

Symfony

API, Microservices et Événements

Maxime Puys

6 janvier 2025

Plan

1 Rappels sur les API et les Microservices

2 API en Symfony

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

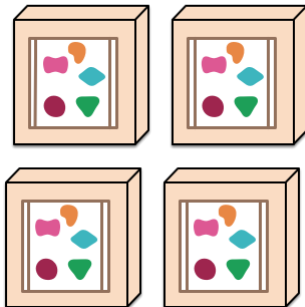


Figure – <https://martinfowler.com/articles/microservices.html>

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

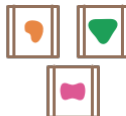
- ❑ Scalabilité améliorée
- ❑ Flexibilité dans l'utilisation de différentes technologies
- ❑ Cycles de déploiement plus rapides

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.

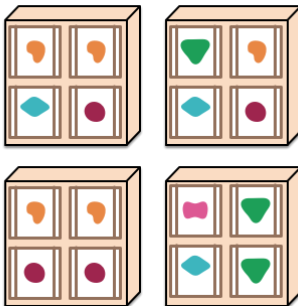


Figure – <https://martinfowler.com/articles/microservices.html>

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

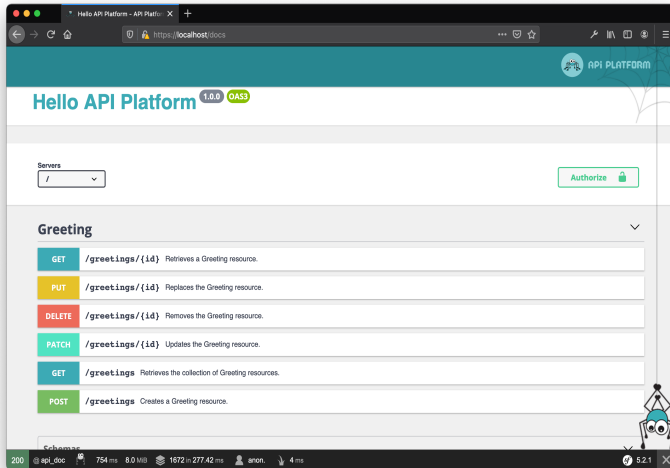
Présentation de API Platform 1/2

- 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 :

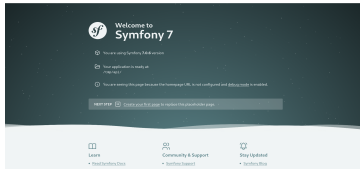
```
1  git clone https://github.com/api-platform/api-platform.git
2  cd api-platform
3  docker compose up -d
```

□ *Ou* Installation avec Symfony CLI :

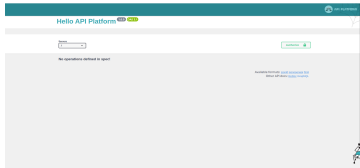
```
1  symfony new mon-api-platform
2  cd mon-api-platform
3  symfony composer require api
4  symfony server:start
```

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

```
1 $ symfony console make:entity
2
3 Class name of the entity to create or update (e.g. GrumpyPuppy):
4 > Book
5
6 Mark this class as an API Platform resource (expose a CRUD API for it) (yes/no)
7 > yes
8
9 created: src/Entity/Book.php
10 created: src/Repository/BookRepository.php
11
12 Entity generated! Now let's add some fields!
13 You can always add more fields later manually or by re-running this command.
14
15 New property name (press <return> to stop adding fields):
16 > title
```

Déclarer une Entité comme Ressource 2/2

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

3 use ApiPlatform\Metadata\ApiResource;

4 #[ApiResource]
5 class Book
6 {
7     private ?int $id = null;
8     private string $title;
9     // autres propriétés
10 }
```

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 Créées 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)

The screenshot displays the API Platform interface. At the top, there's a teal header with the 'API PLATFORM' logo and version '1.0.0 OAS 3.1'. Below this, the text 'Hello API Platform' is visible. The main content area shows a 'Book' resource with a list of endpoints. Each endpoint is represented by a colored square (GET: blue, POST: green, PUT: orange, DELETE: red, PATCH: teal) followed by the HTTP method, the endpoint path, and a brief description. The endpoints are: GET /api/books (Retrieves the collection of Book resources), POST /api/books (Creates a Book resource), GET /api/books/{id} (Retrieves a Book resource), PUT /api/books/{id} (Replaces the Book resource), DELETE /api/books/{id} (Removes the Book resource), and PATCH /api/books/{id} (Updates the Book resource). To the right of the endpoints is an 'Authorize' button with a lock icon. Below the endpoints, there's a 'Schemas' section with an upward arrow.

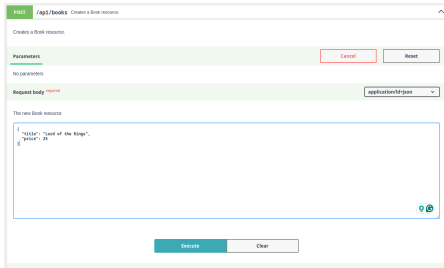
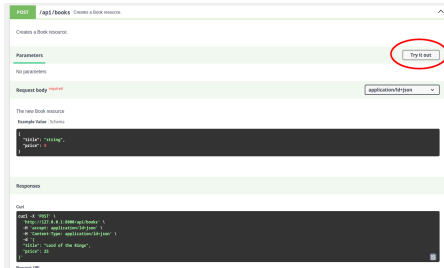
Method	Endpoint	Description
GET	/api/books	Retrieves the collection of Book resources.
POST	/api/books	Creates a Book resource.
GET	/api/books/{id}	Retrieves a Book resource.
PUT	/api/books/{id}	Replaces the Book resource.
DELETE	/api/books/{id}	Removes the Book resource.
PATCH	/api/books/{id}	Updates the Book resource.

Compatibilité avec Doctrine

```
1 // Annotations de Doctrine ORM et API Platform
2 use Doctrine\ORM\Mapping as ORM;
3 use ApiPlatform\Metadata\ApiResource;
4
5 #[ORM\Entity]
6 #[ApiResource]
7 class Book
8 {
9     #[ORM\Id, ORM\GeneratedValue, ORM\Column(type: "integer")]
10     private ?int $id = null;
11
12     #[ORM\Column(type: "string")]
13     private string $title;
14 }
```

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

□ Utilisez "Try It Out" pour POST



□ Sinon : Utilisez la commande `curl` fournie :

```
1 curl -X 'POST' \
2   'http://127.0.0.1:8000/api/books' \
3   -H 'accept: application/json' -H 'Content-Type: application/json' \
4   -d '{
5     "title": "Lord of the Rings",
6     "price": 25
7   }'
```

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

- Utilisez "Try It Out" pour GET

```
1 {  
2   "@context": "/api/contexts/Book",  
3   "@id": "/api/books",  
4   "@type": "hydra:Collection",  
5   "hydra:totalItems": 1,  
6   "hydra:member": [  
7     {  
8       "@id": "/api/books/3",  
9       "@type": "Book",  
10      "id": 3,  
11      "title": "Lord of the Rings",  
12      "price": 25  
13    }  
14  ]  
15 }
```

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

- Requête SQL résultante :

```
1 $ sqlite3 var/data.db
2 sqlite> SELECT * FROM book;
3 1|Lord of the Rings|25
```

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

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

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

Validation des Requêtes avec Symfony 1/2

Rappel : Utilisation du validateur de Symfony

□ Installation : `composer require symfony/validator`

```
1 // src/Entity/Book.php
2 namespace App\Entity;
3 use Symfony\Component\Validator\Constraints as Assert;
4
4 class Book {
5     #[Assert\NotBlank]
6     private string $title;
7 }
```

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
 - ▷ Envoi 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	String Constraints	Comparison Constraints	File Constraints	Other Constraints
• NotBlank • Blank • NotNull • IsTrue • IsFalse • Type	• Email • ExpressionSyntax • Length • Url • Regex • Hostname • Ip • Cidr • Json • Uuid • Ulid • UserPassword • NotCompromisedPassword • PasswordStrength • CssColor • NoSuspiciousCharacters	• EqualTo • NotEqualTo • IdenticalTo • NotIdenticalTo • LessThan • LessThanOrEqual • GreaterThan • GreaterThanOrEqual • Range • DivisibleBy • Unique	• File • Image	• AtLeastOneOf • Sequentially • Compound • Callback • Expression • When • All • Valid • Cascade • Traverse • Collection • Count • UniqueEntity • EnableAutoMapping • DisableAutoMapping
Date Constraints		Number Constraints	Financial and other Number Constraints	Choice Constraints
• Date • DateTime • Time • Timezone		• Positive • PositiveOrZero • Negative • NegativeOrZero	• Bic • CardScheme • Currency • Luhn • Iban • Isbn • Issn • Isin	

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

Validation avec l'interface d'API Platform

- JSON montrant les erreurs de validation :

```
1 {
2   "@context": "/contexts/ConstraintViolationList",
3   "@type": "ConstraintViolationList",
4   "hydra:title": "An error occurred",
5   "hydra:description": "title: This value should not be blank.",
6   "violations": [
7     {
8       "propertyPath": "title",
9       "message": "This value should not be blank."
10    }
11  ]
12 }
```

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

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

Debug du Validateur avec Symfony

- Utilisation de la commande pour debugger le validateur :

```
1 $ symfony console debug:validator 'App\Entity\Book'
```

```
2 App\Entity\Book
```

```
3 -----
```

```
4 +-----+-----+-----+-----+
5 | Property | Name          | Groups          | Options          |
6 +-----+-----+-----+-----+
7 | price    | GreaterThan   | Default, Book   | [
8 |           |               |                 |   "message" => "This value
9 |           |               |                 |   should be greater than
10 |           |               |                 |   {{ compared_value }}.",
11 |           |               |                 |   "payload" =>
12 |           |               |                 |   null,
13 |           |               |                 |   "propertyPath" =>
14 |           |               |                 |   null,
15 |           |               |                 |   "value" => 5
16 |           |               |                 | ]
17 +-----+-----+-----+-----+
```

Interface d'Administration d'API Platform

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

Hello API Platform

Books

Reviews

Reviews List

ADD FILTER

CREATE

EXPORT

REFRESH

☐	Id	Rating	Body	Author	Expiry date	Book	Available	
☐	/reviews/1	5	The year 1984 has come and gone, but George Orwell's prophetic, nightmarish vision in 1949 of the world we were becoming is timelier than ever. 1984 is still the great modern classic of "negative utopia"—a startlingly original and haunting novel that creates an imaginary world that is completely convincing, from the first sentence to the last four words.	John Doe	5/17/2019	/books/1	✓	SHOW EDIT
☐	/reviews/3	4	Enjoyed this book	Jane Doe	5/17/2019	/books/1	✓	SHOW EDIT
☐	/reviews/4	2	Was not that good.	Bernie Sander	7/12/2019	/books/2	✓	SHOW EDIT
☐	/reviews/5	5	Amazing !	Brad Pitt	11/23/2018	/books/2	✓	SHOW EDIT
☐	/reviews/6	5	This is a very very good book !!!	John Lenon	11/23/2018	/books/2	✓	SHOW EDIT
☐	/reviews/7	1	Not that good	Sam Gamji	2/14/2019	/books/1	✓	SHOW EDIT

1-6 of 6

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

The screenshot displays two views of a book management application. On the left, a web browser (QuickTime Player) shows a 'Book List' page with a 'Create' button and a table of books. On the right, a mobile app interface shows a 'List of Books' screen with a 'Add' button and a list of books in a card format.

id	isbn	title	description	author	publicationDate	reviews
/books/1	9799325573610	Accusamus nihil repellat vero omnis voluptates id amet et.	Qui recusandae totam nulla quam ipsam. Cupiditate sed natus debitis voluptas aut. Sit repudiandae esse perspicatis dignissimos error. Itaque quibusdam tempora velit porro ut velit soluta.	Maximillian Larson	2012-10-24T00:00:00+00:00	/reviews/1 /reviews/2 /reviews/3
/books/2	9798946066631	Sint dolore delectus enim ipsum inventore sed libero.	Qui suscipit a deserunt laudantium quibusdam. Nostrum soluta qui ipsam non ipsum. Reiciendis aperiam et fuga doloribus nisi. Cumque est ducimus temporibus modi	Annabell Tromp	1984-05-06T00:00:00+00:00	/reviews/4