

Symfony 4



EMAKINA
building brand experiences

Agenda

PLAN

Structure de Symfony

Traitement des requetes HTTP

Doctrine

Service Container

Templating avec Twig

Formulaire et Validation

Security

Components

Console, Messenger, HttpClient



Structure de Symfony



What is Symfony ?



Symfony est ...

- Un ensemble de **composants** PHP **réutilisable, standalone et découplé.**
- résout les problèmes commun du développement web
- aussi un **framework** web



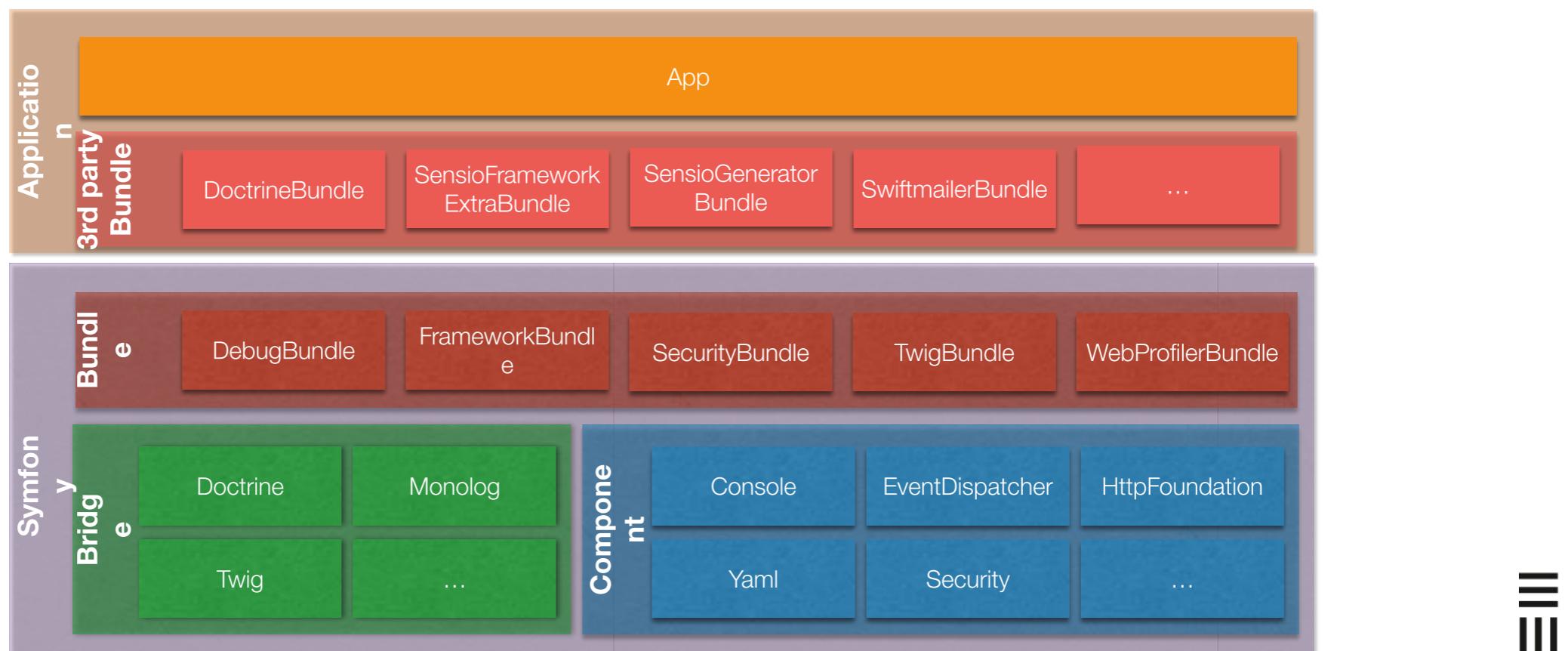
Symfony

Component = Une librairie « indépendantes »

Bridge = Une extension d'une librairie tierce

Bundle = Un groupe de fonctionnalité utilisant des components

Bundle = Mon application



Framework

Un framework est un logiciel qui propose des composants génériques qui facilite de développement d'application et le travail en équipe

Un framework aide les développeurs à développer des applications flexibles et extensibles, écrire du code de qualité et maintenable



Cycle de vie



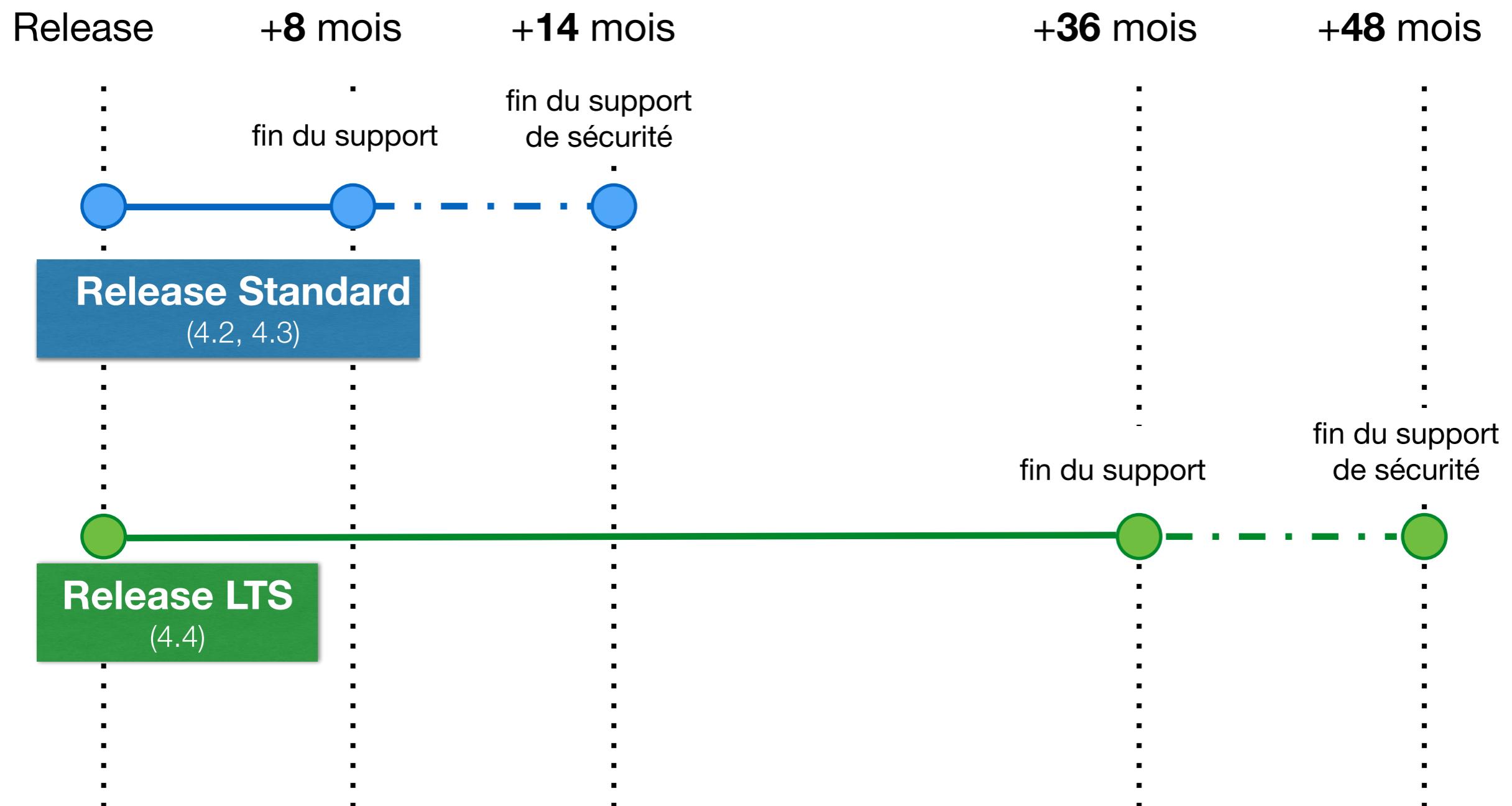
Cycle de vie

- Le développement d'une nouvelle version est divisé en 2 parties
 - Développement (4 mois)
 - Stabilisation (2 mois)
- La plupart des versions de Symfony sont des versions « *standards* »
 - 8 mois de support de bug
 - 14 mois de support sécurité
- Les versions « *Long Term Support* » sont released tous les 2 ans
 - 36 mois de support de bug
 - 48 mois de support sécurité

<http://symfony.com/roadmap>



Cycle de vie



Architecture



Architecture

```
└── bin
    └── console
        └── phpunit
    └── composer.json
    └── composer.lock
    └── config
        ├── bundles.php
        ├── packages
        ├── routes
        ├── routes.yaml
        └── services.yaml
    └── phpunit.xml.dist
    └── public
        └── index.php
    └── src
        ├── Controller
        ├── Entity
        ├── Kernel.php
        ├── Migrations
        └── Repository
    └── symfony.lock
    └── templates
        └── base.html.twig
    └── tests
    └── translations
    └── var
        └── cache
            └── log
    └── vendor
```

config/ est le répertoire qui contient toute la **configuration**

public/ est le répertoire qui contient le **front controller** et les **assets web** (Images, Javascript, CSS)

src/ est le répertoire qui contient le **code PHP**

templates/ est le répertoire qui contient les **vues**

var/ est le répertoire des fichiers générés
(cache, sessions, logs)



Le front controller



Front controller

- Le **front controller** est le seul point d'entrée de l'application

```
|use App\Kernel;
|use Symfony\Component\HttpFoundation\Request;

$kernel = new Kernel($env, $debug);
$request = Request::createFromGlobals();
$response = $kernel->handle($request);
$response->send();
$kernel->terminate($request, $response);
```



Front controller

- Chaque ressource que l'utilisateur veut accéder est accessible depuis le front controller

`http://www.domain.tld/index.php/`

`http://www.domain.tld/index.php/hello/Nicolas`

`http://www.domain.tld/index.php/hello/Toto`

front controller

ressource



Séparation



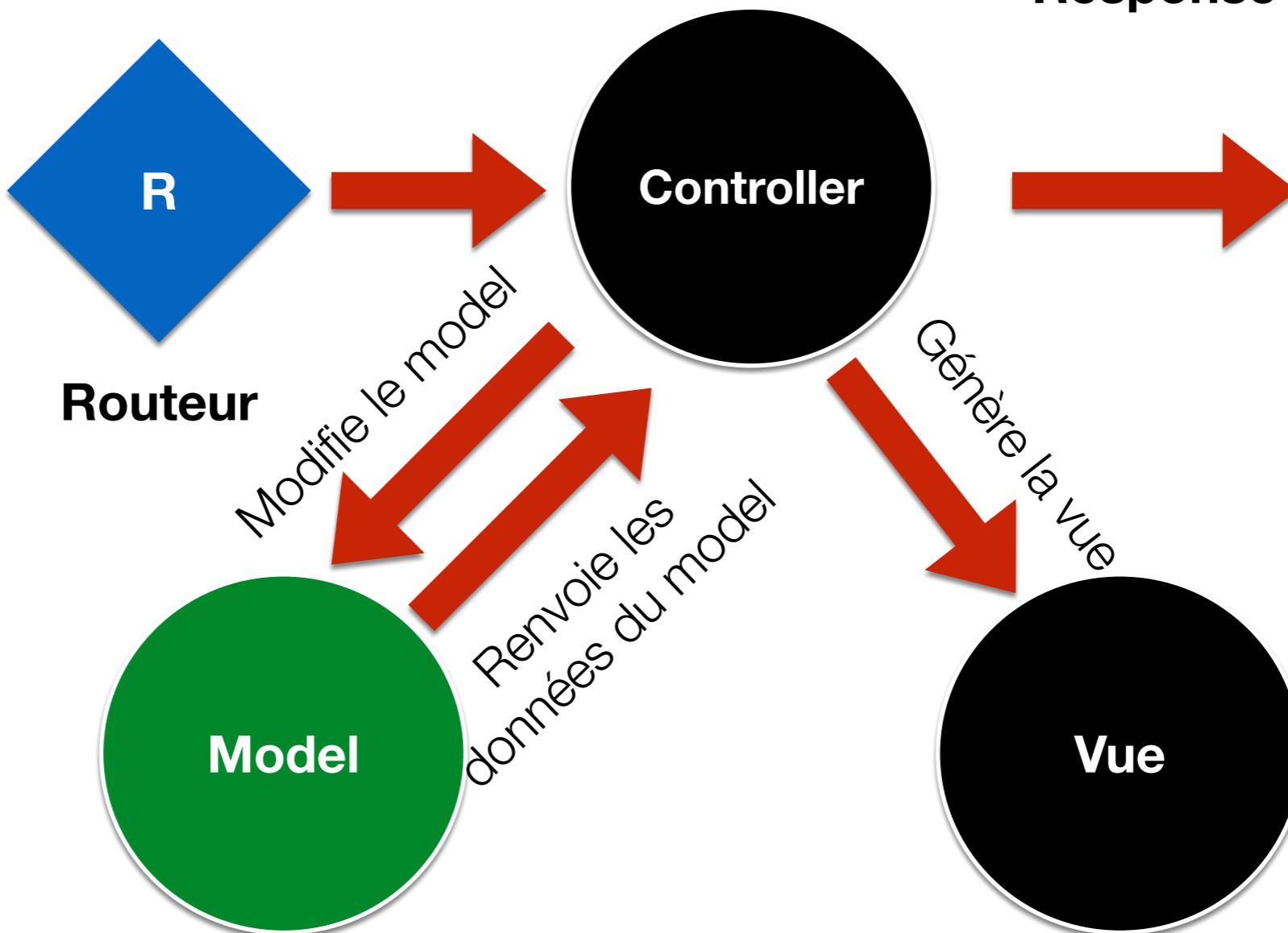
Implementation

Request

→ /hello/nicolas

Le contrôleur analyse la requête utilisateur, appelle le model et passe la donnée à la Vue

Response



Le model stock la logique métier et les données manipulées

La couche de vue formate les données



Environnements

- Un environnement est un ensemble de valeurs de configuration représenté par une chaîne de caractère unique

```
# public/index.php
if (!isset($_SERVER['APP_ENV']) && !isset($_ENV['APP_ENV'])) {
    if (!class_exists( class_name: Dotenv::class)) {
        throw new \RuntimeException( message: '...');
    }
    (new Dotenv())->load(__DIR__ . '/../.env');
}

$env = $_SERVER['APP_ENV'] ?? $_ENV['APP_ENV'] ?? 'dev';
$debug = (bool) ($_SERVER['APP_DEBUG'] ?? $_ENV['APP_DEBUG'] ?? ('prod' !== $env));

$kernel = new Kernel($env, $debug);
```



Configuration



Configuration

Symfony supporte plusieurs formats de fichier de configuration (**Annotation, YAML, XML ou PHP**)

La configuration est découpée en multiples fichiers.

```
.  
  -- bundles.php  
  -- packages  
    -- dev  
    -- doctrine.yaml  
    -- doctrine_migrations.yaml  
    -- framework.yaml  
    -- prod  
    -- routing.yaml  
    -- security.yaml  
    -- sensio_framework_extra.yaml  
    -- swiftmailer.yaml  
    -- test  
    -- translation.yaml  
    -- twig.yaml  
    -- validator.yaml  
  -- routes  
    -- annotations.yaml  
    -- dev  
  -- routes.yaml  
  -- services.yaml
```



Exemple de fichier YAML

```
#config/packages/framework.yaml

framework:
    secret: '%env(APP_SECRET)%'
    session:
        handler_id: ~
    php_errors:
        log: true

twig:
    default_path: '%kernel.project_dir%/templates'
    debug: '%kernel.debug%'
    strict_variables: '%kernel.debug%'
```



Exemple de fichier d'Annotation

```
use Sensio\Bundle\FrameworkExtraBundle\Configuration\Route;
use Sensio\Bundle\FrameworkExtraBundle\Configuration\Method;

class DefaultController
{
    /**
     * @Route("/")
     * @Method()
     */
    public function index(): Response
    {
        // ...
    }
}

/*
 * @Route("/hello/{name}")
*/
/**
 * @Route("/hello/{name}")
*/

```

→ Commentaire PHP

→ Annotation PHP



Bonnes pratiques pour les formats de configuration

- Utilisez les annotations pour le **routing**, **persistance** et **validation**
- Utilisez le YAML pour les fichiers de **configuration** & les **services**
- N'utilisez pas le format PHP si vous n'avez pas besoin d'un control complet sur la configuration



.env

Permet de configurer les variables d'environnements lors du développement.

```
# .env
###> symfony/framework-bundle ###
APP_ENV=dev
APP_SECRET=1ead605c28582803fc8aac598736b8c2
###< symfony/framework-bundle ###

###> doctrine/doctrine-bundle ###
DATABASE_URL=mysql://db_user:db_password@127.0.0.1:3306/db_name
###< doctrine/doctrine-bundle ###

###> symfony/swiftmailer-bundle ###
MAILER_URL=null://localhost
###< symfony/swiftmailer-bundle ###
```



Injecter et transformer des variables d'environnement

Il est possible de référencer des variables d'environnement

parameters:

```
my_api_key: '%env(API_KEY)%'
```

Avec Symfony 4, il est possible de transformer des variables d'environnement. Les processeurs disponibles par défaut sont file, json, base64, resolve, const, bool, float, int et string. Il est possible de les chaîner

parameters:

```
my_api_key: '%env(json:file:API_KEY_FILE)%'
```

#va lire le fichier spécifié par la variable API_KEY_FILE de l'environnement, puis appeler json_decode() sur son contenu.



Le système de Bundle



C'est quoi un bundle ?

Un bundle est tout simplement un **ensemble structuré de fichiers** dans un répertoire qui mettent en œuvre une seule fonction. Il a vocation à être **réutiliser**.

Chaque répertoire contient tout ce qui concerne cette fonctionnalité, y compris les fichiers PHP, des templates, des feuilles de style, les fichiers JavaScript, tests et rien d'autre.

Chaque aspect d'une fonctionnalité existe dans un bundle et chaque fonctionnalité vit dans un bundle.



Activation d'un bundle

Les bundles utilisés sont dans le fichier config/bundles.php

```
# config/bundles.php

return [
    Symfony\Bundle\FrameworkBundle\FrameworkBundle::class => ['all' => true],
    Doctrine\Bundle\DoctrineCacheBundle\DoctrineCacheBundle::class => ['all' => true],
    Sensio\Bundle\FrameworkExtraBundle\SensioFrameworkExtraBundle::class => ['all' => true],
    Doctrine\Bundle\DoctrineBundle\DoctrineBundle::class => ['all' => true],
    Doctrine\Bundle\MigrationsBundle\DoctrineMigrationsBundle::class => ['all' => true],
    Symfony\Bundle\SecurityBundle\SecurityBundle::class => ['all' => true],
    Symfony\Bundle\SwiftmailerBundle\SwiftmailerBundle::class => ['all' => true],
    Symfony\Bundle\WebProfilerBundle\WebProfilerBundle::class => ['dev' => true, 'test' => true],
    Symfony\Bundle\TwigBundle\TwigBundle::class => ['all' => true],
    Symfony\Bundle\MonologBundle\MonologBundle::class => ['all' => true],
    Symfony\Bundle\DebugBundle\DebugBundle::class => ['dev' => true, 'test' => true],
    Symfony\Bundle\MakerBundle\MakerBundle::class => ['dev' => true],
    Symfony\Bundle\WebServerBundle\WebServerBundle::class => ['dev' => true],
];
```



Activation d'un bundle

La méthode **registerBundles()** de la classe **Kernel** fait inclut le fichier de bundles.

src/Kernel.php

```
public function registerBundles()
{
    $contents = require $this->getProjectDir() . '/config/bundles.php';
    foreach ($contents as $class => $envs) {
        if (isset($envs['all']) || isset($envs[$this->environment])) {
            yield new $class();
        }
    }
}
```



Handling a request



Controller



Controller

Un contrôleur est un fichier PHP «**callable** ».

Il prend des informations de la **requête HTTP** et crée et **renvoie** une **réponse HTTP** (un objet **Response** Symfony).

La réponse pourrait être une page HTML, un document XML, un tableau JSON sérialisé, une image, une redirection, une erreur 404 ou toute autre chose.

Le contrôleur contient que logique de l'application pour rendre le contenu d'un page



Un Controller

Un contrôleur Symfony en action.
Celui-ci retourne une page qui affiche « Hello word! »

```
use Symfony\Component\HttpFoundation\Response;
```

```
class HelloController
{
    public function hello()
    {
        return new Response('Hello word!');
    }
}
```



Cycle de vie « Requests, Controller, Response »

Toute demande traitée par un projet Symfony passe par le même cycle de vie simple.

Le framework prend soin de tous les trucs répétitif. Il suffit d'écrire le code personnalisé dans la fonction de contrôleur.



Un simple Controller

Un contrôleur est généralement une méthode dans une classe de contrôleur.

```
// src/Controller/HelloController.php
namespace App\Controller;

use Symfony\Component\HttpFoundation\Response;

class HelloController
{
    public function index($name): Response
    {
        return new Response('<html><body>Hello ' . $name . '!</body></html>');
    }
}
```



La classe Controller de base



Etendre la classe AbstractController

Symfony est livré avec une classe de contrôleur de base en option.
Vous aurez accès à un certain nombre de méthodes *d'aide*. (*Rendering des templates, redirections, services,...*)

Les méthodes « Helper » ne sont que des raccourcis vers les fonctions de base de Symfony.

```
// src/Controller/HelloController.php
namespace App\Controller;

use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;

class HelloController extends AbstractController
{
// ...
}
```



Redirection

Si vous souhaitez rediriger l'utilisateur vers une autre page

```
// ...
public function index(): Response
{
    return $this->redirectToRoute('homepage');
}
```



Rendering Templates

Si vous affichez du HTML, vous aurez envie d'utiliser un template.
La méthode render() affiche le template et met le contenu dans un objet de Response.

```
// ...
public function index(): Response
{
    // renders templates/hello/index.html.twig
    return $this->render('hello/index.html.twig', array('name' => $name));
}
```



Gérer les pages 404 ou d'erreurs

Quand les choses ne sont pas trouvés, vous devez retourner une réponse 404.

Vous allez utiliser un helper d'exception.

```
public function index(): Response
{
    // Récupère l'objet de DB
    $product = "...";
    if (!$product) {
        throw $this->createNotFoundException('The product does not exist');
    }

    return $this->render(...);
}
```

Pour une erreur 500:

```
throw new \Exception('Something went wrong');
```



L'objet Request comme argument du controller

Que faire si vous avez besoin de lire les paramètres de la requête (header, fichier uploadé, session) ?

Toutes ces informations sont stockées dans l'objet Request.

Il suffit de l'ajouter comme argument du controller

```
//...
use Symfony\Component\HttpFoundation\Request;

class HelloController extends AbstractController
{

    public function index($firstName, $lastName, Request $request): Response
    {
        $page = $request->query->get('page', 1);
        // ...
    }
}
```



Gestion des sessions

Symfony fournit un objet de session que vous pouvez utiliser pour stocker des informations sur l'utilisateur.

Par défaut, Symfony stocke les attributs dans les sessions PHP natives

```
// ...
use Symfony\Component\HttpFoundation\Request;

class HelloController extends AbstractController
{

    public function index($firstName, $lastName, Request $request): Response
    {
        $session = $request->getSession();
        // stocker un attribut pour être réutilisé lors d'une prochaine request
        $session->set('foo', 'bar');

        // obtenir l'attribut défini dans une autre request
        $foobar = $session->get('foobar');

        // Utiliser une valeur par défaut si l'attribut n'existe pas
        $filters = $session->get('filters', array());

    }
}
```



Flash Messages

Vous pouvez également stocker des messages spéciaux, appelés messages "flash", sur la session de l'utilisateur.

Les messages flash sont destinés à être utilisés une seule fois.

Ils disparaissent de la session automatiquement dès que vous les récupérer.

```
// ...
use Symfony\Component\HttpFoundation\Request;

class HelloController extends AbstractController

{
    public function index($firstName, $lastName, Request $request): Response
    {
        $this->addFlash('notice', 'Your changes were saved!');
    }
}
```



Flash Messages

Après traitement de la requête, le contrôleur met un message flash dans la session et puis redirige.

Dans le template de la page suivante: lire tous les messages flash de la session.

```
{% for flash_message in app.session.flashBag.get('notice') %}  
  <div class="flash-notice">  
    {{ flash_message }}  
  </div>  
{% endfor %}
```



Request et Response



Request et Response

```
namespace App\Controller;

use Sensio\Bundle\FrameworkExtraBundle\Configuration\Route;
use Symfony\Component\HttpFoundation\Response;

class HelloController
{
    /**
     * @Route("/hello/{name}")
     */
    public function index(Request $request): Response
    {
        // ...
    }
}
```



Récupérer les data de la Request

```
use Symfony\Component\HttpFoundation\Request;

class HelloController
{
    public function index(Request $request): Response
    {
        $request->isXmlHttpRequest(); // is it an Ajax request?
        $request->getPreferredLanguage(array('en', 'fr'));

        // retrieve GET and POST variables respectively
        $request->query->get('page');
        $request->request->get('page');

        // retrieve SERVER variables
        $request->server->get('HTTP_HOST');

        // retrieves an instance of UploadedFile identified by foo
        $request->files->get('foo');

        // retrieve a COOKIE value
        $request->cookies->get('PHPSESSID');

        // retrieve an HTTP request header, with normalized, lowercase keys
        $request->headers->get('host');
        $request->headers->get('content_type');
    }
}
```



L'objet Response

Comme **Request**, l'objet **Response** a également la propriété **header** en public.

Les noms des headers sont normalisées de sorte que l'utilisation de *Content-Type* est équivalent à *content-type* ou *content_type*.

```
use Symfony\Component\HttpFoundation\Response;

class HelloController
{
    public function index($name): Response
    {
        // créer un simple réponse avec un code d'état 200 (default)
        $response = new Response('Hello '.$name, Response::HTTP_OK);

        // créer un réponse CSS avec un code d'état 200
        $response = new Response('<style> ... </style>');
        $response->headers->set('Content-Type', 'text/css');
    }
}
```



Json Helper

La classe de contrôleur de base définit une méthode json() qui crée un objet JsonResponse et encode le contenu automatiquement

```
class HelloController extends AbstractController
{
    public function index(): Response
    {
        // returns '{"username":"jane.doe"}' définit l'en-tête appropriée
        return $this->json(array('username' => 'jane.doe'));
        // trois arguments optionnels
        // return $this->json($data, $status = 200, $headers = array(), $context = array());
    }
}
```



Routing



Les urls par défaut ont des problèmes

~~http://www.domain.com/blog.php?action=show&article_id=123~~

- Elles affichent des informations sensibles
- Elles mentionnent des noms de variables et des valeurs
- Elles ne sont pas optimisés pour le SEO



La réécriture d'URL c'est mieux

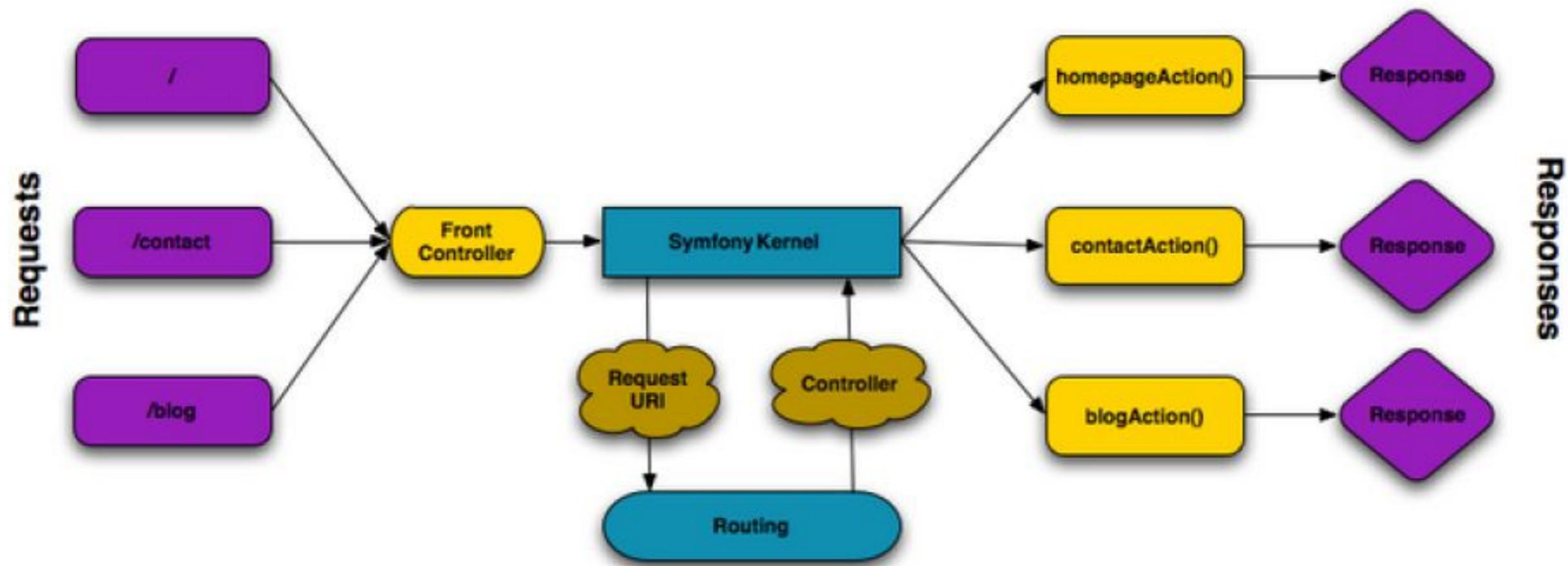
<http://www.domain.com/blog/2014/11/19/symfony-rocks>

De belle URL est un must absolu pour toute application web sérieuse.

- Url propre et smart pour donner des informations utiles
- Aucune information sensible n'est visible
- Configuration des URL facile avec les Annotations, YAMP, PHP, XML



Workflow de Request/Response de Symfony



1. La demande est traitée par le front contrôleur Symfony (`app.php`)
2. Le core Symfony (Kernel) demande au routeur d'inspecter la request;
3. Le routeur correspondant à l'URL entrante, renvoie des informations sur l'itinéraire, y compris le Controller qui doit être exécutée;
4. Le Symfony Kernel exécute le Controller, qui renvoie finalement un objet **Response**.

Créer des routes

Symfony charge toutes les routes de l'application à partir d'un ensemble de fichiers de configuration de routing.

```
└── routes
    ├── annotations.yaml
    └── dev
        ├── twig.yaml
        └── web_profiler.yaml
└── routes.yaml
```

controllers:

```
  resource: ../../src/Controller/
  type: annotation
```

c



Configuration route basique

La définition d'une route est facile, et dans une application typique il y aura beaucoup de routes.

Une route de base se compose de seulement deux parties: le **chemin** et un tableau de **valeurs par défaut**.

```
// src/Controller/MainController.php
namespace MainBundle\Controller;
// ...

$$\begin{array}{l} \text{* } \text{@Route("/") } \\ \text{* } \end{array}$$

public function homepage(): Response
{
    // ...
}
```



Mapping URL et Controller

Ce contrôleur renvoie une page HTML simple. Pour voir cette page dans votre navigateur, vous devez créer une route, qui *mappe* une URL spécifique vers le contrôleur:

```
// src/Controller/HelloController.php
namespace App\Controller;

use Symfony\Component\HttpFoundation\Response;
use Sensio\Bundle\FrameworkExtraBundle\Configuration\Route;

class HelloController
{
    /**
     * @Route("/hello/{name}", name="hello")
     */
    public function index($name): Response
    {
        return new Response('<html><body>Hello ' . $name . '!</body></html>');
    }
}
```



Paramètres de route comme Arguments de Contrôleur

Le contrôleur a un seul argument, **\$name**, qui correspond au placeholder **{name}** de la route.

Lors de l'exécution du controller, Symfony fait correspondre chaque argument avec un placeholder de la route. Ainsi, la valeur de {name} est passé à \$name.

Le placeholder doit avoir le même que le nom que l'argument.

```
// src/Controller/HelloController.php
// ...

use Sensio\Bundle\FrameworkExtraBundle\Configuration\Route;

/**
 * @Route("/hello/{name}", name="hello")
 */
public function index($name): Response
{
// ...
}
```



Paramètres de route comme Arguments de Contrôleur

- L'ordre des arguments du contrôleur n'a pas d'importance
- Chaque argument de contrôleur doit correspondre avec un paramètre de routage
- Tous les paramètres de routage peuvent ne pas être des arguments du contrôleur

<?php

```
// src/Controller/HelloController.php
// ...
use Sensio\Bundle\FrameworkExtraBundle\Configuration\Route;
class HelloController
{
    /**
     * @Route("/hello/{firstName}/{lastName}", name="hello")
     */
    public function index($firstName, $lastName): Response
    {
        // ...
    }
}
```



Routing avec des Placeholders

Le chemin correspond à quelque chose qui ressemble à /blog/*.

La valeur correspondant à la {slug} sera disponible à l'intérieur de votre contrôleur.

Le chemin ne sera pas /blog. Tous les placeholders sont nécessaires.

```
// src/Controller/BlogController.php
namespace App\Controller;
// ...
/**
 * @Route("/blog/{slug}")
 */
public function index($slug): Response
{
    // ...
}
```



Placeholders requis et optionnels

En ajoutant à page la clé **default**, le placeholder {page} n'est plus nécessaire. L'URL /blog correspondra cet itinéraire et la valeur du paramètre de page seront mis à 1.

```
// src/Controller/BlogController.php
namespace App\Controller;
// ...
 $\text{/**}$ 
 $\text{* } \text{@Route("/blog/{page}", defaults={"page" = 1}, name="blog")}$ 
 $\text*/$ 
public function index( $\$page$ ): Response
{
    // ...
}
```

| URL | Route | Parametres |
|---------|-------|------------|
| /blog | blog | {page} = 1 |
| /blog/1 | blog | {page} = 1 |
| /blog/2 | blog | {page} = 2 |



Requirements

Le « \d+ » est une expression régulière dit que la valeur de {page} doit être un chiffre.

```
/**  
 * @Route("/blog/{page}", defaults={"page": 1},  
 * requirements={"page": "\d+" }  
 *)  
 */  
public function index($page): Response  
{  
    // ...  
}
```

| URL | Route | Parametres |
|----------------------|-----------|-------------------------|
| /blog/2 | blog | {page} = 2 |
| /blog/my-blog-post | blog_show | {slug} = my-blog-post |
| /blog/2-my-blog-post | blog_show | {slug} = 2-my-blog-post |



Requirements

Étant donné que les *requirements* sont des expressions régulières, la complexité et la flexibilité de chaque *requirement* est entièrement configurable.

```
/**
 * @Route("/{_locale}", defaults={"_locale": "en"}, requirements={
 *     "_locale": "en|fr"
 * })
 */
public function homepage($_locale): Response
{}
```



Limitation méthode HTTP

Vous pouvez également faire correspondre la méthode de la demande entrante avec un controller (GET, HEAD, POST, PUT, DELETE)

Si aucune méthode n'est spécifiée, la route répondra sur toutes les méthodes

```
// src/Controller/MainController.php
namespace App\Controller;

use Sensio\Bundle\FrameworkExtraBundle\Configuration\Method;

// ...
class BlogApiController extends AbstractController
{
    /**
     * @Route("/api/posts/{id}")
     * @Method({"GET", "HEAD"})
     */
    public function show($id): Response
    {
        // ... return a JSON response with the post
    }

    /**
     * @Route("/api/posts/{id}")
     * @Method("PUT")
     */
    public function edit($id): Response
    {
        // ... edit a post
    }
}
```



Ajouter des formats

```
/*
 *  @Route(
 *      name          = "greet",
 *      path          = "/hello/{name}.{_format}",
 *      defaults     = {
 *          "name"      = "world"
 *          "_format"   = "html"
 *      },
 *      requirements = {
 *          "name"    = << [a-z]+"
 *          "_format" = "html|json|xml"
 *      }
 *  )
 */
```



Configuration des ressources de route



Préfixer une route

Vous pouvez également choisir de fournir un "**prefix**" pour les routes importées.

```
# config/routing.yml
app:
  resource: ../../src/Controller/
  type: annotation
  prefix: /site
```



Visualisation & Debugging des Routes

Tout en ajoutant et personnalisant de routes, il est utile de pouvoir visualiser et d'obtenir des informations détaillées sur vos routes.

Une excellente façon de voir toutes les routes dans votre application via cette commande.

```
php bin/console debug:router
```

| | | |
|-----------------|------|--|
| homepage | ANY | / |
| contact | GET | /contact |
| contact_process | POST | /contact |
| article_show | ANY | /articles/{_locale}/{year}/{title}.{_format} |
| blog | ANY | /blog/{page} |
| blog_show | ANY | /blog/{slug} |



Génération d'URL



Générer des URL depuis un controller

Pour générer une URL, vous devez spécifier le nom de la route et des wildcards.

Avec ces informations, toute URL peut facilement être générée

```
class MainController extends AbstractController
{
    public function show($slug): Response
    {
        // ...
        $url = $this->generateUrl(
            'blog_show',
            array('slug' => 'my-blog-post')
        );
    }
}
```



Génération d'une URL absolue depuis un controller

Par défaut, le routeur va générer des URL relatives (par exemple /blog).

Il suffit de passer UrlGeneratorInterface::ABSOLUTE_URL comme 3eme argument à la méthode generateUrl();

```
use Symfony\Component\Routing\Generator\UrlGeneratorInterface;  
  
$this->generateUrl(  
    'blog_show',  
    ['slug' => 'my-blog-post'],  
    UrlGeneratorInterface::ABSOLUTE_URL  
);  
// http://www.example.com/blog/my-blog-post
```



Exercice

Créer un Controller **Deal**. Ce Controller aura le préfix **/deal** sur l'ensemble de ses routes.

Créer une action **Index** dans le Controller **Deal** :

- Elle répond sur l'url **/deal/list** et à “/”.
- Elle retourne un objet Response.
- Cette route portera le nom de **deal_list**
- Elle n'accepte que la méthode **GET**

Créer une action **Show** dans le Controller **Deal** :

- Elle répond sur l'url **/deal/show/<index>**.
- La route oblige que l'index soit un **digit**.
- Elle retourne un objet Response avec l'index dedans.
- Cette route portera le nom de **deal_show**.
- Elle n'accepte que la méthode **GET**.

Faire un php **bin/console debug:router** pour contrôler les routes.

Exercice

Créer une action **Show** dans le Controller **Deal** :

- Elle répond sur l'url **/deal/show/<index>**.
- La route oblige que l'index soit un **digit**.
- Elle retourne un objet Response avec l'index dedans.
- Cette route portera le nom de **deal_show**.
- Elle n'accepte que la méthode **GET**.

Faire un php **bin/console debug:router** pour contrôler les routes.

Vérifier le comportement de l'application :

- <http://127.0.0.1:8000> et <http://127.0.0.1:8000/deal/list>
- <http://127.0.0.1:8000/deal/show/1204>
- <http://127.0.0.1:8000/deal/show/azerty>
- <http://127.0.0.1:8000/deal/show/>

Database & Doctrine 2



C'est quoi Doctrine ?

- Projet Open Source PHP débuté en 2006
- Spécialisé dans la fonctionnalité de base de données
 - Database Abstraction Layer (DBAL)
 - Database Migrations
 - Object Relational Mapper (DBAL)
 - MongoDB Object Document Manager (ODM)
 - CouchDB Object Document Manager (ODM)



Configuration de la DB

Il faut configurer les informations de connexion de base de données. Par convention, cette information est généralement configuré dans le fichier .env

```
# .env
DATABASE_URL=mysql://db_user:db_password@127.0.0.1:3306/db_name

# config/packages/doctrine.yaml
doctrine:
    dbal:
        driver: 'pdo_mysql'
        server_version: '5.7'
        charset: utf8mb4
        default_table_options:
            charset: utf8mb4
            collate: utf8mb4_unicode_ci

        url: '%env(resolve:DATABASE_URL)%'
```



Configuration de la DB

En séparant les informations de base de données dans un fichier séparé, vous pouvez facilement garder différentes versions du fichier sur chaque serveur.

Vous pouvez aussi facilement stocker la configuration de base de données (ou toute information sensible) à l'extérieur de votre projet.

```
# app/config/config.yml
doctrine:
    dbal:
        driver: pdo_mysql
        host: "%database_host%"
        dbname: "%database_name%"
        user: "%database_user%"
        password: "%database_password%"
```



Création de la DB

Maintenant que la doctrine peut se connecter à la base de données, la commande suivante peut générer automatiquement une base de données

```
php bin/console doctrine:database:create
```



Entities (Entité)

- Objet métier (léger) persistant
- Classe PHP
- Pas besoin d'étendre une classe de Doctrine
- Ne peut pas être une classe « final »
- Prise en charge de l'héritage, des associations polymorphiques et des requêtes polymorphiques.
- Les classes à la fois abstraites et concrètes peuvent être des entités



Entity

La classe est souvent appelé une «entité».

Ce qui signifie une classe de base qui contient des données

Cette classe ne peut pas être encore persisté dans une base de données
Elle est juste une classe PHP simple.

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

```
class Product
{
    private $name;
    private $price;
    private $description;
}
```



Entity

Symfony peut créer des entités simples pour vous.

```
$ php bin/console make:entity
```

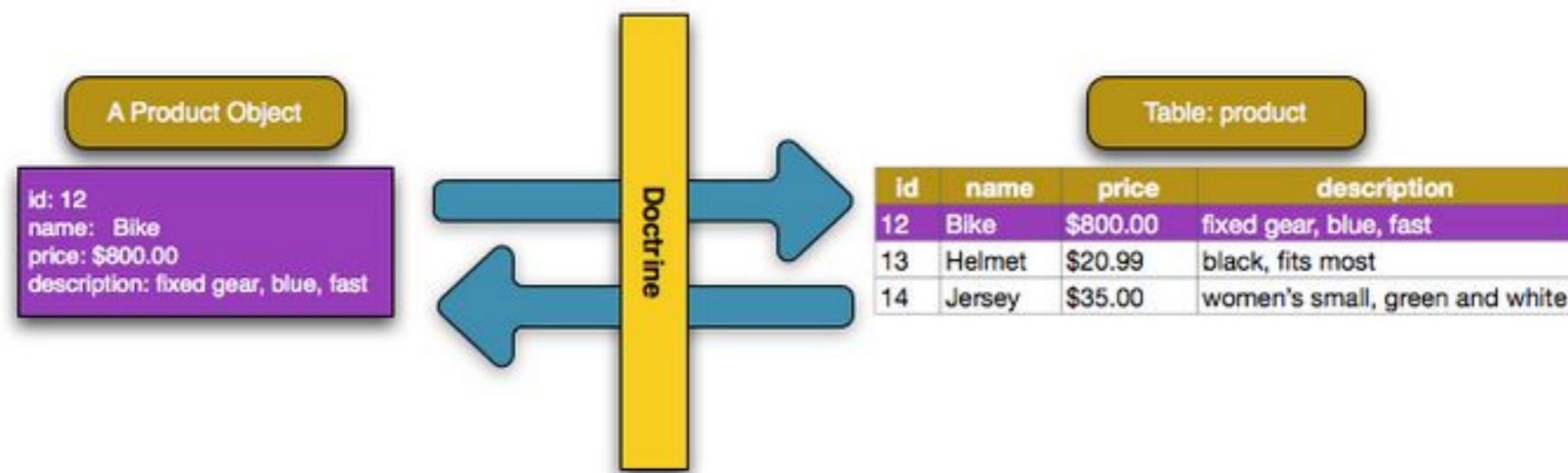


Information de mapping

Doctrine permet d'aller chercher des objets entiers sur la base de données, et de les persister.

Il faut mapper les tables de la DB avec des classes PHP spécifiques

Les colonnes de ces tables doivent être mappées à des propriétés de la classes PHP correspondantes.



Entity

Vous fournissez ces informations de mapping sous forme de métadonnées ».

Un ensemble de règles qui indique Doctrine exactement comment la classe et ses propriétés doivent être mappés.

Ces métadonnées peuvent être spécifiées dans un certain nombre de formats différents, y compris YAML, XML ou directement à l'intérieur de la classe de produit via les **annotations**.



Entity

```
// src/Entity/Product.php
namespace App\Entity;
use Doctrine\ORM\Mapping as ORM;
/**
 * @ORM\Entity
 * @ORM\Table(name="product")
 */
class Product
{
    /**
     * @ORM\Column(type="integer")
     * @ORM\Id
     * @ORM\GeneratedValue(strategy="AUTO")
     */
    private $id;
    /**
     * @ORM\Column(type="string", length=100)
     */
    private $name;
    /**
     * @ORM\Column(type="decimal", scale=2)
     */
    private $price;
    /**
     * @ORM\Column(type="text")
     */
    private $description;
}
```



Mapping Type simple

| Type | Description |
|------------------|---|
| string | Type qui mappe un VARCHAR SQL avec une chaîne dans PHP. |
| integer | Type qui mappe un INT SQL avec un entier de PHP. |
| smallint | Type qui mappe un SMALLINT avec un entier de PHP. |
| bigint | Type qui mappe un BIGINT SQL avec une chaîne dans PHP. |
| boolean | Type qui mappe un booléen SQL (ou TINYINT) avec un booléen dans PHP. |
| decimal | Type qui mappe un DECIMAL SQL avec une chaîne dans PHP. |
| date | Type qui mappe un DATETIME SQL avec un objet DateTime dans PHP. |
| time | Type qui mappe un TIME SQL avec un objet DateTime dans PHP. |
| datetime | Type qui mappe un DATETIME/TIMESTAMP SQL avec un objet DateTime dans PHP. |
| datetimez | Type qui mappe un DATETIME/TIMESTAMP SQL avec un objet DateTime dans PHP. |
| text | Type qui mappe un CLOB avec une chaîne dans PHP. |
| float | Type qui mappe un FLOAT de SQL à un double PHP. |
| blob | Type qui mappe un BLOB avec un flux resource PHP. |



Mapping Type complexe

| Type | Description |
|---------------------|--|
| object | Type qui mappe les types SQL CLOB à un objet PHP en utilisant serialize() et unserialize() |
| array | Type qui mappe les types SQL CLOB à un tableau PHP en utilisant serialize() et unserialize() |
| simple_array | Type qui mappe les types SQL CLOB à un tableau PHP en utilisant implode() et explode().
Ne peut contenir de ',' |
| json_array | Type qui mappe les types SQL CLOB à un tableau PHP en utilisant json_encode() et json_decode() |
| guid | Type qui mappe un GUID / UUID à une chaîne dans PHP. |



Getter et Setter

Même si la doctrine sait maintenant comment persister un objet produit dans la DB, la classe elle-même n'est pas encore vraiment utile.

Il faut générer les getter, setters car les propriétés sont privés.

La commande suivante peut générer ces méthodes automatiquement:

```
$ php bin/console make:entity AppBundle\Entity\Product
```



Creation du schema

Vous avez maintenant une classe `Product` utilisable avec des informations de mapping de telle sorte que la doctrine sait exactement comment la persister.

Bien sûr, vous ne possédez pas encore la table de produit correspondant dans votre base de données.

Doctrine peut créer automatiquement toutes les tables de base de données nécessaires pour chaque entité connue dans votre application avec cette commande.

```
php bin/console doctrine:schema:update --force
```



Persistace d'objets

```
// src/Controller/DefaultController.php
// ...
use App\Entity\Product;
use Symfony\Component\HttpFoundation\Response;

class DefaultController
{
    // ...
    public function create()
    {
        $product = new Product();
        $product->setName('Keyboard');
        $product->setPrice(19.99);
        $product->setDescription('Ergonomic and stylish!');
        $em = $this->getDoctrine()->getManager();
        // Dit à Doctrine que vous voulez (éventuellement) enregistrer le produit
        // (pas de requêtes encore)
        $em->persist($product);
        // Exécute réellement les requêtes(i.e. the INSERT query)
        $em->flush();

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



Lecture d'objets

Récupérer un objet depuis la DB est encore plus facile.

```
// src/Controller/DefaultController.php
// ...
public function show(($productId)
{
    $product = $this->getDoctrine()
        ->getRepository('AppBundle:Product')
        ->find($productId);

    if (!$product) {
        throw $this->createNotFoundException(
            'No product found for id ' . $productId
        );
    }
    // ...
}
```



Repository

Lorsque vous interrogez pour un type d'objet particulier, vous utilisez toujours ce qu'on appelle un "**repository**".

Vous pouvez penser à un dépôt comme une classe PHP dont le seul travail est de vous aider à aller chercher des entités d'une certaine classe.

```
$this->getDoctrine()  
->getRepository('AppBundle:Product')
```



Repository

Une fois que vous avez un objet **repository**, vous pouvez accéder à toutes sortes de méthodes utiles:

// Interroger pour un seul produit par sa clé primaire
\$product = \$repository->find(\$productId);

// Méthodes dynamiques pour trouver un produit basé sur une valeur de colonne
\$product = \$repository->findOneById(\$productId);
\$product = \$repository->findOneByName('Keyboard');

// Méthode dynamique pour trouver une liste de produit basé sur une valeur de colonne
\$products = \$repository->findByPrice(19.99);

// Trouver *tous* les produits
\$products = \$repository->findAll();



Update d'un objet

Une fois que vous avez chargé un objet Doctrine, la mise à jour est facile.

La Mise à jour d'un objet implique trois étapes:

1. Aller chercher l'objet dans Doctrine;
2. modification de l'objet;
3. appeler la méthode flush() de l'entity Manager

```
// src/Controller/DefaultController.php
public function update($productId)
{
    $em = $this->getDoctrine()->getManager();
    $product = $em->getRepository('AppBundle:Product')->find($productId);
    if (!$product) {
        throw $this->createNotFoundException(
            'No product found for id' . $productId
        );
    }
    $product->setName('New product name!');
    $em->flush();

    return $this->redirectToRoute('homepage');
}
```



Suppression d'un objet

La suppression d'un objet est très similaire, mais nécessite un appel à la méthode **remove()** de l'entity Manager

```
// src/Controller/DefaultController.php
public function delete($productId)
{
    $em = $this->getDoctrine()->getManager();
    $product = $em->getRepository('AppBundle:Product')->find($productId);
    if (!$product) {
        throw $this->createNotFoundException(
            'No product found for id ' . $productId
        );
    }
    $em->remove($product);
    $em->flush();

    return $this->redirectRoute('homepage');
}
```



Transactions et performance

```
for ($i = 0; $i < 20; ++$i) {  
    $user = new User;  
    $user->name = "John Doe");  
    $em->persist($user);  
    // $em->flush();  
}  
  
$em->flush();
```



Requêtes



Interrogation des objets avec le DQL

Vous pouvez utiliser le langage DQL, du « SQL like » pour construire une requête:

```
$em = $this->getDoctrine()->getManager();  
  
$query = $em->createQuery(  
    'SELECT p  
    FROM AppBundle:Product p  
    WHERE p.price > :price  
    ORDER BY p.price ASC'  
)->setParameter('price', 19.99);  
  
$products = $query->getResult();
```



Interrogation des objets avec le Query Builder

Le QueryBuilder simplifie la création des requêtes complexes / dynamiques

L'objet QueryBuilder contient toutes les méthodes nécessaires pour construire des requêtes.

```
$repository = $this->getDoctrine()
    ->getRepository('AppBundle:Product');
// createQueryBuilder selection automatiquement FROM AppBundle:Product
// et l'aliases sur "p"
$query = $repository->createQueryBuilder('p')
    ->where('p.price > :price')
    ->setParameter('price', '19.99')
    ->orderBy('p.price', 'ASC')
    ->getQuery();
$products = $query->getResult();
// Pour récupérer un seul résultat:
// $product = $query->setMaxResults(1)->getOneOrNullResult();
```



Custom Repository

Dans les sections précédentes, vous avez commencé à construire et utiliser des requêtes plus complexes à partir de l'intérieur d'un contrôleur. Afin d'isoler, de réutiliser et de tester ces requêtes, il est une bonne pratique pour créer une classe de référentiel personnalisé pour votre entité. Méthodes contenant la logique de votre requête peuvent alors être stockés dans cette classe.

Pour ce faire, ajoutez le nom de la classe de référentiel pour la définition de la cartographie de votre entité:

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

use Doctrine\ORM\Mapping as ORM;

/*
 * @ORM\Entity(repositoryClass="App\Repository\ProductRepository")
 */
class Product
{
//...
}
```



Custom Repository

Afin d'isoler, de réutiliser et de tester les requêtes (précédente), il existe une bonne pratique de créer une classe de repository personnalisé pour l'entity.

Les méthodes de la classe repository contiennent la logique des requêtes.

Pour ce faire, ajoutez le nom de classe repository dans l'annotation de l'entité:

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

use Doctrine\ORM\Mapping as ORM;

/*
 * @ORM\Entity(repositoryClass="App\Repository\ProductRepository")
 */

class Product
{
//...
}
```



Custom Repository

La classe repository contenant la logique métier

```
// src/Repository/ProductRepository.php
namespace App\Repository;

use Doctrine\ORM\EntityRepository;

class ProductRepository extends EntityRepository
{
    public function findAllOrderedByName()
    {
        return $this->getEntityManager()
            ->createQuery(
                'SELECT p FROM Product p ORDER BY p.name ASC'
            )
            ->getResult();
    }
}
```



Custom Repository

Vous pouvez utiliser cette nouvelle méthode, tout comme les méthodes du repository par défaut.

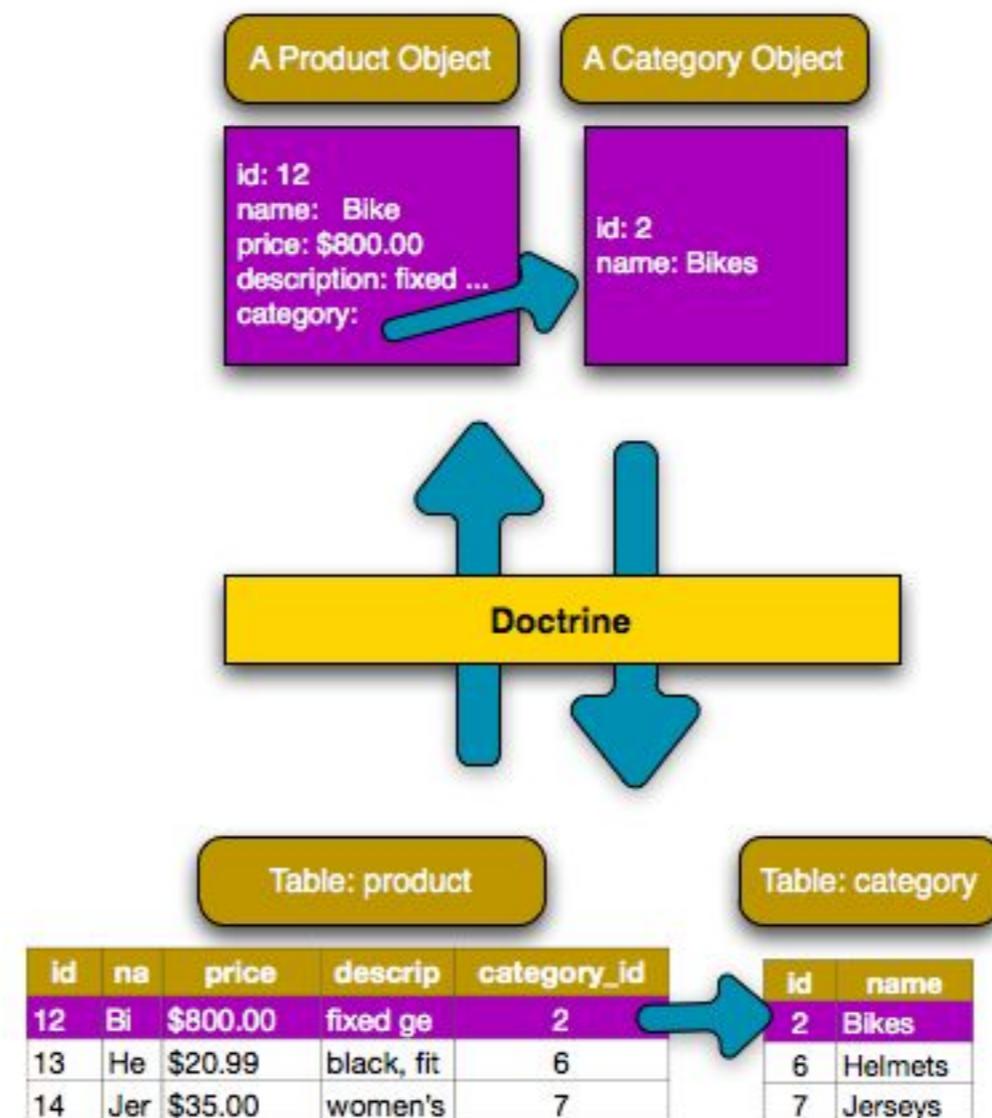
```
$em = $this->getDoctrine()->getManager();
$products = $em->getRepository(Product::class)
    ->findAllOrderedByName();
```



Relations / Associations



Relation / Association



Relation association

Doctrine gère la persistance de cette relation.

Il indique à Doctrine d'utiliser la colonne category_id sur la table product et de relier chaque enregistrement avec un enregistrement de la table category.

```
// src/Entity/Product.php
// ...
class Product
{
// ...
    /**
     * @ORM\ManyToOne(targetEntity="Category", inversedBy="products")
     * @ORM\JoinColumn(name="category_id", referencedColumnName="id")
     */
    private $category;
}
```



Mapping

Un objet Category sera associé à de nombreux objets Product, une propriété \$products peut être ajouté à la classe de Category pour contenir les objets associés.

```
// src/Entity/Category.php
// ...
use Doctrine\Common\Collections\ArrayCollection;

class Category
{
// ...
    /**
     * @ORM\OneToMany(targetEntity="Product", mappedBy="category")
    */
    private $products;

    public function __construct()
    {
        $this->products = new ArrayCollection();
    }
}
```



Enregistrement des entités parentes

Doctrine gère la persistance de cette relation.

```
class DefaultController extends AbstractController
{
    public function createProduct()
    {
        $category = new Category();
        $category->setName('Computer Peripherals');

        $product = new Product();
        $product->setName('Keyboard');
        $product->setPrice(19.99);
        $product->setDescription('Ergonomic and stylish!');

        // relate this product to the category
        $product->setCategory($category);
        $em = $this->getDoctrine()->getManager();

        $em->persist($category);
        $em->persist($product);
        $em->flush();
    }
}
```



Enregistrement des entités parentes

Créer automatiquement l'entité parente

```
/*
 * @ORM\ManyToOne(targetEntity="Category", inversedBy="products", cascade={"persist"})
 * @ORM\JoinColumn(name="category_id", referencedColumnName="id")
 */
private $category;
```

```
class DefaultController extends AbstractController
{
    public function createProduct()
    {
        $category = new Category();
        //...
        $product = new Product();
        //...

        $product->setCategory($category);
        $em = $this->getDoctrine()->getManager();

        // $em->persist($category);
        $em->persist($product);
        $em->flush();
    }
}
```



Lire les objets associés

Lorsque vous avez besoin d'aller chercher des objets associés.

- Il y a une première requête basé sur le productId.
- Cela « hydrate » l'objet \$product.
- Lors de l'appel \$product->getCategory()->getName(), Doctrine fait une seconde requête pour trouver la catégorie qui est lié à ce produit.
- Cela « hydrate » un objet Category.

```
public function show($productId)
{
    $product = $this->getDoctrine()
        ->getRepository(Product::class)
        ->find($productId);

    $categoryName = $product->getCategory()->getName();
    // ...
}
```



Detail des Associations



Associations

Doctrine supporte l'association entre les entités avec les stratégies suivantes:

- **Many To One**

- Unidirectionnel

- **One To One**

- Unidirectionnel
- Bidirectionnel
- Auto-référencement

- **One To Many**

- Unidirectionnel
- Bidirectionnel
- Auto-référencement

- **Many To Many**

- Unidirectionnel
- Bidirectionnel
- Auto-référencement



Associations

```
php bin/console make:entity
```

```
Field type (enter ? to see all types) [string]:  
[ > relation
```

```
What class should this entity be related to?:  
[ > Category
```

What type of relationship is this?

| Type | Description |
|------------|--|
| ManyToOne | Each Post relates to (has) one Category.
Each Category can relate/has to (have) many Post objects |
| OneToMany | Each Post relates can relate to (have) many Category objects.
Each Category relates to (has) one Post |
| ManyToMany | Each Post relates can relate to (have) many Category objects.
Each Category can also relate to (have) many Post objects |
| OneToOne | Each Post relates to (has) exactly one Category.
Each Category also relates to (has) exactly one Post. |

```
Relation type? [ManyToOne, OneToMany, ManyToMany, OneToOne]:
```



Many To One

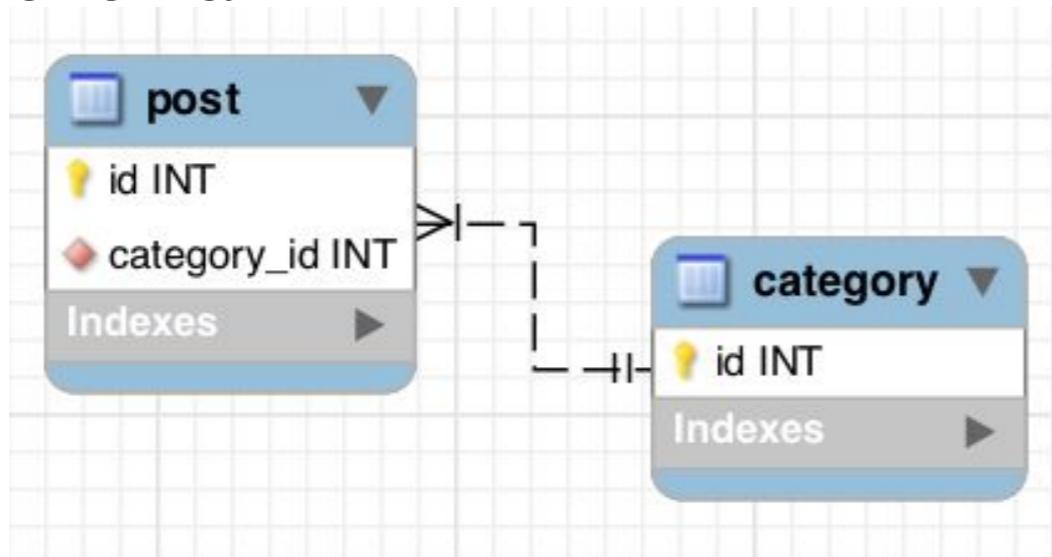
```
/**  
 * @ORM\Entity(repositoryClass="App\Repository\PostRepository")  
 */  
class Post  
{  
    /**  
     * @ORM\ManyToOne(targetEntity="App\Entity\Category", inversedBy="posts")  
     */  
    private $category;  
}
```

```
/**  
 * @ORM\Entity(repositoryClass="App\Repository\CategoryRepository")  
 */  
class Category  
{  
    /**  
     * @ORM\OneToMany(targetEntity="App\Entity\Post", mappedBy="category")  
     */  
    private $posts;  
  
public function __construct()  
{  
    $this->posts = new ArrayCollection();  
}  
}
```



Many To One

Schema



SQL

```
CREATE TABLE Post (
    id INT AUTO_INCREMENT NOT NULL,
    category_id INT DEFAULT NULL,
    PRIMARY KEY(id)
) ENGINE = InnoDB;

CREATE TABLE Category (
    id INT AUTO_INCREMENT NOT NULL,
    PRIMARY KEY(id)
) ENGINE = InnoDB;

ALTER TABLE User ADD FOREIGN KEY (category_id) REFERENCES Category(id);
```



One To Many

```
class Post
{
    /**
     * @ORM\OneToOne(targetEntity="App\Entity\Category", mappedBy="post")
     */
    private $categories;
```

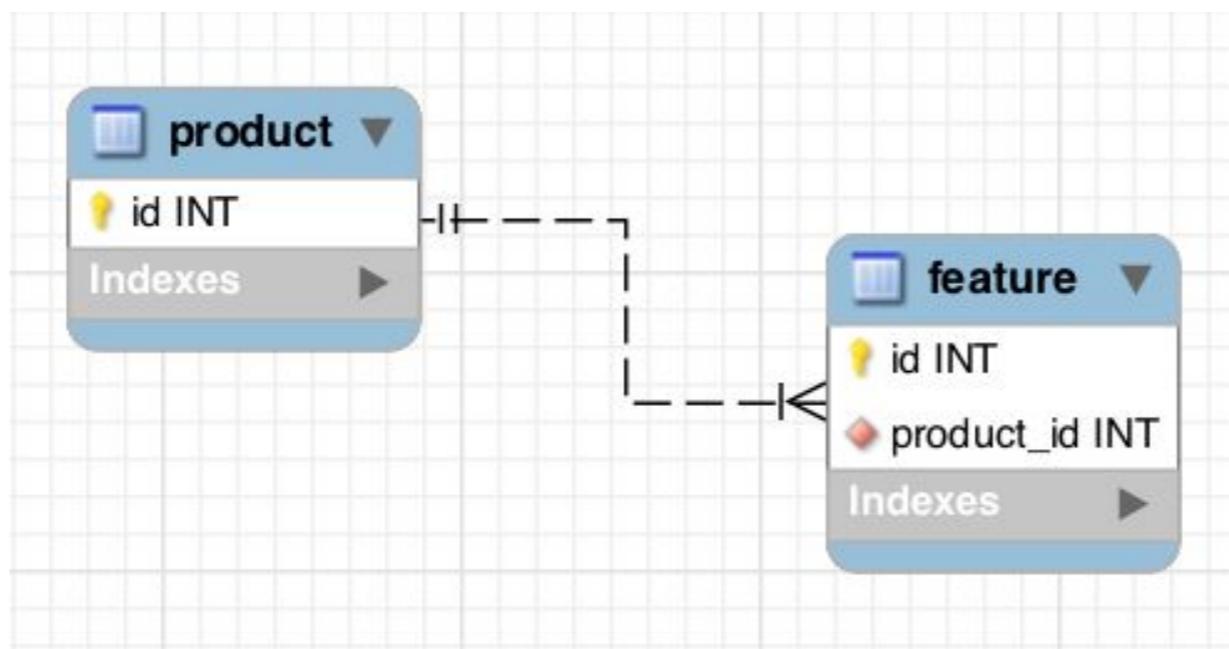
```
public function __construct()
{
    $this->categories = new ArrayCollection();
}
```

```
class Category
{
    /**
     * @ORM\ManyToOne(targetEntity="App\Entity\Post", inversedBy="categories")
     */
    private $post;
```



One To Many

Schema



SQL

```
CREATE TABLE Product (
    id INT AUTO_INCREMENT NOT NULL,
    PRIMARY KEY(id)
) ENGINE = InnoDB;

CREATE TABLE Feature (
    id INT AUTO_INCREMENT NOT NULL,
    product_id INT DEFAULT NULL,
    PRIMARY KEY(id)
) ENGINE = InnoDB;

ALTER TABLE Feature ADD FOREIGN KEY (product_id) REFERENCES Product(id);
```



Many To Many

```
class Post
{
    /**
     * @ORM\ManyToMany(targetEntity="App\Entity\Category", inversedBy="posts")
     */
    private $categories;

    public function __construct()
    {
        $this->categories = new ArrayCollection();
    }
}

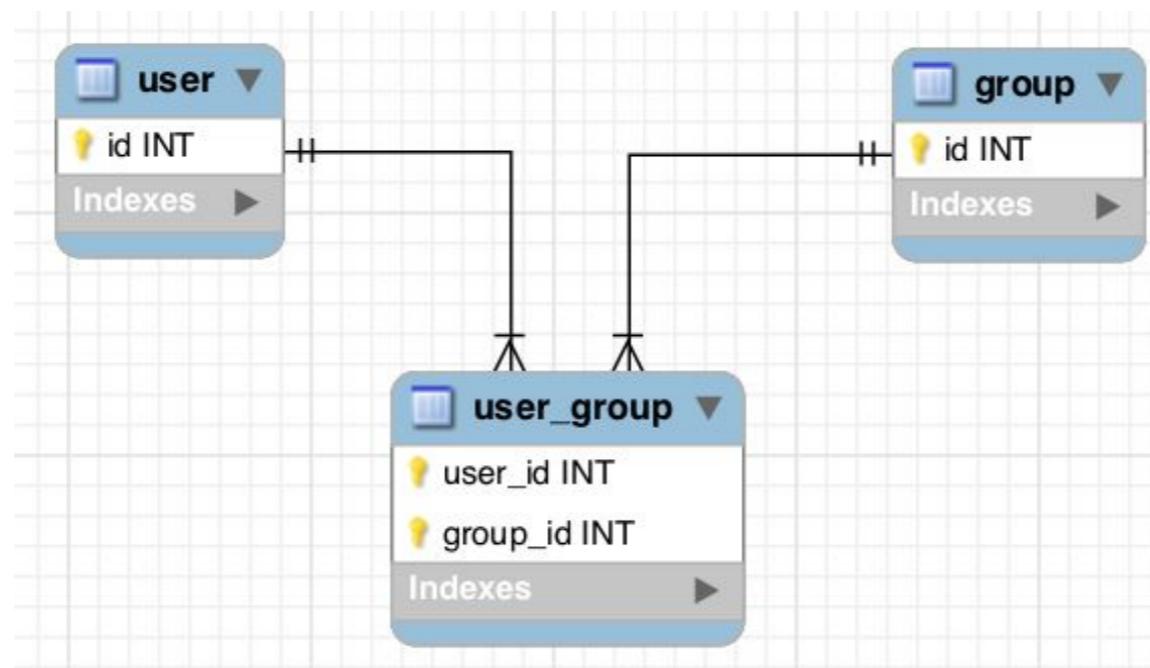
class Category
{
    /**
     * @ORM\ManyToMany(targetEntity="App\Entity\Post", mappedBy="categories")
     */
    private $posts;

    public function __construct()
    {
        $this->posts = new ArrayCollection();
    }
}
```



Many To Many

Schema



SQL

```
CREATE TABLE User (
    id INT AUTO_INCREMENT NOT NULL,
    PRIMARY KEY(id)
) ENGINE = InnoDB;

CREATE TABLE user_group (
    user_id INT NOT NULL,
    group_id INT NOT NULL,
    PRIMARY KEY(user_id, group_id)
) ENGINE = InnoDB;

CREATE TABLE Group (
    id INT AUTO_INCREMENT NOT NULL,
    PRIMARY KEY(id)
) ENGINE = InnoDB;

ALTER TABLE users_group ADD FOREIGN KEY (user_id) REFERENCES User(id);
ALTER TABLE users_group ADD FOREIGN KEY (group_id) REFERENCES Group(id);
```



One To One

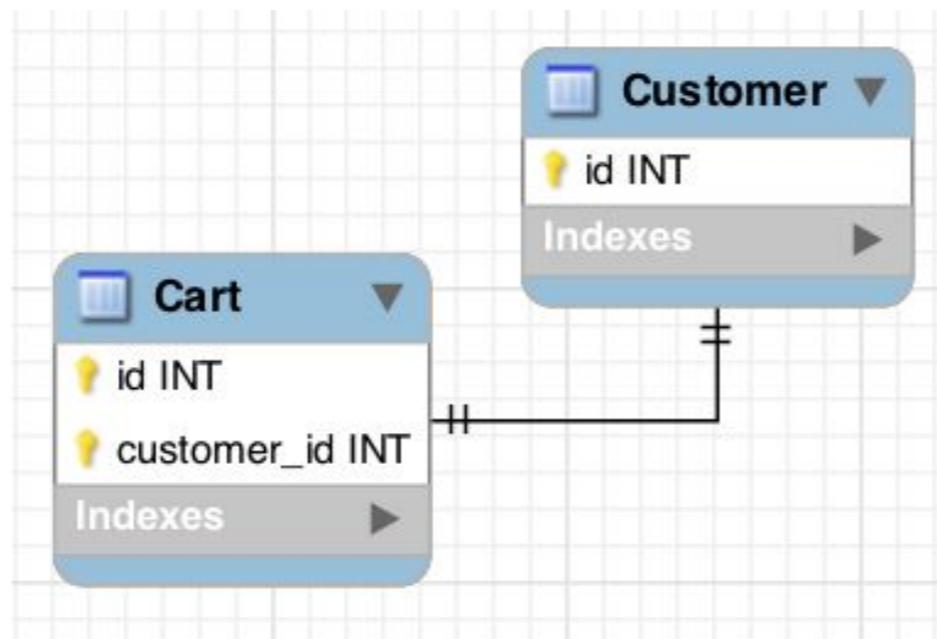
```
class Post
{
    /**
     * @ORM\OneToOne(targetEntity="App\Entity\Category", inversedBy="post", cascade={"persist",
    "remove"})
     */
    private $category;
}

class Category
{
    /**
     * @ORM\OneToOne(targetEntity="App\Entity\Post", mappedBy="category", cascade={"persist",
    "remove"})
     */
    private $post;
}
```



One To One

Schema



SQL

```
CREATE TABLE Cart (
    id INT AUTO_INCREMENT NOT NULL,
    customer_id INT DEFAULT NULL,
    PRIMARY KEY(id)
) ENGINE = InnoDB;

CREATE TABLE Customer (
    id INT AUTO_INCREMENT NOT NULL,
    PRIMARY KEY(id)
) ENGINE = InnoDB;

ALTER TABLE Cart ADD FOREIGN KEY (customer_id) REFERENCES Customer(id);
```



Lifecycle Callbacks



Lifecycle Callbacks

Parfois, vous avez besoin d'effectuer une action juste avant ou après une entité est inséré, mis à jour ou supprimé.

Doctrine déclenche des événements tout au long de la cycle de vie des objets qu'il gère.

- preRemove
- postRemove

- preLoad
- postLoad

- prePersist
- postPersist

- preUpdate
- postUpdate

```
/**
 * @ORM\Entity()
 * @ORM\HasLifecycleCallbacks()
 */
class Product
```



Lifecycle Callbacks

```
/**
 * @ORM\Entity()
 * @ORM\HasLifecycleCallbacks()
 */
class Product
{
    /**
     * @ORM\PrePersist
     */
    public function setCreatedAtValue()
    {
        $this->createdAt = new \DateTime();
    }
}
```



Fixtures

Fixtures

Les **Fixtures** sont utilisées pour charger des données “fake”.

Cela permet de revenir dans un état stable après des tests sur les données ou de construire un environnement utilisable **très rapidement**.

Afin d'installer les composants nécessaire, ajouter le bundle Fixtures :

```
composer require --dev doctrine/doctrine-fixtures-bundle
```

Écrire des Fixtures

Les Fixtures sont des **classes PHP** où on va créer des objets qu'on va **persister** dans la base de données.

Par exemple pour ajouter quelques produits objets à la base de données, il suffira d'écrire une classe fixture qui se chargera d'ajouter des produits.

Exemple d'écriture de Fixtures

```
<?php // src/DataFixtures/AppFixtures.php
namespace App\DataFixtures;

use App\Entity\Product;
use Doctrine\Bundle\FixturesBundle\Fixture;
use Doctrine\Common\Persistence\ObjectManager;

class AppFixtures extends Fixture {
    public function load(ObjectManager $manager) {
        // Création des 20 produits
        for ($i = 0; $i < 20; $i++) {
            $product = new Product();
            $product->setName('product' . $i);
            $product->setPrice(mt_rand(10, 100));
            $manager->persist($product);
        }
        $manager->flush();
    }
}
```

Chargement des Fixtures

L'avantage d'utiliser le bundle Fixtures est qu'il support ORM.

On peut donc charger les données avec la commande de load :

```
php bin/console doctrine:fixtures:load
```

Attention : par défaut la commande load **purge** la base de donnée.

Pour ajouter les données à l'existant il est possible d'utiliser l'option **--append**.

Split des Fixtures

Il est possible de créer toutes les Fixtures dans une seule classe.

Mais il est également possible de séparer la création des Fixtures en différentes classes afin d'éviter d'avoir des classes de création trop longue.

Dans le cas où un objet dépend d'un autre, il est possible de partager des objets entre les différentes Fixtures.

Pour cela il faut utiliser l'objet **Reference** :

La méthode **addReference()** pour donner un nom à l'objet.

La méthode **getReference()** pour récupérer le même l'objet grâce à son nom.

Fixtures utilisant addReference

```
// src/DataFixtures/UserFixtures.php
class UserFixtures extends Fixture
{
    public const ADMIN_USER_REFERENCE = 'admin-user';

    public function load(ObjectManager $manager)
    {
        $userAdmin = new User('admin', 'pass_1234');
        $manager->persist($userAdmin);
        $manager->flush();
        $this->addReference(self::ADMIN_USER_REFERENCE, $userAdmin);
    }
}
```

Fixtures utilisant getReference

```
// src/DataFixtures/GroupFixtures.php
class GroupFixtures extends Fixture
{
    public function load(ObjectManager $manager)
    {
        $userGroup = new Group('administrators');
        $userGroup->addUser($this->getReference(UserFixtures::ADMIN_USER_REFERENCE))
        $manager->persist($userGroup);
        $manager->flush();
    }
}
```

Dépendance des Fixtures

Par défaut le chargement des Fixtures par Doctrine se fait dans l'**ordre alphabétique**.

L'exemple précédent ne fonctionnera donc pas car les Fixtures Group se feront avant les Fixtures de User.

Il est possible de définir des dépendances entre les Fixtures en implémentant l'interface **DependentFixtureInterface** et en ajoutant une méthode **getDependencies()** à la classe Fixtures.

Cette méthode doit retourner un array de classes à charger avant celle là.

Exemple à ajouter sur la classe Group :

```
public function getDependencies() {  
    return array(UserFixtures::class);  
}
```

Exercice 3.1

Créer l'entité **Category** avec l'outil **make**.

Elle doit avoir comme propriété :

- name (string, non null, **unique**)

Créer l'entité **Deal** avec l'outil **make**.

Elle doit avoir comme propriétés :

- name (string, non null)
- description (text, non null)
- created_at (datetime, non null)
- updated_at (datetime)
- price (float, non null)
- enable (boolean, non null)

Observer le fichier généré dans src/Entity/Deal.php

Modifier l'entité pour avoir un **created_at automatiquement** lors de la création de l'objet.

Exercice 3.2

Créer l'entité **Deal** avec l'outil **make**.

Elle doit avoir comme propriétés :

- name (string, non null)
- description (text, non null)
- created_at (datetime, non null)
- updated_at (datetime)
- price (float, non null)
- enable (boolean, non null)

Observer le fichier généré dans src/Entity/Deal.php

Modifier l'entité pour avoir un **created_at automatiquement** lors de la création de l'objet.

Exercice 3.3

- Créer une relation **ManyToMany** entre **Deal** et **Category** :
 - Ajouter la propriété **categories** à l'entity **Deal**
 - Ajouter la propriété **deals** à l'entity **Category**

Exercice 3.4

- Configurer la base de donnée
- Créer la base de données avec les entités avec **doctrine**.

Exercice 3.5

Créer des **Fixtures** pour avoir des données de tests.

Créer deux classes :

- La classe **CategoryFixtures**
 - Elle ajoute au moins **deux objets Category** dans la base de donnée.
 - Chaque objet est ajouté en référence pour l'utiliser dans les Deal.
- La classe **DealFixtures**
 - Elle ajoute au moins **quatre objets Deal** dans la base de donnée.
 - Chaque objet Deal doit être **associé à au moins une Category** qu'on récupère des références.

Exercice 3.6

Créer une action **toggleEnableAction** dans le controller du **Deal**.

L'action prend en paramètre un **id**. Créer une route qui match avec :
deal/toggle/<index>

Vérifier que l'id **existe** bien dans la base de donnée.
Une exception est levé si ce n'est pas le cas.

Elle permet de basculer le **enable** entre **false** et **true**.

Elle affiche le nouveau résultat.

Injection de dépendance / Service Container



La vie sans l'injection de dépendance



Une classe de Logging

```
class Logger
{
    public function log($message, $level = 'INFO')
    {
        $formatter = new XmlFormatter();
        $log = $formatter->format($message, $level);
        $this->writeLog($log);
    }
}
```



Problèmes ?

- Ne supporte que les logs au format XML
- Dépendance hardcodé sur la class XmlFormatter
- Non flexible
- Les tests unitaires sont plus difficile



Un pas de plus vers l'Injection de Dependence



Retrait de la dépendance

```
class Logger
{
    private $formatter;

    public function __construct(XmlFormatter $formatter)
    {
        $this->formatter = $formatter;
    }

    public function log($message, $level = 'INFO')
    {
        $log = $this->formatter->format($message, $level);
        $this->writeLog($log);
    }
}
```



Retrait de la dépendance

```
$logger = new Logger(new XmlFormatter());  
$logger->log('An error to log');  
  
/**  
 * <log>  
 *   <message>An error to log</message>  
 * </log>  
 */
```



Améliorer la flexibilité

```
class Logger
{
    private $formatter;

    public function __construct(Formatter $formatter)
    {
        $this->formatter = $formatter;
    }

    //...
}
```



Améliorer la flexibilité

```
$logger = new Logger(new JsonFormatter());  
$logger->log('An error to log');
```

```
/**  
 * { message: "An error to log" }  
 */
```



Injection de Dependence



What's Dependency Injection?

« L'Injection de dépendance est où les composants indiquent leurs dépendances par leurs constructeurs, méthodes, ou directement dans les champs. »

<http://picocontainer.org/injection.html>



Injection par le constructeur

```
class Logger
{
    private $formatter;

    public function __construct(Formatter $formatter)
    {
        $this->formatter = $formatter;
    }

    //...
}
```



Injection par une méthode

```
class Logger
{
    private $formatter;

    public function setFormatter(Formatter $formatter)
    {
        $this->formatter = $formatter;
    }

    //...
}
```



Injection d'une Interface

```
class Logger
{
    private $formatter;

    public function setFormatter(FormatterInterface $formatter)
    {
        $this->formatter = $formatter;
    }

    /**
     */
}
```



Problème introduit

Le code devient plus découplé et testable mais la construction et l'initialisation des objets deviennent plus complexe ...



The Service Container



Service Container

Ce chapitre est sur un objet PHP Symfony spécial qui vous aide à instancier, organiser et récupérer les nombreux objets de votre application.

Cet objet, appelé le **service container** permet de normaliser et de centraliser la façon dont les objets sont construits dans votre application.

Le container met l'accent sur une architecture qui favorise le code réutilisable et découplé.

Toutes les classes de base de Symfony utilisent le container.



Qu'est ce qu'un service?

Un service est un objet de PHP qui effectue **une sorte de tâche "global"**.

Il est un nom générique utilisé délibérément en informatique pour décrire un objet qui est créé dans un but précis (par exemple fournir des e-mails).

Chaque service est utilisé tout au long de l'application à chaque fois que vous avez besoin des fonctionnalités spécifiques qu'il fournit.

Pour faire un service, il suffit d'écrire une classe PHP avec un code qui accomplit une tâche spécifique.

Structurer l'application autour d'un ensemble de classes de services indépendants est une (très) bonne pratique.



Configuration par default

```
# config/services.yaml
services:
    _defaults:
        autowire: true      # Automatically injects dependencies in your services.
        autoconfigure: true # Automatically registers your services as commands, event subscribers, etc.
        public: false       # Allows optimizing the container by removing unused services; this also means
                            # fetching services directly from the container via $container->get() won't work.
                            # The best practice is to be explicit about your dependencies anyway.
```

App\:

```
resource: '../src/*'
exclude: '../src/{DependencyInjection,Entity,Migrations,Tests,Kernel.php}'
```

```
# controllers are imported separately to make sure services can be injected
# as action arguments even if you don't extend any base controller class
```

App\Controller\:

```
resource: '../src/Controller'
tags: ['controller.service_arguments']
```



Autowiring: true

Cette directive dit à Symfony de regarder le constructeur et les éventuels “setters” de vos services, et d'y injecter *le* service correspondant au type requis par leurs arguments

```
# Symfony 3
# app/config/services.yml
services:
    app.mailer:
        class: App\Service\Mailer
    app.newsletter_manager:
        class: App\Newsletter\NewsletterManager
        arguments: ['@app.mailer']
```

```
# Symfony 4
# config/services.yml
services:
    App\:
        resource: '../src/*'
```



Autowiring: true

```
// src/App/Service/Mailer.php
use App\Service;

class Mailer
{
    public function send($text)
    {
        //..
    }
}

// src/App/Newsletter/NewsletterManager.php
namespace App\Newsletter;

use App\Service\Mailer;

class NewsletterManager
{
    protected $mailer;

    public function __construct(Mailer $mailer)
    {
        $this->mailer = $mailer;
    }
    // ...
}
```



Autoconfigure: true

Lorsque cette option est activée, Symfony va regarder les types implémentés par chacun de vos services (classes et interfaces), et leur appliquer des tags automatiquement en fonction. Ainsi, si une de vos classes implémente Command, le service correspondant se verra attribuer le tag `console.command`.

```
#config/services.yml
services:
    App\Task\:
        resource: '../src/Task'
        tags: ['app.task']
```



Définition partielle des services

Il est également possible de définir partiellement un service, en laissant à l'autowiring le soin de compléter :

```
#config/services.yml
services:
  App\Newsletter\NewsletterManager:
    arguments:
      $dryRun: true
```



bind

```
// src/App/Newsletter/NewsletterManager.php
namespace App\Newsletter;

use App\Service\Mailer;

class NewsletterManager
{
    protected $mailer;

    public function __construct(Mailer $mailer, bool $dryRun=false)
    {
        $this->mailer = $mailer;
        // $dryRun vaudra true
    }
    // ...
}
```



bind

Définition de « variables » applicable à l'ensembles des services.

Désormais, pour tous les services définis, si un argument porte le nom `$dryRun`, sa valeur sera injectée comme spécifié.

```
#config/services.yml
services:
    _defaults:
        autowire: true # Auto-detection des services à injecter
        autoconfigure: true #Applique des tags en fonctions des classes étendues
        public: false #Chaque service inutilisé doit être nettoyé
    bind:
        $dryRun: true
        $recipient: "%env(RECIPIENT)%"
```



Qu'est ce que le service container?

Le **service container** (ou *dependency injection container*) est simplement un objet PHP qui gère l'instanciation des services (à savoir des objets).

Sans service container:

```
use App\Mailer;
```

```
$mailer = new Mailer('sendmail');  
$mailer->send('jdoe@example.com', ...);
```



Créer / Configurer un service

Une meilleure réponse est de laisser le conteneur de service créer l'objet Mailer pour vous.

Pour que cela fonctionne, il faut « enseigner » au container comment créer le service Mailer.

Cela se fait par l'intermédiaire de configuration (YAML, XML ou PHP)

```
<!-- services.xml -->
<services>
    <service id="App\Mailer" class="App\Mailer">
        <argument id="sendmail" type="service"/>
    </service>
</services>
```

```
# services.yml
services:
    App\Mailer:
        arguments: [sendmail]
```



Utilisation d'un service

Lorsque vous appelez le service « app.mailer » du container, le container construit l'objet et le renvoie.

Un service n'est jamais construit jusqu'à ce qu'il soit nécessaire.

Le service est créé une seule fois et la même instance est retourné à chaque fois que vous le demandez.

```
class HelloController extends AbstractController
{
    // ...
    public function sendEmail(App\Mailer $mailer)
    {
        // ...
        $mailer->send('ryan@foobar.net', ...);
    }
}
```



Les paramètres d'un service

Définir des paramètres dans les services, les rend plus organisés et flexibles

En enfermant la chaîne de app .mailer.transport avec des pourcents (%), le container sait qu'il doit trouver un paramètre avec ce nom.

Lorsque le container est construit, il remplace le paramètre défini du service par sa valeur.

```
# app/config/services.yml
parameters:
    app.mailer.transport: sendmail

services:
    App\Mailer:
        arguments: [%app.mailer.transport%]
```



Injecter des services

La classe « NewsletterManager » va utiliser la classe « Mailer » pour envoyer tous les mails.

```
// src/Newsletter/NewsletterManager.php
namespace App\Newsletter;

use App\Mailer;

class NewsletterManager
{
    protected $mailer;
    public function __construct(Mailer $mailer)
    {
        $this->mailer = $mailer;
    }
    // ...
}
```



Injecter des services

En YAML, la syntaxe spéciale `@app.mailer` indique au container de trouver un service nommé `app.mailer` et de transmettre cet objet au constructeur du service `NewsletterManager`.

Cependant, le service `app.mailer` doit exister. Si ce n'est pas le cas, une exception sera levée.

```
# app/config/services.yml
services:
    app.mailer:
        # ...
        App\Newsletter\NewsletterManager
        arguments: ['@app.mailer']
```



Injecter des services via « expression language »

Le container de services prend en charge une "expression" qui permet d'injecter des valeurs très spécifiques dans un service.

```
# app/config/config.yml
services:
  my_mailer:
    class: App\Mailer
    arguments: ["@=service('mailer_configuration').getMailerMethod()"]

<!-- services.xml -->
<service id="my_mailer" class="App\Mailer">
  <argument type="expression">service('mailer_configuration').getMailerMethod()</argument>
</service>

services:
  my_mailer:
    class: App\Mailer
    arguments: ["@=container.hasParameter('some_param') ? parameter('some_param') : 'default_value'"]
```



Accéder à l'objet Request dans un service

Si vous avez besoin d'accéder à la *Request* courante dans un service.
Il faut injecter le service *request_stack* et appeler la méthode
`Request::getCurrentRequest()`

```
namespace App\Newsletter;  
  
use Symfony\Component\HttpFoundation\RequestStack;  
  
class NewsletterManager  
{  
    protected $requestStack;  
  
    public function __construct(RequestStack $requestStack)  
    {  
        $this->requestStack = $requestStack;  
    }  
  
    public function anyMethod()  
    {  
        $request = $this->requestStack->getCurrentRequest();  
        //...  
    }  
}
```



Mettre une référence comme facultative

Un des services peut avoir une dépendance facultative, ce qui signifie que la dépendance n'est pas nécessaire pour que votre service fonctionne correctement.

Définition des dépendances manquantes à null

```
<!-- app/config/services.xml -->
<?xml version="1.0" encoding="UTF-8" ?>
<container xmlns="http://symfony.com/schema/dic/services"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://symfony.com/schema/dic/services
http://symfony.com/schema/dic/services/services-1.0.xsd">
    <services>
        <service id="app.mailer">
            <!-- ... -->
        </service>
        <service id="app.newsletter_manager" class="App\Newsletter\NewsletterManager">
            <argument type="service" id="app.mailer" on-invalid="null" />
        </service>
    </services>
</container>
```



Tags

Les services configurés dans un conteneur peuvent également être taggés. Dans le conteneur de service, un tag implique que le service est destiné à être utilisé dans un but spécifique.

Les tags sont une façon de dire à Symfony ou à d'autres bundles tiers que votre service doit être enregistré ou utilisé de façon particulière par le bundle.

```
# app/config/services.yml
services:
    foo.twig.extension:
        class: App\Extension\FooExtension
        public: false
        tags:
            - { name: twig.extension }
```



« Transformer » un formulaire en service

Il faut créer un service et de lui associer le tag « form.type »

```
<!-- src/Resources/config/services.xml -->
<!-- ... -->
<service id="app.form.type.task" class="App\Form\TaskType">
    <tag name="form.type" />
</service>
```

La construction reste identique

```
// src/Controller/DefaultController.php
namespace App\Controller;
// ...
use App\Form\TaskType;

class DefaultController extends AbstractController
{
    public function new()
    {
        $task = new Task();
        $form = $this->createForm(TaskType::class, $task);

        //...
    }
}
```



Service « abstrait » // Service parent

Vous pouvez avoir des classes liées qui partagent les mêmes dépendances. Par exemple, vous pouvez avoir plusieurs classes qui ont besoin du service doctrine.entity_manager

```
<!-- Comme aucune classe n'est configurée, le service parent DOIT être résumé-->
<service id="app.base_doctrine_repository" abstract="true">
    <argument type="service" id="doctrine.entity_manager" />
</service>

<!-- Étend le service app.base_doctrine_repository -->
<service id="app.user_repository"
    class="App\Repository\DoctrineUserRepository"
    parent="app.base_doctrine_repository"
/>

<service id="app.post_repository"
    class="App\Repository\DoctrineUserRepository"
    parent="app.base_doctrine_repository"
/>
```



Debugger un service

```
php app/console container:debug  
--show-private
```

[container] Public services	Service Id	Scope	Class Name
acme.demo.listener	acme.demo.listener	container	Acme\DemoBundle\ControllerListener
annotation_reader	annotation_reader	container	Doctrine\Common\Annotations\FileCacheReader
assetic.asset_manager	assetic.asset_manager	container	Assetic\Factory\LazyAssetManager
assetic.controller	assetic.controller	prototype	Symfony\Bundle\AsseticBundle\Controller\AsseticController
assetic.filter.cssrewrite	assetic.filter.cssrewrite	container	Assetic\Filter\CssRewriteFilter
assetic.filter_manager	assetic.filter_manager	container	Symfony\Bundle\AsseticBundle\FilterManager
assetic.request_listener	assetic.request_listener	container	Symfony\Bundle\AsseticBundle\EventListener\RequestListener
cache_warmer	cache_warmer	container	Symfony\Component\HttpKernel\CacheWarmer\CacheWarmerAggregate
controller_resolver	controller_resolver	container	Symfony\Bundle\FrameworkBundle\Controller\TraceableControllerResolver
data_collector.request	data_collector.request	container	Symfony\Bundle\FrameworkBundle\DataCollector\RequestDataCollector
database_connection	n/a alias for doctrine.dbal.default_connection	n/a	alias for doctrine.dbal.default_connection
debug.controller_resolver	debug.controller_resolver	n/a	alias for controller_resolver
debug.event_dispatcher	debug.event_dispatcher	n/a	alias for event_dispatcher
debug.stopwatch	debug.stopwatch	container	Symfony\Component\HttpKernel\Debug\Stopwatch
debug.templating.engine.twig	debug.templating.engine.twig	n/a	alias for templating
doctrine	doctrine	container	Symfony\Bundle\DoctrineBundle\Registry
doctrine.dbal.connection_factory	doctrine.dbal.default_connection	container	Symfony\Bundle\DoctrineBundle\ConnectionFactory
doctrine.dbal.default_connection	doctrine.dbal.default_connection	container	stdClass
doctrine.orm.default_entity_manager	doctrine.orm.entity_manager	container	Doctrine\ORM\EntityManager
doctrine.orm.entity_manager	n/a alias for doctrine.orm.default_entity_manager	n/a	alias for doctrine.orm.default_entity_manager
doctrine.orm.validator.unique	doctrine.orm.validator.unique	container	Symfony\Bridge\Doctrine\Validator\Constraints\UniqueEntityValidator
doctrine.orm.validator_initializer	doctrine.orm.validator_initializer	container	Symfony\Bridge\Doctrine\Validator\DoctrineInitializer
event_dispatcher	event_dispatcher	container	Symfony\Bundle\FrameworkBundle\Debug\TraceableEventDispatcher
file_locator	file_locator	container	Symfony\Component\HttpKernel\Config\FileLocator
filesystem	filesystem	container	Symfony\Component\HttpKernel\Util\Filesystem
form.csrf_provider	form.csrf_provider	container	Symfony\Component\Form\Extension\Csrf\CsrfProvider\SessionCsrfProvider
form.factory	form.factory	container	Symfony\Component\Form\FormFactory
form.type.birthday	form.type.birthday	container	Symfony\Component\Form\Extension\Core\Type\BirthdayType
form.type.checkbox	form.type.checkbox	container	Symfony\Component\Form\Extension\Core\Type\CheckboxType
form.type.choice	form.type.choice	container	Symfony\Component\Form\Extension\Core\Type\ChoiceType
form.type.collection	form.type.collection	container	Symfony\Component\Form\Extension\Core\Type\CollectionType
form.type.country	form.type.country	container	Symfony\Component\Form\Extension\Core\Type\CountryType
form.type.csrf	form.type.csrf	container	Symfony\Component\Form\Extension\Csrf\Type\CsrfType

Vous pouvez obtenir des informations plus détaillées sur un service particulier en spécifiant son identifiant:

```
php bin/console debug:container app.mailer
```



Exercice

- Créer un Service **RandomSlogan**.
Ce service aura une méthode **getSlogan** qui enverra un slogan aléatoire.
- Faire une injection du **Logger** dans le constructeur du service
- Injecter le Service **RandomSlogan** dans un Controller et afficher le message aléatoire dans le template Twig.

Templeting avec Twig



<http://twig.sensiolabs.org>

a SENSIOLABS  product

TWIG

The flexible, fast, and secure template engine for PHP

ABOUT DOCUMENTATION BLOG DEVELOPMENT CONTRIBUTORS



Twig is a modern template engine for PHP

- **Fast:** Twig compiles templates down to plain optimized PHP code. The overhead compared to regular PHP code was reduced to the very minimum.
- **Secure:** Twig has a sandbox mode to evaluate untrusted template code. This allows Twig to be used as a template language for applications where users may modify the template design.
- **Flexible:** Twig is powered by a flexible lexer and parser. This allows the developer to define its own custom tags and filters, and create its own DSL.

[LEARN MORE →](#)

[INSTALL NOW ↓](#)

Stable: [1.13.1](#)



Twig

```
<?php include 'header.php' ?>
```

Lorem ipsum dolor sit amet, consectetur adipisicing 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

- Foo
- Bar
- Baz
- Ping
- Pong

```
Home > Blog ←----- Contextual content  


php foreach ($posts as $post) : ?&gt;<br/  <h2>  
    <?php echo htmlspecialchars($post['title']) ?>  
  </h2>  
  <?php include 'post.php' ?> ←----- No isolation  
  <?php echo substr($post['body'], 255).'...' ?>  
  <?php endforeach ?>  
</div>


```

Manual escaping

No separation of concerns

```
<?php include 'footer.php' ?>
```



Syntaxe



Syntaxe concise

{# ... commente quelque chose ... #}

{% ... fait quelque chose ... %}

{{ ... affiche quelque chose ... }}



Variable



Afficher une variable

{{ article.titre }} =>

- echo \$article['titre'];
- echo \$article->titre;
- echo \$article->titre();
- echo \$article->getTitre();
- echo \$article->isTitre();



Définition d'une variable

Vous pouvez attribuer des valeurs aux variables à l'intérieur des blocs de code en utilisant la balise **set**

```
{% set foo = 'foo' %}  
{% set foo = [1, 2] %}  
{% set foo = {'foo': 'bar'} %}
```



Filtres et fonctions



Les filters

Les variables peuvent être modifiées par les **filters**.

Les filters sont séparés de la variable par un symbole *pipe* (|) et peut avoir des arguments optionnels entre parenthèses.

Plusieurs filters peuvent être **enchaînés**.

La sortie d'un filtre est **appliquée à l'autre**.



Les filtres

L'exemple suivant supprime toutes les balises et mets le texte en majuscule

```
{{ '<h1>Nicolas</h1>'|striptags|upper }}
```

Les filtres qui acceptent des arguments ont des parenthèses autour des arguments.

```
{{ list|join(',') }}
```

<http://twig.sensiolabs.org/doc/filters/index.html>



Les functions

Les fonctions peuvent être appelées pour générer du contenu. Les fonctions sont appelées par leur nom suivi de parenthèses et peuvent présenter des arguments.

Affiche la valeur la plus petite de la séquence

```
{{ min(1, 3, 2) }}
```

```
{{ constant('DATE_W3C') }}
```

On peut lire les constantes depuis l'instance d'un objet

```
{{ constant('RSS', date) }}
```



Génération d'un lien

La fonction twig « **path()** » genre une URL relative

```
<a href="{{ path('blog_show', {'slug': 'my-blog-post'}) }}">  
    Read this blog post.  
</a>
```



Les functions "Arguments nommés"

```
{% for i in range(low=1, high=10, step=2) %}  
    {{ i }},  
{% endfor %}
```

En utilisant les arguments nommés cela rend vos template plus explicite sur le sens des valeurs que vous passez comme arguments

```
{ data|convert_encoding('UTF-8', 'iso-2022-jp') }  
{# versus #}  
{ { data|convert_encoding(from='iso-2022-jp', to='UTF-8') }}
```



Les functions "Arguments nommés"

Les arguments nommés vous permettent également de sauter certains arguments pour lesquels vous ne voulez pas changer la valeur par défaut valeur:

{# le premier argument est le format de date, ce qui est par défaut le format global de date si null est passé #}

```
{{ "now"|date(null, "Europe/Paris") }}
```

{# ou ignorer la valeur de format en utilisant un argument nommé pour le fuseau horaire #}

```
{{ "now"|date(timezone="Europe/Paris") }}
```



Filtrages de la sortie



Filtrage automatique

Hello {{ name }}!

La variable est automatiquement filtré si elle contient une chaîne de caractère



Filtrage automatique

Hello {{ name }}!

name = John

Sortie attendue:

John

Vraie sortie:

John

;



Filtrages de la sortie

Si échappement manuel est activé, il est de votre responsabilité d'échapper les variables si nécessaire.

En utilisant le filtre « **e** »

```
{{ user.username|e }}
```



Filtrages de la sortie

Par défaut, le filtre d'échappement utilise la stratégie html, mais en fonction du contexte échappement, vous voudrez peut-être d'utiliser explicitement d'autres stratégies disponibles

```
 {{ user.username|e('js') }}  
 {{ user.username|e('css') }}  
 {{ user.username|e('url') }}  
 {{ user.username|e('html_attr') }}
```



Structure de contrôle



Prendre une décision

Une structure de contrôle se réfère à toutes ces choses qui contrôlent le flux d'un programme - conditionals (if / elseif / else), les boucles ...

Les structures de contrôle apparaissent à l'intérieur des tags { % ... % }.

{# pour afficher une liste d'utilisateurs prévus dans une variable appelée users, utilisez le tag for #}

```
{% if users|length > 0 %}
<h1>Members</h1>
<ul>
    {% for user in users %}
        <li>{{ user.username|e }}</li>
    {% endfor %}
</ul>
{% endif %}
```



Boucles imbriquées

Parfois, lors de l'utilisation de boucles imbriquées, vous devez accéder au contexte parent. Le contexte parent est toujours accessible via la variable **loop.parent**.

```
{% for topic, messages in topics %}
```

```
* {{ loop.index }}: {{ topic }}
```

```
{% for message in messages %}
```

```
- {{ loop.parent.loop.index }}.{{ loop.index }}: {{ message }}
```

```
{% endfor %}
```

```
{% endfor %}
```

```
$data = array(  
    'topics' => array(  
        'topic1' => array('Message 1 of topic 1', 'Message 2 of topic 1'),  
        'topic2' => array('Message 1 of topic 2', 'Message 2 of topic 2'),  
    ),  
);
```



Boucles imbriquées

La sortie sera:

- * 1 : topic1
 - 1.1 : The message 1 of topic 1
 - 1.2 : The message 2 of topic 1
- * 2 : topic2
 - 2.1 : The message 1 of topic 2
 - 2.2 : The message 2 of topic 2



Environnements

Variables	Descriptions
loop.index	L'iteration courante dans la boucle. (commence à 1)
loop.index0	L'iteration courante dans la boucle. (commence à 0)
loop.revindex	Le nombre d'iteration depuis la fin de la boucle(commence à 1)
loop.revindex0	Le nombre d'iteration depuis la fin de la boucle(commence à 0)
loop.first	True s'il s'agit de la premiere itération
loop.last	True s'il s'agit de la dernière itération
loop.length	Le nombre d'élément de la séquence
loop.parent	Le contexte du parent



Génération d'une URL absolue depuis Twig

La fonction twig « **url()** » fonctionne de la même manière que « **path()** », mais génère des URLs absolues

Dans un flux RSS ou
un email

```
<a href="#">{ { url('blog_show', { 'slug': 'my-blog-post' }) } } >
```

Read this blog post.

```
</a>
```



Expressions



Expressions

Twig permet d'utiliser des expressions partout. Celle-ci ont un fonctionnement très similaire à PHP.

```
{% set greeting = 'Hello ' %}  
{% set name = 'Nicolas' %}  
{ { greeting ~ name|lower } } {# Hello nicolas #}
```

{# Utiliser des parenthèses pour modifier la priorité #}
{ { (greeting ~ name)|lower } } {# hello nicolas #}



Comparaisons

Les opérateurs de comparaison suivants sont pris en charge dans une expression: ==, !=, <, >, >=, et <=.

Vous pouvez également vérifier si une chaîne commence par ou se termine par une autre chaîne:

```
{% if 'Nicolas' starts with 'F' %}  
{% endif %}
```

```
{% if 'Nicolas' ends with 'n' %}  
{% endif %}
```



Opérateur de confinement

L'opérateur **in** effectue le test de *confinement*.

Il renvoie vrai si l'opérande de gauche est contenu dans le droit:

```
 {{ 1 in [1, 2, 3] }}
```

```
 {{ 'cd' in 'abcde' }}
```

Pour effectuer un test négatif, utilisez **not** pas dans l'opérateur:

```
{% if 1 not in [1, 2, 3] %}
```

```
{% if not (1 in [1, 2, 3]) %}
```



Opérateur de test

L'opérateur **is** effectue des tests. Les tests peuvent être utilisés pour tester une variable contre une expression commune. L'opérande de droite est le nom du test:

```
{% if post.status is constant('Post::PUBLISHED') %}
```

String
Interpolation

Pour effectuer un test négatif, utilisez **not** pas dans l'opérateur:

```
{% if post.status is not constant('Post::PUBLISHED') %}
```

```
{% if not (post.status is constant('Post::PUBLISHED')) %}
```



String Interpolation

{expression} permet à toute expression valide d'apparaître dans une chaîne entre guillemets doubles.

Le résultat de l'évaluation de cette expression est insérée dans la chaîne:

```
 {{ "foo #{bar} baz" }}
```

```
 {{ "foo #{1 + 2} baz" }} # output 'foo 3 baz' #}
```



Macro



Definition d'une macro

Les macros sont compatibles avec les fonctions dans les langages de programmation.

Elles sont utiles pour réutiliser des fragments HTML souvent utilisés pour ne pas avoir à les répéter.

Une macro est définie par la balise **macro**

```
{% macro input(name, value, type, size) %}  
  <input type="{{ type|default('text') }}"  
        name="{{ name }}" value="{{ value|e }}"  
        size="{{ size|default(20) }} />  
{% endmacro %}
```



Utilisation d'une macro

Les macros peuvent être définies dans n'importe quel template, et doivent être "importés" via le tag **import** avant d'être utilisé.

```
{% import "forms.html" as forms %}
```

```
<p>{{ forms.input('username') }}</p>
```



Inclusion de template



Inclusion de template

La fonction `include()` évalue un template Twig et retourne le contenu généré.

```
{{ include('sidebar.html') }}
```

Par défaut, les templates inclus ont accès au même contexte que le template qui les comprend. Ce signifie que toute variable définie dans le template principal sera disponible dans le template inclus aussi



Inclusion de template

Le contexte est passé par défaut au template mais vous pouvez également passer des variables supplémentaires

```
{{ include('template.html', {foo: 'bar'}) }}
```

Vous pouvez désactiver l'accès au contexte en définissant **with_context** à **false**:

```
{% include('template.html', {foo: 'bar'}, with_context = false) %}
```

```
{% include('template.html', with_context = false) %}
```



Heritage de template



Heritage de template

La partie la plus puissante de Twig est l'héritage de gabarit.

L'héritage de Template permet de construire un template de base "squelette" qui contient tous les éléments communs du site et définit les blocs que les Template enfants peuvent surcharger.



Heritage de template

Voici un template simple, « *base.html* », qui définit un squelette HTML que vous pouvez utiliser pour une page à deux colonnes

```
<!DOCTYPE html>
<html>
<head>
    {% block head %}
        <link rel="stylesheet" href="style.css"/>
        <title>{% block title %}{% endblock %} - My Webpage</title>
    {% endblock %}
</head>
<body>
<div id="content">{% block content %}{% endblock %}</div>
<div id="footer">
    {% block footer %}
        &copy; Copyright 2011 by <a href="http://domain.invalid/">you</a>.
    {% endblock %}
</div>
</body>
</html>
```



Heritage de template

```
{% extends "base.html" %}           ← Etends le template parent  
  
{% block title %}Index{% endblock %}   ← S'affichera dans le block  
« title » du template parent  
  
{% block head %}  
{{ parent() }}                      ← Réutilise la valeur par défaut du  
<style type="text/css">  
    .important { color: #336699; }  
</style>  
{% endblock %}  
  
{% block content %}  
    <h1>Index</h1>  
    <p class="important">  
        Welcome to my awesome homepage.  
    </p>  
{% endblock %}
```



Objet globaux



Objet globaux

{ { app.request } }

{ { app.session } }

{ { app.security } }

{ { app.user } }



Exercice 1

- Créer un template **deal/index.html.twig** qui va étendre le template **base.html.twig**

(Vous pouvez ajouter bootstrap pour faire de la mise en page rapidement)

- Ce template sera utilisé pour surcharger le block **body**.
- Le title dans <head> va contenir **{{ site.name }}**.
- Définir le paramètre **app.site.name** dans le **services.yaml** et l'utiliser comme variable **global** dans le **twig.yaml**.
- Dans le **DealController**, récupérer tous les Deals grâce à son **repository**.
- Renvoyer le résultat dans le template twig : **deal/index.html.twig**
- Faire en sorte que le template twig affiche tous les deals.
- Appliquer un **filtre** date sur le **createdAt** pour le mettre en format **m/d/Y**

Exercice 2

- Modifier le Controller et le Template pour afficher toutes les catégories avec tous les deals disponibles avec **deux for imbriqués**.
- Appliquer un filtre sur le nom du Category pour le mettre en majuscule.
- Le rendu sera sous la forme :

NOM CATEGORY 1 :

- Nom Deal 1 : Prix [Description] - 'date en format m/d/y'
- Nom Deal 2 : Prix [Description] - 'date en format m/d/y'
- Nom Deal 3 : Prix [Description] - 'date en format m/d/y'

NOM CATEGORY 2 :

- Nom Deal 4 : Prix [Description] - 'date en format m/d/y'
- Nom Deal 5 : Prix [Description] - 'date en format m/d/y'

Exercice 3

- Créer une extension Twig (make:twig-extension) **RandomExtension**
- Modifier l'extension pour avoir une fonction custom “my_random”

Formulaire & Validation



Formulaire

Traiter avec des formulaires HTML est l'un des tâches les plus communes et plus difficiles pour un développeur web.

Symfony intègre un composant de forme qui rend le traitement des formes faciles.

Ce composant est « standalone », il peut être donc utilisé dans des projets non Symfony.



L'objet métier



Création de la classe métier

```
<?php

// src/Entity/Task.php
namespace App\Entity;

class Task
{
    protected $task;
    protected $dueDate;

    public function getTask()
    {
        return $this->task;
    }

    public function setTask($task)
    {
        $this->task = $task;
    }

    public function getDueDate()
    {
        return $this->dueDate;
    }

    public function setDueDate(\DateTime $dueDate = null)
    {
        $this->dueDate = $dueDate;
    }
}
```



Creation d'un formulaire



Création de la classe métier

```
<?php
// src/Controller/DefaultController.php
namespace App\Controller;

use App\Entity\Task;
use Symfony\Component\HttpFoundation\Request;
use Symfony\Component\Form\Extension\Core\Type\TextType;
use Symfony\Component\Form\Extension\Core\Type\DateType;
use Symfony\Component\Form\Extension\Core\Type\SubmitType;

class DefaultController extends AbstractController
{
    public function new(Request $request)
    {
        // Creation d'une tache avec des datas fake
        $task = new Task();
        $task->setTask('Write a blog post');
        $task->setDueDate(new \DateTime('tomorrow'));

        $form = $this->createFormBuilder($task)
            ->add('task', TextType::class)
            ->add('dueDate', DateType::class)
            ->add('save', SubmitType::class, array('label' => 'Create Task'))
            ->getForm();

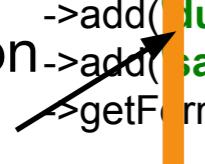
        return $this->render(
            'default/new.html.twig',
            array(
                'form' => $form->createView(),
            )
        );
    }
}
```

Définition des champs

return \$this->render(
 'default/new.html.twig',
 array(
 'form' => \$form->createView(),
)
);

Utilisation du FormBuilder

Création de la vue pour être envoyée à Twig



Type de champ disponible

Nom			
Birthday	Entity	Password	Time
Button	File	Percent	Timezone
Checkbox	Form	Radio	Url
Choice	Hidden	Repeated	
Collection	Integer	Reset	
Country	Language	Search	
Csrf	Locale	Submit	
Date	Money	Text	
Datetime	Number	Textarea	



Afficher le formulaire



Affiche d'un formulaire: mode bisounours

```
{# app/Resources/views/default/new.html.twig #}
{{ form_start(form) }}
{{ form_widget(form) }}
{{ form_end(form) }}
```

The screenshot shows a user interface for creating a blog post. At the top, there is a text input field with the placeholder "Write a blog post". Below it is a label "Due date" followed by three dropdown menus: one for the year (2014), one for the month (Aug), and one for the day (6). At the bottom of the form is a blue "Create Post" button.

form_start(form)

- Affiche la balise de début du formulaire, y compris l'attribut enctype correct lors de l'utilisation des téléchargements de fichiers.

form_widget(form)

- Affiche tous les champs, qui comprend l'élément de champ lui-même, le label et tous les messages d'erreur de validation pour le champ.

form_end(form)

- Affiche la balise de fin du formulaire et tous les champs qui n'ont pas encore été affichés, au cas où vous avez affichés chaque champ vous-même. Ceci est utile pour afficher les champs cachés et tirer profit de la protection automatique CSRF.



Traitement d'un formulaire

```
public function new(Request $request)
{
    // Il suffit de configurer un nouvel objet $task
    $task = new Task();
    $form = $this->createFormBuilder($task)
        ->add('task', TextType::class)
        ->add('dueDate', DateType::class)
        ->add('save', SubmitType::class, array('label' => 'Create Task'))
        ->getForm();
    $form->handleRequest($request);
    if ($form->isSubmitted() && $form->isValid()) {
        // ... exécuter certaines actions, telles que l'enregistrement de la tâche dans la base de données
        return $this->redirectToRoute('task_success');
    }

    return $this->render(
        'default/new.html.twig',
        array(
            'form' => $form->createView(),
        )
    );
}
```



Traitement d'un formulaire

Ce contrôleur suit un comportement commun de traitement de formulaires et comporte trois chemins possibles

1. Lors du chargement initial de la page dans un navigateur, le formulaire est simplement créé et affiché. HandleRequest() reconnaît que le formulaire n'a pas été envoyé et ne fait rien. IsSubmitted() renvoie false si le formulaire n'a pas été envoyé.
2. Lorsque l'utilisateur envoie le formulaire, handleRequest() le reconnaît et réécrit immédiatement les données soumises dans les propriétés task et dueDate de l'objet \$task. Ensuite, cet objet est validé. Si il est invalide, isValid() renvoie false, donc le formulaire est rendu avec toutes les erreurs de validation
3. Lorsque l'utilisateur soumet le formulaire avec des données valides, les données soumises sont de nouveau écrites dans le formulaire, mais cette fois isValid () renvoie true. Maintenant, vous avez la possibilité d'effectuer certaines actions à l'aide de l'objet \$task (par exemple, de le conserver dans la base de données) avant de rediriger l'utilisateur vers une autre page



Classe de formulaire



Creation d'une classe de formulaire dédié

Afin de rendre le controller le plus petit possible et avoir un formulaire réutilisable, la bonne pratique est de déplacer le code de création d'un formulaire dans un **classe dédié**



Creation d'une classe de formulaire dédié

```
// src/Form/PostType.php
namespace App\Form;

use App\Entity\Post;
use Symfony\Component\Form\AbstractType;
use Symfony\Component\Form\FormBuilderInterface;
use Symfony\Component\OptionsResolver\OptionsResolver;

class PostType extends AbstractType
{
    public function buildForm(FormBuilderInterface $builder, array $options)
    {
        $builder
            ->add('title')
            ->add('publishDate')
        ;
    }

    public function configureOptions(OptionsResolver $resolver)
    {
        $resolver->setDefaults([
            'data_class' => Post::class,
        ]);
    }
}
```



Creation d'une classe de formulaire dédié

```
/**
 * @Route("/post")
 */
class PostController extends AbstractController
{
    /**
     * @Route("/new", name="post_new", methods="GET|POST")
     */
    public function new(Request $request): Response
    {
        $post = new Post();
        $form = $this->createForm(PostType::class, $post);
        $form->handleRequest($request);

        if ($form->isSubmitted() && $form->isValid()) {
            // ...
            return $this->redirectToRoute('post_index');
        }

        return $this->render('post/new.html.twig', [
            'post' => $post,
            'form' => $form->createView(),
        ]);
    }
}
```



Génération de la classe de formulaire

```
$php bin/console make:form
```



Ajouter des contraintes de validations



Validation par Annotation

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

use Symfony\Component\Validator\Constraints as Assert;

class Task
{
    /**
     * @Assert\NotBlank()
     */
    public $task;

    /**
     * @Assert\NotBlank()
     * @Assert\Type("DateTime")
     */
    protected $dueDate;
}
```

Dans Symfony, la validation est appliquée à l'objet sous-jacent (par exemple, Tâche). En d'autres termes, la question n'est pas de savoir si le "form" est valide, mais si l'objet \$task est valide après que le formulaire a appliqué les données soumises à celle-ci.

Appeler \$form->isValid () est un raccourci qui demande à l'objet \$task si oui ou non il a des données valides.



Contraintes disponibles

Nom			
All	Email	Language	Optional
Blank	EqualTo	Length	Range
Callback	Existence	LessThan	Regex
CardScheme	False	LessThanOrEqual	Required
Choice	GroupSequence	Locale	Time
Collection	Iban	Luhn	True
Count	IdenticalTo	NotBlank	Type
Country	Image	NotEqualTo	Url
Currency	Ip	NotIdenticalTo	Valid
Date	Isbn	NotNull	
DateTime	Issn	Null	



Groupes de validation

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

use Symfony\Component\Validator\Constraints as Assert;

class Task
{
    /**
     * @Assert\NotBlank()
     */
    public $task;
    /**
     * @Assert\NotBlank()
     * @Assert\Type("DateTime")
     */
    protected $dueDate;
}
```

Dans Symfony, la validation est appliquée à l'objet sous-jacent (par exemple, Tâche). En d'autres termes, la question n'est pas de savoir si le "form" est valide, mais si l'objet \$task est valide après que le formulaire a appliqué les données soumises à celle-ci.

Appeler \$form->isValid () est un raccourci qui demande à l'objet \$task si oui ou non il a des données valides.



Rendu d'un formulaire dans un Template



Rendu d'un formulaire dans un Template

```
 {{ form_start(form) }}  
   {{ form_errors(form) }}  
  
   {{ form_row(form.task) }}  
   {{ form_row(form.dueDate) }}  
  
 {{ form_end(form) }}
```



Affiche avancé d'un formulaire

Nom	Description
form_start(form)	Affiche la balise <form>
form_errors(form)	Affichage des messages d'erreur
form_row(form.price)	Affiche le label, erreurs et widget pour un champ
form_end(form)	Affiche la balise </form> et les champs non affichés



Rendu de chaque champ à la main

form_row est très utile parce qu'il permet très rapidement d'afficher chaque champ.
Par contre il faut par moment pouvoir manuellement afficher chaque champ
L'affichage qui suit est le même que lorsque vous avez utilisé le **form_row**

```
{ { form_start (form) } }
{ { form_errors (form) } }
<div>
{ { form_label (form.task) } }
{ { form_errors (form.task) } }
{ { form_widget (form.task) } }
</div>
<div>
{ { form_label (form.dueDate) } }
{ { form_errors (form.dueDate) } }
{ { form_widget (form.dueDate) } }
</div>
<div>
{ { form_widget (form.save) } }
</div>
{ { form_end (form) } }
```



Affiche avancé d'un formulaire

Nom	Description
form_label(form.price)	Affiche le label du champ « price »
form_errors(form.price)	Affiche les messages d'erreur du champ « price »
form_widget(form.price)	Affiche le widget du champ « price »



Tweaking du comportement du formulaire

```
{ { form(form, {  
    'action' : '...',  
    'method' : 'GET'  
}) } }
```

```
{ { form_start(form, {  
    'action' : '...',  
    'method' : 'GET'  
}) } }
```



Formulaires et Doctrine



Formulaires et Doctrine

Le concept à comprendre est que lorsque le formulaire est soumis, les données soumises sont transférées immédiatement à l'objet sous-jacent. Si vous souhaitez persister ces données, vous devez simplement persister l'objet lui-même (qui contient déjà les données soumises).

```
if ($form->isSubmitted() && $form->isValid()) {  
    $em = $this->getDoctrine()->getManager();  
    $em->persist($post);  
    $em->flush();  
  
    return $this->redirectToRoute('post_index');  
}
```



Formulaires imbriqués



Formulaires imbriqués

Souvent, vous voudrez construire un formulaire qui inclura des champs de beaucoup d'objets différents. Par exemple, un formulaire d'enregistrement peut contenir des données appartenant à un objet Utilisateur ainsi qu'à de nombreux objets Adresse.

```
// src/Entity/Category.php                                     // src/Entity/Task.php
class Category
{
    /**
     * @Assert\NotBlank()
     */
    public $name;
}

class Task
{
    /**
     * @Assert\Type(type="App\Entity\Category")
     * @Assert\Valid()
     */
    protected $category;

    public function getCategory()
    {
        return $this->category;
    }

    public function setCategory(Category $category = null)
    {
        $this->category = $category;
    }
}
```



Formulaires imbriqués

L'objectif final est de permettre la modification de la catégorie d'une tâche à l'intérieur du formulaire de tâche lui-même. Pour ce faire, ajoutez un champ de catégorie à l'objet TaskType dont le type est une instance de la classe CategoryType

//...

```
class CategoryType extends AbstractType
{
    public function buildForm(FormBuilderInterface $builder, array $options)
    {
        $builder->add('name');
    }

    public function configureOptions(OptionsResolver $resolver)
    {
        $resolver->setDefaults(
            array(
                'data_class' => 'App\Entity\Category',
            )
        );
    }
}
// src/Form/TaskType.php
```

```
class TaskType extends AbstractType
{
    public function buildForm(FormBuilderInterface $builder, array $options)
    {
        $builder
            ->add('task')
            ->add('dueDate', null, array('widget' => 'single_text'))
            ->add('save', SubmitType::class)
            ->add('category', CategoryType::class);
    }
}
```



Formulaires imbriqués

Les champs de CategoryType peuvent maintenant être affichés aux côtés de ceux de la classe TaskType. Afficher les champs de catégorie de la même manière que les champs de tâche d'origine:

```
{# ... #}
<h3>Category</h3>
<div class="category">
{{ form_row(form.category.name) }}
</div>
{# ... #}
```



Exercice 1

- Créer une **classe de formulaire dédiée** pour l'entité Deal. Le formulaire présentera les champs :
 - Name
 - Description
 - Price
 - Categories
 - Un bouton save
- Créer le template pour afficher le formulaire. Il étend le template de base.
- Ajouter dans le template de base de quoi afficher les messages flash.

Exercice 2

- Ajouter une action dans le Controller Deal permettant d'afficher le formulaire avec l'adresse **/deal/new**.
- Modifier l'action pour gérer l'enregistrement des données du formulaire une fois validé.
Enable aura false comme valeur.
- Une fois l'enregistrement validé :
 - Ajouter un message flash de type **success**.
 - Faire une **redirection** vers la liste des deals.
- Ajouter une validation sur le champ Price pour ne prendre que des valeurs supérieure à 0.

Security



But

Fournir des mécanismes
d'authentification et
d'autorisation



Configuration initiale de security.yml (Authentification)

Le système de sécurité est configuré dans app/config/packages/security.yml.
La configuration par défaut ressemble à ceci:

```
security:  
    firewalls:  
        dev:  
            pattern: ^/(_profiler|wdt)|css|images|js/  
            security: false  
        main:  
            anonymous: true
```

La clé **firewall** est au cœur de la configuration de sécurité.

Le firewall de **dev** n'est pas important, il s'assure simplement que les outils de développement de Symfony qui sont sous URLs comme /_profiler et /_wdt ne sont pas bloqués par la sécurité.

Toutes les autres URL seront traitées par le pare-feu par défaut (aucune clé ne correspond à toutes les URL).



Configuration de l'authentification de vos utilisateurs

Le rôle principal d'un pare-feu consiste à configurer la façon dont vos utilisateurs vont s'authentifier.

Pour l'activer l'authentification HTTP, il faut ajouter la clé http_basic sous votre pare-feu:

```
# app/config/security.yml
security:
    # ...
    firewalls:
        # ...
        default:
            anonymous: ~
            http_basic: ~
```



Configurer la façon dont les utilisateurs sont chargés

Lorsque vous tapez votre nom d'utilisateur, Symfony doit charger les informations de cet utilisateur quelque part. C'est ce qu'on appelle un «user provider».

```
security:  
    providers:  
        app_user_provider:  
            entity:  
                class: App\Entity\User  
                property: email
```



Codage du mot de passe de l'utilisateur

Que les utilisateurs soient stockés dans security.yml, dans une base de données ou ailleurs, il faut encoder leurs mots de passe. Le meilleur algorithme à utiliser est **argon2i**

```
# /config/packages/security.yml
security:
    # ...
    encoders:
        App\Entity\User:
            algorithm: argon2i
```



Roles

Lorsqu'un utilisateur se connecte, il reçoit un ensemble de rôles.

Dans l'exemple ci-dessus, ceux-ci sont codés en dur dans security.yml.

Vous pouvez charger des utilisateurs à partir de la base de données.

Tous les rôles attribués à un utilisateur doivent commencer par le préfixe ROLE_.
Sinon, ils ne seront pas traités par le système de sécurité de Symfony.

Les rôles sont essentiellement des chaînes que vous inventez et utilisez au besoin.

Vous pouvez également spécifier une hiérarchie de rôles dans laquelle certains rôles signifient automatiquement que vous avez également d'autres rôles.

```
# app/config/security.yml
security:
    # ...
    role_hierarchy:
        ROLE_ADMIN: ROLE_USER
        ROLE_SUPER_ADMIN: [ROLE_ADMIN, ROLE_ALLOWED_TO_SWITCH]
```



Ajouter du code pour refuser l'accès

Il y a deux façons de refuser l'accès à quelque chose:

1. **access_control** dans security.yml permet de protéger des patterns d'URL (/admin/*). C'est facile, mais moins flexible.
2. Fans votre code via le service **security.authorization_checker**.



Sécurisation des patterns d'URL (access_control)

La façon la plus simple de sécuriser une partie d'une application consiste à sécuriser un pattern d'URL entier.

```
# app/config/security.yml
security:
# ...
firewalls:
# ...
default:
# ...
access_control:
# require ROLE_ADMIN for /admin*
- { path: ^/admin/users, roles: ROLE_SUPER_ADMIN }
- { path: ^/admin, roles: ROLE_ADMIN }
```



Sécurisation des contrôleurs et autres codes

Vous pouvez facilement refuser l'accès à l'intérieur d'un contrôleur:

```
public function hello($name)
{
    // le second paramètre est utilisé pour spécifier sur quel objet le rôle est testé.
    $this->denyAccessUnlessGranted('ROLE_ADMIN', null, 'Unable to access this page!');
}
```

Utilisation des annotations

```
use Sensio\Bundle\FrameworkExtraBundle\Configuration\Security;

/**
 * @Security("has_role('ROLE_ADMIN')")
 */
public function hello($name)
{
    // ...
}
```



Contrôle d'accès dans les templates

Pour vérifier si l'utilisateur actuel a un rôle dans un modèle, utilisez la fonction d'aide `is_granted()`

```
{% if is_granted('ROLE_ADMIN') %}  
  <a href="#">Delete</a>  
{% endif %}
```



Récupération de l'objet utilisateur

Après l'authentification, l'objet User de l'utilisateur actuel peut être consulté via le service **security.token_storage**.

```
public function index()
{
    if (!$this->get('security.authorization_checker')->isGranted('IS_AUTHENTICATED_FULLY')) {
        throw $this->createAccessDeniedException();
    }
    $user = $this->getUser();
    // Meme chose qu'au dessus
    $user = $this->get('security.token_storage')->getToken()->getUser();
}
```



Toujours vérifier si l'utilisateur est connecté

Il est important de vérifier si l'utilisateur est authentifié en premier.

Vérifiez toujours si l'utilisateur est connecté avant d'utiliser l'objet User et utilisez la méthode isGranted (ou access_control):

```
// Vérification si le user est connecté
if (!$this->get('security.authorization_checker')->isGranted('IS_AUTHENTICATED_FULLY')) {
    throw $this->createAccessDeniedException();
}
```

```
// N'est pas la bonne solution pour voir si le user est connecté
if ($this->getUser()) {
```



Récupération de l'utilisateur dans Template

Dans un modèle Twig, le User peut être consulté via la variable app.user

```
{% if is_granted('IS_AUTHENTICATED_FULLY') %}  
    <p>Username: {{ app.user.username }}</p>  
{% endif %}
```



Déconnexion

Le **firewall** peut gérer automatiquement la déconnexion lorsque vous activez le paramètre de configuration logout:

```
# app/config/security.yml
security:
    # ...
    firewalls:
        secured_area:
            # ...
            logout:
                path: /logout
                target: /
```

Création d'une route

```
# app/config/routing.yml
logout:
    path: /logout
```



Codage dynamique d'un mot de passe

Si, vous stockez des utilisateurs dans la base de données, vous devez coder les mots de passe des utilisateurs avant de les insérer.

Peu importe l'algorithme que vous configurez pour votre objet utilisateur, le mot de passe haché peut toujours être déterminé de la façon suivante à partir d'un contrôleur

```
$user = new App\Entity\User();
$plainPassword = 'ryanpass';
$encoder = $this->container->get('security.password_encoder');
$encoded = $encoder->encodePassword($user, $plainPassword);
$user->setPassword($encoded);
```



Security

Le système de sécurité de Symfony est incroyablement puissant, mais il peut aussi être source de confusion pour la mise en place.

Le système de sécurité Symfony est très flexible

2 commandes facilitent la configuration

```
$ php bin/console make:user
```

```
$ php bin/console make:auth
```



Exercice 1

Créer une entité **User** qui doit avoir comme propriétés :

- **username** (string, unique, non null)
- **password** (string, non null)
- **roles** (json)

Ajouter l'annotation **@UniqueEntity** sur le champ **username** avec un **message**.

User implémente les classes :

- **Symfony\Component\Security\Core\User\UserInterface**
- **\Serializable**

Ajouter les méthodes manquantes.

Vous pouvez retourner null pour le **getSalt()** et ne rien mettre dans le
eraseCredentials()

Exercice 2

Créer des **fixtures** pour **User**.

Injecter la classe **UserPasswordEncoderInterface** dans le constructeur pour le passwordEncoder.

Créer au moins un user de chaque type : **ROLE_ADMIN & ROLE_USER**

Lancer la création des fixtures.

Exercice 3

Créer un controller **User** avec deux fonctions:

- **logout** qui aura la route **/logout**. La fonction lèvera une exception, Symfony va intercepter la fonction avant l'exécution.
- **login** qui aura la route **/login**. La fonction fait appel à un template pour le formulaire de login. (par exemple user/login.html.twig)

Créer le template avec le formulaire classique : `<form action='...' method='post'><input> ...`

Une fois connecté, utiliser le **profiler** symfony pour avoir des informations sur l'utilisateur.

Exercice 5

Pour aller plus loin :

Gérer les messages d'erreurs dans le template de login :

- Utiliser l'helper **AuthenticationUtils** avec la fonction **getLastAuthenticationError** dans le controller pour l'envoyer au template.
- Afficher les erreurs dans le template twig.

Modifier le template de base :

- Si l'utilisateur est **connecté** : Afficher son nom et ajouter un lien pour se déconnecter.
- Si l'utilisateur est un **admin** : Ajouter un lien vers le dashboard admin.
- Si l'utilisateur n'est **pas connecté** : Ajouter un lien vers le formulaire de connection.

Exercice 6

Pour aller encore plus loin :

Créer un formulaire pour s'enregistrer en utilisant un form **UserType**.

Modifier le template de base pour afficher le lien d'enregistrement quand l'utilisateur n'est pas connecté.

Components



Messenger



Messenger

Un composant pour l'envoi de messages

- Un bus de messages
- Un bus de commande
- Un bus événementiel
- Un bus d'interrogation
- Une façon d'exécuter du code de façon asynchrone



Prise en main par l'exemple

Création d'une application de "Système de commande de cafés"

POST /coffee/order

```
{  
  "topping": "cream",  
  "size": "medium",  
  "recipient": "Nicolas"  
}
```



Prise en main par l'exemple

POST /coffee/order

```
{  
    "topping": "cream",  
    "size": "medium",  
    "recipient": "Nicolas"  
}
```

```
//src/Controller/CoffeeController.php  
/**  
 * @Route ("/coffee/order", methods={"POST"})  
 */  
public function index(Request $request)  
{  
    $data = json decode($request->getContent(), true);  
    $topping = $data['topping'] ?? 'none';  
    $size = $data['size'] ?? 'large';  
    $recipient = $data['recipient'] ?? 'John';  
    // ...  
}
```



Prise en main par l'exemple

POST /coffee/order

```
{  
    "type": "black",  
    "size": "medium",  
    "recipient": "Nicolas"  
}
```

```
//src/Controller/CoffeeController.php  
/**  
 * @Route("/coffee/order", methods={"POST"})  
 */  
public function index(Request $request)  
{  
    $data = json_decode($request->getContent(), true);  
    $type = $data['type'] ?? 'expresso';  
    $size = $data['size'] ?? 'large';  
    $recipient = $data['recipient'] ?? 'John';  
    // ...  
}
```



Notre machine à café exclusive et secrète

```
public function index(Request $request)
{
    // ...
    $this->logger->info(sprintf(
        'Put a %s amount of coffee in the espresso machine',
        $size
    ));
    usleep(500000);
    $this->logger->info('Adding some sugar');
    usleep(500000);
    if ($topping !== 'none') {
        usleep(500000);
        $this->logger->info(sprintf(
            'Adding topping: '.$topping
        ));
    }
    $this->logger->info('Delivering to '.$recipient);
    usleep(500000);
}
```



Notre machine à café exclusive et secrète

```
public function index(Request $request)
{
    // ...
    $this->logger->info(sprintf(
        'Machine',
        ) );
    usleep(100000);
    $this->logger->info('
    if (
        )
    }
    $this->logger->info('
    usleep(500000);
}
```



Délicieux ! Pouvons-nous nous organiser ?

```
<?php
namespace App\Message;

class CoffeeOrder
{
    private $topping;
    private $size;
    private $recipient;

    public function __construct(string $topping, string $size, string $recipient)
    {
        $this->topping = $topping;
        $this->size = $size;
        $this->recipient = $recipient;
    }
    // Getters
}
```



Délicieux ! Pouvons-nous nous organiser ?

```
/*
* @Route("/coffee/order", methods={"POST"})
*/
public function index(Request $request)
{
// ...
    $order = new CoffeeOrder(
        $stopping,
        $size,
        $recipient
    );

    $this->logger->info(sprintf(
        'Put a %s amount of coffee in the espresso machine',
        $order->getSize()
    ));
    usleep(500000);
// ...
}
```



Organisation: peut-être un service pour la logique ?

```
<?php
namespace App\MessageHandler;

use Psr\Log\LoggerInterface;

class CoffeeOrderHandler
{
    private $logger;

    public function __construct(LoggerInterface $logger)
    {
        $this->logger = $logger;
    }
}
```



Organisation: peut-être un service pour la logique ?

```
/*
* @Route("/coffee/order", methods={"POST"})
*/
public function index(Request $request, CoffeeOrderHandler $handler)
{
    // ...
    $order = new CoffeeOrder(
        $stopping,
        $size,
        $recipient
    );
    $handler->handleOrder($order);
    return new Response(204);
}
```



Messenger?

Appeler un service

Hé, CoffeeHandler !

S'il vous plaît, faites-moi un petit café pour Nicolas !

```
$coffeeHandler->handleOrder (  
    'plain',  
    'small',  
    'Nicolas'  
);
```



Utilisation d'un "message"

Hé CoffeeHandler !

s'il vous plaît, prenez cette CoffeeOrder !

```
$order = new CoffeeOrder(...)  
$coffeeHandler->handleOrder($order);
```



Message BUS

Hé "Message Bus" !

Appelez la personne qui s'occupe de ce message.

```
$order = new CoffeeOrder(...)  
$messageBus->dispatch(  
    $order  
) ;
```



> composer require messenger

What's next?

- * You're ready to use the Messenger component. You can define your own message buses or start using the default one right now by injecting the `message_bus` service or type-hinting `Symfony\Component\Messenger\MessageBusInterface` in your code.
- * To send messages to a transport and handle them asynchronously:
 1. Uncomment the `MESSENGER_TRANSPORT_DSN` env var in `.env` and `framework.messenger.transports.async` in `config/packages/messenger.yaml`;
 2. Route your message classes to the async transport in `config/packages/messenger.yaml`.
- * Read the documentation at <https://symfony.com/doc/current/messenger.html>

Connecting messages & handlers

Comment Messenger sait-il qu'il faut appeler CoffeeOrderHandler lorsque nous envoyons un CoffeeOrder ?



autoconfigure: Tell Messenger this is a handler

```
<?php

namespace App\MessageHandler;

use Psr\Log\LoggerInterface;
use
Symfony\Component\Messenger\Handler\MessageHandlerInterface;

class CoffeeOrderHandler implements MessageHandlerInterface
{
    private $logger;

    public function __construct(LoggerInterface $logger)
    {
        $this->logger = $logger;
    }
}
```



Type-hint: Tell Messenger what object it handles

```
<?php
```

```
namespace App\MessageHandler;

use Symfony\Component\Messenger\Handler\MessageHandlerInterface;

class CoffeeOrderHandler implements MessageHandlerInterface
{
    public function __invoke(CoffeeOrder $order)
    {
        // ...
    }
}
```



Appeler le BUS

```
public function index(Request $request, MessageBusInterface $messageBus)
{
    // ...

    $order = new CoffeeOrder(
        $stopping,
        $size,
        $recipient
    );
    $messageBus->dispatch($order);
    $handler->handleOrder($order);

    return new Response(204);
}
```



Traitement asynchrone

La centralisation des "messages" par le biais d'un système (bus) a de gros avantages

Possibilités de centralisation

- Logger chaque fois qu'un "message" est envoyé
- Envelopper chaque "andler à l'intérieur d'une transaction
- Stockez le message ailleurs... puis exécutez une commande pour le lire et le manipuler plus tard



Transports : où envoyer/lire les messages

```
//.env  
###> symfony/messenger ###  
# Choose one of the transports below  
# MESSENGER_TRANSPORT_DSN=amqp://guest:guest@localhost:5672/%2f/messages  
# MESSENGER_TRANSPORT_DSN=doctrine://default  
# MESSENGER_TRANSPORT_DSN=redis://localhost:6379/messages  
###< symfony/messenger ###
```

```
# config/packages/messenger.yaml  
framework:  
    messenger:  
        transports:  
            async: '%env(MESSENGER_TRANSPORT_DSN)%'
```



Prêt pour le transport ! Mais nous ne l'utilisons pas encore.

framework:

messenger:

transports:

async: '%env(MESSENGER_TRANSPORT_DSN)%'

routing:

'App\Message\CoffeeOrder' : **async**



Traitement asynchrone

POST /coffee/order

```
{  
  "topping": "cream",  
  "size": "medium",  
  "recipient": "Nicolas"  
}
```

... le café n'est pas préparé !



Traitements asynchrones

```
mysql> select * FROM messenger_messages\G
***** 1. row *****
    id: 1
  body: O:36:"Symfony\Component\Messenger\Envelope":2:{s:44:"\0Symfony\Component\Messenger\Envelope\0stamps";a:2:{s:46:"Symfony\Component\Messenger\Stamp\BusNameStamp";a:1:{i:0;O:46:"Symfony\Component\Messenger\Stamp\BusNameStamp":1:{s:55:"\0Symfony\Component\Messenger\Stamp\BusNameStamp\0busName";s:21:"messenger.bus.default";}}s:43:"Symfony\Component\Messenger\Stamp\SentStamp";a:1:{i:0;O:43:"Symfony\Component\Messenger\Stamp\SentStamp":2:{s:56:"\0Symfony\Component\Messenger\Stamp\SentStamp\0senderClass";s:64:"Symfony\Component\Messenger\Transport\Doctrine\DoctrineTransport";s:56:"\0Symfony\Component\Messenger\Stamp\SentStamp\0senderAlias";s:5:"async";}}s:45:"\0Symfony\Component\Messenger\Envelope\0message";O:23:"App\Message\CoffeeOrder":3:{s:32:"\0App\Message\CoffeeOrder\0topping";s:5:"plain";s:29:"\0App\Message\CoffeeOrder\0size";s:6:"medium";s:34:"\0App\Message\CoffeeOrder\0recipient";s:7:"Nicolas";}}
  headers: []
queue_name: default
created_at: 2019-11-26 13:05:29
available_at: 2019-11-26 13:05:29
delivered_at: NULL
1 row in set (0,00 sec)
```



Manipulation/Consommation

Comment pouvons-nous dire à Messenger que nous *sommes* prêts à traiter le message ? (worker)

bin/console messenger:consume -vv

```
[OK] Consuming messages from transports "async".
```

```
// The worker will automatically exit once it has received a stop signal via the messenger:stop-workers command.  
// Quit the worker with CONTROL-C.  
  
13:09:33 INFO      [messenger] Received message App\Message\CoffeeOrder ["message" => App\Message\CoffeeOrder^ { ... }, "class" => "App\Message\CoffeeOrder"]  
13:09:33 INFO      [messenger] Message App\Message\CoffeeOrder handled by App\MessageHandler\CoffeeOrderHandler::__invoke  
["message" => App\Message\CoffeeOrder^ { ... }, "class" => "App\Message\CoffeeOrder", "handler" => "App\MessageHandler\CoffeeOrderHandler::__invoke"]  
13:09:33 INFO      [messenger] App\Message\CoffeeOrder was handled successfully (acknowledging to transport). ["message" => App\Message\CoffeeOrder^ { ... }, "class" => "App\Message\CoffeeOrder"]  
^C
```



Console



Console

Le Symfony fournit beaucoup de commandes via le script bin/console. Ces commandes sont créées avec le composant Console. Vous pouvez également l'utiliser pour créer vos propres commandes.

Ce composant est « standalone », il peut être donc utilisé dans des projets non Symfony.



Création d'une commande

Les commandes sont définies dans des classes qui doivent être créées dans le répertoire « Command » (par exemple, App\Command) et leurs noms doivent se terminer par le suffixe Command.

```
<?php

// src/Command/CreateUserCommand.php
namespace App\Command;

use Symfony\Component\Console\Command\Command;
use Symfony\Component\Console\Input\InputInterface;
use Symfony\Component\Console\Output\OutputInterface;

class CreateUserCommand extends Command
{
    protected function configure()
    {
        //...
    }

    protected function execute(InputInterface $input, OutputInterface $output)
    {
        //...
    }
}
```



Configuration de la commande

Il faut configurer le nom de la commande dans la méthode configure().
Il faut éventuellement définir un message d'aide et les options et arguments d'entrée

```
protected function configure()
{
    $this
        // le nom de la commande (the part after "bin/console")
        ->setName('app:create-users')
        // la courte description affichée lors de l'exécution "php bin/console list"
        ->setDescription('Creates new users.')
        // la description de la commande complète affichée lors de l'exécution de la commande
        // "--help"
        ->setHelp("This command allows you to create users...");
```



Execution de la commande

Il suffit de l'executer via bin/console

```
php bin/console app:create-users
```



Utiliser les services à partir du service container

Si la commande doit accéder à certains services. Cela peut être fait en faisant hériter la commande de la classe ContainerAwareCommand

```
use Symfony\Bundle\FrameworkBundle\Command\ContainerAwareCommand;

class CreateUserCommand extends ContainerAwareCommand
{
    // ...

    protected function execute(InputInterface $input, OutputInterface $output)
    {
        // ...

        // Accéder au conteneur en utilisant getContainer()
        $userManager = $this->getContainer()->get('app.user_manager');

        // ...
    }
}
```



Utilisation des arguments

Les arguments sont des chaînes « séparées par des espaces » qui viennent après le nom de commande elle-même.
Ils sont ordonnés, et peuvent être facultatifs ou requis.

```
use Symfony\Component\Console\Input\InputArgument;

class GreetCommand extends Command
{
    // ...

    protected function configure()
    {
        $this
            // ...
            ->addArgument('name', InputArgument::REQUIRED, 'Who do you want to greet?')
            ->addArgument('last_name', InputArgument::OPTIONAL, 'Your last name?');
    }

    protected function execute(InputInterface $input, OutputInterface $output)
    {
        $text = 'Hi ' . $input->getArgument('name');

        $lastName = $input->getArgument('last_name');
        // ...
    }
}
```



Utilisation des arguments

La commande peut maintenant être utilisée de l'une des manières suivantes

```
php bin/console app:greet John
```

```
php bin/console app:greet John Doe
```



Types des arguments

InputArgument::REQUIRED

L'argument est obligatoire. La commande ne s'exécute pas si l'argument n'est pas fourni;

InputArgument::OPTIONAL

L'argument est facultatif et peut donc être omis

InputArgument::IS_ARRAY

L'argument peut contenir un nombre quelconque de valeurs. Pour cette raison, il doit être utilisé à la fin de la liste des arguments



Utilisation des options

Contrairement aux arguments, les options ne sont pas ordonnées et sont spécifiées avec deux tirets (par exemple --dir).

Les options sont toujours facultatives et peuvent être configurées pour accepter une valeur ou simplement comme un indicateur booléen sans valeur

```
use Symfony\Component\Console\Input\InputOption;

class GreetCommand extends Command
{
    // ...

    protected function configure()
    {
        $this
            // ...
            ->addOption(
                'iterations',
                null,
                InputOption::VALUE_REQUIRED,
                'How many times should the message be printed?',
                1
            );
    }

    protected function execute(InputInterface $input, OutputInterface $output)
    {
        $iteration = $input->getOption('iterations');
    }
}
```



Utilisation des options

La commande peut maintenant être utilisée de l'une des manières suivantes

```
# Valeur par défaut 1  
php bin/console app:greet John
```

```
php bin/console app:greet John Doe --iteration=5
```



Types des options

InputOption::VALUE_IS_ARRAY

Cette option accepte plusieurs valeurs (--dir=/foo
--dir=/bar)

InputOption::VALUE_NONE

Ne pas accepter d'entrée pour cette option (--dry-run)

InputOption::VALUE_REQUIRED

Cette valeur est requise, l'option elle-même est optionnelle
(--iterations = 5)

InputOption::VALUE_OPTIONAL

Cette option peut ou non avoir une valeur (--yell or
--yell=loud)



Console output

Objet permettant l'affichage de message dans la console

```
protected function execute(InputInterface $input, OutputInterface $output)
{
    // Affiche plusieurs lignes vers la console (en ajoutant "\n" à la fin de chaque ligne)
    $output->writeln([
        'User Creator',
        '=====',
        '',
    ]);

    // Affiche un message suivi d'un "\n"
    $output->writeln('Whoa!');

    // Affiche un message sans ajouter un "\n" à la fin de la ligne
    $output->write('You are about to ');
    $output->write('create a user.');
}
```

```
$ php bin/console app:create-user
User Creator
=====
Whoa!
You are about to create a user.
```



Helpers

Le composant Console est fourni avec quelques assistants utiles.

Ces aides contiennent des fonctions pour faciliter certaines tâches courantes.

- Formatter Helper
- Process Helper
- Progress Bar
- Question Helper
- Table
- Debug Formatter Helper



Helper / Table

Lors de la construction d'une application console, il peut être utile d'afficher des données tabulaires

```
use Symfony\Component\Console\Helper\Table;
// ...

public function execute(InputInterface $input, OutputInterface $output)
{
    $table = new Table($output);
    $table
        ->setHeaders(array('ISBN', 'Title', 'Author'))
        ->setRows(
            array(
                array('99921-58-10-7', 'Divine Comedy', 'Dante Alighieri'),
                array('9971-5-0210-0', 'A Tale of Two Cities', 'Charles Dickens'),
                array('960-425-059-0', 'The Lord of the Rings', 'J. R. R. Tolkien'),
                array('80-902734-1-6', 'And Then There Were None', 'Agatha Christie'),
            )
        );
    $table->render();
}
```

1	ISBN	Title	Author
2			
3			
4	99921-58-10-7	Divine Comedy	Dante Alighieri
5	9971-5-0210-0	A Tale of Two Cities	Charles Dickens
6	960-425-059-0	The Lord of the Rings	J. R. R. Tolkien
7	80-902734-1-6	And Then There Were None	Agatha Christie
8			



Helper / Progress Bar

Lors de l'exécution de commandes longues, il peut être utile de montrer des informations de progression, qui sont mises à jour à mesure que votre commande s'exécute.

```
use Symfony\Component\Console\Helper\ProgressBar;

protected function execute(InputInterface $input, OutputInterface $output)
{
    // Créer une nouvelle barre de progression (50 units)
    $progress = new ProgressBar($output, 50);

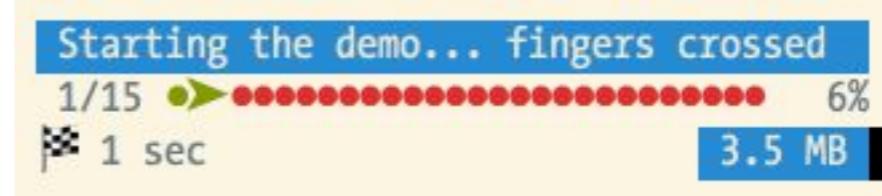
    // Démarre et affiche la barre de progression
    $progress->start();

    $i = 0;
    while ($i++ < 50) {
        // ...

        // Faire avancer la barre de progression d'1
        $progress->advance();

        // Vous pouvez également avancer la barre de progression de plus vite
        // $progress->advance(3);
    }

    //Assurez-vous que la barre de progression est à 100%
    $progress->finish();
}
```



Helper / Question

Le QuestionHelper fournit des fonctions pour demander à l'utilisateur de plus amples informations.

- ConfirmationQuestion
- Question
- ChoiceQuestion

```
use Symfony\Component\Console\Input\InputInterface;
use Symfony\Component\Console\Output\OutputInterface;
use Symfony\Component\Console\Question\ConfirmationQuestion;

// ...

public function execute(InputInterface $input, OutputInterface $output)
{
    $helper = $this->getHelper('question');
    $question = new ConfirmationQuestion('Continue with this action?', false);

    if (!$helper->ask($input, $output, $question)) {
        return;
    }
}
```



Exercice

Créer une commande **deal:price:increase** avec le composant Console.

La commande prend un **argument** price qui sera l'augmentation du prix à appliquer.

La commande prend en option deux options : **--id=xxxx** (optionnel) et **--all** (sans valeur).

- Si les deux options sont utilisées, on lève une erreur dans la console.
- Si l'option **--id** est utilisée alors seul un deal aura le changement. Afficher l'ancien et le nouveau prix en sortie de console.
- Si l'option **--all** est utilisée, mettre à jour tout le prix de tous les Deals en fonction de l'argument. Afficher un tableau récapitulatif à la fin du processus contenant :
 - l'ID du Deal
 - l'ancien prix
 - le nouveau prix

Httpclient



HttpClient

Le composant HttpClient est un client HTTP de bas niveau.

Il fournit des outils pour consommer les API et prend en charge les opérations synchrones et asynchrones.

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



Appel en GET

```
use Symfony\Component\HttpClient\HttpClient;  
  
$client = HttpClient::create();  
  
$url = 'https://symfony.com/versions.json';  
$response = $client->request('GET', $url);  
  
$symfonyVersions = $response->toArray();
```

```
array:25 [▼  
    "lts" => "4.4.0"  
    "latest" => "5.0.0"  
    "dev" => "5.1.0"  
    "2.0" => "2.0.25"  
    "2.1" => "2.1.13"  
    "2.2" => "2.2.11"  
    "2.3" => "2.3.42"  
    "2.4" => "2.4.10"  
    "2.5" => "2.5.12"  
    "2.6" => "2.6.13"  
    "2.7" => "2.7.51"  
    "2.8" => "2.8.52"
```



Headers, body

```
$client->request('POST', 'https://...', [
    'headers' => [ 'Content-type' => 'text/plain' ],
    'body' => $payload
]) ;
```



Query

```
$response = $client->request('GET', $url, [  
    'query' => [ 'acces_token' => $token]  
] );
```

`http://...?access_token=xxx`



JSON, Bearer

```
$client->request('POST', $url, [
    'auth_bearer' => $token,
    'json'        => $array
]) ;
```



Form

```
$client->request('POST', $url, [
    'body' => [
        'secret' => $secret,
        'response' => $token,
    ]
]) ;
```



Asynchrone

```
for ($i = 0; $i < 379; ++$i) {
    $uri= "https://http2.akamai.com/demo/tile-$i.png";
    $responses[] = $client->request('GET', $uri);

}

foreach($client->stream($responses) as $response=> $chunk) {
    if ($chunk->isLast()) {
        // a $response completed
    } else {
        // a $response's got network activity or timeout
    }
}
```



MERCI



nfa@emakina.fr