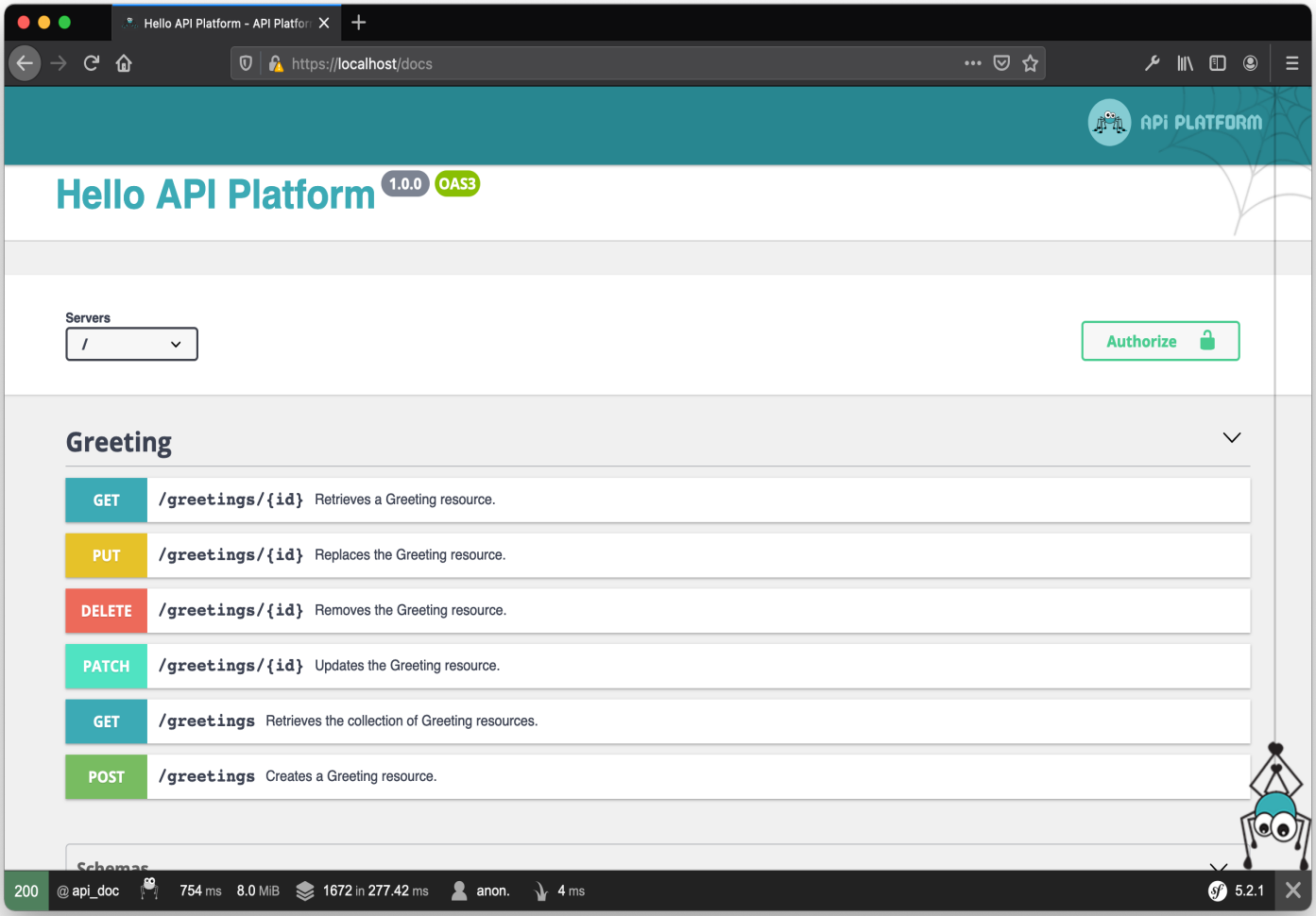
Présentation de API Platform 1/2

- API Platform : framework avancé pour la création d'APIs
  - Utilisable avec Symfony
  - Supporte REST et GraphQL
  - Intégration avec Docker et Kubernetes pour le déploiement



Présentation de API Platform 2/2

- Routes automatiques pour des fonctions CRUD sur des entités
- Visualisation avec Swagger UI



Présentation de OpenAPI

- **Standard ouvert pour les interfaces RESTful** :
  - Interface standardisée pour description et documentation d'APIs REST.
  - Description de l'API en format YAML.
- **Génération automatique de documentation** :
  - Génération automatique de documentation et pages de test avec Swagger UI.
- **Interopérabilité entre différents outils et langages de programmation** :
  - Compatibilité d'OpenAPI avec une variété de langages de programmation comme Java, Python, Node.js, etc.
  - Les outils de génération de code peuvent créer des serveurs et des clients API dans plusieurs langages à partir d'un fichier OpenAPI.

Installation 1/2

- **Installation d'API Platform avec Docker** :

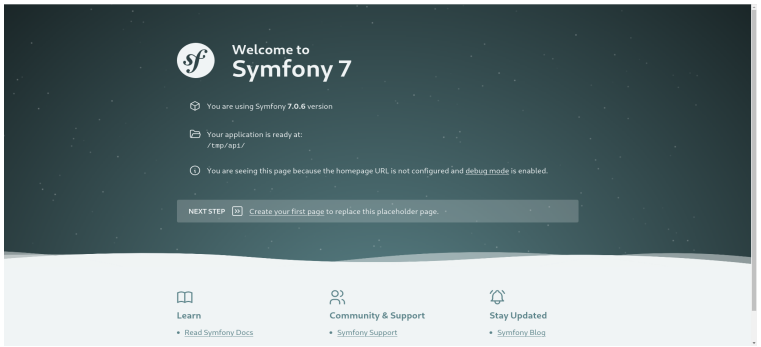
```
git clone https://github.com/api-platform/api-platform.git
cd api-platform
docker compose up -d
```

- ***Ou* Installation avec Symfony CLI** :

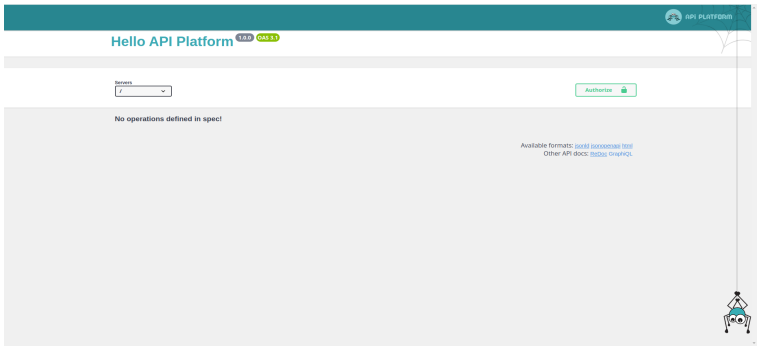
```
symfony new mon-api-platform
cd mon-api-platform
symfony composer require api
symfony server:start
```

Installation 2/2

- `http://127.0.0.1:8000` : Page d'accueil de Symfony



- `http://127.0.0.1:8000/api` : Page d'accueil d'API Platform



- Attention : API Platform est un bundle
  - → Modifie le fonctionnement de Symfony,
  - Ex : `/api` est un évènement sur le kernel et pas une route déclarée

Déclarer une Entité comme Ressource 1/2

- **Rappel** : MakerBundle pour générer du code

```
$ symfony console make:entity

Class name of the entity to create or update (e.g. GrumpyPuppy):
> Book

Mark this class as an API Platform resource (expose a CRUD API for it) (yes/no)
> yes

created: src/Entity/Book.php
created: src/Repository/BookRepository.php

Entity generated! Now let's add some fields!
You can always add more fields later manually or by re-running this command.

New property name (press <return> to stop adding fields):
> title
```

Déclarer une Entité comme Ressource 2/2

```php
// api/src/Entity/Book.php
namespace App\Entity;

use ApiPlatform\Metadata\ApiResource;

#[ApiResource]
class Book
{
    private ?int $id = null;
    private string $title;
    // autres propriétés
}
```

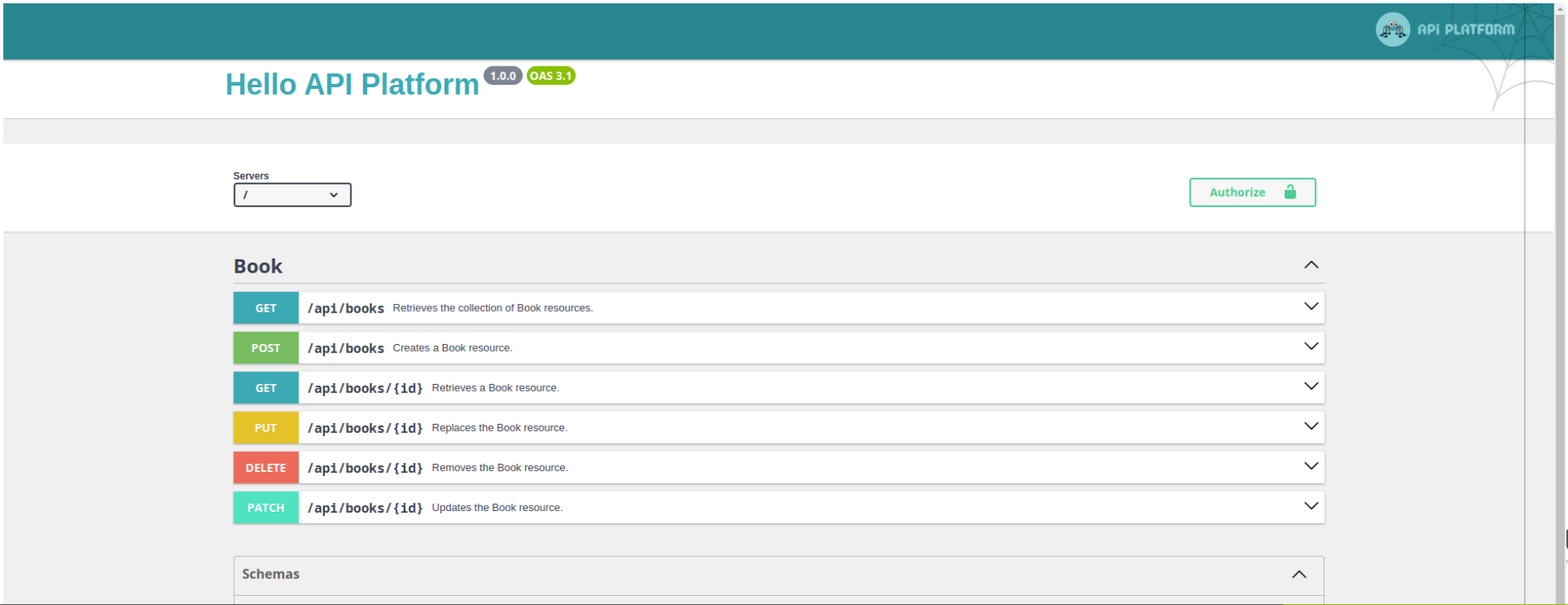
Création de la BDD

- **Configuration supplémentaire** :
  - Si nécessaire, configurer la base de données et d'autres paramètres dans le fichier `.env` ou `.env.local`.
  - Créer la base de données et appliquer les migrations :

```
1    symfony console doctrine:database:create
2    symfony console make:migration
3    symfony console doctrine:migrations:migrate
```

Routes Crées Automatiquement

- GET (un seul élément)
- PUT (remplacer un élément)
- DELETE (supprimer un élément)
- PATCH (modifier un champ spécifique d'un élément)
- GET (tous les éléments)
- POST (créer un nouvel élément)

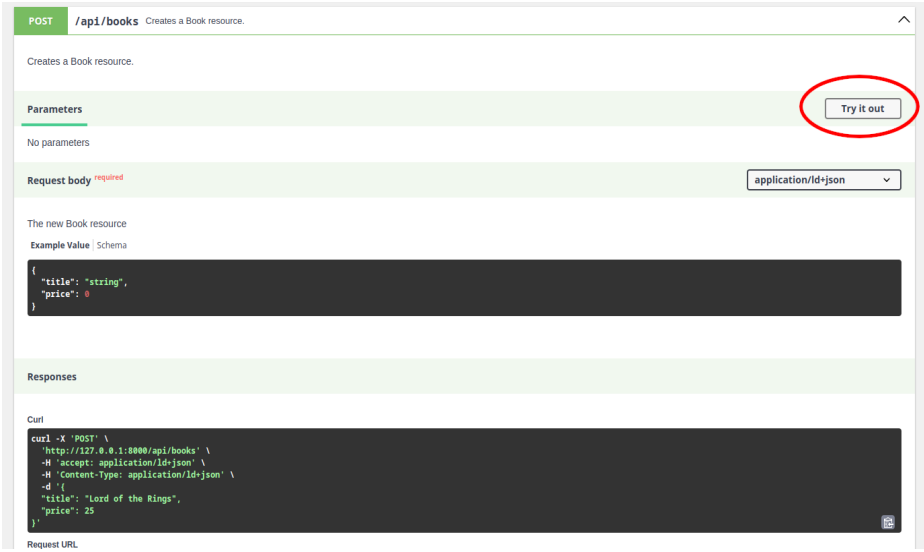


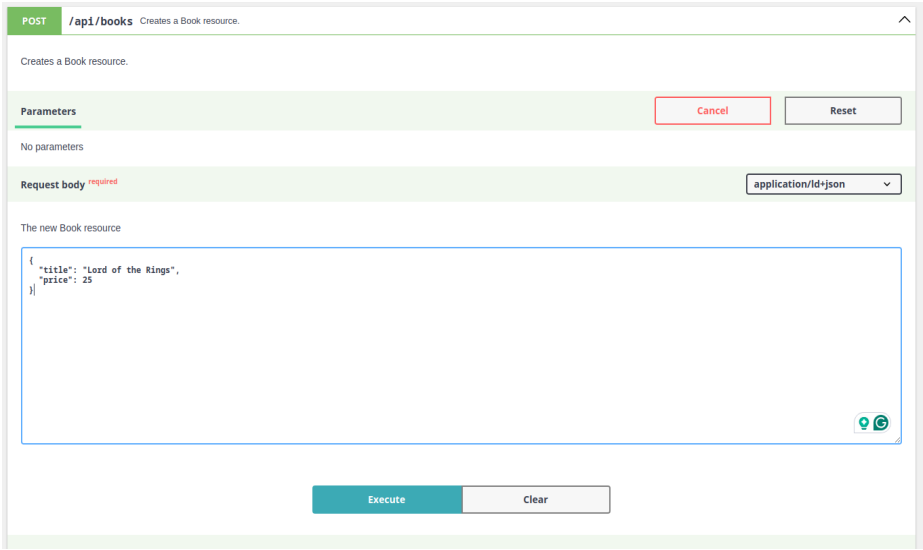
Compatibilité avec Doctrine

```php
// Annotations de Doctrine ORM et API Platform
use Doctrine\ORM\Mapping as ORM;
use ApiPlatform\Metadata\ApiResource;

#[ORM\Entity]
#[ApiResource]
class Book
{
    #[ORM\Id, ORM\GeneratedValue, ORM\Column(type: "integer")]
    private ?int $id = null;

    #[ORM\Column(type: "string")]
    private string $title;
}
```

Ajout d'une Instance de l'Entité avec OpenAPI 1/3

- Utilisez "Try It Out" pour POST





- Sinon : Utilisez la commande `curl` fournie :

```
curl -X 'POST' \
  'http://127.0.0.1:8000/api/books' \
  -H 'accept: application/ld+json' -H 'Content-Type: application/ld+json' \
  -d '{
  "title": "Lord of the Rings",
  "price": 25
}'
```

Ajout d'une Instance de l'Entité avec OpenAPI 2/3

- Utilisez "Try It Out" pour GET

```
{
  "@context": "/api/contexts/Book",
  "@id": "/api/books",
  "@type": "hydra:Collection",
  "hydra:totalItems": 1,
  "hydra:member": [
    {
      "@id": "/api/books/3",
      "@type": "Book",
      "id": 3,
      "title": "Lord of the Rings",
      "price": 25
    }
  ]
}
```

Ajout d'une Instance de l'Entité avec OpenAPI 3/3

- Requête SQL résultante :

```
$ sqlite3 var/data.db
sqlite> SELECT * FROM book;
1|Lord of the Rings|25
```

- GET pour récupérer les instances : `GET /api/books` et `GET /api/books/{id}`

```
$ curl -X 'GET' \
  'http://127.0.0.1:8000/api/books' \
  -H 'accept: application/ld+json'
```

```
$ curl -X 'GET' \
  'http://127.0.0.1:8000/api/books/1' \
  -H 'accept: application/ld+json'
```

Validation des Requêtes avec Symfony 1/2

Rappel : Utilisation du validateur de Symfony

- Installation : `composer require symfony/validator`

```php
// src/Entity/Book.php
namespace App\Entity;
use Symfony\Component\Validator\Constraints as Assert;

class Book {
    #[Assert\NotBlank]
    private string $title;
}
```

Validation des Requêtes avec Symfony 2/2

Attention !

- Ne garantit pas que l'assertion sera tout le temps vraie, il faudra appeler le validateur !

```
1 public function check(ValidatorInterface $validator): array {
2     $book = new Book(...);
3     $errors = $validator->validate($author);
4     return $errors;
5 }
```

Validation avec API Platform

- Normalement :
  - Assertions dans l'entité
  - Validation dans le contrôleur en fonction des requêtes
  - Envoie de la réponse normale ou d'une erreur par le contrôleur
- Avec API Platform :
  - Pas de contrôleur (API Platform = bundle qui joue le rôle du contrôleur)
  - API Platform fait la validation tout seul ( )

Liste des Contraintes Implémentées

Basic Constraints

- NotBlank
- Blank
- NotNull
- IsNull
- IsTrue
- IsFalse
- Type

Date Constraints

- Date
- DateTime
- Time
- Timezone

String Constraints

- Email
- ExpressionSyntax
- Length
- Url
- Regex
- Hostname
- Ip
- Cidr
- Json
- Uuid
- Ulid
- UserPassword
- NotCompromisedPassword
- PasswordStrength
- CssColor
- NoSuspiciousCharacters

Comparison Constraints

- EqualTo
- NotEqualTo
- IdenticalTo
- NotIdenticalTo
- LessThan
- LessThanOrEqual
- GreaterThan
- GreaterThanOrEqual
- Range
- DivisibleBy
- Unique

Number Constraints

- Positive
- PositiveOrZero
- Negative
- NegativeOrZero

File Constraints

- File
- Image

Financial and other Number Constraints

- Bic
- CardScheme
- Currency
- Luhn
- Iban
- Isbn
- Issn
- Isin

Choice Constraints

- Choice
- Language
- Locale
- Country

Other Constraints

- AtLeastOneOf
- Sequentially
- Compound
- Callback
- Expression
- When
- All
- Valid
- Cascade
- Traverse
- Collection
- Count
- UniqueEntity
- EnableAutoMapping
- DisableAutoMapping

https://symfony.com/doc/current/validation.html#constraints

Validation avec l'interface d'API Platform

- JSON montrant les erreurs de validation :

```
{
  "@context": "/contexts/ConstraintViolationList",
  "@type": "ConstraintViolationList",
  "hydra:title": "An error occurred",
  "hydra:description": "title: This value should not be blank.",
  "violations": [
    {
      "propertyPath": "title",
      "message": "This value should not be blank."
    }
  ]
}
```

- Utile pour afficher un message d'erreur dans un template Twig :

```
json_decode($json, true)["violations"];
```

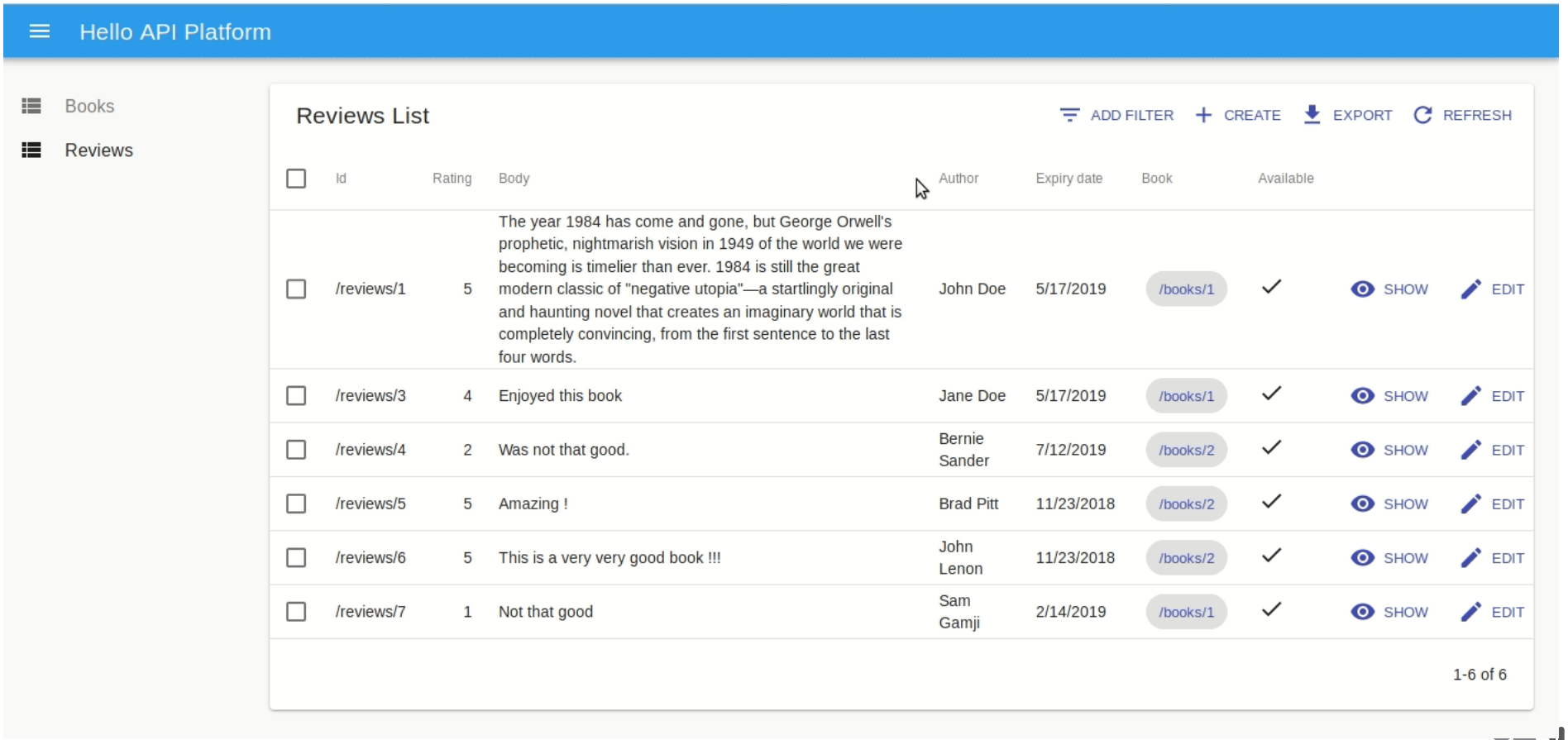
Debug du Validateur avec Symfony

- Utilisation de la commande pour debugger le validateur :

```
$ symfony console debug:validator 'App\Entity\Book'

App\Entity\Book
---------------

+----------+-------------+---------------+-----------------------------+
| Property | Name        | Groups        | Options                     |
+----------+-------------+---------------+-----------------------------+
| price    | GreaterThan | Default, Book | [                           |
|          |             |               |   "message" => "This value  |
|          |             |               | should be greater than      |
|          |             |               | {{ compared_value }}.",     |
|          |             |               |   "payload" =>              |
|          |             |               | null,                       |
|          |             |               |   "propertyPath" =>         |
|          |             |               | null,                       |
|          |             |               |   "value" => 5              |
|          |             |               | ]                           |
+----------+-------------+---------------+-----------------------------+
```

Interface d'Administration d'API Platform

- URL : `http://127.0.0.1:8000/admin`
- Code en React : `https://api-platform.com/docs/admin/`



Génération Automatique de Codes de Clients

- Génère le code front et les routes basé sur les métadonnées de l'API.
- Support : Next.js, Nuxt, Quasar, Vuetify, React, React Native, Vue.js
- APIs en Temps Réel et Asynchrones