Symfony

Romain Clair

Formateur consultant informatique
contact@romainclair.com

2018

Symfony

- Framework PHP

  2005 : Version 1
  2011 : Version 2
  2015 : Version 3
  2017 : Version 4

  - Version 3.4.4 de novembre 2017 – LTS
  - Version 4.0.4 de novembre 2017 – Recommandée

- Logiciel libre
- Sensiolabs
- Modèle MVC

Framework

Définition

- Cadre applicatif (générique)
- Ensemble de composants logiciels (bibliothèques)
- Modèle d'application (architecture logiciel)
- Mise en place de bonnes pratiques
- Apprentissage long et spécifique

2 approches de Symfony

Bibliothèque

- Ensemble de composants logiciels ($\simeq 30$)
- Indépendants
- Réutilisables

Framework

- Symfony *Full Stack*
- Utilisation conjointe de l'ensemble de composants

Les nouveautés

- Symfony Flex pour l'installation automatique avec composer
- Approche micro-framework (modulaire)
- Arborescence plus standard
- Chargement automatique des services

Pré-requis

- HTML/CSS
- PHP
- Bases de la programmation objet
- Espace de noms

Plan

Pré-requis pour l'installation

Symfony 4

- Serveur web
  - PHP version 7.1 (ou plus)
- Serveur de base de donnée (ou sqlite)
  - Avec driver PDO
- Composer

Plan

composer

- Gestionnaire de dépendance PHP
- Logiciel libre
- Inspiré de *npm* (*Node.js*) et *bundler* (*Ruby*)
- Commande CLI PHP
- composer.phar / composer.json / composer.lock

Fichiers PHAR

- PHAR : PHP Archive
- Application complète (Ensemble de fichiers)
- Dans un fichier Archive

Testons composer

- Consulter la documentation :
  - `https://getcomposer.org/download/`
- Quelques remarques :
  - `php -r` : execution de code php (CLI)
- Installer composer
- Explorer les différentes commandes : `php composer.phar`
  - Exemple : mise à jour de composer
  - `php composer.phar self-update`

Installer Symfony

Avec composer

- `php composer.phar create-project symfony/website-skeleton chemin/MonProjet`
- Création d'un dossier `MonProjet` dans le répertoire `chemin`
- Récupère les fichiers nécessaires à un projet Symfony minimal

Installons symfony

- Créer un premier projet Symfony

Le serveur de test

Symfony PHP web server

- Dans le répertoire de votre projet
- `composer require server --dev`
  - Installe le serveur autonome Symfony
- Serveur pour tests seulement

Installons le serveur

- Récuperer le serveur pour effectuer vos tests

Plan

La console

- Commande CLI PHP
- `php bin/console`
- Automatisation de tâches

À vous de jouer !

Testons la console

- Dans le répertoire de votre projet
- Lancer la commande
  - `php bin/console security:check`
- Aperçu des différentes commandes :
  - `php bin/console`

Lancer le serveur

Symfony PHP web server

- Dans le répertoire de votre projet
- Mode interactif :
  - `php bin/console server:run`
- En tâche de fond :
  - `php bin/console server:start`
  - `php bin/console server:status`
  - `php bin/console server:stop`

Vérifier les pré-requis

Installation du script de vérification

- Dans le répertoire de votre projet
- Installe et execute le script
  - `composer require requirements-checker`

Installons le script

- Récuperer le script de vérification
- Vérifier la sortie console

À vous de jouer !

Premier accès

- Lancer le serveur
  - `php bin/console server:run`
- Acceder à la page de vérification
  - `http://127.0.0.1:8000/check.php`
- Attention : public/check.php a supprimer avant mise en production

Plan

Test de l'application démo

- Dans un autre répertoire
- Récuperer l'application de démo
  - `composer create-project symfony/symfony-demo`
- Lancer le serveur de test
  - `cd ApplicationDemo`
  - `php bin/console server:run`
- Acceder l'application de démo
  - `http://127.0.0.1:8000`

Arborescence d'un projet Symfony

assets : Feuilles de styles, scripts, images...
bin : Exécutables (console)
config : Configuration
public : Racine web
src : Notre code !
templates : *Templates* HTML
tests : Tests automatique (unitaires)
translations : Traductions (site multilingue)
var : Fichiers générés automatiquement
vendor : Modules importés

Plan

World Wide Web (www)

- Sir Tim Berners-Lee
- Système hypertexte (hyperliens)
- Une application d'Internet – 90's
- 3 éléments
  - Adresses web : URI (*Uniform Resource Identifier*)
  - Protocole de communication : HTTP (*Hypertext Transfer Protocol*)
  - Format de données : HTML (*HyperText Markup Language*)
- 2 organismes
  - IETF (*Internet Engineering Task Force*)
  - W3C (*World Wide Web Consortium*)

Adresse : protocole : http://, adresse ip, n° de port et répertoire
Nota : remplacé par nom de domaine

Les URI

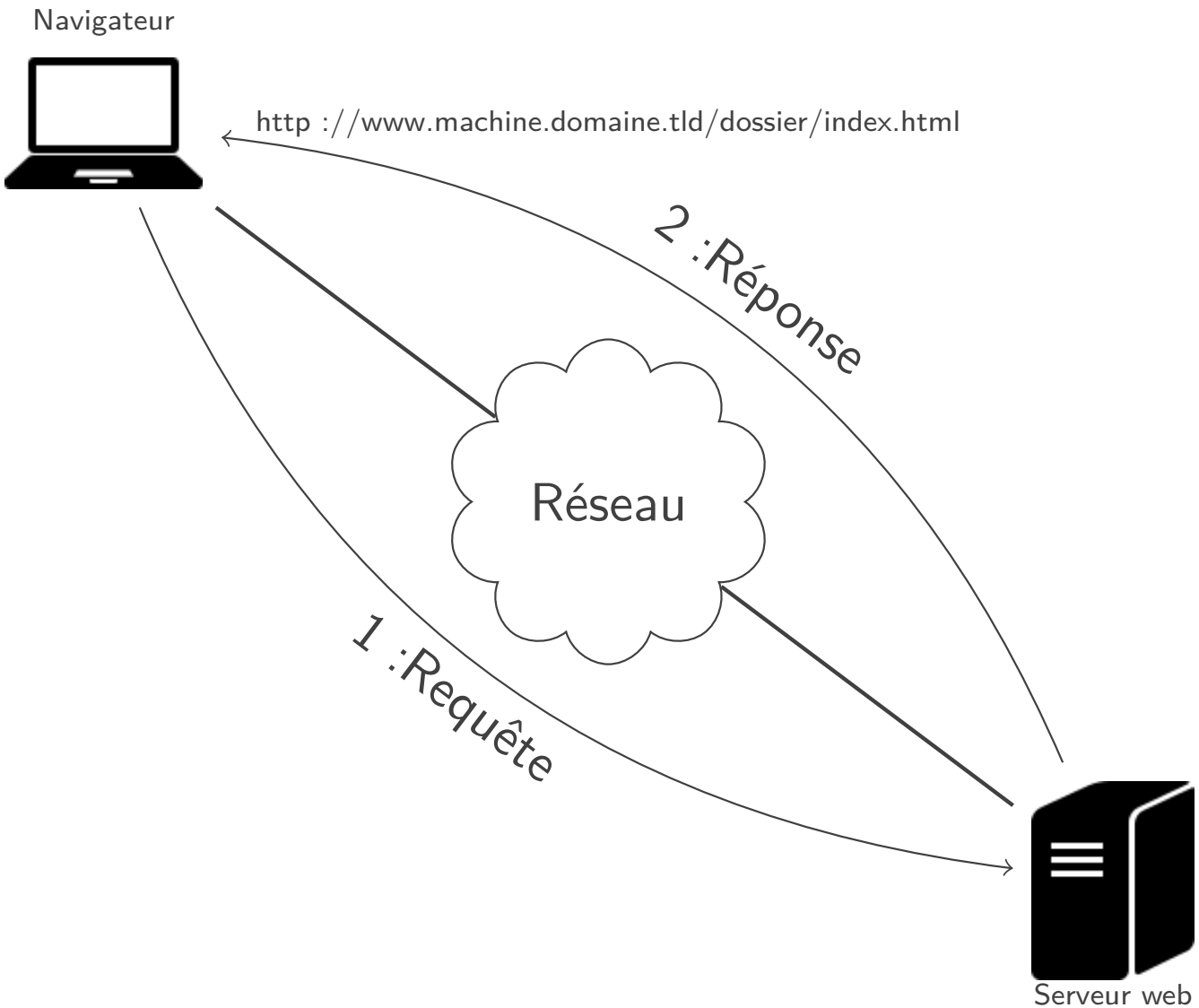
```
scheme:[//[user:password@]host[:port]][/]path[?query][#fragment]
```

URI

- scheme : type d'URI (protocole)
- host : nom d'hôte (DNS)
- path : chemin d'accès sur l'hôte

Protocoles du web

- IP : *Internet Protocol*
- TCP/UDP : *Transmission Control Protocol*/*User Datagram Protocol*
- TLS : *Transport Layer Security*
- HTTP : *Hypertext Transfer Protocol* – 80/tcp
- DNS : *Domain Name System* – 53/udp 53/tcp
- HTTPS : Secure HTTP – 443/tcp


Navigateur
http ://www.machine.domaine.tld/dossier/index.html
2 :Réponse
Réseau
1 :Requête
Serveur web

Le modèle Requête/Réponse

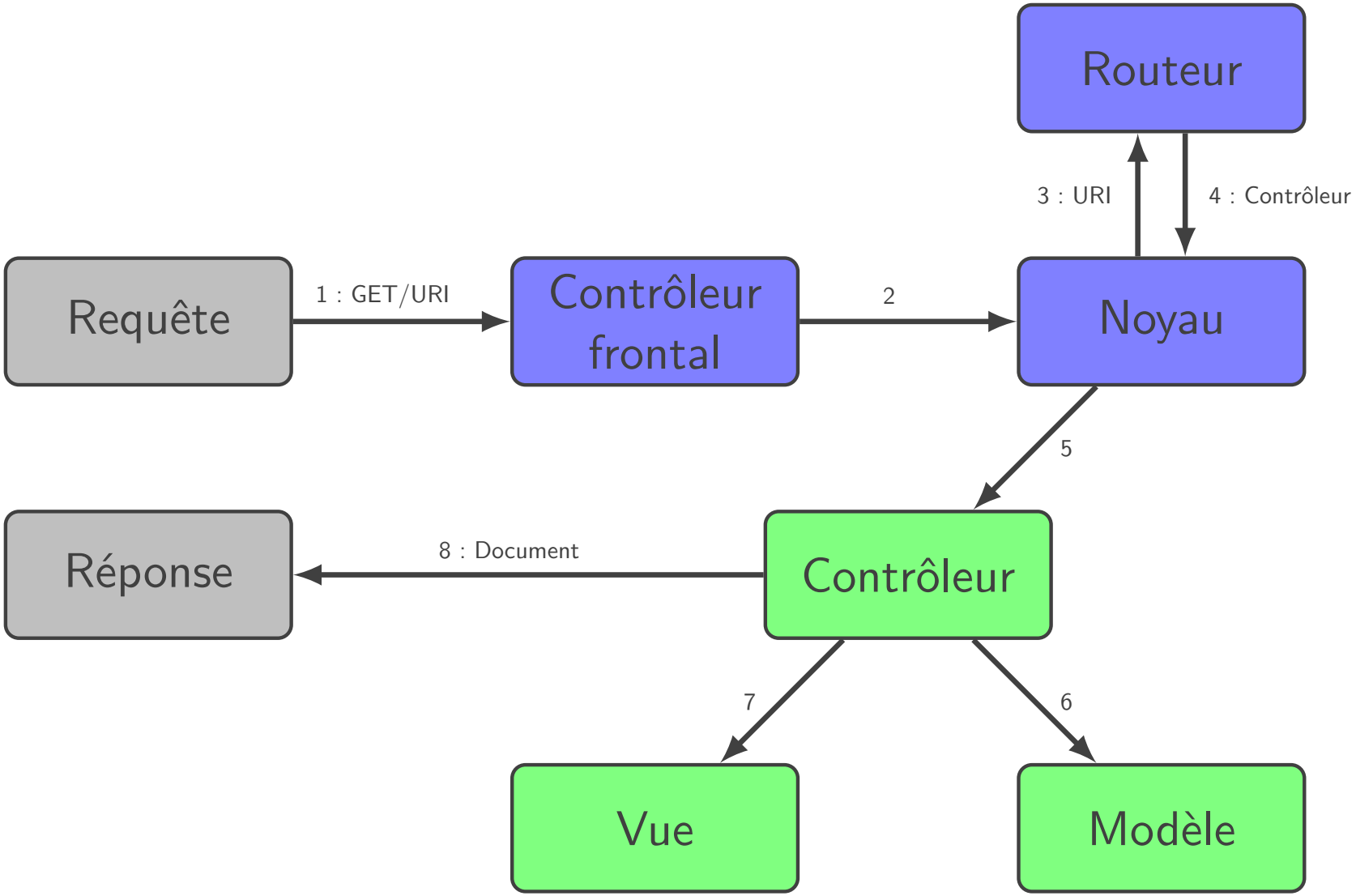
La requête

- Envoyée par le navigateur
- Contient :
  - URL : Chemin/Paramètres
  - Méthode : GET/POST
  - Entêtes HTTP...

La réponse

- Générée par Symfony
- Contient :
  - Ressource demandée : Document
  - Code de statut : 200/OK, 404/Not found
  - Entêtes HTTP...

Fonctionnement général



Plan

Le contrôleur frontal

Front Controller

- Le contrôleur frontal
  - public/index.php
- Notion d'environnement
  - Production : prod
  - Developpement : dev
- Script PHP
  - Instancier le noyau
  - Récupérer la requête
  - Obtenir la réponse
  - Envoyer la réponse

À vous de jouer !

Faites connaissance avec le contrôleur frontal

- Sur l'application symfony-demo
- `http://127.0.0.1:8000/`
- Jetez un coup d'oeil au fichier `public/index.php`

Le profiler

Symfony profiler

- Accessible en développement uniquement
- `index.php/_profiler`
- Informations de développement
- Absent en prod (environnement)

Gestion du cache

- Enregistrement d'informations pré-calculées
- Accélérer le traitement des requêtes
- Informations stockée dans `var`
- Information invalide ?
  - Régénération automatique (dev)
  - Nettoyer le cache
  - `php bin/console cache:clear [--env=prod]`
- En dev : Nettoyer le cache quand dysfonctionnement
- En prod : Nettoyer le cache à chaque modification

À vous de jouer !

Cache et profiler

- Nettoyez le cache
- Explorez les informations du profiler

Le routeur

Gestion des URL

- Composant Symfony
- Analyser l'URL
- Définir le contrôleur correspondant
- Passage de paramètres
- URL « jolies »

Configuration du routeur

- Dans le code : annotations
- Autres formats possibles :
  - Yaml, XML, PHP
  - Dans `/config/routes.[yaml|xml|php]`

Les contrôleurs

Controller

- Routeur : association URL $\Leftrightarrow$ Contrôleur .. Méthode
- Contrôleur : méthode
- Regroupées en classes de contrôleurs
- Paramètres

Le travail d'un contrôleur

- Interroger le modèle
- Fabriquer une réponse en utilisant une vue

À vous de jouer !

Explorer le fonctionnement général

- Sur l'application symfony-demo
- Jetez un coup d'oeil au fichier `src/Controller/BlogController.php`
  - Classe `BlogController` dérivée de `AbstractController`
  - Méthode pour chaque « page » (index, postShow...)
  - Annotations `@Route` **C'est bien dans un commentaire !!!**
    - URL avec paramètres entre accolades
    - Nom
  - Appel à méthode render()
    - Paramètre : `xxx.format.twig`

Twig

- Moteur de *template* (patron)
- Séparé le HTML des données dynamique
- Syntaxe particulière
- Échappement automatique
- Héritage de *templates*

À vous de jouer !

Explorer le fonctionnement général

- Sur l'application symfony-demo
- Jetez un coup d'oeil au fichier `templates/blog/post_show.html.twig`
- Puis à `templates/base.html.twig`
  - Du html
  - Du code twig : `{{ }}`, `{% %}` ou `{# #}`

Plan

À vous de jouer !

Première page

- Reprendre le premier projet
- Acceder au site via le serveur de dev
- Des remarques ?

Une première page

Générer une réponse

- Objet *Response*
  - Constructeur paramétré par le document
- Création d'un contrôleur
- Association avec une route (URL)

À vous de jouer !

Première page

- Installer les annotations
  - `composer require annotations`
- Créer une classe de contrôleurs
  - `src/Controller/HelloController.php`
- Création d'une méthode/contrôleur
- Création d'une route

À vous de jouer !

```php
<?php
// src/Controller/HelloController.php

namespace App\Controller;

use Symfony\Component\HttpFoundation\Response;
use Symfony\Component\Routing\Annotation\Route;

class HelloController
{
    /**
     * @Route("/hello")
     */
    public function hello()
    {
        return new Response('Hello world !');
    }
}
```

À vous de jouer !

Testons

- Lancez votre nouvelle page
  - `http://127.0.0.1:8000/hello`
- Créez un second contrôleur/route qui envoi un hello HTML correct
- Des remarques ?
- Jetez un coup d'oeil aux informations du profiler
- Créer 2 ou 3 autres pages de votre choix

Classe *Controller*

- Les contrôleurs sont méthodes (*PHP Callable*)
  - Associée à une(des) route(s) (annotations)
  - Passage de paramètres
- Réunis dans des classes
  - Nom : `NomController.php` avec suffixe `Controller`
  - Répertoire `src/Controller/`
  - Une classe par fonctionnalité/section du site

La classe Controller

Les contrôleurs

- Hérite de la classe `Controller`
  - `Symfony\Bundle\FrameworkBundle\Controller\`
- Hérite de méthodes

render()	Générer la réponse avec un vue
redirect()	Rediriger vers URI en paramètre
redirectToRoute()	Rediriger avec nom de route

- Gestion d'erreurs
- Accès aux services

À vous de jouer !

```php
<?php
// src/Controller/CompleteController.php

namespace App\Controller;

use Symfony\Component\HttpFoundation\Response;
use Symfony\Component\Routing\Annotation\Route;
use Symfony\Bundle\FrameworkBundle\Controller\Controller;

class CompleteController extends Controller
{
    /**
     * @Route("/redirect")
     */
    public function redirection()
    {
        return $this->redirect('hello');
    }
}
```

À vous de jouer !

Testez la redirection

- Créer une nouvelle classe de contrôleurs
- Ajouter un contrôleur (et une route) qui redirige sur une autre page
- Testez

Générer une réponse

Méthode `render()`

- Méthode héritée de la classe `Controller`
- 2 paramètres
  - Nom de la vue (String)
  - Tableau associatif de paramètres (optionnel)

Nom de vue

- Fichiers Twig
- `templates/dossier_vue/nom_vue.format.twig`
- Convention : snake_case

Plan

Fichiers twig

- Dans `/templates/`
- Regroupées en dossier
  - Un par classe de contrôleurs (par exemple)
- `render("hello/hello.html.twig")`
  - Utilise `hello.html.twig`
  - Qui se trouve dans `/templates/hello/`
- Contient :
  - du HTML
  - du code twig (dynamique)
- Format : permet de produire n'importe quel fichier texte (HTML, CSS, JSON, XML, CSV, LaTeX...)

À vous de jouer !

```php
/**
 * @Route("/hello_twig", name="hello_twig")
 */
public function helloTwig()
{
    return $this->render('hello/hello.html.twig');
}
```

À vous de jouer !

Construire une vue

- Dans hello.html.twig
- Faire un Hello World HTML Propre
- Testez

Héritage de patron

Template inheritance & layouts

- Mutualisation des parties communes
- Héritage de patron
- Mise en page générale (*layout*)
- Utilisation de blocs personnalisables
- À chaque niveau de l'héritage
  - Déclarer des blocs
  - Définir ou redéfinir des blocs
  - Réutiliser le contenu du bloc parent

Exemple de *layout*

```
{# Commentaire en twig #}
{# templates/layout.html.twig #}
<doctype html>
<html>
  <head>
    <title>
      {% block titre %}
        Titre par défaut
      {% endblock %}
    </title>
  </head>
  <body>
    {% block corps %}
    {% endblock %}
  </body>
</html>
```

Exemple d'héritage

```
{# /templates/dossier_vue/vue.html.twig #}
{% extends 'layout.html.twig' %}

{% block titre %}
        {{ parent() }} - Ajout
{% endblock %}

{% block body %}
        {# Contenu du block #}
{% endblock %}
```

.. Reprends le bloc titre de la classe parent et le texte « - Ajout »

À vous de jouer !

Héritage des vues

- Mettez en place un *layout* général pour votre site
- Utilisez ce *layout* pour créer votre vue
- Testez

Héritage de patron – Bonnes pratiques

Approche à 3 niveaux

- `templates/base.html.twig`
  - Contient le patron général du site
- `templates/section/layout.html.twig`
  - Contient le patron commun à une Section du site
  - Hérite de `base.html.twig`
- `templates/section/page.html.twig`
  - Contient la vue spécifique d'une page
  - Hérite de `section/layout.html.twig`

À vous de jouer !

Un vrai site

- Construire une classe de contrôleurs pour les pages d'accueil et d'information de votre site
- Ajouter les contrôleurs et route pour
  - Accueil
  - Mentions légales
  - Contacts
  - Politique de confidentialité
  - ...
- Structurer vos vues selon le modèle à 3 niveaux

Quelques fonctions Twig

Fonction

parent() : Récupérer le contenu du bloc parent

path("nom_route") : Créer une URI

asset("document") : Créer le chemin vers un document

absolute_url(chemin) : Créer le chemin absolu correspondant au chemin relatif donné

Exemples

```
{# Faite un lien relatif vers une route #}
<a href="{{ path('nom_route') }}"></a>

{# Faite un lien absolu vers une route #}
<a href="{{ absolute_url(path('nom_route')) }}"></a>

{# Lier une feuille de style #}
<link rel="stylesheet" href={{ asset("css/style.css") }}>
{# Emplacement : /public/css/style.css #}
```

À vous de jouer !

La navigation

- Dotez votre site de menu de navigation
- Mettez en place une feuille de style

Résumé de syntaxe Twig

En général

{# #} Commentaires

{{ }} Affichage de valeurs

{% %} Instruction

Instructions

block/endblock déclarer/définir un bloc

extends hériter d'un patron

Plan

Annotations

- En commentaire des contrôleurs
- Syntaxe : `@Route("chemin", name="nom_route")`
- Chemin : relatif à la racine (ou au prefixe)
  - Définition d'un prefixe pour les contrôleurs d'une classe
  - `@Route("un_prefixe")` en commentaire de la classe
- nom_route : unique et *snake_case*
- Routeur : utilise la première route qui correspond

Paramètres de contrôleurs

Routes

- Chemin : `/chemin/{nom}`
- nom : identifiant paramètre du contrôleur
- Contraintes[1] : `requirements={"nom": "regex"}`

1. Pour aller plus loin : `https://symfony.com/doc/current/routing/requirements.html`

Exemple de route

```
/**
 * @Route("/article/{id}", name="blog_article",
     requirements={"id": "\d+"})
 */
public function lireArticle($id)
{
    // TODO
}
```

\d+ : que ce soit au moins un chiffre

Paramètres des vues

- Parametre supplémentaire de la méthode `render()`
- Tableau associatif

```
/**
 * @Route("/article/{id}", name="blog_article",
    requirements={"id": "\d+"})
 */
public function lireAction($id)
{
    return $this->render(':blog:article.html.twig',
                         array("id"=> $id) );
}
```

À vous de jouer !

Le blog

- Créer un nouveau paquet pour ajouter un blog à votre site
- Configurer la délégation de route avec le préfixe /blog
- Ajouter 2 squelettes de contrôleurs pour :
  - La page d'accueil de la partie blog
  - La lecture d'un article

À vous de jouer !

Hello You

- Ajouter :
  - un contrôleur
  - une route avec un paramètre
  - une vue
- Objectif
  - afficher hello avec le nom récupéré en paramètre dans l'URL

À vous de jouer !

Le blog

- Créer une nouvelle classe de contrôleurs pour ajouter un blog à votre site
- Les routes auront le préfixe /blog
- Ajouter 2 squelettes de contrôleurs pour :
  - La page d'accueil de la partie blog
  - La lecture d'un article

Plan

Doctrine

- ORM : *Object-Relational Mapping*
- Logiciel libre
- Pas spécifique à Symfony
- Pas obligatoire mais bien intégré dans Symfony
- Objets persistants
- Abstraction de la base de données

Installation de Doctrine

Doctrine & maker

- `composer require doctrine maker`
- Installe Doctrine et maker dans le projet courant
- maker aide à la génération de code

Configuration de Doctrine

Accès à une base

- Fichier `.env`
- Accès à MySQL

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

Mise en place de la base

Dans la base

- Création du compte dans la base
- Configuration des privilèges

Avec la console Symfony

- Destruction d'un schema
  - `php bin/console doctrine:database:drop`
- Création d'un schema
  - `php bin/console doctrine:database:create`
- Bonne configuration du schema (*default charset & collation*)

À vous de jouer !

Mise en place de la base

- Configurer un compte dans votre base
- Accordez les privilèges adéquats
- Configurer l'accès à la base dans
  - `.env`
- Détruire (le cas échéant) et créer le schema
  - `php bin/console doctrine:database:drop`
  - `php bin/console doctrine:database:create`

Entity

- Des objets persistants
- Doctrine s'occupe de la base
- Répertoire `src/Entity`
- Création de classes (Modèle)
  - Classe ⇒ Table
  - Objet ⇒ Enregistrement
  - Attribut ⇒ Champ
  - Correspondance de type [1]
- Correspondance avec des annotations

1. `http://docs.doctrine-project.org/projects/doctrine-orm/en/latest/reference/basic-mapping.html`

Exemple d'entité

```php
<?php
// src/Entity/Entite.php
namespace App\Entity;

use Doctrine\ORM\Mapping as ORM;

/**
 * @ORM\Table(name="nom_table")
 * @ORM\Entity
 */
class Entite
{
    /**
     * @ORM\Column(name="nomAttribut", type="typeDoctrine")
     */
    private $nomAttribut;

    public function setnomAttribut($valeur)
    { ... }
    public function getnomAttribut()
    { ... }
}
```

Doctrine utilise les annotations aussi

Les commandes doctrine/make

Console Symfony

- Générateur d'entité
  - `php bin/console make:entity NomEntite`
- Création/Mise à jour des tables
  - `php bin/console doctrine:migrations:diff`
  - `php bin/console doctrine:migrations:migrate`

À vous de jouer !

Les articles

- Construisez l'entité Article
- Pour tester progressivement la manipulation des entités :
  - Créer une nouvelle route/contrôleur pour ajouter un article
  - Instanciez un objet article
  - Initialisez ses attributs
- Note : pour acceder aux articles depuis un contrôleur
  - `use App\Entity\Article;`

Parenthèse Twig

- Valeur objet dans le tableau des paramètres twig
- Dans la vue
  - Accès aux attributs d'un objet via `objet.attribut`
  - Utilise les getters si attribut privé
    - `attribut`
    - `attribut()`
    - `getAttribut()`
    - `isAttribut()`
    - `hasAttribut()`
  - Permet aussi l'accès aux cases de tableaux

Parenthèse Twig

- Application de filtre à une variable Twig [1]
- variable|filtre
- Exemple de filtres
  - variable|length : nombre d'éléments
  - variable|date("format") : affichage des dates
  - variable|upper : mise en majuscule
  - variable|lower : mise en minuscule

1. cf : `http://twig.sensiolabs.org/doc/2.x/filters/index.html`

Exemple d'affichage d'objet

```
{# Fichier Twig #}
<h1>{{ article.titre|upper }}</h1>
<em>Date : {{ article.date|date("d/m/Y") }}</em>
```

À vous de jouer !

Afficher des articles

- Dans le contrôleur d'ajout d'article
  - Utilisez une vue en lui passant en paramètre l'article créé
- Créé cette vue qu'elle affiche l'article

Enregistrer les objets

Persistance

- Composant doctrine : *entity manager*
- Récupération dans un contrôleur :
  - `$this->getDoctrine()->getManager()`
  - Accès au service Doctrine
- Confier une entité au gestionnaire
  - `$gestionnaire->persist($entite)`
- Enregistrer les changements
  - `$gestionnaire->flush()`

À vous de jouer !

Enregistrer des articles

- Dans le contrôleur d'ajout d'article
  - Enregistrez l'article créé
  - Affichez l'identifiant de l'article dans la vue
- Testez et vérifiez

Récupérer un objet existant

Repository

- Utilisation d'un dépôt (un par objet)
- Existe par défaut – personnalisable
- Récupération du dépôt :
  - `$this->getDoctrine()->getRepository('nom')`
  - Nom : `Entite::class`
- Requête sur le dépôt
  - `$depot->find($id)`

Exception Symfony

- Tentative d'accès à une ressource inexistante
- Soulever une exception
- Erreur HTTP 404
- Méthode `createNotFoundException(description)`
- Classe `Controller`

À vous de jouer !

Lecture des articles

- Dans le contrôleur de lecture d'article
  - Aller chercher l'article avec l'identifiant de l'URI
  - Déclenchez une erreur si l'identifiant n'existe pas
  - Affichez correctement l'article
- Testez

Récupération automatique

ParamConverter

- Composant ParamConverter
- Récupération automatique d'entité (cas simple)

```php
use App\Entity\Article;
class BlogController extends Controller
{
   /**
    * @Route("/lire/{id}", name="blog_affiche_article",
     requirements={"id"="\d+"})
    */
   public function afficheArticle(Article $article)
   {
       return
     $this->render('blog/affiche_article.html.twig',
     array("article" => $article));
   }
}
```

La classe *EntityRepository*

Les méthodes

- `find(id)` : Requête par id
- `findAll()` : Tous les enregistrements
- Méthode générée automatiquement pour chaque attribut
  Attribut : doit être remplacé par le nom de l'attribut
  - `findOneByAttribut(valeur)` : Le premier enregistrement
  - `findByAttribut(valeur)` : Tous les enregistrements correspondants
- Avec un tableau de paramètres
  - `findOneBy(tableau)` : Le premier enregistrement
  - `findBy(tableau)` : Tous les enregistrements correspondants

Requêtes paramétrées

`findOneBy/findBy`

- 1er paramètre : tableau
  - `"attribut" => "valeur"`
- 2e paramètre : tableau
  - `"attribut" => "ASC|DESC"`
- Pour `findBy` seulement
  - 3e paramètre : nombre d'éléments (*limit*)
  - 4e paramètre : premier élément (*offset*)

Exemple de requêtes

```
// Dans un controleur
{
  // Recuperation du depot
  $depot = $this->getDoctrine()
                ->getRepository('Entite::class');

  // Recuperation du tableau de toutes les entites
  $listeEntites = $depot->findAll();

  // Recuperation des 5 plus recents article de John
  $listArticleJohn = $depot->findBy(
                        array("auteur" => "john"),
                        array("date" => "desc"), 5, 0);
}
```

Parenthèse Twig

- Boucle
  - `{% for element in collection %} {% endfor %}`
- Condition
  - `{% if test %} {% else %} {% endif %}`
- Tests [1]
  - `not|and|or`
  - `{% if variable is defined %}`
  - `{% if variable is empty %}`
  - `{% if variable is null %}`

1. `http://twig.sensiolabs.org/doc/2.x/tests/index.html`

Parenthèse Twig

- Fonction twig `path()`
- 1er paramètre : nom de la route
- 2e paramètre : tableau twig
  - `{"clé" => valeur}`

```
<a href="{{ path('nom_route', {'id': 1}) }}"></a>
```

À vous de jouer !

Lecture des articles

- Afficher l'ensemble des articles sur la page d'accueil du blog
- Ajouter un menu qui affiche une liste des articles

Supprimer un objet

Entity Manager

- Récupérer l'objet
- Demander la suppression
  - `$gestionnaire->remove($objet)`
- Modification effectuée au `flush()`
  - `$gestionnaire->flush()`

Le *FlashBag*

- Messages à usage unique
- Stockés dans la session
- Dans un contrôleur
  - `$this->addFlash("type","message")`
- Dans une vue twig
  - `{% for message in app.session.flashBag.get('type') %}`
  - `{# ... #}`
  - `{% endfor %}`

À vous de jouer !

Suppression des articles

- Ajouter un bouton pour supprimer un article

Plan

Le composant `Form`

- Permet la création et la gestion des formulaires
- Validation des données
- Protection CSRF (*Cross-Site Request Forgery*)
- Intégration twig
- Initialiser les attributs d'un objet

Installer le composant Form

- `composer require form`

Le composant *FormBuilder*

- Créer un formulaire correspondant à l'objet à initialiser
  - `createFormBuilder($objet)`
- Faire des correspondances attributs/champs de formulaires
  - `$formulaire->add($attribut, Classe)`
  - Type : `Symfony\Component\Form\Extension\Core\Type\`
- Créer le formulaire
  - `$formulaire->getForm()`
- Envoyer le formulaire à la vue
  - `$formulaire->createView()`

Form types[1]

- `TextType`
- `TextareaType`
- `DateType`
- `DateTimeType`
- `SubmitType`
- `...`

1. Pour aller plus loin : `https://symfony.com/doc/current/reference/forms/types.html`

Exemple de formulaire

```php
// Controleur
use Symfony\Component\Form\Extension\Core\Type\TextType;
use Symfony\Component\Form\Extension\Core\Type\DateType;
use Symfony\Component\Form\Extension\Core\Type\SubmitType;

{
  $objet = new Objet();

  $formulaire = $this->createFormBuilder($objet)
                     ->add('texte', TextType::class)
                     ->add('date', DateType::class)
                     ->add('save', SubmitType::class)
                     ->getForm();
  return $this->render('vue.html.twig',
      array('formulaire' => $formulaire->createView()));
}
```

Afficher les formulaires

Parenthèse Twig

- Fonction form()
  - `{{ form(formulaire) }}` – Entier
- Personnaliser le formulaire
  - `{{ form_start(formulaire) }}`
  - `{{ form_end(formulaire) }}` – Éléments manquants
- Afficher les erreurs
  - `{{ form_errors(formulaire) }}` – générales
  - `{{ form_errors(formulaire.champ) }}` – spécifiques
- Afficher les éléments
  - `{{ form_row(formulaire.champ) }}` – champ entier
  - `{{ form_label(formulaire.champ) }}`
  - `{{ form_widget(formulaire.champ) }}`
  - `{{ form_widget(formulaire) }}` – Tous les éléments

1. `https://symfony.com/doc/current/reference/forms/twig_reference.html`

À vous de jouer !

Ajout des articles

- Dans le contrôleur d'ajout d'article
  - Créer un formulaire pour gerer l'ajout d'article
  - Afficher le formulaire
- Modifier la vue en conséquence

Traitement de la requête

- Objet *Request*
  - `use Symfony\Component\HttpFoundation\Request;`
- En paramètre du contrôleur
  - `$public function controleur(Request $requete)`
- Associer le formulaire à la requête
  - `$formulaire->handleRequest($requete)`
- Méthodes
  - `$formulaire->isSubmitted()`
  - `$formulaire->isValid()`
- Traitement : l'objet associé à été modifié

Exemple de traitement

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

public function ajout(Request $requete)
{
  $objet = new Objet();

  $formulaire = $this->createFormBuilder($objet)
                     ->add('texte', TextType::class)
                     ->add('date', DateType::class)
                     ->add('save', SubmitType::class)
                     ->getForm();

  $formulaire->handleRequest($requete);
  if( $formulaire->isSubmitted() && $formulaire->isValid() )
    // Traitement du formulaire
  else
    // Affichage du formulaire
}
```

À vous de jouer !

Ajout des articles

- Ajouter le traitement du formulaire d'ajout d'article au contrôleur

Les contraintes [1]

- `composer require validator`
- Annotations/YAML/XML/PHP
- Annotation dans la classe de l'objet à remplir
  - `use Symfony\Component\Validator\Constraints as Assert;`
- En commentaire au dessus des attributs
  - `@Assert\contrainte`
- Contraintes :
  - `NotBlank()`
  - `NotNull()`
  - `Type()`
  - `Length(min=X,max=Y)` – Chaînes
  - `Email()`

1. Pour aller plus loin : `https://symfony.com/doc/current/reference/constraints.html`

À vous de jouer !

Ajout des articles

- Ajouter les contraintes nécessaires à vos articles

Externaliser les formulaires

- Création de classes de formulaire
  - Dans `src\Form\`
  - `FormulaireType` (Suffixe *Type*)
  - Hérite de `Symfony\Component\Form\AbstractType`
- Possède une méthode qui construit le *FormBuilder*
  - `public function buildForm( FormBuilderInterface $builder, array $options)`
- Dans le contrôleur on récupère directement le formulaire
  - `use App\Form\FormulaireType;`
  - `$this->createForm(FormulaireType::class, $objet);`
- Génération d'un squelette (console)
  - `php bin/console make:form`

À vous de jouer !

Le formulaire ArticleType

- Générez automatiquement le formulaire pour l'entité Article
- Vérifiez et ajoutez les types de champs
- Remplacer dans le contrôleur d'ajout
- Bonne idée : ajoutez le bouton *submit* dans la vue

Mise à jour des entités

Parenthèse Doctrine

- Pour mettre à jour une entité :
  - Récupérer l'entité
  - Utilisez les *setters*
  - Appelez `flush()` sur le gestionnaire d'entité
- Les modifications faites sur un objet entité récupéré depuis doctrine sont répercutées en cas de `flush()`

À vous de jouer !

Modifier les articles

- Ajouter la possibilité de modifier les articles à votre paquet

Plan

Relation entre entités

- Différent type d'associations
  - *One to One*
  - *One to Many / Many to One*
  - *Many to Many*
- L'association doit être déclarée dans l'entité propriétaire de la relation
  - Celle qui contient la clé étrangère
  - Les mises à jour doivent être faites du coté propriétaire
- Peut être déclarée dans l'autre entité aussi
- Doctrine : annotations

Exemple d'association *Many to One*

```php
class EntiteLiee
{
    // ...

    /**
     * @ORM\ManyToOne(targetEntity="Entite",
     *                inversedBy="entitesLiees")
     * @ORM\JoinColumn(name="entite_id",
     *                 referencedColumnName="id")
     */
    private $entite;
}
```

Exemple d'association *One to Many*

```php
use Doctrine\Common\Collections\ArrayCollection;

class Entite
{
    // ...

    /**
     * @ORM\OneToMany(targetEntity="EntiteLiee",
     *                mappedBy="entite")
     */
    private $entitesLiees;

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

À vous de jouer !

Les commentaires

- Ajoutez la possibilité de commenter les articles

Plan

Fichier `security.yml` [1]

- `/config/pakages/security.yaml`
- `providers`
  - Base d'utilisateur
- `firewalls`
  - `dev` : exclusion (profiler)
  - `main` : Règles de filtrage

1. Pour aller plus loin : https://symfony.com/doc/current/reference/configuration/security.html

La clé `pattern`

- Dans un `firewall`
- Indique les URL prisent en charge par ce `firewall`
- Expression rationelle
- Examen dans l'ordre
- Absence : toutes les URL
- Filtrage par hôte/méthode (HTTP)

Authentication & Authorization

- Symfony distingue 2 parties
- L'authentification (*Authentication*)
  - Vérification de l'identité d'une personne
  - Association avec un objet *UserInterface*
- Le contrôle d'accès (*Authorization*)
  - Interdire l'accès à certaines parties du site pour certains utilisateurs

La base d'utilisateur (*Authentication*)

La clé `providers`

- Où sont les données d'authentifications
- Login/mots de passe
- Association avec des rôles
  - Au choix (Préfixe ROLE_)
  - Droits d'accès (groupe)

La clé `encoders`

- Comment sont stockés les mots de passe
- Recommandation : bcrypt (hash)

Exemple

```yaml
# app/config/security.yml
security:
    providers:
        in_memory:
            memory:
                users:
                    un_login:
                        password: truc_illisible
                        roles: 'ROLE_ADMIN'
    encoders:
        Symfony\Component\Security\Core\User\User:
            algorithm: bcrypt
```

Génération de mot de passe

Console Symfony

- Une fois l'encodeur défini :
  - `php bin/console security:encode-password`

À vous de jouer !

Les utilisateurs

- Créez une petite base d'utilisateurs pour votre application

Restreindre l'accès

- 3 stratégies [1] :
  - Dans `config/packages/security.yaml`
    - Contrôle d'accès à un ensemble d'URI
  - Les annotations `@Security`
    - Restreindre l'accès à un contrôleur
  - Le service `security.authorization_checker`
    - Cas les plus complexes

1. Pour aller plus loin : https://symfony.com/doc/current/best_practices/security.html

Exemple de contrôle d'accès

```yaml
# /config/packages/security.yaml

security:
    access_control:
        - { path: ^/admin, roles: ROLE_ADMIN }
```

^ veut dire « commence par… »

Exemple d'annotation de sécurité

```
// Dans une classe de controleurs

use
  Sensio\Bundle\FrameworkExtraBundle\Configuration\Security;

/**
 * Ce controleur n'est accessible que pour ROLE_ADMIN
 * @Security("has_role('ROLE_ADMIN')")
 */
public function adminAction()
{
    // ...
}
```

À vous de jouer !

Contrôle d'accès

- Permettre uniquement à certains utilisateurs d'écrire des articles

Formulaire de connexion

- Définir un formulaire de connexion
  - Dans `config/packages/security.yaml`
  - Ajouter les chemins login_path et login_check
- Créer la route et le contrôleur
- Créer une vue
  - Avec le formulaire

Exemple

```yaml
# config/packages/security.yaml

security:
  firewalls:
    main:
      form_login:
        login_path: login
        check_path: login
        # Ajout de la protection CSRF
        csrf_token_generator: security.csrf.token_manager
```

Exemple de contrôleur

```
use
    Symfony\Component\Security\Http\Authentication\AuthenticationU

/**
 * @Route("/login", name="login")
 */
public function login(Request $requete, AuthenticationUtils
    $authentificateur)
{
  // On utilise le service d'authentification

  // On recupere le message d'erreur
  $erreur = $authentificateur->getLastAuthenticationError();

  // On recupere le nom entre precedemment
  $login = $authentificateur->getLastUsername();

  // On fait le rendu d'une vue qui affiche les erreurs et
  // le login entre precemment le cas echeant
  return $this->render('nomVue.html.twig',
                       array("login"=>$login,
                             "erreur"=>$erreur));
}
```

Exemple de vue

```
{# nomVue.html.twig #}

{% if erreur %}
  <p>{{ erreur.messageKey|trans(erreur.messageData,
    'security') }}</p>
{% endif %}
<form action="{{ path('login_check') }}" method="post">
  <label for="username">Nom d'utilisateur:</label>
  <input type="text" id="username" name="_username"
    value="{{ login }}" />
  <br>
  <label for="password">Mot de passe:</label>
  <input type="password" id="password" name="_password" />
  <input type="hidden" name="_csrf_token" value="{{
    csrf_token('authenticate') }}">
  <br>
  <button type="submit">login</button>
</form>
```

À vous de jouer !

Le formulaire de connexion

- Ajoutez un formulaire de connexion à votre site
- Vérifiez si vous pouvez vous connectez
- Vérifiez les droits d'accès (rédaction d'article)

Déconnexion

Se déconnecter

- Dans `config/packages/security.yaml`
  - Ajouter un chemin logout
  - Choisir la redirection
- Dans `config/routing.yaml`
  - Créer une route

Exemple de déconnexion

```yaml
# app/config/security.yml

security:
  firewalls:
    main:
      logout:
        path: /logout
        target: /

# app/config/routing.yml

logout:
  path: /logout
```

À vous de jouer !

Se déconnecter

- Mettez en place un lien de deconnexion

Personnaliser l'affichage

- Fonction `is_granted('ROLE')`
  - Vrai si l'utilisateur est connecté et possède le rôle `ROLE`
- Rôles particuliers

  IS_AUTHENTICATED_ANONYMOUSLY : Tout le monde

  IS_AUTHENTICATED_REMEMBERED : Utilisateurs authentifiés (éventuellement via un cookie)

  IS_AUTHENTICATED_FULLY : Connecté

  Utilisateurs authentifiés durant cette session

À vous de jouer !

Liens de connexion

- Ajoutez des liens pour se connecter et se déconnecter le cas échéant
- Personnalisez les liens pour n'afficher que ceux auxquels l'utilisateur à accès.

Hierarchie des rôles

`config/packages/security.yaml`

- Définir une inclusions de rôles
  - Clé `role_hierarchy:`
  - Contient pour chaque rôle l'ensemble des rôles inclus
    - `ROLE_ADMIN: ROLE_USER` (un seul)
    - `ROLE_SUPER: [ROLE_ADMIN, ROLE_AUTRE]` (un ensemble)

À vous de jouer !

Contrôle des commentaires

- Seul les utilisateurs authentifiés peuvent commenter
- Les rédacteurs peuvent aussi commenter

Plan

Plan

Les *Repository*

- Pour chaque entité
- `src/Repository/NomEntiteRepository.php`
- Contient les requêtes pour l'entité
  - Si les `find*()` ne suffisent pas
- Construction des requêtes :
  - *QueryBuilder* **On appelle une méthode**
  - DQL – *Doctrine Query Language*
  - SQL

Exemple de jointure

```php
<?php
namespace App\Repository;
use App\Entity\Client;
use Doctrine\Bundle\DoctrineBundle\Repository...
                        ...\ServiceEntityRepository;
use Symfony\Bridge\Doctrine\RegistryInterface;

class ClientRepository extends ServiceEntityRepository
{
  public function findOneClientWithContacts($id)
  {
                    // SELECT et FROM a partir de l'entite
                    // Defini un alias
    return $this->createQueryBuilder('client')
                ->where('client.id = :id')
                ->setParameter('id', $id)
                ->leftJoin('client.contacts','contact')
                ->addSelect('contact')
                ->getQuery()
                ->getOneOrNullResult();
  }
}
```

Alias

=Select from client

Attention, where ecrase le where d'origine

Alias

Les méthodes d'un *QueryBuilder*

where('clause') utilisation des paramètres (PDO)

setParameter('nom',valeur) instanciation des paramètres

leftJoin('alias1.attribut','alias2') (join/innerJoin)

addSelect('alias2') ajoute les champs au select

orderBy('alias.attribut','sens') ou addOrderBy si plusieurs

groupBy('alias.attribut') addGroupBy si plusieurs

having('clause') andHaving/orHaving si plusieurs

1. Pour aller plus loin : http://docs.doctrine-project.org/projects/doctrine-orm/en/latest/reference/query-builder.html

Récupérer les résultats

Les méthodes de *Query*

getResult() Tableau d'objets résultats (même si 1 seul)

getArrayResult() Récupérer les résultats en tableaux (rapide mais lecture seule)

getScalarResult() Récupère un tableau de valeurs (même si 1 seule)

getOneOrNullResult() Récupère un objet résultat ou null (Exception si plusieurs)

getSingleResult() Récupère un objet résultat (Exception)

getSingleScalarResult() Récupère une valeur (Exception)

Plan

Modules autonomes

- Programmation modulaire
- Module : *Bundle* ($\simeq$Plug-in)
- Bibliothèques
  - Les composants du framework (/vendor)
  - Des composants importés (/vendor)
- Une arborescence de fichiers

Installer un *bundle*

Installer SwiftMailer

- `composer require mailer`

Les commentaires

- Installez Swift Mailer pour votre projet

Envoyer des mails

Swift Mailer[1]

- Configuration
  - Dans `config/packages/swiftmailer.yaml`
  - Et dans `.env`
- Création et envoi de mail (contrôleurs)
- Utilisation de vue

1. Pour aller plus loin : `https://symfony.com/doc/current/email.html`

Configuration de *Swift Mailer*

```
# .env

###> symfony/swiftmailer-bundle ###
# For Gmail as a transport, use:
    "gmail://username:password@localhost"
# For a generic SMTP server, use:
    "smtp://localhost:25?encryption=&auth_mode="
# Delivery is disabled by default via "null://localhost"
MAILER_URL=null://localhost
# Utilisation d'un serveur SMTP classique
MAILER_URL=smtp://localhost:25?encryption=ssl
    &auth_mode=login&username=&password=

###< symfony/swiftmailer-bundle ###
```

Envoi d'un mail

```
public function envoiMail(\Swift_Mailer $mailer)
{
  // Creation d'un mail
        $message = (new \Swift_Message('Le sujet'))
              ->setFrom('expediteur@domain.tld')
              ->setTo('destinataire@domain.tld')
               // Format texte
    ->setBody($this->renderView('emails/test.txt.twig'),
          'text/plain');
               // Format HTML
    ->setBody($this->renderView('emails/test.html.twig'),
          'text/html');

  // Envoi du mail
  $mailer->send($message);
}
```

À vous de jouer !

Le formulaire de contact

- Configurez `disable_delivery: true` dans `config/packages/test/swiftmailer.yaml`
- Ajoutez un formulaire pour vous envoyer des mails dans la partie contact
- Testez
- Facultatif : testez avec un serveur SMTP (si possible)

Plan

Poursuivre l'exploration

- Standards et conventions
  - `https://symfony.com/doc/current/contributing/code/standards.html`
- Bonnes pratiques
  - `https://symfony.com/doc/current/best_practices/index.html`
- Configuration et paramètres
  - `https://symfony.com/doc/current/configuration.html`
- Les services
  - `https://symfony.com/doc/current/service_container.html`
- La mise en production
  - `https://symfony.com/doc/current/deployment.html`

Doctrine

- *Repository* et personnalisation des requêtes
  - `https://symfony.com/doc/current/doctrine.html#doctrine-queries`
  - Avec *DQL* : `http://docs.doctrine-project.org/projects/doctrine-orm/en/latest/reference/dql-doctrine-query-language.html`
- Les relations *OneToOne* et *ManyToMany*
  - `http://docs.doctrine-project.org/en/latest/reference/association-mapping.html`
- Les *Lifecycle callbacks*
  - `https://symfony.com/doc/current/doctrine/lifecycle_callbacks.html`

- Inclusion de *templates*
  - `https://symfony.com/doc/current/templating.html#index-7`
- Inclusion de contrôleurs
  - `https://symfony.com/doc/current/templating/embedding_controllers.html`
- Approfondir Twig et les *templates*
  - `https://symfony.com/doc/current/components/templating.html`

Autres composants

- Internationalisation
  - `https://symfony.com/doc/current/translation.html`
- Les tests unitaires
  - `https://symfony.com/doc/current/testing.html`
- Gestionnaire d'événements
  - `https://symfony.com/doc/current/event_dispatcher.html`

Pour aller plus loin

- YAML – `http://www.yaml.org/`
  - Introduction – `http://sweetohm.net/article/introduction-yaml.html`
- Twig – `http://twig.sensiolabs.org/`
- Composer – `https://getcomposer.org/`
- Doctrine – `http://www.doctrine-project.org/`
- PHPDoc – `https://www.phpdoc.org/`

Bibliographie

- Site de SensioLabs
- PHP Standards Recommandations (PSR)