Piste de Documentation API REST ET SYMFONY API PLATFORM

1) API REST

Une API REST se doit d'être sans état ou stateless en anglais. La communication entre le client et le serveur ne doit pas dépendre d'un quelconque contexte provenant du serveur. Ainsi, chaque requête doit contenir l'ensemble des informations nécessaires à son traitement. Cela permet au de traiter indifféremment les requêtes de plusieurs clients via de multiples instances de serveurs.

➢ Notion d'API

L'API (Application Programming Interface), est la partie du programme qu'on expose officiellement au monde extérieur. L'API est au développeur ce que l'User Interface est à l'utilisateur. Cette dernière permet d'entrer des données et de les récupérer la sortie d'un traitement. Initialement, une API regroupe un ensemble de fonctions ou méthodes, leurs signatures et ordre d'usage pour obtenir un résultat. La mise en place d'une API permet d'opérer une séparation des responsabilités entre le client et le serveur. Cette séparation permet donc une portabilité et évolutivité grandement améliorées.

➢ Protocole HTTP (fonctionnement, les verbes)

HTTP « Hypertext Transfer Protocol»est le protocole utilisé pour livrer des docu ments sur le Web. Ces données sont des fichiers HTML, mais peuvent également être d es fichiers d'image, de vidéo, etc. HTTP est un protocole de la couche Application qui utilise TCP pour transporter ces messages. ll utilise les fonctions de ce protocole de transport pour offrir un transfert fiable, ce qui permet à HTTP de se concentrer sur des aspects d'échange et de négociation des types de données échangées. L'une des forces de HTTP est sa simplicité. Son ensemble de commandes et de réponses est réduit, mais accompagné de divers en-têtes qui permettent aux clients et aux serveurs d'échanger des informations supplémentaires sur les données.

➢ Architecture Rest basé sur le protocole http

REST « Representational State Transfer » ou RESTful est un style d'architecture permettant de construire des applications (Web, Intranet, Web Service). Il s'agit d'un ensemble de conventions et de bonnes pratiques à respecter et non d'une technologie à part entière. L'architecture REST utilise les spécifications originelles du protocole HTTP, plutôt que de réinventer une surcouche (comme le font SOAP ou XML-RPC par exemple).

Cette architecture est base sur 5 regles.

➢ Contrainte Architecture de API REST

Pour considérer qu'une API est RESTful, 6 contraintes ont été proposées par Roy Fielding:

Client-Server – Un mode de communication avec séparation de rôles entre client et serveur.

Stateless Server Les requêtes doivent contenir toutes les informations nécessaires au traitement. Il ne doit pas y avoir une notion de session côté serveur. Cette contrainte est indispensable pour rendre une API scalable.

Cache La réponse du serveur doit être cacheable côté client.

Uniform interface La méthode de communication entre client et serveur doit être uniforme avec des ressources identifiables, représentables et auto-descriptives. Autrement dit, en vocabulaire HTTP, avec une URL et une réponse contenant un body et une entête.

Layered System Le système doit permettre le rajout de couches intermédiaires *(proxy server, firewall, CDN, etc ...)*.

Code-on-Demand Architecture (optionnelle) – L'architecture doit permettre d'exécuter du code côté client.

Réellement, beaucoup d'APIs se considérant RESTful ne valident pas la totalité de ces contraintes. C'est ainsi qu'un modèle de maturité a été proposé par Leonard Richardson afin de les classifier.

➢ API RESTFULL: Modèle de Maturité de Richardson

Le modèle de maturité de Richardson définit 4 niveaux de développement dans les APIs REST *(de 0 à 3)* dont le niveau 3 représente une API totalement RESTful.

La différenciation entre les 4 niveaux peut se baser sur 4 critères:

	Identifiants de ressource	Verbes HTTP	Codes retour	HATEOAS
Niveau 0	NON	POST	200	NON
Niveau 1	OUI	GET	200	NON
Niveau 2	OUI	GET, POST, PUT, DELETE	2XX, 4XX	NON
Niveau 3	OUI	GET, POST, PUT, DELETE	2XX, 4XX	OUI

Les identifiants de ressource : REST est une architecture orientée ressources où chaque ressource est accessible via un identifiant unique *(URI)*.

Les Verbes http Fondé sous le principe KISS *(Keep it simple, stupid)*, REST a su profiter des verbes HTTP pour définir des spécifications d'échange facilement intégrables à tout système.

- GET Permet de récupérer les données d'une ou plusieurs entités dans une ressource. Il est possible de filtrer par des paramètres dans l'url.

- POST Permet de rajouter des entités à une ressource. Les données sont envoyées dans le corps de la requête. Par défaut, une requête POST intègre le

format Content-Type:application/x-www-form-urlencoded. Ce format est déconseillé avec les APIs REST car il ne correspond pas aux formats de réponse utilisés *(JSON, XML)*.

- PUT permet de modifier les données d'une entité dans une ressource.
  L'url doit indiquer l'identifiant de l'entité. Si l'identifiant est inexistant, l'entité devrait être créée.

- PATCH (OPTIONNEL) Permet de modifier en partie les données d'une entité dans une ressource.
  Ce verbe ne fait pas partie des opérations de base pour la persistance des données *(CRUD)* mais pourrait être considéré comme une extension au verbe PUT lui permettant de simplifier les mises à jour partielles.
- DELETE Permet de supprimer une entité dans une ressource.

LES CODES RETOUR HTTP D'API REST Les codes retours permettent de donner un statut aux réponses renvoyées par l'API.

HATEOAS *« Hypermedia As The Engine Of Application State »* est une contrainte sur la présence de liens au niveau des ressources pour permettre une navigation exploratoire. Malgré l'omniprésence de cette contrainte dans le Web *(liens hypertexte)*, elle est rarement intégrée dans les APIs REST car beaucoup considèrent qu'une bonne documentation est suffisante pour mieux explorer une API. D'ailleurs, plusieurs grands noms d'Internet dont Facebook, Twitter et Amazon n'ont pas pris en considération ce niveau 3.
Toutefois, HATEOAS est devenu un terme définissant une API complètement mature.

2) Notion de Sérialisation

➢ Notion de Decode et Encode

Le processus de codage convertit une donnée en une autre représentation (un autre codage, si vous voulez être technique à ce sujet

➢ Notion de sérialisation et désérialisation

La sérialisation est le processus de conversion d'un objet en un flux d'octets pour stocker l'objet ou le transmettre à la mémoire, à une base de données, ou dans un fichier. Son principal objectif est d'enregistrer l'état d'un objet afin de pouvoir le recréer si nécessaire. La désérialisation est, par conséquent, le rétablissement à sa forme première d'un objet sérialisé.

➢ Notion de normalisation et dénormalisation

La normalisation et la dénormalisation sont les méthodes utilisées dans les bases de données. Les termes sont différenciables lorsque la normalisation est une technique permettant de minimiser les anomalies d'insertion, de suppression et de mise à jour en éliminant les données redondantes. Alors que la dénormalisation est le processus inverse de la normalisation dans lequel la redondance est ajoutée aux données afin d'améliorer les performances de l'application spécifique et l'intégrité des données.
La normalisation évite le gaspillage d'espace disque en minimisant ou en éliminant la redondance.

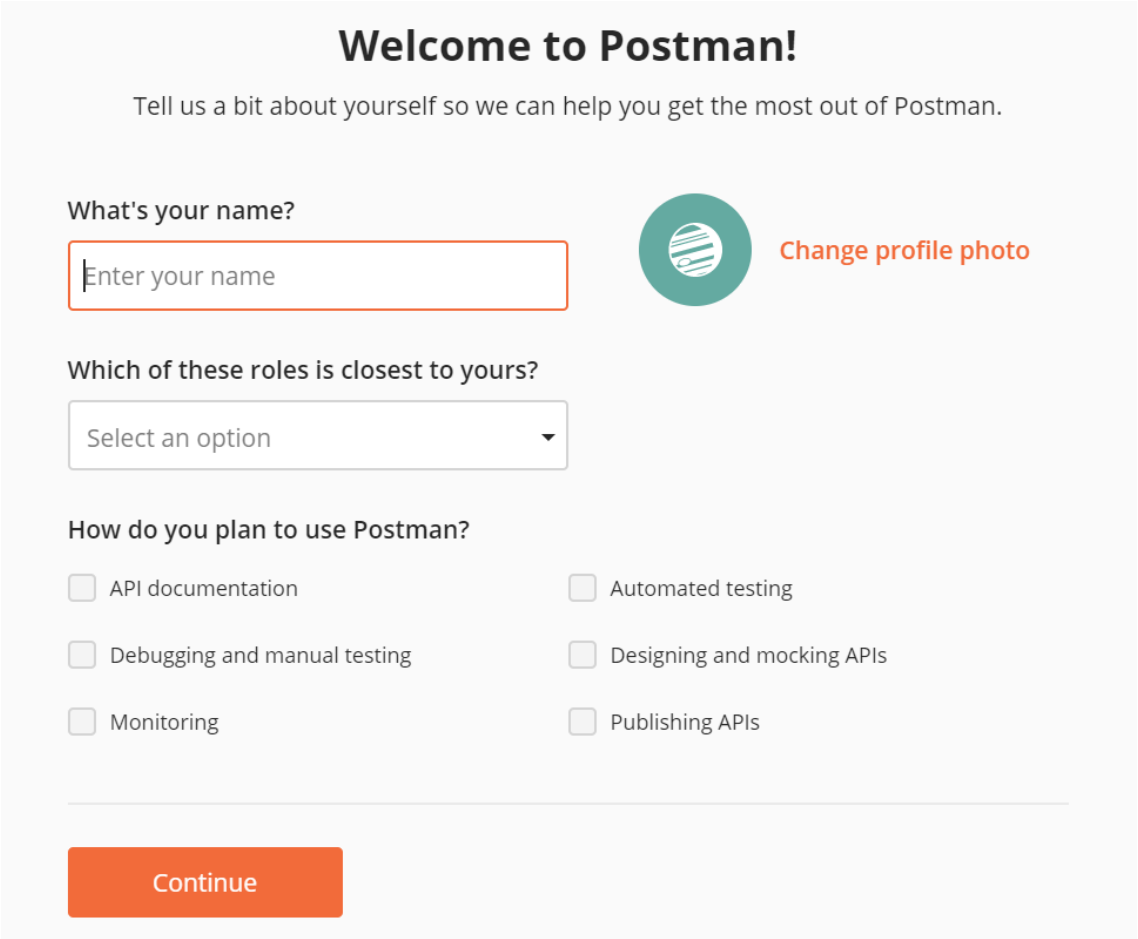
3) Notion de Fixtures

Les fixtures sont utilisés pour charger de « fausses » ensemble de données dans une base de données qui peut ensuite être utilisées pour des testes ou pour fournir des données intéressantes au cours du développement de notre application. En symfony on utilise cette commande pour générer des fixtures.

```
> composer require --dev orm-fixtures
```

4) Installation Postman pour tester les API

PostMan est un logiciel permettant de tester nos API mais aussi de les enregistrer c'est-à-dire qu'au lieu de les tester sur le navigateur et de découvrir vos informations non formaliser, nous allons utiliser postman pour tester les requêtes, les voir sous le bon format et aussi enregistrer la requête.
Pour installer PostMan , cliquer sur télécharger l'application et choisissez votre architecture d'ordinateur.
Une fois le téléchargement fini, lancer l'installateur, l'installation devrait se faire automatiquement et PostMan vas se lancer tout seul.



5) Bundle API PlatForm

- Installation

Bundle API platform peut être installer dans un projet existant a l'aide de composer avec la commande :

```
$ composer require --dev api-platform/schema-generator
```

- Référence circulaire
- Api Auto-decouvrable
- Notion de Tri
- Notion de Filter(RangeFilter,SearchFilter, OrderFilter, PropertyFilter)

API platform fournis un système générique pour appliquer des filtres et des critères de trie sur les collections. Les filtres peuvent être personnalisé avec une implémentation des interfaces fournis par la bibliothèque. Les filtres sont des services qui peuvent être lié à une ressource de deux manières.

```php
<?php
// api/src/Entity/Offer.php

namespace App\Entity;

use ApiPlatform\Core\Annotation\ApiFilter;
use ApiPlatform\Core\Annotation\ApiResource;
use ApiPlatform\Core\Bridge\Doctrine\Orm\Filter\DateFilter;

/**
 * @ApiResource
 * @ApiFilter(DateFilter::class, properties={"dateProperty"})
 */
class Offer
{
    // ...
}
```

➢ Pagination

API Platform prend en charge nativement les collections paginées. La pagination est activée par défaut pour toutes les collections. Chaque collection contient 30 éléments par page. L'activation de la pagination et le nombre d'élément par page peuvent être configurés du cote serveur et cote client via le paramètre GET.

```yaml
# api/config/packages/api_platform.yaml
api_platform:
    collection:
        pagination:
            page_parameter_name: _page
```

La pagination est active par défaut mais peut aussi être désactivé. Cette image l'illustre.

```yaml
# api/config/packages/api_platform.yaml
api_platform:
    collection:
        pagination:
            enabled: false
```

➢ Notion de Api SubRessource

Une subRessource est une collection ou un element qui appartient à un autre ressource

➢ itemsOperation

- CollectionsOperation

6) Gestion de l'authentification

- Notion JWT

Un jeton JWT est une chaîne de caractères que l'on va envoyer à chaque requête que l'on souhaite effectuer auprès d'une API afin de s'authentifier. Il contient toutes les informations nécessaires à notre identification (n'oubliez pas que nous sommes dans un système Stateless et que le serveur ne stocke pas par exemple en session nos informations d'identifications !).

- Architecture du JWT

Ce jeton est composé de trois parties :

- L'en-tête, qui contient l'algorithme utilisé pour la signature du jeton,

- La charge utile du jeton, qui contient nos données utilisateurs, par exemple, notre nom d'utilisateur, ainsi que d'autres informations utiles telles que la date d'expiration du jeton, sa date de création, etc,

- Et pour finir, la signature du jeton. Les deux premières parties sont des objets JSON encodés en base 64 et la dernière partie est le résultat de l'encodage des deux premières parties avec l'algorithme défini dans l'en-tête.

- Notion Provider

Le monde des API commence par un individu, une entreprise ou une organisation, avec une sorte de contenu, de données ou d'autres ressources numériques, souhaitant les rendre disponibles, sur Internet, d'une manière lisible par machine, afin que d'autres puissent créer des sites, et applications autour d'elle. Même si les API utilisent Internet pour la communication, cela ne signifie pas qu'elles sont accessibles à tous. La fourniture d'API est très similaire à la fourniture de sites Web, vous pouvez contrôler quels utilisateurs y ont accès, où ils peuvent aller et ce qu'ils peuvent faire - les API sont uniquement destinées à d'autres sites Web, applications Web et développeurs mobiles.

- Firewall

Représente une ressource de règle de pare-feu.

Les règles de pare-feu autorisent ou refusent le trafic entrant et sortant du trafic de vos instances.

Représentation JSON

```
{
    "id" : chaîne ,
    "creationTimestamp" : chaîne ,
    "nom" : chaîne ,
    "description" : chaîne ,
```

```
"réseau" : chaîne ,
"priorité" : entier ,
"sourceRanges" : [ chaîne ] ,
"destinationRanges" : [ chaîne ] ,
"sourceTags" : [ chaîne ] ,
"targetTags" :[ chaîne ] ,
"sourceServiceAccounts" : [ chaîne ] , "targetServiceAccounts" : [
chaîne ] ,
"autorisé" : [
        {
                "IPProtocol" : chaîne ,
                "ports" : [ chaîne ]
        }
] ,
"refusé" : [
        {
                "IPProtocol" : chaîne ,
                "ports" : [ chaîne ]
        }
] ,
"direction" : enum ,
"logConfig" :{
        "enable" : boolean
} ,
"disabled" : boolean ,
"selfLink" : string ,
"kind" : string
}
```

Des champs

id string (fixed64 format)
[Sortie uniquement] L'identifiant unique de la ressource. Cet identifiant est défini par le serveur.
creationTimestamp string
[Sortie uniquement] Horodatage de création au format texte RFC3339 .

name string
Nom de la ressource; fourni par le client lors de la création de la ressource. Le nom doit être composé de 1 à 63 caractères et conforme à la RFC1035 . Plus précisément, le nom doit comporter de 1 à 63 caractères et correspondre à l'expression régulière ` az ?. Le premier caractère doit être une lettre minuscule et tous les caractères suivants (à l'exception du dernier caractère) doivent être un t lettre minuscule ou un chiffre. Le dernier caractère doit être une lettre minuscule ou un chiffre.
description string
Une description facultative de cette ressource. Fournissez ce champ lorsque vous créez la ressource.
network string
URL de la ressource réseau pour cette règle de pare-feu. S'il n'est pas spécifié lors de la création d'une règle de pare-feu, le defaultréseau est utilisé:
global / réseaux / par défaut Si vous choisissez de spécifier ce champ, vous pouvez spécifier le réseau comme partielle. Par exemple, les URL suivantes sont toutes valides:
https://www.googleapis.com/compute/v1/projects/myproject/global/networks/my network
projects/myproject/global/networks/my-network
global/networks/default
priority integer
Priorité pour cette règle. Il s'agit d'un entier entre 0et 65535, tous deux inclus. La valeur par défaut est 1000. Les priorités relatives déterminent quelle règle prend effet si plusieurs règles s'appliquent. Des valeurs plus faibles indiquent une priorité plus élevée. Par exemple, une règle avec priorité 0a une priorité plus élevée qu'une règle avec priorité 1. Les règles DENY ont priorité sur les règles ALLOW si elles ont une priorité égale. Notez que les réseaux VPC ont des règles implicites avec une priorité de 65535. Pour éviter les conflits avec les règles implicites, utilisez un numéro de priorité inférieur à 65535.
sourceRanges[] string
Si des plages source sont spécifiées, la règle de pare-feu s'applique uniquement au trafic qui a une adresse IP source dans ces plages. Ces plages doivent être exprimées au format deux sourceRangeset sourceTagspeuvent être définis. Si les deux champs sont définis, la règle s'applique au trafic qui a une adresse IP source dans sourceRangesOU une adresse IP source à partir d'une ressource avec une balise correspondante répertoriée dans le sourceTags connexion n'a pas besoin de correspondre aux deux champs pour que la règle s'applique. est pris en charge.
destinationRanges[] string
Si des plages de destination sont spécifiées, la règle de pare-feu s'applique uniquement au trafic dont l'adresse IP de destination se trouve dans ces plages. Ces plages doivent être exprimées au format CIDR . Seul IPv4 est pris en charge.

sourceTags[] string
Si des balises source sont spécifiées, la règle de pare-feu s'applique uniquement au trafic avec des adresses IP source qui correspondent aux interfaces réseau principales des instances de VM qui ont la balise et se trouvent dans le même réseau VPC. Les balises source ne peuvent pas être utilisées pour contrôler le trafic vers l'adresse IP externe d'une instance, elles s'appliquent uniquement au trafic entre les instances du même réseau virtuel. Parce que les balises sont associées à des instances, pas à des adresses IP. Un ou les deux sourceRangeset sourceTagspeuvent être définis. champs sont définis, le pare-feu s'applique au trafic qui a une adresse IP source dans sourceRangesOU une IP source à partir d'une ressource avec une balise correspondante répertoriée dans le sourceTagschamp. La connexion n'a pas besoin de correspondre aux deux champs pour que le pare-feu s'applique.
targetTags[] string
Une liste de balises qui contrôle les instances auxquelles la règle de pare-feu s'applique. S'ils targetTagssont spécifiés, la règle de pare-feu s'applique uniquement aux instances du réseau VPC qui ont l'une de ces balises. Si aucun targetTagsn'est spécifié, la règle de pare s'applique à toutes les instances du réseau spécifié.
Des champs
sourceServiceAccounts[] string
Si des comptes de service source sont spécifiés, les règles de pare-feu s'appliquent uniquement au trafic provenant d'une instance avec un compte de service dans cette liste. Les comptes de service source ne peuvent pas être utilisés pour contrôler le trafic vers l'adresse IP externe d'une instance car les comptes de service sont associés à une instance et non à une adresse IP. sourceRanges être réglé en même temps que sourceServiceAccounts. Si les deux sont définis, le pare s'applique au trafic qui a une adresse IP source dans le sourceRangesOU une IP source qui appartient à une instance avec un compte de service répertorié dans sourceSer connexion n'a pas besoin de correspondre aux deux champs pour que le pare-feu s'applique. sourceServiceAccountsne peut pas être utilisé en même temps que sourceTagsou targetTags.
targetServiceAccounts[] string
Une liste de comptes de service indiquant des ensembles d'instances situées dans le réseau qui peuvent établir des connexions réseau comme spécifié dans allowed[]. targetServiceAccountsne peut pas être utilisé en même temps que targetTagsou sourceTags. Si ni l'un targetServiceAccountsni l' autre ne targetTagssont spécifiés, la règle de pare-feu s'applique à toutes les instances sur le réseau spécifié.
allowed[] object
La liste des règles ALLOW spécifiées par ce pare-feu. Chaque règle spécifie un tuple de protocole et de plage de ports qui décrit une connexion autorisée.
allowed[].IPProtocol string

Protocole IP auquel cette règle s'applique. Le type de protocole est requis lors de la création d'une règle de pare-feu. Cette valeur peut être soit l' une des chaînes de protocole bien conn (tcp, udp, icmp, esp, ah, ipip, sctp) ou le numéro de protocole IP.

allowed[].ports[] string

Une liste facultative des ports auxquels cette règle s'applique. Ce champ s'applique uniquement au protocole UDP ou TCP. Chaque entrée doit être soit un entier soit une plage. Si elle n'est pas spécifiée, cette règle s'applique aux connexions via n'importe quel port.

Entrées comprennent par exemple: ["22"], ["80","443"]et ["12345-12349"]

Des champs

denied[] object

La liste des règles DENY spécifiées par ce pare-feu. Chaque règle spécifie un tuple de protocole et de plage de ports qui décrit une connexion refusée.

denied[].IPProtocol string

Protocole IP auquel cette règle s'applique. Le type de protocole est requis lors de la création règle de pare-feu. Cette valeur peut être soit l' une des chaînes de protocole bien connues suivantes (tcp, udp, icmp, esp, ah, ipip, sctp) ou le numéro de protocole IP.

denied[].ports[] string

Une liste facultative des ports auxquels cette règle s'applique. Ce champ s'applique uniquement au protocole UDP ou TCP. Chaque entrée doit être soit un entier soit une plage. Si elle n'est pas spécifiée, cette règle s'applique aux connexions via n'importe quel port.

Entrées comprennent par exemple: ["22"], ["80","443"]et ["12345-12349"]

direction enum

Direction du trafic auquel s'applique ce pare-feu, soit INGRESSou EGRESS. La valeur par défaut est INGRESS. Pour le INGRESStrafic, vous ne pouvez pas spécifier le destinationRanges pour le EGRESStrafic, vous ne pouvez pas spécifier les champs sourceRangesou

logConfig object

Ce champ indique les options de journalisation pour une règle de pare-feu particulière. journalisation est activée, les journaux seront exportés vers Cloud Logging.

logConfig.enable boolean

Ce champ indique s'il faut activer la journalisation pour une règle de pare-feu particulière.

disabled boolean

Indique si la règle de pare-feu est désactivée. Lorsqu'il est défini sur true, la règle de pare appliquée et le réseau se comporte comme s'il n'existait pas. Si cela n'est pas spécifié, la règle de pare-feu sera activée.

selfLink string

Des champs

[Sortie uniquement] URL définie par le serveur pour la ressource.

kind string
[Sortie uniquement] Type de ressource. Toujours compute#firewallpour les règles de pare
Les méthodes
delete Supprime le pare-feu spécifié.
get Renvoie le pare-feu spécifié.
insert Crée une règle de pare-feu dans le projet spécifié en utilisant les données incluses dans la demande.
list Récupère la liste des règles de pare-feu disponibles pour le projet spécifié.
patch Met à jour la règle de pare-feu spécifiée avec les données incluses dans la demande.
update Met à jour la règle de pare-feu spécifiée avec les données incluses dans la demande.

➢ Encoder

L'API Encoding fournit un mecanisme de traitement de texte dans plusieurs character encodings, incluant les encodings non-UTF-8.

L'API fournit quatres interfaces: TextDecoder, TextEncoder, TextDecoderStream et TextEncoderStream.

➢ Access control

L'accès aux API est le processus visant à garantir que les appels avec des connexions authentifiées peuvent entrer dans les API. Une passerelle API est au cœur d'une solution de gestion d'API. Les passerelles garantissent que les appels API sont traités de manière appropriée. Ils gèrent également le style de vie des API. De plus, les produits API sont un bon moyen de contrôler l'accès à un ensemble spécifique de ressources.
Les API Google sont un exemple d'accès aux API. Ces API fournissent un apprentissage automatique et des analyses. Ils permettent également d'accéder aux données utilisateur lorsque l'autorisation est accordée. Ces interfaces de programmation d'applications permettent aux services Google tels que Google Maps, la recherche, la traduction et Gmail de communiquer avec les services Google.
La solution de gestion d'API d'Apigee vous permet d'autoriser ou de refuser l'accès à vos API, en utilisant des adresses IP spécifiques. Étant donné que les produits API sont le mécanisme central d'autorisation et de contrôle d'accès à vos API, Apigee aide à leur fournir des clés API.
Lorsqu'une application tente d'accéder à un produit API, l'autorisation est appliquée par Apigee lors de l'exécution. À l'aide des mesures de contrôle d'accès de l'API Apigee, vous pouvez:
- Autoriser l'accès à une ressource API particulière - Vérifiez et assurez-vous que l'application demandeuse n'a pas dépassé le quota autorisé - Faites correspondre les portées OAuth avec celles associées aux jetons d'accès donnés par l'application.

Apigee vous couvre si vous êtes architecte de sécurité, développeur, ingénieur d'exploitation ou propriétaire de produit API. Découvrez comment Apigee gère l'accès aux API en téléchargeant notre eBook.

7) API Platform et Evenements Doctrine

➢ Evenements Doctrine

Dans certains cas, vous pouvez avoir besoin d'effectuer des actions juste avant ou juste après la création, la mise à jour ou la suppression d'une entité. Par exemple, si vous stockez la date d'édition d'une annonce, à chaque modification de l'entitéAdvert il faut mettre à jour cet attribut juste avant la mise à jour dans la base de données. Ces actions, vous devez les faire à chaque fois. Cet aspect systématique a deux impacts. D'une part, cela veut dire qu'il faut être sûrs de vraiment les effectuer à chaque fois pour que votre base de données soit cohérente. D'autre part, cela veut dire qu'on est bien trop fainéants pour se répéter !

C'est ici qu'interviennent les évènements Doctrine. Plus précisément, vous les trouverez sous le nom de callbacks du cycle de vie (lifecycle en anglais) d'une entité. Un callback est une méthode de votre entité, et on va dire à Doctrine de l'exécuter à certains moments.

On parle d'évènements de « cycle de vie », car ce sont différents évènements que Doctrine déclenche à chaque moment de la vie d'une entité : son chargement depuis la

base de données, sa modification, sa suppression, etc. On en reparle plus loin, je vous dresserai une liste complète des évènements et de leur utilisation.

➢ Methodes PrePersist, PreUpdate, PreRemove, PreSubscriber

PrePersist

L'évènement PrePersist se produit juste avant que l'EntityManager ne persiste effectivement l'entité. Concrètement, cela exécute le callback juste avant un $em->persist($entity). Il ne concerne que les entités nouvellement créées. Du coup, il y a deux conséquences : d'une part, les modifications que vous apportez à l'entité seront persistées en base de données, puisqu'elles sont effectives avant que l'EntityManager n'enregistre l'entité en base. D'autre part, vous n'avez pas accès à l'id de l'entité si celui-ci est autogénéré, car justement l'entité n'est pas encore enregistrée en base de données, et donc l'id pas encore généré.

PreUpdate

L'évènement preUpdate se produit juste avant que l'EntityManager ne modifie une entité. Par modifiée, j'entends que l'entité existait déjà, que vous y avez apporté des modifications, puis un $em->flush(). Le callback sera exécuté juste avant le flush(). Attention, il faut que vous ayez modifié au moins un attribut pour que l'EntityManager génère une requête et donc déclenche cet évènement. Vous avez accès à l'id autogénéré (car l'entité existe déjà), et vos modifications seront persistées en base de données.

PreRemove

L'évènement PreRemove se produit juste avant que l'EntityManager ne supprime une entité, c'est-à-dire juste avant un $em->flush() qui précède un $em->remove($entite). Attention, soyez prudents dans cet évènement, si vous souhaitez supprimer des fichiers liés à l'entité par exemple, car à ce moment l'entité n'est pas encore effectivement supprimée, et la suppression peut être annulée en cas d'erreur dans une des opérations à effectuer dans le flush().

8) API Platform et Controllers personnalisés

L'utilisation de contrôleurs personnalisés avec API Platform est déconseillée . De plus, GraphQL n'est pas pris en charge . Pour la plupart des cas d'utilisation, de meilleurs points d'extension, fonctionnant à la fois avec REST et GraphQL, sont disponibles .

La plate-forme API peut tirer parti du système de routage Symfony pour enregistrer les opérations personnalisées liées aux contrôleurs personnalisés. Ces contrôleurs personnalisés peuvent être n'importe quel contrôleur Symfony valide , y compris les

contrôleurs Symfony standard étendant la Symfony\Bundle\FrameworkBundle\Controller\AbstractController classe d'assistance.

Cependant, API Platform recommande d'utiliser des classes d'actions au lieu des contrôleurs Symfony classiques. En interne, API Platform implémente le modèle ActionDomain-Responder (ADR), un raffinement spécifique au Web de MVC .

La distribution d'API Platform facilite également la mise en œuvre du modèle ADR: il enregistre automatiquement les classes d'actions stockées en api/src/Controllertant que services câblés automatiquement.

Grâce à la fonctionnalité de câblage automatique du conteneur Symfony Dependency Injection, les services requis par une action peuvent être indiqués dans son constructeur, il sera automatiquement instancié et injecté, sans avoir à le déclarer explicitement.

Dans les exemples suivants, l' GETopération intégrée est enregistrée ainsi qu'une opération personnalisée appelée post_publication.

Par défaut, API Platform utilise la première GETopération définie dans itemOperationspour générer l'IRI d'un élément et la première GETopération définie dans collectionOperationspour générer l'IRI d'une collection.

Si vous créez une opération personnalisée, vous souhaiterez probablement la documenter correctement. Voir la partie OpenAPI de la documentation pour le faire.

Commençons par créer votre opération personnalisée:

```
<?php
// api/src/Controller/CreateBookPublication.php

namespace App\Controller;

use App\Entity\Book;

class CreateBookPublication
{
    private $bookPublishingHandler;

    public function __construct(BookPublishingHandler $bookPublishingHandler)
    {
        $this->bookPublishingHandler = $bookPublishingHandler;
    }

    public function __invoke(Book $data): Book
    {
        $this->bookPublishingHandler->handle($data);

        return $data;
    }
}
```

Cette opération personnalisée se comporte exactement comme l'opération intégrée: elle renvoie un document JSON-LD correspondant à l'ID passé dans l'URL.

Ici, nous considérons que le câblage automatique est activé pour les classes de contrôleur (valeur par défaut lors de l'utilisation de la distribution API Platform). Cette action sera automatiquement enregistrée en tant que service (le nom du service est le même que le nom de la classe:) App\Controller\CreateBookPublication.

La plate-forme API récupère automatiquement l'entité PHP appropriée à l'aide du fournisseur de données, puis désérialise les données utilisateur, POSTet PUTdemande et met à jour l'entité avec les données fournies par l'utilisateur.

Attention: le __invoke() paramètre de méthode DOIT être appelé$data , sinon, il ne sera pas rempli correctement!

Les services ($bookPublishingHandlerici) sont automatiquement injectés grâce à la fonction de câblage automatique. Vous pouvez taper n'importe quel service dont vous avez besoin et il sera également câblé automatiquement.

La __invoke méthode de l'action est appelée lorsque l'itinéraire correspondant est atteint. Il peut renvoyer soit une instance de Symfony\Component\HttpFoundation\Response(qui sera immédiatement affichée au client par le noyau Symfony), soit, comme dans cet exemple, une instance d'une entité mappée en tant que ressource (ou une collection d'instances pour les opérations de collecte). Dans ce cas, l'entité passera par tous les écouteurs d'événements intégrés d'API Platform. Il sera automatiquement validé, conservé et sérialisé en JSON-LD. Ensuite, le noyau Symfony enverra le document résultant au client.

Le routage n'a pas encore été configuré car nous l'ajouterons au niveau de la configuration des ressources:

```
<?php

// api/src/Entity/Book.php


use ApiPlatform\Core\Annotation\ApiResource;

use App\Controller\CreateBookPublication;
```

```
/**
 * @ApiResource(itemOperations={
 *     "get",
 *     "post_publication"={
 *         "method"="POST",
 *         "path"="/books/{id}/publication",
 *         "controller"=CreateBookPublication::class,
 *     }
 * })
 */
class Book
{
    //...
}
```

Il est obligatoire de définir les attributs method, pathet controller. Ils permettent à l'API Platform de configurer respectivement le chemin de routage et le contr