

Développement front avancé

MMI 3 – TP#5 S6

Danielo **JEAN-LOUIS**

Symfony

- Framework PHP gratuit et open source
- Développé en France par SensioLabs
- Première version : décembre 2005
 - Dernière version (01/2025) : Version 7.2

Source(s) :

- <https://symfony.com/> - anglais

Symfony

- Complet et populaire
 - Fonctionne aussi bien pour un blog qu'un projet complexe
- Utilise le moteur de template Twig
- Sert de base à de nombreux projets :
 - Laravel, Ez Publish, PrestaShop...

Source(s) :

- <https://symfony.com/> - anglais

Symfony

- Propose de nombreux modules
 - Communauté très développée
- Nécessite PHP 8.2+
- Utilise la POO
 - Programmation Orientée Objet
- Basé sur le patron MVC

Source(s) :

- <https://symfony.com/> - anglais

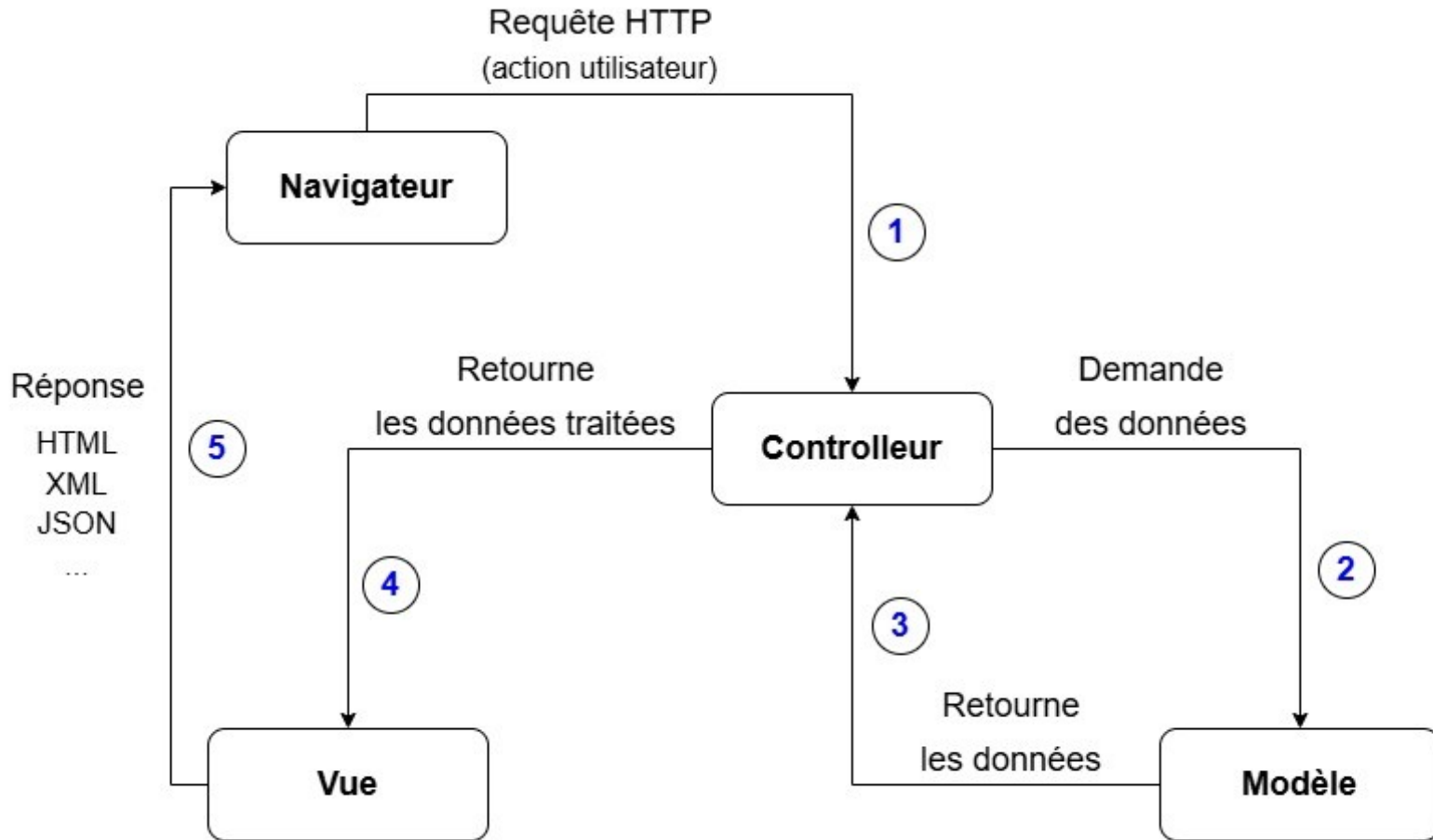
MVC – Modèle-vue-contrôleur

- Patron d'architecture logicielle
 - Un des plus connus (pour le pas dire le plus connu)
- Chaque partie a sa responsabilité
 - Le code est mieux structuré et plus lisible

MVC – Modèle-vue-contrôleur

- Composé de trois parties :
 - Modèle : Données à afficher
 - Vue : Interface graphique
 - Contrôleur : Contient la logique. Fait le lien entre le modèle et la vue
 - Injecte les données dans la vue
 - Modifie / lit le modèle

MVC – Modèle-vue-contrôleur – Schéma



MVC – Avantages

- Découplage du code
 - Chaque partie a sa responsabilité
 - Code plus facile à faire évoluer
- Code plus facilement testable au niveau des tests unitaires

MVC – Cas d'usage : Connexion utilisateur

- Formulaire de connexion (IHM)
 - Vue
- Récupération des données du formulaire (POST) et envoi des données vers le modèle
 - Contrôleur
- Vérifie si la paire utilisateur / mdp est bonne
 - Modèle

Installation

- Utilisation de la ligne de commandes
 - Commandes différentes en fonction de l'OS

Source(s) :

- <https://symfony.com/download> - anglais

Pratiquons ! - Symfony (Partie 1)

Pré-requis :

- Avoir la ressource `ressources/symfony`

A télécharger ici :

https://github.com/DanYellow/cours/raw/refs/heads/main/developement-web-et-dispositif-interactif-s6/travaux-pratiques/numero-5/developpement-web-et-dispositif-interactif-s6_travaux-pratiques_numero-5.ressources.zip

Sécurité

- Symfony possède une couche de sécurité protégeant des failles suivantes :
 - XSS, CSRF, Injection SQL
- Seuls les assets et templates (via leur route) sont accessibles depuis le navigateur grâce au routing

Source(s) :

- https://fr.wikipedia.org/wiki/Cross-site_scripting
- https://fr.wikipedia.org/wiki/Cross-site_request_forgery
- https://fr.wikipedia.org/wiki/Injection_SQL

Routing

- Aiguillage du projet
 - Appelle un contrôleur en fonction de l'URL courante. Les deux sont liés
- Gère l'URL rewriting
 - URL plus élégantes
 - URL mieux référençables

Source(s) :

- <https://symfony.com/doc/current/routing.html>

Routing



```
http://localhost:8000/region/bretagne/ville/brest
```

Le routing de symfony permet d'utiliser cette url au lieu de celle en dessous



```
http://localhost:8000/?region=bretagne&ville=brest
```

Source(s) :

- <https://symfony.com/doc/current/routing.html>

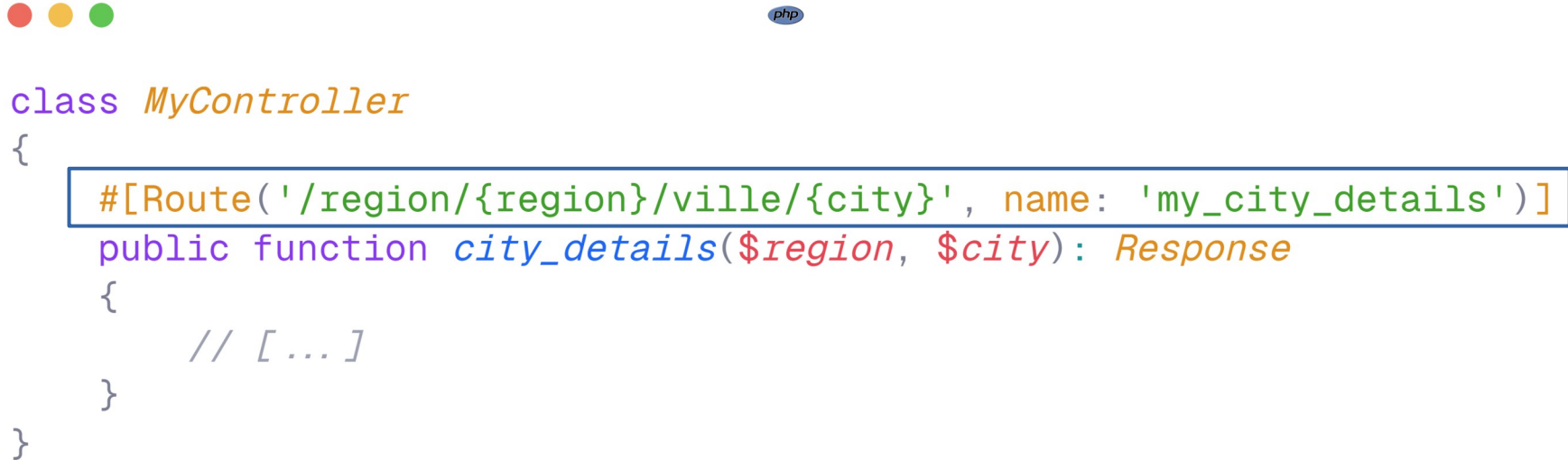
Routing

- Nommage possible des routes
 - Manipulation aisée dans les templates
- Possibilité de :
 - Passer des paramètres et des query string
 - Définir des conditions sur les paramètres

Source(s) :

- <https://symfony.com/doc/current/routing.html>

Routing



```
class MyController
{
    #[Route('/region/{region}/ville/{city}', name: 'my_city_details')]
    public function city_details($region, $city): Response
    {
        // [...]
    }
}
```

Exemple de route avec deux paramètres **obligatoires** nommée “my_city_details”

Source(s) :

- <https://symfony.com/doc/current/routing.html>

Pratiquons ! - Symfony (Partie 2)

Pré-requis :

- Avoir la ressource `ressources/symfony`

A télécharger ici :

https://github.com/DanYellow/cours/raw/refs/heads/main/developement-web-et-dispositif-interactif-s6/travaux-pratiques/numero-5/developpement-web-et-dispositif-interactif-s6_travaux-pratiques_numero-5.ressources.zip

Routing - Notes

- Il est possible d'écrire les routes dans d'autres formats (php, yml, xml), mais préférez les attributs
- Si l'URL courante n'est pas "capturée" par le routeur → page 404

Source(s) :

- <https://symfony.com/doc/current/routing.html>

Routing - Notes

- La première URL qui est “capturée” voit sa fonction appelée
 - Pensez à ordonner vos routes
 - Ou utiliser l’attribut “priority”

Source(s) :

- <https://symfony.com/doc/current/routing.html>

Routing - Notes

- Une route dynamique peut avoir des valeurs par défaut
- Il est possible de préfixer toutes les routes d'un contrôleur pour éviter la répétition

Source(s) :

- <https://symfony.com/doc/current/routing.html>

Contrôleur

- Gestionnaire de la logique d'une app sf
- Classe PHP
 - Suffixée "Controller" par convention
- Doit impérativement retourner une Reponse
 - Reponse → classe Symfony

Source(s) :

- <https://symfony.com/doc/current/controller.html>

Contrôleur

- Constitué de méthodes php qui réagissent (ou non) en fonction de l'URL courante
- Peut hériter de la classe "AbstractController"
 - Ajoute de nouvelles méthodes dont la gestions de templates

Source(s) :

- <https://symfony.com/doc/current/controller.html>

Contrôleur

```

<?php
namespace App\Controller;

// [...]
use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;

class FrontController extends AbstractController
{
    #[Route('/')]
    public function index(): Response
    {
        return $this->render('index.html.twig');
    }
}
```

Source(s) :

- <https://symfony.com/doc/current/controller.html>

Templates

- Responsables de la Vue dans le modèle MVC
- Gère l'affichage de la donnée, affiché dans le navigateur
- Peut recevoir des données en provenance du contrôleur

Source(s) :

- <https://symfony.com/doc/current/templates.html>

Templates

- Placés dans le dossier templates/
 - Pas besoin de mettre “templates/” dans le chemin quand vous le chargez
- Plusieurs formats possibles :
 - php, html et twig
 - Évitez le format php

Source(s) :

- <https://symfony.com/doc/current/templates.html>

Templates / Contrôleur - Connexion

```

<?php
namespace App\Controller;

// [...]
use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;

class FrontController extends AbstractController
{
    #[Route('/lucky')]
    public function index(): Response
    {
        // [...]
        return $this->render('index.html.twig', [
            'formation' => "MMI",
            'iut' => "CY Paris Université - Site de Sarcelles",
            'students_list' => $studentsList,
        ]);
    }
}
```

Templates / Contrôleur - Connexion



```
<ul>
  {% for student in students_list %}
    <li>{{ student.lastname }} - {{ iut }}</li>
  {% endfor %}
</ul>
```

On affiche les données du contrôleur dans le template Twig

Pratiquons ! - Symfony (Partie 3)

Pré-requis :

- Avoir la ressource `ressources/symfony`

A télécharger ici :

https://github.com/DanYellow/cours/raw/refs/heads/main/developement-web-et-dispositif-interactif-s6/travaux-pratiques/numero-5/developpement-web-et-dispositif-interactif-s6_travaux-pratiques_numero-5.ressources.zip

Contrôleur – Bonnes pratiques

- Doivent contenir le moins de logique possible. Passez par des classes externes
- Faites hériter vos contrôleurs de la classe “AbstractController”
 - Symfony injectera plein d'options utiles dans votre contrôleur

Source(s) :

- https://symfony.com/doc/current/best_practices.html#controllers

Twig dans Symfony

- Symfony injecte des variables globales dans tous les templates
 - Ex : `app.request.method`
- Possède des fonctions supplémentaires
 - Ex : `path()` → génère des urls relatives



```
<a href="{{ path('my_city_details', { region: 'bretagne', city: 'brest' }) }}">
    Voir détails sur la ville de Brest
</a>
```

Source(s) :

- <https://symfony.com/doc/current/templates.html#the-app-global-variable>
- <https://symfony.com/doc/current/templates.html#linking-to-pages>

Twig dans Symfony

- Possède des fonctions supplémentaires
 - `path()` → génère des urls relatives



```
<a href="{{ path('my_city_details', { region: 'bretagne', city: 'brest' }) }}">
    Voir détails sur la ville de Brest
</a>
```

- `asset()` → accès au contenu des dossiers public/ et assets/
- `form()` → génère un formulaire

Source(s) :

- <https://symfony.com/doc/current/templates.html#the-app-global-variable>
- <https://symfony.com/doc/current/templates.html#linking-to-pages>

assets/ ou public/ ?

- Dossiers qui gèrent les assets du projet à une différence près :
 - assets/ : à préférer pour les assets qui peuvent changer. Ex : css du projet
 - public/ : pour les assets statiques. Ex : reset.css ou favicon

Source(s) :

- https://symfony.com/doc/current/frontend/asset_mapper.html

assets/ ou public/ ?

- Dossier assets/ se base sur l'AssetMapper
 - Équivalent de Symfony de Vite. Permet d'importer du javascript dans le js ou d'utiliser les modules js
 - Installation des modules js via une commande (voir consignes)

Source(s) :

- https://symfony.com/doc/current/frontend/asset_mapper.html

Templates / Twig – Notes

- On utilisera la snake_case pour nommer templates et dossiers de templates
- Impossible d'utiliser du php dedans mais Twig propose un ensemble de fonctions
 - Vous pouvez créer les vôtres

Source(s) :

- <https://symfony.com/doc/current/templates.html>
- https://fr.wikipedia.org/wiki/Snake_case
- <https://twig.symfony.com/>

Templates / Twig – Notes

- Inutile de mettre “templates” dans le chemin, Symfony cherchera toujours dans ce dossier
- Le même template peut être réutilisé

Source(s) :

- <https://symfony.com/doc/current/templates.html>
- https://fr.wikipedia.org/wiki/Snake_case
- <https://twig.symfony.com/>

Pratiquons ! - Symfony (Partie 4)

Pré-requis :

- Avoir la ressource `ressources/symfony`

A télécharger ici :

https://github.com/DanYellow/cours/raw/refs/heads/main/developement-web-et-dispositif-interactif-s6/travaux-pratiques/numero-5/developpement-web-et-dispositif-interactif-s6_travaux-pratiques_numero-5.ressources.zip

Nous avons vu la gestion du contrôleur et de la vue, maintenant, découvrons la gestion du modèle dans Symfony

Modèle

- Gère les données
- Souvent connecté à une base de données
- Symfony utilise un ORM → Doctrine 2
 - Object Relation Mapping
 - Manipulation d'une base de données sous forme de classes

Source(s) :

- <https://symfony.com/doc/current/doctrine.html>
- <https://www.doctrine-project.org/>

Doctrine

- ORM agnostique
 - Fonctionne avec n'importe quel type de base de données relationnelle (SGBDR)
 - Fonctionne également avec des bases de données NoSQL
- Les tables sont appelées “entités”

Source(s) :

- <https://symfony.com/doc/current/doctrine.html>
- <https://www.doctrine-project.org/>

Doctrine

- Utilisé par défaut par Symfony
- Tables générées par la ligne de commandes
 - Doctrine gère automatiquement les relations entre les entités et les fichiers d'entité
- Possibilité de mettre en place des règles

Source(s) :

- <https://symfony.com/doc/current/doctrine.html>
- <https://www.doctrine-project.org/>

Pratiquons ! - Symfony (Partie 5)

Pré-requis :

- Avoir la ressource `ressources/symfony`

A télécharger ici :

https://github.com/DanYellow/cours/raw/refs/heads/main/developement-web-et-dispositif-interactif-s6/travaux-pratiques/numero-5/developpement-web-et-dispositif-interactif-s6_travaux-pratiques_numero-5.ressources.zip

Doctrine / Contrôleur

- Le contrôleur manipule les modèles l'Entity Manager :
 - Création, mise à jour, suppression...
 - Entity Manager : Objet responsable de la gestion de la base de données

Source(s) :

- <https://symfony.com/doc/current/doctrine.html>
- <https://www.doctrine-project.org/>

Doctrine / Contrôleur – Création entité

```

<?php

namespace App\Controller;

// [ ... ]
use App\Entity\Gallery;
use Doctrine\ORM\EntityManagerInterface;

final class GalleryController extends AbstractController
{
    #[Route('/gallery/{name}', name: 'app_gallery')]
    public function createGallery(EntityManagerInterface $entityManager, $name = "Festival MMI"): Response
    {
        $gallery = new Gallery();
        $gallery->setName($name);
        // [ ... ]
        $entityManager->persist($gallery);
        $entityManager->flush();

        // [ ... ]
    }
}
```

Voir ressource pour le code complet ainsi que la mise à jour et suppression d'entité

DQL

- Utilisation du DQL
 - Doctrine Query Language pour requêter
- Langage propre à Doctrine permettant de manipuler la base de données
 - Facultatif, mais indispensable pour des requêtes complexes

Source(s) :

- <https://www.doctrine-project.org/projects/doctrine-orm/en/current/reference/dql-doctrine-query-language.html>
- <https://symfony.com/doc/current/doctrine.html#querying-for-objects-the-repository>

Pratiquons ! - Symfony (Partie 6)

Pré-requis :

- Avoir la ressource `ressources/symfony`

A télécharger ici :

https://github.com/DanYellow/cours/raw/refs/heads/main/developement-web-et-dispositif-interactif-s6/travaux-pratiques/numero-5/developpement-web-et-dispositif-interactif-s6_travaux-pratiques_numero-5.ressources.zip

Formulaires

- Fonctionnalité clé en main dans Symfony
- Permet de construire, afficher et valider un formulaire
- Symfony propose un ensemble de champs
 - Champs appelés “FieldType”
 - Possibilité de créer ses propres champs

Source(s) :

- <https://symfony.com/doc/current/forms.html>
- <https://symfony.com/doc/current/reference/forms/types.html>

Formulaires

```
<?php

namespace App\Controller;

use App\Entity\Gallery;

use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;
use Symfony\Component\Form\Extension\Core\Type\SubmitType;
use Symfony\Component\Form\Extension\Core\Type\TextType;
// [...]

final class GalleryController extends AbstractController
{
    #[Route('/gallery/new', name: 'create_gallery', methods: ['GET'])]
    public function newGallery(): Response
    {
        // Instancie une galerie
        $task = new Gallery();
        // Valeur par défaut du champ "name"
        $task->setName('Ma super galerie');

        $form = $this->createFormBuilder($task)
            ->add('name', TextType::class)
            ->add('save', SubmitType::class, ['label' => 'Créer galerie'])
            ->getForm();

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

Source(s) :

- <https://symfony.com/doc/current/forms.html>

Formulaires

```
<?php

namespace App\Controller;

use App\Entity\Gallery;

use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;
use Symfony\Component\Form\Extension\Core\Type\SubmitType;
use Symfony\Component\Form\Extension\Core\Type\TextType;
// [...]

final class GalleryController extends AbstractController
{
    #[Route('/gallery/new', name: 'create_gallery', methods: ['GET'])]
    public function newGallery(): Response
    {
        // Instancie une galerie
        $task = new Gallery();
        // Valeur par défaut du champ "name"
        $task->setName('Ma super galerie');

        $form = $this->createFormBuilder($task)
            ->add('name', TextType::class)
            ->add('save', SubmitType::class, ['label' => 'Créer galerie'])
            ->getForm();

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

Nom du champ
dans l'entité

Limite la route qu'à
la méthode GET

Source(s) :

- <https://symfony.com/doc/current/forms.html>

Formulaires



```
{{ form(form) }}
```

Le formulaire complet sera affiché dans le template
(Note : il est possible de “séparer” le formulaire)

Source(s) :

- <https://symfony.com/doc/current/forms.html>

Formulaires

Si le formulaire est correct, on :

- Récupère les données
- Crée une entité
- Crée un flash message
- Redirige l'internaute

Source(s) :

- <https://symfony.com/doc/current/forms.html>

```
<?php

namespace App\Controller;

use App\Entity\Gallery;

use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;
use Symfony\Component\HttpFoundation\Request;
// [...]

final class GalleryController extends AbstractController
{
    #[Route('/gallery/new', name: 'create_gallery')]
    public function newGallery(Request $request): Response
    {
        $gallery = new Gallery();
        $form = $this->createFormBuilder($gallery);
        // [...]
        $form->handleRequest($request);

        if ($form->isSubmitted() && $form->isValid()) {
            $gallery = $form->getData();
            // Persistance de l'entité
            // [...]

            $this->addFlash(
                'notice',
                "Galerie {$gallery->getName()} créée"
            );

            return $this->redirectToRoute('list_galleries');
        }

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

Flash message

- Message stocké dans la session utilisateur
- Affiché qu'une seule fois
- Utile pour afficher une notification
 - Ex : Entité modifiée
- Accessible dans les templates Twig

Source(s) :

- <https://symfony.com/doc/current/session.html#flash-messages>

Pratiquons ! - Symfony (Partie 7)

Pré-requis :

- Avoir la ressource `ressources/symfony`

A télécharger ici :

https://github.com/DanYellow/cours/raw/refs/heads/main/developement-web-et-dispositif-interactif-s6/travaux-pratiques/numero-5/developpement-web-et-dispositif-interactif-s6_travaux-pratiques_numero-5.ressources.zip

Commande magique

```
php bin/console make:crud {NOM_ENTITE}
```

- Génère tout le CRUD pour votre entité :
 - Templates
 - Contrôleur
 - Formulaire (sous forme de FormType)
 - Repository

Source(s) :

- <https://symfony.com/doc/current/reference/constraints.html>

Formulaires - Validation

- Plusieurs possibilités :
 - Gestion au niveau du contrôleur
 - Gestion au niveau de l'entité
 - Gestion au niveau de la classe du formulaire

Évitez de mélanger les possibilités, Symfony affichera toutes les erreurs

Source(s) :

- <https://symfony.com/doc/current/reference/constraints.html>

Formulaires - Validation

- Utilisation de règles de validation définies
 - Vous pouvez créer les vôtres

Source(s) :

- <https://symfony.com/doc/current/reference/constraints.html>

Formulaires – Validation (contrôleur)

```
<?php

namespace App\Controller;

use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;
// [...]

use Symfony\Component\Validator\Constraints\NotBlank;

final class GalleryController extends AbstractController
{
    #[Route('/gallery/{name}', name: 'create_gallery')]
    public function create(EntityManagerInterface $entityManager, Request $request): Response
    {
        // [...]
        $form = $this->createFormBuilder($entity)
            ->add('name', TextType::class, [
                'label' => "Nom",
                'constraints' => [
                    new NotBlank([
                        'message' => "Ce champ doit être rempli"
                    ]),
                ],
            ],
        ]),
        ->add('save', SubmitType::class, ['label' => 'Créer galerie'])
        ->getForm();
        // [...]
    }
}
```

Source(s) :

- <https://symfony.com/doc/current/forms.html>
- <https://symfony.com/doc/current/reference/constraints.html>

Pratiquons ! - Symfony (Partie 8)

Pré-requis :

- Avoir la ressource `ressources/symfony`

A télécharger ici :

https://github.com/DanYellow/cours/raw/refs/heads/main/developement-web-et-dispositif-interactif-s6/travaux-pratiques/numero-5/developpement-web-et-dispositif-interactif-s6_travaux-pratiques_numero-5.ressources.zip

Formulaires - Validation

- Utilisation de règles de validation définies
 - Vous pouvez créer les vôtres

Source(s) :

- <https://symfony.com/doc/current/reference/constraints.html>

Utilisateurs

- Symfony propose une solution clé en main
 - Stockage dans la base de données
 - Utilisation de LDAP
 - Dans la mémoire
- Gestion d'un ACL
 - Access Control List

Source(s) :

- <https://symfony.com/doc/current/reference/constraints.html>

Utilisateurs

- Génération des fichiers nécessaires pour le bon fonctionnement via commandes (voir consignes) :
 - Formulaires (connexion et inscription)
 - Quel attribut devra être unique (e-mail...)
 - Possibilité d'avoir plusieurs attributs

Source(s) :

- <https://symfony.com/doc/current/reference/constraints.html>

Pratiquons ! - Symfony (Partie 9)

Pré-requis :

- Avoir la ressource `ressources/symfony`

A télécharger ici :

https://github.com/DanYellow/cours/raw/refs/heads/main/developement-web-et-dispositif-interactif-s6/travaux-pratiques/numero-5/developpement-web-et-dispositif-interactif-s6_travaux-pratiques_numero-5.ressources.zip

Bundle

- Extension d'un projet Symfony
 - Ajoute de nouvelles fonctionnalités
- Développé par SensioLabs et la communauté
 - Vous pouvez créer le vôtre

Source(s) :

- <https://symfony.com/bundles>
- <https://symfony.com/doc/current/bundles.html>

EasyAdmin et SonataAdmin

- Bundles permettant de générer un back-office facilement
- Gèrent le CRUD, l'interface (personnalisable) et les utilisateurs (avec ACL*)
- Sonata est bien plus complet, proche de Wordpress. Possibilité de configurer les pages. Documentation plus obscure

Source(s) :

- <https://symfony.com/bundles>
- <https://symfony.com/doc/current/bundles.html>

*ACL : Access Control List

Mise en production

- Préférez une CI pour éviter d'uploader les vendors (très lourds)
- Étapes :
 - Passer le projet en mode production
 - Créer un fichier `.env.prod` avec `APP_ENV=prod`
 - Upload du code (sans les vendors)
 - Migrer la base de données
 - Compilation du dossier assets/

Source(s) :

- <https://symfony.com/doc/current/deployment.html>

Symfony, c'est aussi

- La gestion des e-mails via le bundle symfony/mailer
- symfonycasts pour apprendre Symfony
 - Freemium
- Une certification payante
- Sylius : Framework e-commerce

Source(s) :

- <https://sylius.com/>
- <https://symfonycasts.com/>
- <https://symfony.com/doc/current/mailer.html>

Questions ?

