

e-application - 2020

1

3. Symphony 5

Matthieu Kowalczyk

Ingénieur en technologies de l'information @CGI

Nathan Levy

Ingénieur en technologies de l'information @CGI



matthieu.kowalczyk@cgi.com
nathan.levy@cgi.com

CGI



**UNIVERSITÉ
DE MONTPELLIER**

Agenda

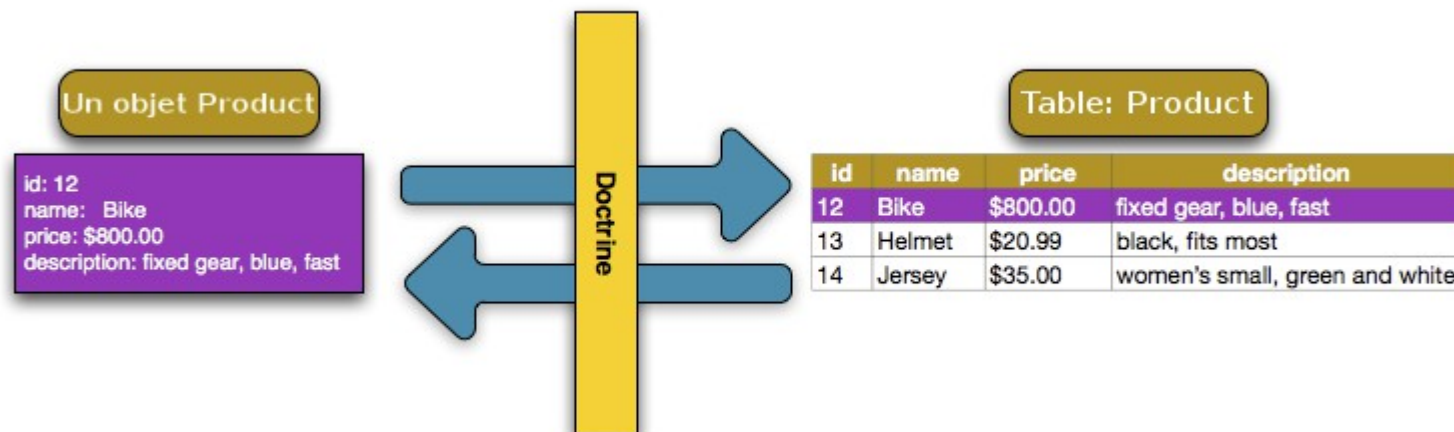
1. L'ORM de Sf5 : Doctrine
2. Les APIs
3. TP n°4
4. La traduction dans Sf5
5. Déployer une application Sf5 dans le cloud
6. TP n°5
7. Installation d'un bundle tiers
8. TP n°6

1 - L'ORM de Sf5 : Doctrine

- Le concept d'ORM
- Configuration de Doctrine
- L'entité
- Utiliser Doctrine
- Aller plus loin

Qu'est ce que l'ORM

Un mapping objet-relationnel (en anglais object-relational mapping ou ORaM) est une technique de programmation informatique qui crée l'illusion d'une base de données orientée objet à partir d'une base de données relationnelle en définissant des correspondances entre cette base de données et les objets du langage utilisé. On pourrait le désigner par « correspondance entre monde objet et monde relationnel » (© Wikipedia).



Doctrine & ...

- A partir de Sf4, Symfony n'est compatible qu'avec 1 seul ORM : **Doctrine**.
- **Propel** était l'ORM historique de Symfony mais il a laissé sa place à Doctrine, ORM par défaut de Symfony depuis la 1.2 Puis seul ORM de Sf4.

Configuration de la base de données

- Jusqu'à présent nous avons installé Sf5 sans nous préoccuper de la persistance des données au sein d'une base de données.
- Tout d'abord, le prérequis pour travailler avec Doctrine est d'être sur une base de données utf8_general_ci.
- La configuration de l'accès à la base de données se fait via le fichier .env et le fichier config/package/doctrine.yaml :

Configuration de la DB sqlite

- Pour faciliter nos dev et le portabilité de notre application, nous pouvons utiliser la base de données fichier **SQLite**.
- Pour manipuler une base SQLite, on peut utiliser l'extension Firefox **SQLite Manager**.
- Pour cela changer la variable `DATABASE_URL` du fichier `.env` :
 - `DATABASE_URL=sqlite:///kernel.project_dir%/var/blog.db`
- Une fois configurer, créez votre base avec :

`php bin/console doctrine:database:create`

Création d'une Entity

- On crée une classe dans le namespace App\Entity qui va correspondre une table dans notre BDD.
- On ajoute des attributs à cette classe qui vont correspondre aux champs de notre table
- Ensuite on rajoute les information de mapping. Il existe plusieurs méthode pour cela (YAML, XML) mais ce que je vous conseil est d'utiliser les annotations
- La listes des types de champs est disponible [ici](#).
- Les attributs sont en private, il faut donc ajouter un getter et un setter pour chaque attribut.

```
// src/Entity/Product.php
namespace App\Entity;

use App\Repository\ProductRepository;
use Doctrine\ORM\Mapping as ORM;

/**
 * @ORM\Entity(repositoryClass=ProductRepository::class)
 */
class Product
{
    /**
     * @ORM\Id()
     * @ORM\GeneratedValue()
     * @ORM\Column(type="integer")
     */
    private $id;

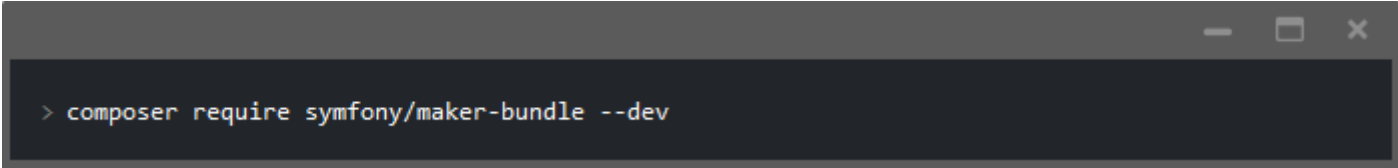
    /**
     * @ORM\Column(type="string", length=255)
     */
    private $name;

    /**
     * @ORM\Column(type="integer")
     */
    private $price;

    public function getId(): ?int
    {
        return $this->id;
    }

    // ... getter and setter methods
}
```


Symfony MakerBundle



```
> composer require symfony/maker-bundle --dev
```

- Symfony MakerBundle est un bundle qui va vous permettre de générer automatiquement plusieurs type de fichier en ligne de commande.
- Génération automatique des Commands, Controllers, Forms, Entities, etc...
- Une fois installer taper la commande suivante pour voir la liste des différentes chose que vous pouvez créer avec le maker.

symfony console list make

console make:entity

- Grâce au maker de Symfony, il est très simple de créer nos Entity.
- On va lui préciser les différents attributs de notre Entity et leurs types.
- Le maker va automatiquement générer une Classe avec les différents attributs, les différentes annotations Doctrine, ainsi que les getters et setters pour chacun de nos attributs.
- Cela va aussi générer le repository lié a votre Entity.

```
1  $ php bin/console make:entity
2
3  Class name of the entity to create or update:
4  > Product
5
6  New property name (press <return> to stop adding fields):
7  > name
8
9  Field type (enter ? to see all types) [string]:
10 > string
11
12 Field length [255]:
13 > 255
14
15 Can this field be null in the database (nullable) (yes/no) [no]:
16 > no
17
18 New property name (press <return> to stop adding fields):
19 > price
20
21 Field type (enter ? to see all types) [string]:
22 > integer
23
24 Can this field be null in the database (nullable) (yes/no) [no]:
25 > no
26
27 New property name (press <return> to stop adding fields):
28 >
29 (press enter again to finish)
```

Les migrations

- Le bundle DoctrineMigrationBundle va nous permettre de mettre à jour notre base de donnée grace aux fichiers de Migration.
- L'intérêt principal des migrations est de pouvoir versionner sa base de donnée.
- Chaque fois qu'on modifie une entité, il faut générer un nouveau fichier de migration.
- **console doctrine:migrations:diff** (cela va générer un fichier de migration en comparant l'état de vos entités et l'état de votre BDD).
- **console doctrine:migrations:migrate** (cela va exécuter toute les migrations qui n'ont pas encore été faites et mettre à jour votre BDD).
- Une nouvelle table à été créer dans votre BDD afin de suivre l'historique des migrations exécuter.

Repository

- Le repository est une classe qui est lié à une Entity.
- Cette classe hérite d'une classe mère qui contient déjà plusieurs méthode qui vont vous permettre de récupérer des données depuis votre base de données.
- Vous pouvez ensuite rajouter vos propre méthodes custom.

```
// src/Repository/ProductRepository.php

// ...
class ProductRepository extends ServiceEntityRepository
{
    public function __construct(ManagerRegistry $registry)
    {
        parent::__construct($registry, Product::class);
    }

    /**
     * @return Product[]
     */
    public function findAllGreaterThanOrPrice($price): array
    {
        $entityManager = $this->getEntityManager();

        $query = $entityManager->createQuery(
            'SELECT p
            FROM App\Entity\Product p
            WHERE p.price > :price
            ORDER BY p.price ASC'
        )->setParameter('price', $price);

        // returns an array of Product objects
        return $query->getResult();
    }
}
```

Manipuler des entités

- Créer ou manipuler des objets avec l'EntityManager.

```
class ProductController extends AbstractController
{
    /**
     * @Route("/product", name="create_product")
     */
    public function createProduct(): Response
    {
        // you can fetch the EntityManager via $this->getDoctrine()
        // or you can add an argument to the action: createProduct(EntityManagerInterface $entityManager)
        $entityManager = $this->getDoctrine()->getManager();

        $product = new Product();
        $product->setName('Keyboard');
        $product->setPrice(1999);
        $product->setDescription('Ergonomic and stylish!');

        // tell Doctrine you want to (eventually) save the Product (no queries yet)
        $entityManager->persist($product);

        // actually executes the queries (i.e. the INSERT query)
        $entityManager->flush();

        return new Response('Saved new product with id '.$product->getId());
    }
}
```

find(), findOneBy(), findBy() et findAll()

```
$repository = $this->getDoctrine()->getRepository(Product::class);

// Look for a single Product by its primary key (usually "id")
$product = $repository->find($id);

// Look for a single Product by name
$product = $repository->findOneBy(['name' => 'Keyboard']);
// or find by name and price
$product = $repository->findOneBy([
    'name' => 'Keyboard',
    'price' => 1999,
]);

// Look for multiple Product objects matching the name, ordered by price
$products = $repository->findBy(
    ['name' => 'Keyboard'],
    ['price' => 'ASC']
);

// Look for *all* Product objects
$products = $repository->findAll();
```

Aller plus loin avec le DQL

- Bien sûr, Doctrine vous permet également d'écrire des requêtes plus complexes en utilisant le Doctrine Query Language (DQL). Le DQL est très ressemblant au SQL excepté que vous devez imaginer que vous requêtez un ou plusieurs objets d'une classe d'entité (ex: Product) au lieu de requêter des lignes dans une table (ex: product).

```
public function findAllGreaterThanPrice($price, $includeUnavailableProducts = false): array
{
    // automatically knows to select Products
    // the "p" is an alias you'll use in the rest of the query
    $qb = $this->createQueryBuilder('p')
        ->where('p.price > :price')
        ->setParameter('price', $price)
        ->orderBy('p.price', 'ASC');

    if (!$includeUnavailableProducts) {
        $qb->andWhere('p.available = TRUE');
    }

    $query = $qb->getQuery();

    return $query->execute();

    // to get just one result:
    // $product = $query->setMaxResults(1)->getOneOrNullResult();
}
```

Aller encore plus loin

- Il va de soit que doctrine permet encore plus de choses :
 - **C**reate,
 - **R**ead,
 - **U**ppdate,
 - **D**eleter.
- Jointure complexe,
- La documentation de Doctrine pour Sf5 est riche : <https://symfony.com/doc/current/doctrine.html>
- Mais celle de doctrine l'est encore plus : <http://www.doctrine-project.org/>

2 – Les API

- Définition
- Implémentation
- API Platform

API : Définition

- En informatique, une interface de programmation d'application ou interface de programmation applicative (souvent désignée par le terme API pour Application Programming Interface) est un ensemble normalisé de classes, de méthodes, de fonctions et de constantes qui sert de façade par laquelle un logiciel offre des services à d'autres logiciels. Elle est offerte par une bibliothèque logicielle ou un service web, le plus souvent accompagnée d'une description qui spécifie comment des programmes consommateurs peuvent se servir des fonctionnalités du programme fournisseur.
- De manière plus générale, on parle d'API à partir du moment où une entité informatique cherche à agir avec ou sur un système tiers, et que cette interaction se fait de manière normalisée en respectant les contraintes d'accès définies par le système tiers. On dit que le système tiers « expose une API. » À ce titre, des choses aussi diverses que la signature d'une fonction, une URL, un RPC... sont parfois considérés comme des API (ou micro-API) à part entière. (@Wikipedia)

API : APIRESTful, c'est quoi ?

- REST est un style d'architecture permettant à des systèmes distribués de centraliser des services partagés.
- Nécessite un langage commun au niveau des échanges (JSON, XML)
- Basé sur 5 contraintes obligatoires:
 - **Client-Server** : séparation des rôles client et serveur lors de la communication
 - **Stateless server** : toutes les informations nécessaires au traitement sont présentes dans la requête. Pas de notion de session côté serveur.
 - **Cache** : la réponse du serveur doit être cacheable côté client
 - **Uniform interface** : en HTTP cela veut dire, 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 ...)

API : APIRESTful, c'est quoi ?

- Plusieurs niveaux d'implémentations :

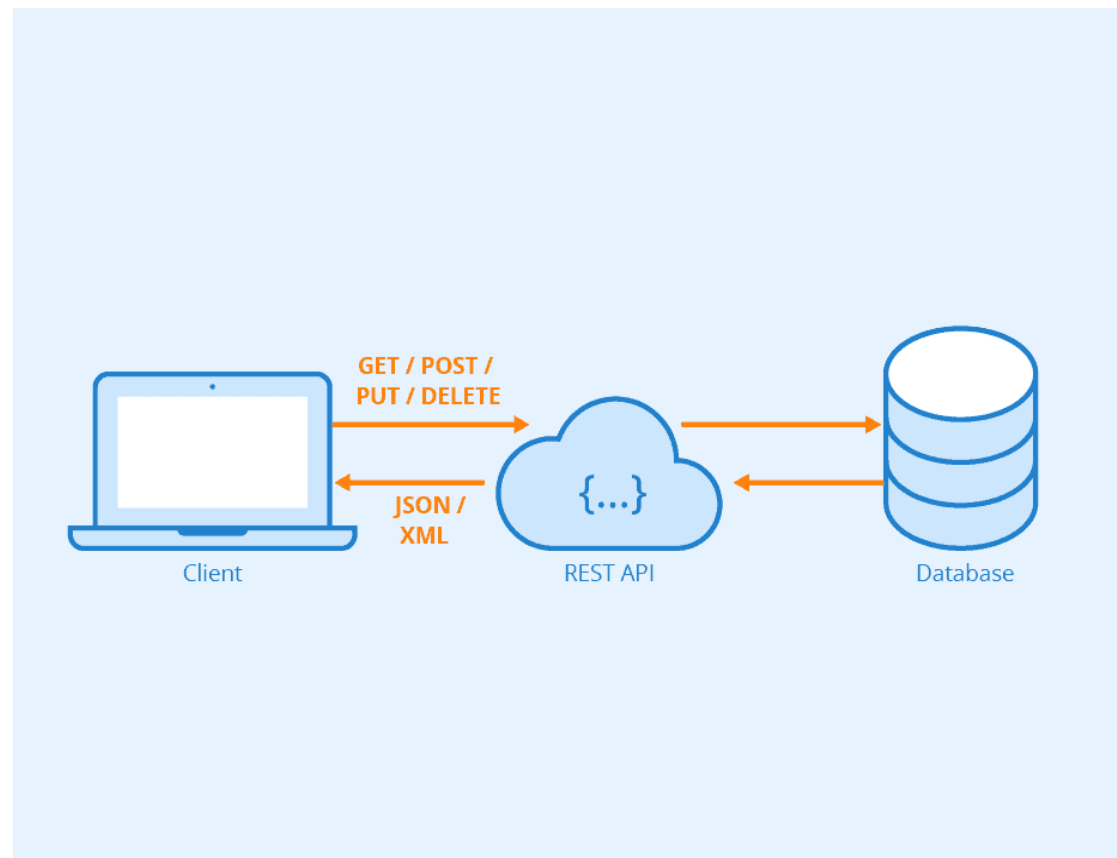
	Identifiants de ressource	Verbes HTTP	Codes retour	HATEOAS
Niveau 0	NON	POST	200	NON
Niveau 1	OUI	GET	200	NON
Niveau 2	OUI	GET, POST, PUT, DELETE	2XX, 4XX	NON
Niveau 3	OUI	GET, POST, PUT, DELETE	2XX, 4XX	OUI

- Les méthodes : GET, POST, PUT, DELETE, (PATCH, OPTION, ...)
- HATEOAS (Hypermedia As The Engine Of Application State) : contrainte sur la présence de liens au niveau des ressources pour permettre une navigation exploratoire.

API : Pourquoi ?

- Permet d'interagir facilement avec tout type de techno (PHP, JAVA, Python, etc).
- Fournir des services et données à des applications tierces.
- Moyen de communication simple d'utilisation encadré par un contrat d'interface (CI)
- Sur une même application, séparer les données du backend et du frontend (données client protégées côté serveur (backend) par exemple)
- Cloud data, permet d'externaliser sa BDD.

Fonctionnement d'un API Rest



Différentes methods

- **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.
- **PUT** : Permet de modifier les données d'une entité dans une ressource.
- **DELETE** : Permet de supprimer une entité dans une ressource.

API Public et API Privée

- Une API privée va rajouter une surcouche de sécurité afin d'éviter les appels anonymes.
- Le client doit être authentifié pour faire un appel vers l'API.
- 3 façons principales de sécuriser son API
 - HTTP Basic Authentication (simple mais moins sécurisé)
 - Oauth (complexe mais très sécurisé)
 - OpenID (en mode SSO, ressemble a Oauth).

API : Exemple Implémentation Symfony

```
/**
 * @Route("/api/articles", name="article", methods={"GET"})
 */
public function index(EntityManagerInterface $em): JsonResponse
{
    $articles = $em->getRepository(Article::class)->findAll();
    $serializedArticles = [];
    foreach($articles as $article) {
        $serializedArticles[] = [
            'id' => $article->getId(),
            'title' => $article->getTitle(),
            'content' => $article->getContent(),
        ];
    }

    return new JsonResponse(['data' => $serializedArticles, 'items' => count($serializedArticles)]);
}
```

```
1 {
2     "data": [
3         {
4             "id": 1,
5             "title": "Title 0",
6             "content": "Lorem ipsum dolor sit amet, consectetur adipiscing
elit, sed do eiusmod tempor incididunt ut labore et dolore
magna aliqua. Ut enim ad minim veniam, quis nostrud
exercitation ullamco laboris nisi ut aliquip ex ea commodo
consequat. Duis aute irure dolor in reprehenderit in
voluptate velit esse cillum dolore eu fugiat nulla
pariatur. Excepteur sint occaecat cupidatat non proident,
sunt in culpa qui officia deserunt mollit anim id est
laborum."
7         },
8         ...
9     ],
10     "items": 30
11 }
```

API : API Platform

- Fournit une façon simplifiée d'exposer des API.
- Fonctionne principalement par annotation sur les entités
- Simple et rapide à mettre en place, à personnaliser
- Fournit un swagger

```
/**
 * @ApiResponse()
 * @ORM\Entity(repositoryClass=ArticlesRepository::class)
 */
class Article
```

Article

GET /api/articles Retrieves the collection of Article resources.

POST /api/articles Creates a Article resource.

GET /api/articles/{id} Retrieves a Article resource.

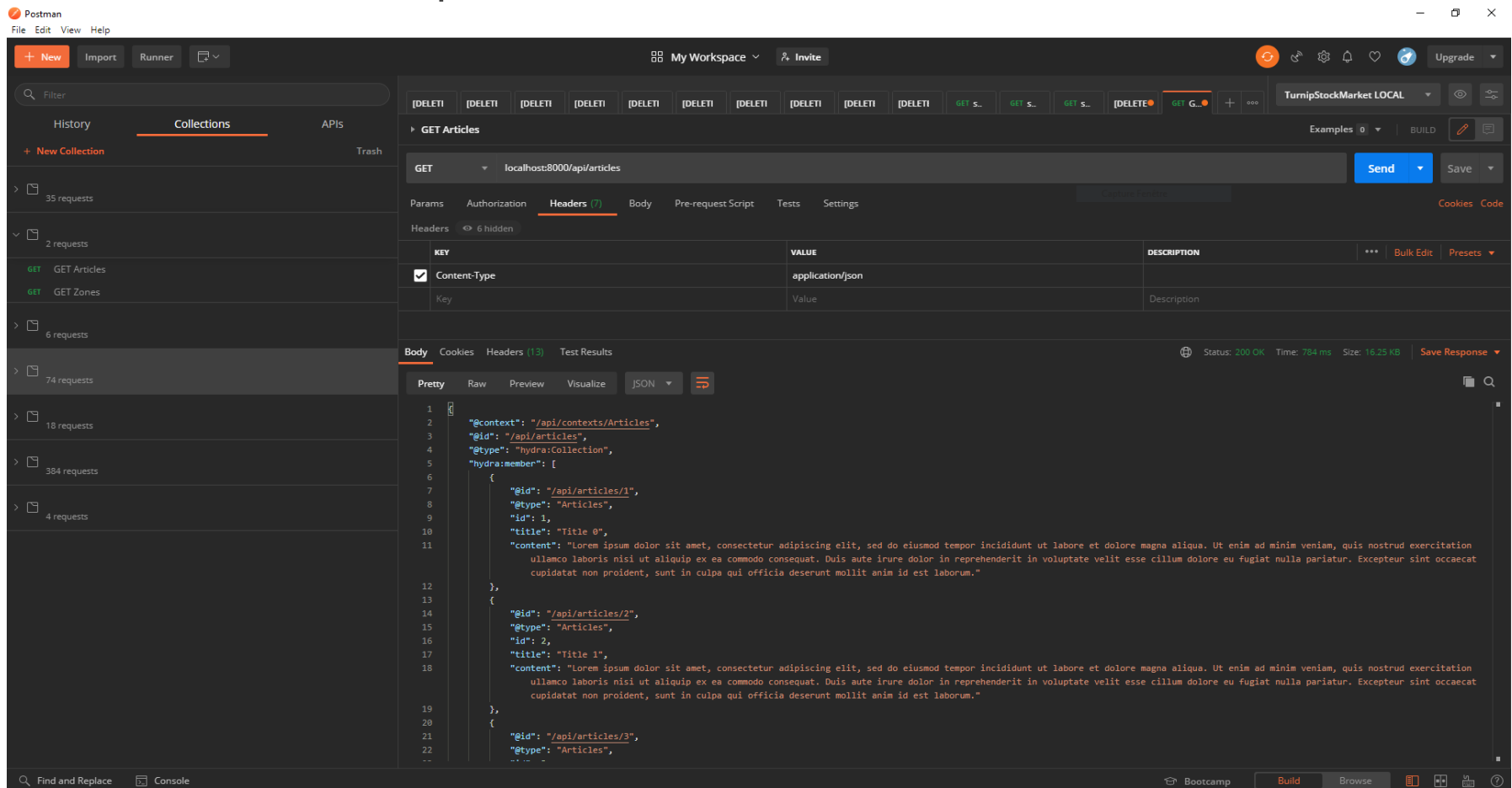
DELETE /api/articles/{id} Removes the Article resource.

PUT /api/articles/{id} Replaces the Article resource.

PATCH /api/articles/{id} Updates the Article resource.

Outils

- **Postman** est un outil pour tester ses API.



3 – Doctrine :

TP n°4

- Créer son modèle
- Créer ses entités
- Manipuler

A rendre pour le :
04/01/2020

2 personnes max !

Description du projet final

29

Dans le cadre de ce cours et afin de mettre en pratique l'utilisation du framework Symfony, nous allons réaliser un blog.

Celui-ci devra être hébergé sur la plateforme Cloud Heroku.

Le blog devra être :

En HTML5 (**sémantique** et **valide**). Utilisant un **framework CSS** (Bootstrap, pure, etc).

Le blog comprend 2 contrôleurs (Blog & Crud) et 5 routes (Action) :

homepage => page d'accueil (/) listant les x derniers articles (page à page),

post => page article (/post/\$id) affichant le contenu d'un article

newPost => Ajout

editPost => Edition

deletePost => Suppression

Le blog implémente FosUser (ou autre système

doit être habillée) et les routes CRUD doivent être protégées.

Le blog implémente aussi StofDoctrineExtensionsBundle pour rendre vos articles timestampable et sluggable

Récupérer les données provenant d'une API externe et mettre en place une API pour récupérer les 5 articles les plus récents de votre blog.

Le code source et le blog seront stockés dans le « cloud ».

En gris les sujet à traiter lors du prochain c

Modèle de données de mon blog

Notre blog sera simple. Il se composera uniquement d'une table « post » composée des champs suivants :

- titre : titre de l'article.
- url_alias (ou slug ou etc.) : URL de l'article.
- content : contenu de l'article.
- published : date de publication.

A l'aide des connaissances acquises en cours, écrivez l'entité et déployez le modèle en base.

Alimenter la base

Afin de partir avec de vraies données, renseignez quelques billets dans votre base directement depuis phpMyAdmin de SA.P.IENS (Voir le TP du cours 2 sur comment utiliser SA.P.IENS).

Autre solution, installer le FixturesBundle de doctrine en mode « dev » et créer des fichiers de fixtures :

<https://symfony.com/doc/3.1/bundles/DoctrineFixturesBundle/index.html>

Exploitation des données

Maintenant que nous avons une classe entité, nous allons modifier les contrôleurs créés précédemment :

- Le contrôleur `indexAction` va récupérer les 10 derniers billets et les passer aux templates.
- Le contrôleur `postAction` va récupérer le billet en fonction de l'`url_alias` passé en URL et le passer aux templates.

5 – La traduction dans Sf5

- Activation
- Les fichiers de traduction
- Utilisation

Activer le service de traduction

- Les traductions sont traitées par le **service** *Translator* qui utilise la locale de l'utilisateur pour chercher et retourner les messages traduits. Avant de l'utiliser, activez le Translator dans votre configuration :

```
1 # app/config/config.yml
2 framework:
3     translator: { fallback: en }
```

- L'option `fallback` définit la locale de secours à utiliser quand une traduction n'existe pas dans la locale de l'utilisateur.

Différents fichier de traduction

- Comme toujours :-(, Symfony5 propose plusieurs formats de fichiers. La traduction n'échappe pas à cela et propose 3 formats :
 - **XML / XLIFF**,
 - **PHP**,
 - **YAML**.
- La forme la plus simple et la plus recommandée par la doc officielle étant le **YAM**. Cependant le format **XLIFF** qui est plus complexe est recommandée pour les applications générant les traductions avec des programmes spéciaux.

XLIFF VS YAML

```
1 <!-- messages.fr.xliff -->
2 <?xml version="1.0"?>
3 <xliff version="1.2" xmlns="urn:oasis:names:tc:xliff:document:1.2">
4   <file source-language="en" datatype="plaintext" original="file.ext">
5     <body>
6       <trans-unit id="1">
7         <source>Symfony2 is great</source>
8         <target>J'aime Symfony2</target>
9       </trans-unit>
10    </body>
11  </file>
12 </xliff>
```

```
1 # messages.fr.yml
2 Symfony2 is great: J'aime Symfony2
```

Utilisation du service de traduction

- L'utilisation du service de traduction est alors simple :

```
class DefaultController extends Controller
{
    /**
     * @Route("/", name="homepage")
     */
    public function indexAction(TranslatorInterface $translator)
    {
        $translationMessage = $translator->trans('Symfony is great');

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

Aller plus loin

Nous venons de voir les bases de la traduction dans Sf4. Mais l'outil va bien plus loin :

- Paramètres de substitution,
- Utilisation de mots clef,
- Utilisation de domaines,
- Formes plurielles complexes (ex : Russe),
- Traduction au sein des templates Twig (utilisation du pipe « trans »).

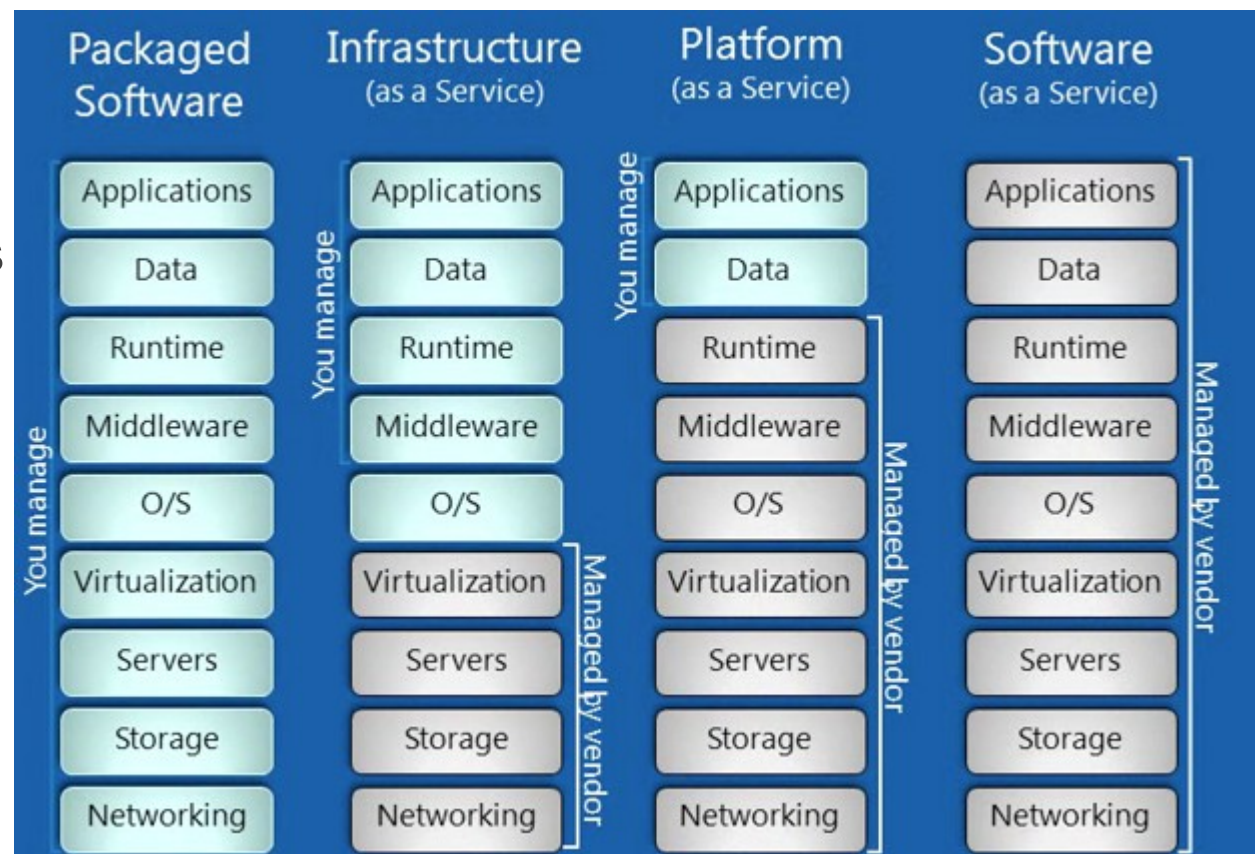
Aller plus loin : [Translation](#).

6 – Déployer une application Sf5 dans le cloud

- Le cloud
- Pourquoi utiliser le PaaS
- Les différents PaaS pour symfony
- Heroku

Le cloud ?

« Le cloud computing c'est de pouvoir utiliser des ressources informatiques sans les posséder »



Pourquoi utiliser le PaaS ?

- Approche no ops : le talent des développeurs 100 % consacré aux tâches à valeur ajoutée
- Une mise en production quasi instantanée
- Des capacités d'expérimentation illimitées
- Une fluidité dans l'organisation
- Une meilleur sécurité

Iterate Faster

Write Code. We Run It.

Les différents PaaS pour symfony

- Microsoft Azure Website Cloud
- Fortrabbit
- Platform.sh
- Heroku Cloud

Heroku Cloud

- Heroku est un service de cloud computing de type plate-forme en tant que service. Créé en 2007, il était l'un des tout premiers services cloud, puis a été racheté par Salesforce.com.
- À l'origine dévolue aux applications web programmées en Ruby et utilisant Rack (typiquement, des applications Ruby on Rails ou Sinatra), l'offre s'est ensuite étendue à d'autres runtimes : node.js, Java, Spring et Play framework, Clojure, Python et Django, Scala, API Facebook, ainsi que PHP. Les différents runtimes coexistent aujourd'hui dans un même stack polyglotte, nommé Cedar, tournant sur une base d'Ubuntu. Le service permet le déploiement très rapide d'applications web dans le cloud, avec une gestion très souple du scaling horizontal au travers d'un modèle de gestion des processus emprunté à Unix et adapté au Web.

7 – TP n°5

- Déployer son application sur HEROKU

Déployer son application sur Heroku 1/5 : Sur Heroku

- Aller sur <https://www.heroku.com/> et créez y un compte.
- Une fois connecté :
 - Cliquer sur « *Create a new App* »,
 - Pour la App-name veuillez saisir : nom1-nom2-blog (ex : levy-kowalczyk-blog),
 - Pour la région choisir « *Europe* ».
- Une fois l'application créée :
 - Cliquer sur « Settings » puis sur « Add buildpack »,
 - Choisir « *heroku/php* ».
- Ajout de la variable d'environnement :
 - Aller dans « settings » puis dans « Config Variables » ajoutez
 - APP_ENV: prod

Déployer son application sur Heroku 2/5 : Le client Heroku

- Installer le client Heroku (version standalone?) :
<https://devcenter.heroku.com/articles/heroku-cli#standalone>
- Création de la ssh key :
 - Allez sur « account settings »,
 - Ajouter votre clé publique dans la section SSH Keys.
- Ajouter le support de PostgreSQL :

```
heroku addons:add heroku-postgresql
```

Déployer son application sur Heroku 3/5 : Configurez votre application

Important

- Ajouter un fichier « Procfile » à la racine de votre projet :
`web: vendor/bin/heroku-php-apache2 public/`
- Activer les logs dans Heroku : <https://devcenter.heroku.com/articles/deploying-symfony4#logging>
- Nettoyez le fichier **config/packages/doctrine.yaml** en supprimant cette partie :

```
# configure these for your database server
# use postgresql for PostgreSQL
# use sqlite for SQLite
driver: 'mysql'
server_version: '5.7'
```

```
# only needed for MySQL
charset: utf8mb4
```

```
default_table_options:
  charset: utf8mb4
  collate: utf8mb4_unicode_ci
```

- Ajouter la gestion de l'URL rewriting en répondant bien « yes » pour déployer la *recipe* :
<https://devcenter.heroku.com/articles/deploying-symfony4#url-rewrites>
- Ajouter à composer la mise à jour automatique de la BDD post installation

Déployer son application sur Heroku 4/5 :

Push de l'application sur heroku

- Allez à la racine de votre application et lancer les commande suivante :

```
git init
```

```
heroku git:remote -a {nom1}-{nom2}-blog
```

```
git add .
```

```
git commit -am "Init project on Heroku"
```

```
git push heroku master
```


Déployer son application sur Heroku 5/5

- L'application est déployer sur :
<https://{nom1}-{nom2}-blog.herokuapp.com/>
- Pour finir aller dans Access et cliquer sur « Add collaborateur »
 - nathan.levy.um2@gmail.com
 - matthieu.kowalczyk.um2@gmail.com

Buts :

- Avoir votre application qui fonctionne sur heroku

Personal > loic-cariou-blog



Open app

More

GitHub loic-cariou/blog

[Overview](#) [Resources](#) [Deploy](#) [Metrics](#) [Activity](#) [Access](#) [Settings](#)

Installed add-ons \$0.00/month

[Configure Add-ons](#)Heroku Postgres Hobby Dev
postgresql-fitted-28339

Dynamo formation \$0.00/month

[Configure Dynos](#)

This app is using free dynos

web vendor/bin/heroku-php-apache2 public/

ON

Collaborator activity

[Manage Access](#)

cariou.loic.um2@gmail.com


3 deploys

Latest activity

[All Activity](#)cariou.loic.um2@gmail.com: Deployed 9996d05f
Today at 5:58 PM · v10 [Compare diff](#)cariou.loic.um2@gmail.com: Build succeeded
Today at 5:58 PM · [View build log](#)cariou.loic.um2@gmail.com: Deployed 9996d05f
Today at 5:57 PM · v9 [Compare diff](#)cariou.loic.um2@gmail.com: Build succeeded
Today at 5:57 PM · [View build log](#)cariou.loic.um2@gmail.com: Deployed 9996d05f
Today at 5:56 PM · v8cariou.loic.um2@gmail.com: Build succeeded
Today at 5:56 PM · [View build log](#)cariou.loic.um2@gmail.com: Remove SYMFONY_ENV config var
Today at 5:56 PM · v7cariou.loic.um2@gmail.com: Set APP_ENV config var
Today at 5:56 PM · v6cariou.loic.um2@gmail.com: Build failed
Today at 5:55 PM · [View build log](#)

Personal  >  loic-cariou-blog

Open app

More GitHub  loic-cariou/blog

Overview

Resources

Deploy

Metrics

Activity

Access

Settings

Free Dynos

Change Dyno Type

web vendor/bin/heroku-php-apache2 public/



\$0.00



Add-ons

Find more add-ons

 Quickly add add-ons from ElementsHeroku Postgres Attached as DATABASE 

Hobby Dev

Free



Estimated Monthly Cost

\$0.00

Name

Name management is not available

Only the owner, cariou.loic.um2@gmail.com, can rename this app.

Config Vars


Config vars change the way your app behaves. In addition to creating your own, some add-ons come with their own.


Config Vars

Hide Config Vars

APP_ENV


prod






DATABASE_URL

postgres://jxhmm1easovkfk:f46e7402eab89l








KEY

VALUE

Add

Info

Region	 Europe
Stack	heroku-18
Framework	 PHP
Slug Size	23.5 MiB of 500 MiB
GitHub Repo	 loic-cariou/blog
Heroku Git URL	<code>https://git.heroku.com/loic-cariou-blog.git</code>

Buildpacks



8 – Installation d'un bundle tiers

- Comment
- Où les trouver

Installer un bundle tiers

Deux façons de faire:

1. Ajout du bundle à composer.json et faire un composer install

ou

2. composer require monbundle

```
{  
    'require': {  
        'friendsofsymfony/user-bundle': "2.0.*@dev"  
    }  
}
```

```
> composer require stof/doctrine-extensions-bundle
```

Trouver des bundles tiers

- [Symfony Recipes](#) propose un large éventail de bundles.
- Les Bundles que vous allez regretter de ne pas avoir connu plus tôt.

Les meilleurs bundles...

LES BUNDLES

QUE VOUS ALLEZ REGRETTER DE
N'ÊTRE PAS AVOIR CONNU PLUS TÔT

Une présentation par [Damien Alexandre](#) / JoliCode
Symfony Live Paris 2013

9 – TP n°6

- Installation d'un bundle tiers.

Installation de StofDoctrineExtensionsBundle

- Lisez la [documentation de StofDoctrineExtensionsBundle](#),
- Appliquez la.

Buts :

- Avoir des entités timestampable et sluggable
- Apprendre à lire / utiliser / appliquer une documentation.

Mise en place du security.yml

- Lisez la [documentation Security de Symfony](#),
- Appliquez la et suivez les liens et instructions pour la créations de form login, user (pensez au maker bundle)

Buts :

- Avoir une entité User enregistrée en base avec des rôles, et droits d'accès particuliers.
- Apprendre à lire / utiliser / appliquer une documentation.

Annexes

Guide de survit de bin/console

- Vider le cache :

`bin/console cache:clear`

- Générer mes getter et setter de mes entities :

`php bin/console make:entity App/Entity/Product`

Webographie

- Site officiel de Symfony : <http://symfony.com/>
- Le manuel PHP : <http://php.net/manual/fr/index.php>
- La doc : <http://symfony.com/doc/current/book/index.html>

Bibliographie

- Toutes la documentation et les « books » officiels sont librement téléchargeables
<http://symfony.com/doc/current/index.html>

Licence

La documentation Symfony5 étant bien faite et sous Licence libre (MIT), des illustrations tout comme des bouts de textes de ce cours en sont extraits.