Symfony 4.4

Framework PHP

Introduction

Qu’est-ce que Symfony ?

- Composants open-source (Modulaire)
- Framework MVC

Pourquoi utiliser un framework PHP ?

- Structure le code
- Permet le travail en équipe
- Évite de réinventer la roue
- Produit du code de qualité
- Permet une rapidité de production

Les alternatives à Symfony

- Laravel
- Zend Framework
- Slim
- CodeIgniter, CakePHP, Phalcon, Yii

- WordPress
- Joomla!
- Drupal

Pourquoi choisir Symfony ?

- Très utilisé en France
- Communauté hyperactive
- Qualité de code
- Excellente gestion du projet

Symfony 4.4

- Novembre 2019
- LTS
- Installation avec Flex

Configuration du serveur

- PHP 7.2 +
- Apache
- MySQL

- WampServer, XAMPP, MAMP

WampServer 64 Bits

PhpStorm (IDE)

- Navigation dans le code
- Autocomplétion
- Adapté à Symfony
- Puissant
- Payant

PhpStorm

Installation

Symfony Flex

- composer create-project symfony/skeleton:^4.4 MyProject

- composer req requirements-checker
- composer req apache-pack
- composer req annotations
- composer req make

- symfony server:start --port=8000 --no-tls -d

Download Symfony Framework and Components

You're seeing this page because you haven't configured any homepage URL.

Welcome to Symfony 4.4.2

/home/vagrant/Project/MyProject/

Your application is now ready and you can start working on it.

Documentation

Guides, components, references

Tutorials

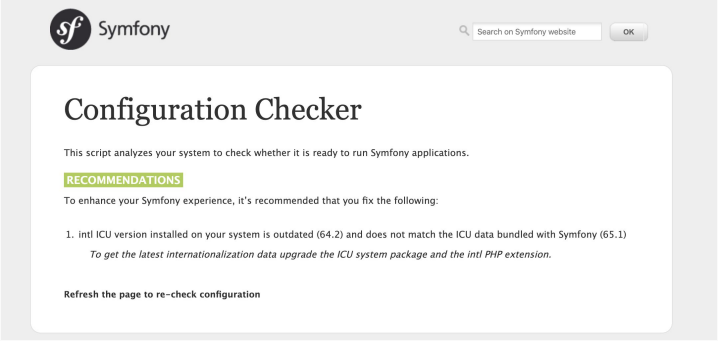
Create your first page

Community

Connect, get help, or contribute

Vérification de l’installation

- **composer req requirements-checker**



- **composer remove requirements-checker**

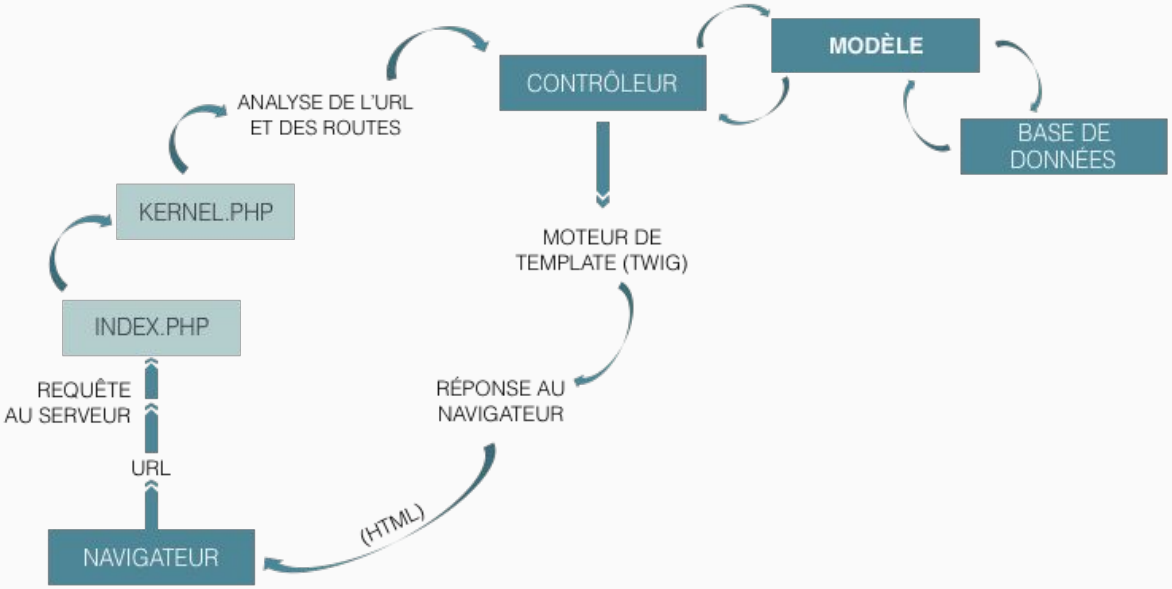
Arborescence

Les répertoires du projet

- **bin**: console et autres exécutables
- **config**: fichiers de configuration de l'application
- **public:** accessible par les navigateurs
- **src:** le code PHP (Controller, Entity, Form, Service, Repository ...)
- **var**: cache et logs
- **vendor**: dépendances du projet, géré par Composer

Routage et Contrôleurs

Parcours d’une requête HTTP sous Symfony



La requête au serveur

- Par une URL
- Le .htaccess redirige sur index.php, le contrôleur frontal
- L'objet ...\HttpFoundation\Request est créé

Request - Symfony 4.4

Le Kernel

- src/Kernel.php
- Le noyau de l'application
- Instancié par le contrôleur frontal
- Chargé de gérer la requête et de retourner une réponse

Response - Symfony 4.4

Le système de routage

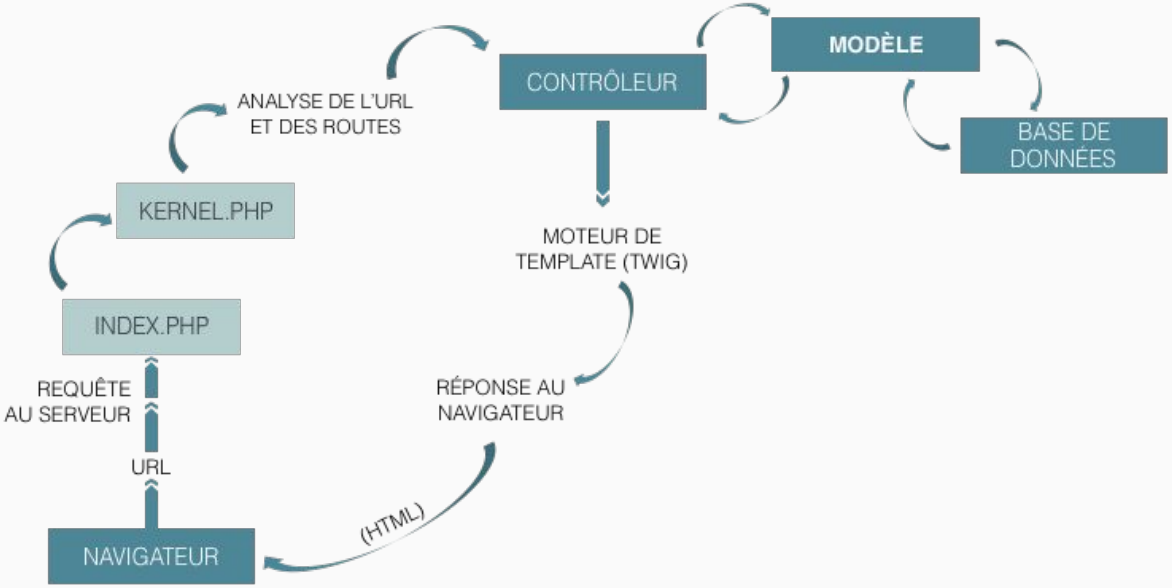
- Une URL => Une fonction

- http://localhost:8000/contact => contact()
- http://localhost:8000/home => home()
- http://localhost:8000/produits/ => showProducts()

Les contrôleurs

- Contiennent les fonctions appelées par le système de routage
- Héritent du contrôleur de Symfony
- Retournent un objet ...\HttpFoundation\Response

Parcours d’une requête HTTP sous Symfony



Les routes

- Correspondance URL / Fonction
- Permettent de facilement créer de jolies URL
- Système souple

- PHP, YAML, XML ou annotations
- Avantage aux annotations

Les annotations

- Commentaires interprétés
- Au-dessus du code PHP associé

```
/**
 * @Route("home", name="home_index", methods={"GET"})
 */
```

- **composer req annotations**

Les paramètres d’annotations de routes

- Précise l’URL menant à la méthode
- Doit être unique
- Forme libre

```
/**
 * @Route("home", name="home_index", methods={"GET"})
 */
```

Les jokers dans l’URL

- Expression rationnelle correspondant au paramètre d’URL

```php
class UserController extends AbstractController
{
    /**
     * @Route("user/{id}", name="user_profile", requirements={"id"="\d+"}, methods={"GET"})
     *
     * @param Request $request
     * @return Response
     */
    public function profile(Request $request): Response
    {
        // Récupération du paramètre
        $id = $request->get( key: 'id');

        return new Response($id);
    }

}
```

Les noms des routes

- Très fortement recommandés
- Permettent de générer les URL correspondant aux routes
- Convention: nom du contrôleur + méthode en snake_case
- Uniques

- Afficher la liste des routes: **php bin/console debug:router**
- Afficher les détails d'une route : **php bin/console debug:router nom**
- Tester le match entre une URL et une route : **php bin/console router:match /url**

Les contrôleurs

- Simple classe PHP
- Hérite du contrôleur de Symfony
- Nombre illimité
- Une méthode ⇔ une page
- Accès facile à HttpFoundation\Request
- Obligation de retourner un objet HttpFoundation\Response

Les contrôleurs de base Symfony

- Réponses
  a. render()
  b. json()
  c. file()
  d. createNotFoundException()

- Routage
  a. redirectToRoute
  b. generateURL()
  c. redirect()

- Sécurité
  a. getUser()
  b. createAccessDeniedException()
  c. isGranted()

- Modèle et données
  a. createForm()
  b. addFlash()

Controller - Symfony 4.4

Exemple d’un contrôleur

```php
namespace App\Controller;


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

class HomeController extends AbstractController
{
    /**
     * @Route("home", name="home_index", methods={"GET"})
     *
     * @param Request $request
     * @return Response
     */
    public function index(Request $request): Response
    {

        return new Response( content: "Hello World");
    }

}
```

TWIG: le moteur de template

Le moteur de template

- Remplace PHP
- Simplifie la lecture et l'écriture
- Génère du HTML
- Twig

- **composer req twig**

Créer une vue Twig (Conventions)

- Sous templates/
- Extension .html.twig
- Un fichier Twig par méthode de contrôleur
- Fichiers Twig nommés comme la méthode de contrôleur associée
- Placé dans un sous-dossier nommé comme le contrôleur
- Pas de majuscules

Exemple d’un contrôleur

```php
namespace App\Controller;


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

class HomeController extends AbstractController
{
    /**
     * @Route("home", name="home_index", methods={"GET"})
     *
     * @param Request $request
     * @return Response
     */
    public function index(Request $request): Response
    {

        return $this->render( view: 'Home/index.html.twig');
    }

}
```

Twig, du HTML, mais pas seulement

- Un fichier HTML normal est un fichier Twig valide
- 3 délimiteurs permettent d'exécuter du code
- Compilés en PHP
- Avantages de Twig :
    a. Minimaliste
    b. Lisible
    c. Système d'héritage
    d. Performant

Documentation - Twig - The flexible, fast, and secure PHP template engine

Les délimiteurs TWIG

- Servent à séparer le code Twig du code HTML
- Il est possible d'utiliser plusieurs délimiteurs sur une même page

- {# ... #}: permet d'écrire des commentaires ⇔ <? /* un commentaire */ ?>
- {{ ... }}: affiche quelque chose ⇔ <? echo "du texte"; ?>
- {% ... %} : exécute du code ⇔ <?php ?>
    a. Créer des variables
    b. Réaliser une boucle
    c. Tester une condition
    d. Exécuter des fonctions

Exemple d’une vue

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <title>Symfony 4.4</title>
    </head>
    <body>

        {# Initialise une variable et lui affecte une valeur #}
        {% set maVariable =  "Hello World" %}

        {# Afficher la valeur de la variable #}
        {{ maVariable }}

        {# Teste la valeur de la variable #}
        {% if maVariable == "Hello World"  %}
            <p>Condition valide !</p>
        {% endif %}

    </body>
</html>
```

Les balises TWIG (Tags)

- Permettent:
  a. l'itération sur des variables, objets et tableaux
  b. l'ajout de métier dans la vue
  c. une syntaxe simple

Les balises TWIG (Tags)

- apply
- block
- extends
- flush
- for
- from
- if
- import
- include
- set
- use
- verbatim
- with
- ...

Documentation - Twig - The flexible, fast, and secure PHP template engine

Les balises TWIG (Filtres)

- Modifient une variable
- Peuvent s’enchaîner
- Ont parfois des options, des arguments
- Utilisés avec un | (pipe) à la suite d'une variable

Les balises TWIG (Filtres)

- abs
- capitalize
- date
- escape
- first
- format
- join
- json_encode
- length
- lower
- number_format
- round
- trim
- ...

Documentation - Twig - The flexible, fast, and secure PHP template engine

Les blocs et l’héritage

```twig
<!DOCTYPE html>
<html lang="fr">
    <head>
        <meta charset="UTF-8">
        <title>{% block title %}Titre par défaut{% endblock %}</title>
    </head>
    <body>
        {% block main %}{% endblock %}
    </body>
</html>
```

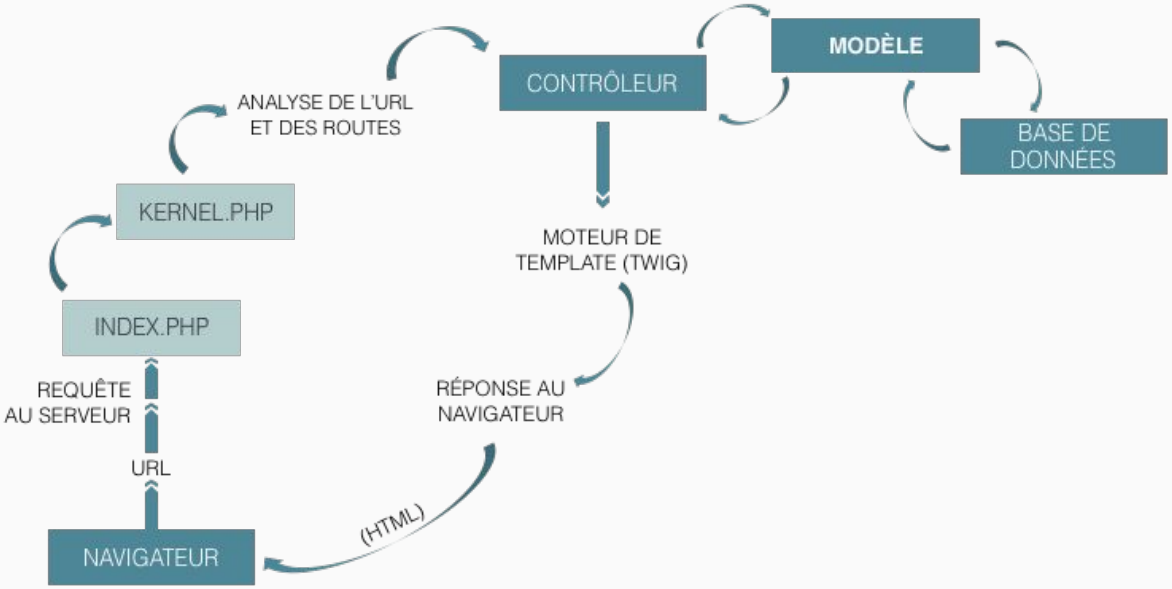
```twig
{% extends "base.html.twig" %}

{% block title %}Titre de ma page index{% endblock %}

{% block main %}
    <p>Contenu de ma page index !</p>
{% endblock %}
```

```twig
{% extends "base.html.twig" %}

{% block title %}Titre de ma page contact{% endblock %}

{% block main %}
    <p>Contenu de ma page contact !</p>
{% endblock %}
```

Gestion des URL internes

- Même problème que pour les assets
- Souplesse du système de routage
- Toujours utiliser:
  a. {% path('home') %} ⇔ URL Relative
  b. {% url('home') %} ⇔ URL Absolue

Parcours d’une requête HTTP sous Symfony



Passer des données du contrôleur vers TWIG

```php
/**
 * @Route("home", name="home_index", methods={"GET"})
 *
 * @param Request $request
 * @return Response
 * @throws \Exception
 */
public function index(Request $request): Response
{

    // Tableau de fruits
    $fruits = ["Orange", "Pomme", "Fraise"];

    return $this->render( view: 'Home/index.html.twig', ["fruits" => $fruits, "date" => new DateTime( time: 'now')]);
}
```

```twig
{% extends "base.html.twig" %}

{% block title %}Titre de ma page contact{% endblock %}

{% block main %}

    <ul>
        {% for fruit in fruits %}
            <li>{{ fruit }}</li>
        {% endfor %}
    </ul>

    <p>La date du jour est: {{ date | date("d/m/Y") }}</p>


{% endblock %}
```

- Orange
- Pomme
- Fraise

La date du jour est: 17/01/2020

Images, Javascript et CSS

- Utilisation du composant WebPack Encore

- **composer req symfony/webpack-encore-bundle**
- **yarn install**
- **yarn add @symfony/webpack-encore --dev**

- https://symfony.com/doc/4.4/frontend/encore/simple-example.html

Installing Encore in Symfony Applications

Les attaques XSS

- Cross-Site Scripting
- Injection de code malveillant dans le code HTML
- Protection en PHP avec <?php echo htmlentities($myVar); ?>

Les attaques XSS sous TWIG

- Protection automatique avec {{ myVar }}

- Filtre raw pour désactiver la protection
- Filtre striptags
- HTML Purifier ou autres (Bundle)

Débogage avec le VarDumper

- Le composant VarDumper et dump()
- Alternative à var_dump() ou print_r()
- Le composant Profiler
- Debug Bar
- Inspection du déroulement de la requête

- **composer req debug**

http://192.168.2.100:8000/home/5

Method: GET **HTTP Status:** 200 **IP:** 192.168.2.1 **Profiled on:** Fri, 17 Jan 2020 12:21:52 +0100 **Token:** ac6d03

Last 10 | Latest | Search

- Request / Response
- Performance
- Exception
- Logs 2
- Events
- Routing
- Cache
- Twig
- Debug
- Configuration

Settings

HomeController :: index

Request | Response | Cookies | Session | Flashes | Server Parameters

GET Parameters

No GET parameters

POST Parameters

No POST parameters

Uploaded Files

No files were uploaded

Request Attributes

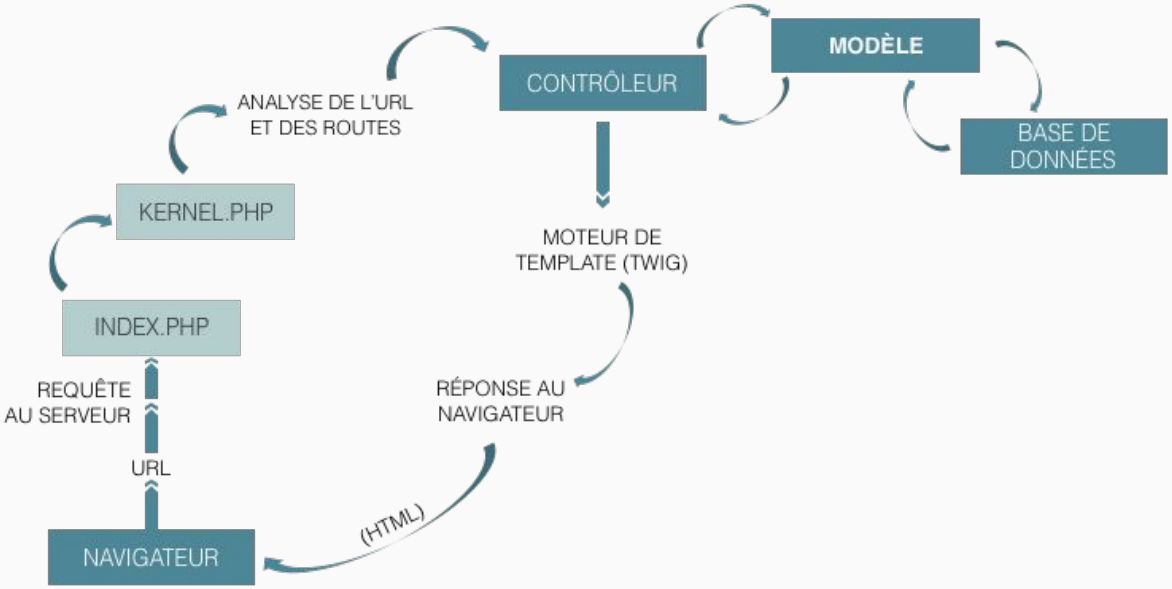
Key	Value
_controller	"App\Controller\HomeController::index"
_route	"home_index"
_route_params	[▶]
id	"5"

Request Headers

Header	Value
accept	"text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9"
accept-encoding	"gzip, deflate"
accept-language	"fr-FR,fr;q=0.9,en-US;q=0.8,en;q=0.7"
cache-control	"max-age=0"
connection	"keep-alive"
content-length	""
content-type	""
cookie	"tarteaucitron=!gajs=wait"
host	"192.168.2.100:8000"

Données et Doctrine

Parcours d’une requête HTTP sous Symfony



Les données

- Problème
  a. Codes répétitifs
  b. Les tableaux sont peu fiables
  c. Différents SGBD
- Solution
  a. Utiliser des objets
  b. Automatiser les tâches répétitives
  c. Abstraire la communication avec le SGBD

Installation de Doctrine

- Dossiers:
  a. src/Entity/
  b. src/Migrations/
  c. src/Repository/
- Configuration
  a. /.env
  b. /config/packages/doctrime.yaml
- **composer req symfony/orm-pack**

Configuration de Doctrine

- Paramètres de la base de données dans .env

```
###> doctrine/doctrine-bundle ###
DATABASE_URL=mysql://root:root@127.0.0.1:3306/MyProject
###< doctrine/doctrine-bundle ###
```

- **php bin/console doctrine:database:create**
- Modifier l'interclassement dans phpMyAdmin

Documentation - Doctrine 2.7

Les entités

- Une classe PHP représentant les données
- Une classe ⇔ une table
- Une propriété de classe ⇔ champ d'une table

- On indique à Doctrine quelles propriétés doivent être sauvegardées et de quelle manière

Exemple d’une entité

```php
/**
 * @ORM\Entity()
 * @ORM\Table(name="User")
 */
class User
{
    /**
     * @ORM\Id
     * @ORM\GeneratedValue(strategy="AUTO")
     * @ORM\Column(name="id", type="integer", options={"unsigned"=true}))
     */
    private $id;

    /**
     * @var string
     *
     * @ORM\Column(name="name", type="string", length=50)
     */
    private $nom;

    /**
     * @var string
     *
     * @ORM\Column(name="first_name", type="string", length=50)
     *
     */
    private $prenom;

    /**
     * @var string
     *
     * @ORM\Column(name="email", type="string", length=150, nullable=true)
     *
     */
    private $email;
}
```

Générer des entités

- Le bundle Maker

- Créer l'entité avec **php bin/console make:entity**
- Ajouter des champs dans la classe
- Générer les getters et setters

- **composer req maker**

Le schéma

- Ensemble des configurations des entités
- Le cœur du modèle

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

L’EntityManager de Doctrine

- EntityManager ⇔ Gestionnaire d’entités
- Pour:
  a. Create
  b. Update
  c. Delete

- Récupération de l’EntityManager en utilisant l’injection de dépendances
- Un EntityManager pour toutes les entités

Exemple de l’injection de l’EntityManager

```php
/**
 * @Route("home/{id}", name="home_index", requirements={"id", "\d+"}, methods={"GET"})
 *
 * @param Request $request
 * @param EntityManagerInterface $entityManager
 * @return Response
 */
public function index(Request $request, EntityManagerInterface $entityManager): Response
{

    return $this->render( view: 'Home/index.html.twig');
}
```

Méthodes principales de l’EntityManager

- persist($object)
- remove($object)
- flush()

```php
/**
 * @Route("home/{id}", name="home_index", requirements={"id", "\d+"}, methods={"GET"})
 *
 * @param Request $request
 * @param EntityManagerInterface $entityManager
 * @return Response
 */
public function index(Request $request, EntityManagerInterface $entityManager): Response
{
    $user = new User();

    // Insertion de l'utilisateur en base
    $entityManager->persist($user);
    $entityManager->flush();

    // Suppression de l'utilisateur en base
    $entityManager->remove($user);
    $entityManager->flush();

    return $this->render( view: 'Home/index.html.twig');
}
```

Les repository de Doctrine

- Repository ⇔ Dépôt
- Pour le R du CRUD (Select)

- Méthode pour le récupérer depuis un contrôleur :
  a. $em->getRepository(Objet::class)

- Un Repository par entité

Les méthodes principales du Repository

- $repository->findAll()
- $repository->find($id)
- $repository->findOneBy(["name" => "lorem"])
- $repository->findBy([], ["price" => "DESC"], 30, 0)
- $repository->count()

Exemple de création d’un Repository

```php
/**
 * @ORM\Entity(repositoryClass="App\Repository\UserRepository")
 * @ORM\Table(name="User")
 */
class User
{
    /**
     * @ORM\Id
     * @ORM\GeneratedValue(strategy="AUTO")
     * @ORM\Column(name="id", type="integer", options={"unsigned"=true}))
     */
    private $id;
}
```

```php
namespace App\Repository;

use Doctrine\ORM\EntityRepository;

/**
 * Class UserRepository
 * @package App\Entity
 */
class UserRepository extends EntityRepository
{

}
```

```php
public function index(Request $request, EntityManagerInterface $entityManager): Response
{
    $user = $entityManager->getRepository( className: 'App:User')->find($request->get( key: 'id',  default: 0));

    return $this->render( view: 'Home/index.html.twig', ["user" => $user]);
}
```

Les requêtes personnalisées

- Les Repository sont modifiables
- On y ajoute les requêtes complexes
- 2 "langages" possibles :
  a. DQL
  b. QueryBuilder
- Une méthode par requête

- Les requêtes se font aux classes PHP et non aux tables !

Le DQL

- Doctrine Query Language
- Très inspiré du SQL

```php
/**
 * @param string $name
 * @return User[]
 */
public function findUserByName($name) {

    $entityManager = $this->getEntityManager();

    $dql = "SELECT u
            FROM App\Entity\User u
            WHERE u.nom LIKE :nom
            ORDER BY u.nom ASC";


    $query = $entityManager->createQuery($dql);
    $query->setParameter( key: 'nom', $name);
    $query->setMaxResults( maxResults: 10);

    return $query->getResult();
}
```

DQL - Doctrine 2.7

Le QueryBuilder

- Classe de Doctrine
- Génère du DQL grâce à une API Orienté Objet

```php
/**
 * @param string $name
 * @return User[]
 */
public function findUserByName($name) {

    $queryBuilder = $this->createQueryBuilder( alias: 'u')
        ->where( predicates: 'u.nom LIKE :nom')->setParameter( key: 'nom', $name)
        ->setMaxResults( maxResults: 10);

    return $queryBuilder->getQuery()->getResult();
}
```

Query Builder - Doctrine 2.7

Les différentes méthodes de résultats

- getResult()
- getArrayResult()
- getScalarResult()
- getOneOrNullResult()
- getSingleResult()
- getSingleScalarResult()

Les Formulaires

Les problèmes des formulaires classiques

- Codes répétitifs
- Validation difficile
- HTML illisible
- Beaucoup d'erreurs possibles

Le composant Form de Symfony

- Automatise les tâches
- Validation très simple
- HTML ultra-limpide

- **composer req form**

Créer une classe de formulaire

- **php bin/console make:form**
- Associer le formulaire à la classe
- Ajouter les champs souhaités

Les types de champs

- Plus d'une trentaine de types de champs
- Les plus courants :
  a. Text
  b. TextArea
  c. Email
  d. Choice
  e. Entity
  f. Date
  g. File

Form - Symfony 4.4

Les options communes et essentielles

- attr
- disabled
- label
- mapped
- placeHolder
- required
- trim

Form - Symfony 4.4

Exemples d’une classe formulaire

```php
class UserType extends AbstractType {

    public function buildForm(FormBuilderInterface $builder, array $options)
    {
        $builder->add( child: 'nom',  type: TextType::class, [
            'required' => true,
            'label' => "Nom",
            'attr' => ['placeHolder' => "Nom", 'maxlength' => 50]
        ]);

        $builder->add( child: 'prenom',  type: TextType::class, [
            'required' => true,
            'label' => "Prénom",
            'attr' => ['placeHolder' => "Prénom", 'maxlength' => 50]
        ]);

        $builder->add( child: 'submit',  type: SubmitType::class, [
            'label' => "Envoyer",
        ]);
    }

    public function configureOptions(OptionsResolver $resolver)
    {
        parent::configureOptions($resolver);

        $resolver->setDefaults(array(
            'data_class' => User::class,
            'trim' => true,
        ));
    }
}
```

```php
class UserType extends AbstractType {

    public function buildForm(FormBuilderInterface $builder, array $options)
    {
        $builder->add( child: 'email',  type: EmailType::class, [
            'required' => false,
            'label' => "Email",
            'attr' => ['placeHolder' => "Email", 'maxlength' => 150]
        ]);

        $builder->add( child: 'submit',  type: SubmitType::class, [
            'label' => "Envoyer",
        ]);
    }

    public function configureOptions(OptionsResolver $resolver)
    {
        parent::configureOptions($resolver);

        $resolver->setDefaults(array(
            'data_class' => User::class,
            'trim' => true,
        ));
    }
}
```

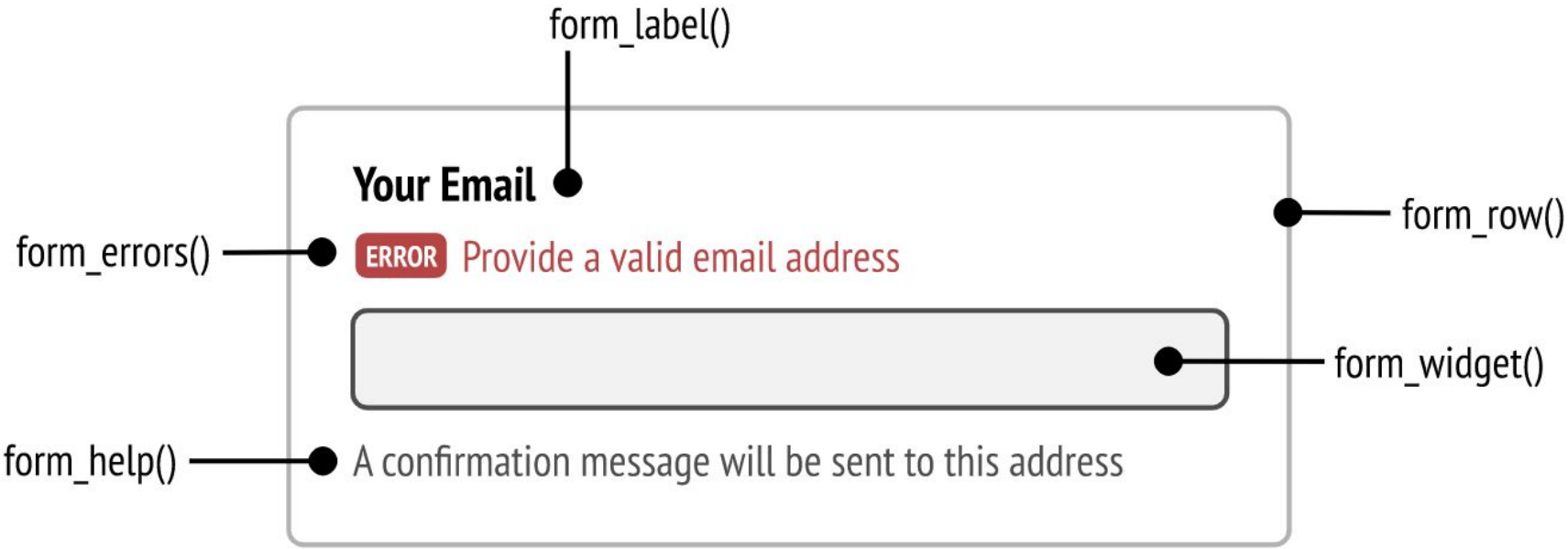
Les étapes pour afficher un formulaire

- Définir la classe de formulaire
- Dans un contrôleur :
  a. Créer une instance de l'entité
  b. Créer une instance du formulaire
  c. Passer le formulaire à Twig
- Déclencher l'affichage dans Twig

Exemple d’un formulaire (Création)

```php
class UserController extends AbstractController
{
    /**
     * @Route("user/create", name="user_create", methods={"GET", "POST"})
     *
     * @param Request $request
     * @return Response
     */
    public function create(Request $request): Response
    {
        // Création de l'utilisateur
        $user = new User();

        // Création du formulaire
        $formUser = $this->createForm( type: UserType::class, $user);

        // Appel de la vue
        return $this->render( view: "User/create.html.twig", [
            'formUser' => $formUser->createView()
        ]);
    }

}
```

Exemple d’un formulaire (Types)



Exemple d’un formulaire (Minimaliste)

```twig
{% extends "base.html.twig" %}

{% block title %}Création d'un utilisateur{% endblock %}

{% block main %}

    {# Affichage du formulaire #}
    {{ form(formUser) }}

{% endblock %}
```

Exemple d’un formulaire (Décomposition 1)

```twig
{% extends "base.html.twig" %}

{% block title %}Création d'un utilisateur{% endblock %}

{% block main %}

    {# Affichage avec décomposition par ligne  #}
    {{ form_start(formUser) }}

        {{ form_row(formUser.nom) }}
        {{ form_row(formUser.prenom) }}
        {{ form_row(formUser.submit) }}

    {{ form_end(formUser) }}

{% endblock %}
```

Exemple d’un formulaire (Décomposition 2)

```twig
{% extends "base.html.twig" %}

{% block title %}Création d'un utilisateur{% endblock %}

{% block main %}

    {# Affichage avec décomposition par ligne  #}
    {{ form_start(formUser) }}

    <div class="group">
        {{ form_label(formUser.nom) }}
        {{ form_widget(formUser.nom) }}
        {{ form_errors(formUser.nom) }}
    </div>

    {{ form_row(formUser.submit) }}

    {{ form_end(formUser) }}

{% endblock %}
```

Exemple d’un formulaire (Thème)

```yaml
twig:
    form_themes: ['bootstrap_4_horizontal_layout.html.twig']
```

```twig
{% extends "base.html.twig" %}

{% block title %}Création d'un utilisateur{% endblock %}

{% block main %}

    {% form_theme formUser 'bootstrap_4_horizontal_layout.html.twig' %}

    <div class="container">

        {# Affichage du formulaire #}
        {{ form(formUser) }}

    </div>

{% endblock %}
```

Traitement du formulaire

- Dans le contrôleur
- Sur la même page
- Objectifs:
  a. Tester si le formulaire est soumis
  b. Récupérer les données
  c. Traiter les données
  d. Afficher un message à l'utilisateur
- Symfony injecte les données dans l'entité

Exemple d’un formulaire (Traitement)

```php
/**
 * @Route("user/create", name="user_create", methods={"GET", "POST"})
 *
 * @param Request $request
 * @return Response
 */
public function create(Request $request): Response
{
    // Création de l'utilisateur
    $user = new User();

    // Création du formulaire
    $formUser = $this->createForm( type: UserType::class, $user);

    // Vérification du formulaire
    $formUser->handleRequest($request);
    if ($formUser->isSubmitted() && $formUser->isValid()) {
        dump($user);
        exit();
    }

    // Appel de la vue
    return $this->render( view: "User/create.html.twig", [
       'formUser' => $formUser->createView()
    ]);
}
```

Messages Flash

- Stockés en session
- Détruits dès qu'ils sont affichés

Flash Messages - Symfony 4.4

Exemple d’un Message Flash

```php
// Vérification du formulaire
$formUser->handleRequest($request);
if ($formUser->isSubmitted() && $formUser->isValid()) {

    // Création du message Flash
    $this->addFlash( type: "success",  message: "Utilisateur créé !");

    // Redirection vers le contrôleur de la page d'accueil
    return $this->redirectToRoute( route: "home_index");
}
```

```twig
{% extends "base.html.twig" %}

{% block title %}Page d'accueil{% endblock %}

{% block main %}

    {# Affichage du message flash  #}
    {% for label, messages in app.flashes %}
        {%  for message in messages %}
            <div class="alert-{{ label }}">
                {{ message }}
            </div>
        {% endfor %}
    {% endfor %}

{% endblock %}
```

Les attaques CSRF

- Cross Site Request Forgery
- Lorsqu'on soumet sans le vouloir un formulaire vers un site sur lequel on est connecté

- Protection : ajouter un champ caché aux formulaires dont nous seuls connaissons la valeur

Les attaques CSRF sous Symfony

- Tester si le formulaire est valide
- Configurer dans les packages

- **composer req security-csrf**

CSRF - Symfony 4.4

Validation des données

- S’assurer que l’utilisateur saisit des données valides
- Fastidieux à développer

- Sous Symfony, on valide l’entité, configuration en annotations

- **composer req validator**

Contraintes de validation

- NotBlank
- Type
- Email
- Length
- Url
- Regex
- Range

- LessThan / GreaterThan
- Date
- Choice
- UniqueEntity
- File
- Image
- ...

Validation - Symfony 4.4

Appliquer les contraintes de validation

- Dans l'entité
- En annotation
- Ne pas oublier le use

```php
namespace App\Entity;

use Doctrine\ORM\Mapping as ORM;
use Symfony\Component\Validator\Constraint as Assert;

/**
 * @ORM\Entity(repositoryClass="App\Repository\UserRepository")
 * @ORM\Table(name="User")
 */
class User
{
```

Exemple de contraintes de validation

```php
/**
 * @var string
 *
 * @Assert\NotBlank(message="Veuillez renseigner un nom !")
 * @Assert\Length(
 *     min="2", max="50",
 *     minMessage = "2 caractères minimum !",
 *     maxMessage = "50 caractères maximum !"
 * )
 *
 * @ORM\Column(name="name", type="string", length=50)
 */
private $nom;

/**
 * @var string
 *
 * @Assert\Email(
 *     message = "L'adresse email '{{ value }}' n'est pas valide !"
 * )
 *
 * @ORM\Column(name="email", type="string", length=150, nullable=true)
 */
private $email;
```

Relations des entités

La Relation OneToOne

- Un objet peut être associé à un seul autre objet
- Relation OneToOne

Exemple de Relations entre objet PHP

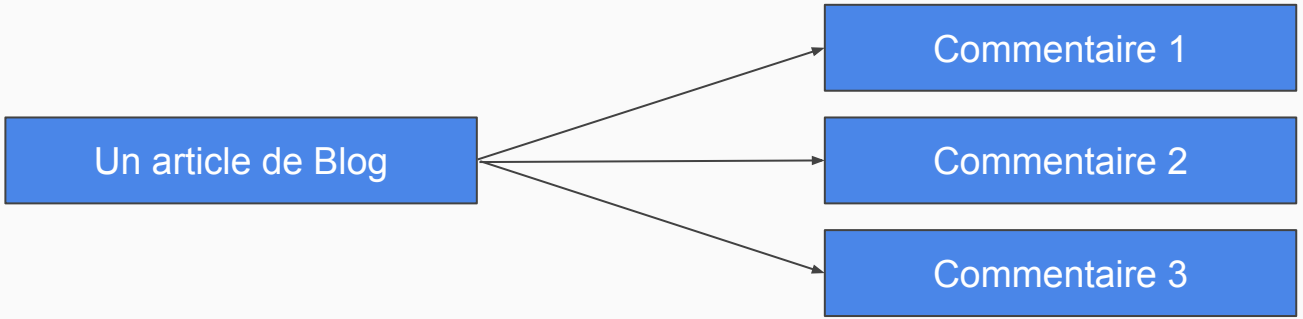
- Relation OneToOne

```php
class User {

    private ?int $id;
    private string $email;
    private string $password;

    // Contient l'objet "Profile"
    private Profile $profile;

}
```

```php
class Profile {

    private ?int $id;
    private string $firstName;
    private string $lastName;
    private ?string $biography;
    private ?string $picture;

    // Contient l'objet "User"
    private User $user;

}
```

La Relation OneToMany / ManyToOne

- Un objet peut être associé à plusieurs autres objets
- Relation OneToMany / ManyToOne



Exemple de Relations entre objet PHP

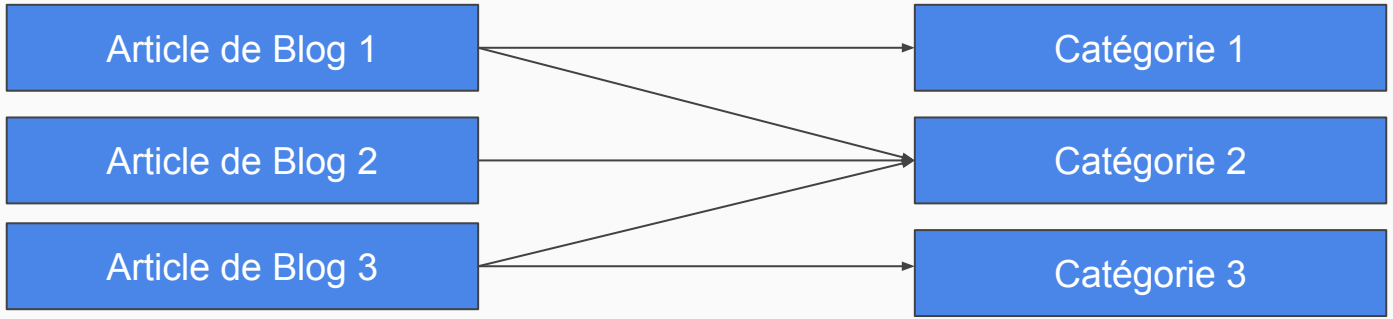
- Relation OneToMany

```php
class Post {

    private ?int $id;
    private string $title;
    private string $description;
    private string $author;

    // Contient un tableau d'objets "Comment"
    private array $comments;

}
```

```php
class Comment {

    private ?int $id;
    private DateTime $date;
    private string $author;
    private string $text;

    // Contient l'objet "Post"
    private Post $post;

}
```

La Relation ManyToMany

- Plusieurs objets peuvent être associés à plusieurs objets
- Relation ManyToMany



Exemple de Relations entre objet PHP

- Relation ManyToMany

```php
class Post {

    private ?int $id;
    private string $title;
    private string $description;
    private string $author;

    // Contient un tableau d'objets "Category"
    private array $categories;
}
```

```php
class Category {

    private ?int $id;
    private string $title;

    // Contient un tableau d'objets "Post"
    private array $posts;
}
```

Les relations avec Doctrine

- Travailler avec les objets
- Jamais avec les id !
- Etapes:
  a. Définir les relations en annotations
  b. Générer les getters et setters manquants
  c. Mettre à jour la base de données: **php bin/console doctrine:schema:update --force**
  d. Tester

Relations - Doctrine 2.7

Exemple de Relations One To One

```php
/** @Entity */
class User {

    private ?int $id;
    private string $email;
    private string $password;

    /** @OneToOne(targetEntity="App\Entity\Profile", mappedBy="user") */
    private Profile $profile;
}
```

```php
/** @Entity */
class Profile {

    private ?int $id;
    private string $firstName;
    private string $lastName;
    private ?string $biography;
    private ?string $picture;

    /**
     * @OneToOne(targetEntity="App\Entity\User", inversedBy="profile")
     * @JoinColumn(name="user_id", referencedColumnName="id")
     */
    private User $user;
}
```

Relations One To One - Doctrine 2.7

Exemple de Relations One To Many

```php
/** @Entity */
class Post {

    private ?int $id;
    private string $title;
    private string $description;
    private string $author;

    /** @OneToMany(targetEntity="App\Entity\Comment", mappedBy="post") */
    private Collection $comments;

    /** Post constructor */
    public function __construct() {
        $this->comments = new ArrayCollection();
    }
}
```

```php
/** @Entity */
class Comment {

    private ?int $id;
    private DateTime $date;
    private string $author;
    private string $text;

    /**
     * @ManyToOne(targetEntity="App\Entity\Post", inversedBy="comments")
     * @JoinColumn(name="post_id", referencedColumnName="id")
     */
    private Post $post;
}
```

Relations One To Many - Doctrine 2.7

Exemple de Relations Many To Many

```php
/** @Entity */
class Post {

    private ?int $id;
    private string $title;
    private string $description;
    private string $author;

    /**
     * @ManyToMany(targetEntity="App\Entity\Category", inversedBy="posts")
     * @JoinTable(name="posts_categories")
     */
    private Collection $categories;

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

```php
/** @Entity */
class Category {

    private ?int $id;
    private string $title;

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

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

Relations Many To Many - Doctrine 2.7

Récupérer une entité associée à une autre

- Doctrine se charge des requêtes (Via le Getter)
- LAZY Loading
- Attention au nombre de requêtes
- Jointure “Manuelle”
- Paginer les résultats

```
/** @OneToOne(targetEntity="App\Entity\Profile", mappedBy="user", fetch="LAZY") */
private Profile $profile;
```