Symfony

Framework PHP open-source qui permet de développer
des applications et des sites web de manière rapide et efficace
en utilisant des composants réutilisables.

L'ÉCOLE ATYPIQUE

Symfony

Introduction

Symfony, c'est quoi ?

- Symfony est un **framework PHP open source** créé en 2005 par Fabien Potencier.
- Il s'agit d'un ensemble de **composants réutilisables** qui facilitent le développement d'applications web.
- Symfony est basé sur le **design pattern MVC** (Modèle-Vue-Contrôleur) et suit les principes de la programmation orientée objet.
- Il est utilisé par de nombreuses entreprises et organisations dans le **monde entier**, telles que Spotify, BlaBlaCar, et le gouvernement français.

L'ÉCOLE ATYPIQUE

Symfony

Introduction

Pourquoi utiliser un framework ?

- Un framework fournit une **structure de base** pour une application web, ce qui permet de gagner du temps en évitant de réinventer la roue.
- Il offre une séparation claire des préoccupations entre les **différentes couches de l'application** (modèle, vue, contrôleur), ce qui facilite la maintenance et l'évolution du code.
- Les frameworks proposent souvent des **fonctionnalités avancées** telles que la gestion de la sécurité, la validation de formulaires, la pagination, etc.
- En utilisant un framework **populaire** et bien **documenté** comme Symfony, il est plus facile de trouver des ressources et de l'aide en ligne.

Symfony

Introduction

Les avantages de Symfony

- Symfony est **très flexible** et peut être utilisé pour développer autant des petits sites vitrines ou des applications complexes et à fort trafic.
- Il est basé sur des composants réutilisables, ce qui permet de créer des **applications sur mesure** en utilisant uniquement les composants dont on a besoin.
- Symfony est très bien **documenté** et dispose d'une **communauté** active et dynamique.
- Il est **compatible** avec de nombreux systèmes de gestion de bases de données (MySQL, PostgreSQL, SQLite, etc.) et peut être déployé sur différents environnements.
- Symfony suit les **meilleures pratiques** de développement web et respecte les normes de développement.

Symfony

Installation

Prérequis

Avant de pouvoir installer Symfony, vous devez vous assurer que votre poste de travail dispose des éléments suivants :

- PHP 7.2.5 ou supérieur
- Composer (gestionnaire de dépendances PHP)

Sur Windows des outils commes Laragon peuvent simplifier l'installation de l'environnement de travail.

L'ÉCOLE ATYPIQUE

Symfony

Installation

Création d'un nouveau projet

Ouvrez votre **invite de commandes** et naviguez jusqu'au répertoire où vous souhaitez créer votre projet Symfony.

Tapez la commande suivante pour créer un nouveau projet :

```
composer create-project symfony/website-skeleton my_project
```

Une fois l'installation terminée, naviguez jusqu'au répertoire de votre projet :

```
cd my_project
```

Tapez la commande suivante pour lancer le serveur web intégré à Symfony :

```
php bin/console server:start
```

Ouvrez votre navigateur web et accédez à l'adresse suivante : http://localhost:8000 et vous devriez voir une page Symfony.

Symfony

Structure

La structure de fichiers

- **bin**/ : contient les exécutables de Symfony (console, etc.)
- **config**/ : contient les fichiers de configuration de l'application
- **public**/ : contient les fichiers accessibles au public (images, CSS, JS, etc.)
- **src**/ : contient les fichiers sources de l'application (contrôleurs, modèles, vues, etc.)
- **var**/ : contient les fichiers temporaires et les logs de l'application
- **composer.json** : le fichier de configuration de Composer (dépendances, autoload, etc.)
- **.env** : le fichier de configuration de l'environnement de l'application
- **.gitignore** : le fichier de configuration de Git (fichiers à ignorer)
- **phpunit.xml.dist** : le fichier de configuration de PHPUnit (tests unitaires)

Symfony

Structure

Le rôle du fichier .env

Le fichier .env est un **fichier de configuration** contenant des paramètres d'environnement de l'application :

- **APP_ENV** : l'environnement de l'application (dev, prod, test, etc.)
- **APP_SECRET** : la clé secrète de l'application (pour la sécurité)
- **DATABASE_URL** : l'URL de la base de données (pour la connexion)
- **MAILER_URL** : l'URL du service de messagerie (pour l'envoi d'emails)
- …

Symfony

Modèle-vue-contrôleur

Création d'un contrôleur

Un contrôleur est un fichier PHP dans le dossier **src/Controller**. Par exemple, nous allons créer un contrôleur HomeController pour gérer la page d'accueil de notre site.

```
class HelloController extends AbstractController {

	/**
	* @Route("/hello", name="hello")
	*/
	public function index() {
		return $this->render('hello/index.html.twig', [
			'controller_name' => 'HelloController',
		]);
	}
}
```

Dans cet exemple, nous avons défini une action index qui renvoie une réponse HTTP contenant le rendu de la vue index.html.twig.

Symfony

Modèle-vue-contrôleur

Création d'une vue Twig

Une vue Twig est un **fichier HTML** qui contient des balises spécifiques pour afficher les données dynamiques. Par exemple :

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

{% block title %}Hello Controller{% endblock %}

{% block body %}
        <h1>Hello, World!</h1>
{% endblock %}
```

Cette vue Twig **étend la vue** base.html.twig et définit deux blocs : title et body. Le bloc title contient le titre de la page et le bloc body contient le contenu de la page.

Le principe d'**héritage en Twig** permet de créer un gabarit (template) parent qui définit une structure commune à plusieurs pages, puis de créer des gabarits enfants qui héritent de cette structure et la personnalisent en remplaçant des blocs spécifiques.

Symfony

Doctrine

Doctrine : ORM (Object-Relational Mapping)

Il permet de **faciliter les interactions** avec la base de données en utilisant des objets PHP plutôt que d'écrire du SQL brut.

Doctrine est un **composant clé** de Symfony et est largement utilisé dans le développement web et se compose de deux parties principales :

- **DBAL** (Database Abstraction Layer) : **Abstraction de base** de données qui permet d'interagir avec différents systèmes.

- **ORM** (Object-Relational Mapping) : **Mappe les objets PHP** aux tables de la base de données. Cela permet de manipuler les données de la base de données comme des objets PHP.

Avantages de Doctrine

- **Productivité** : Doctrine réduit la quantité de code à écrire.
- **Maintenance** : Doctrine facilite la maintenance du code.

Symfony

Doctrine

Création d'une entité Doctrine

Une entité est un objet PHP qui représente **une ligne dans une table** de la base de données. Un exemple d'entité Doctrine :

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

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

        // getters et setters…
}
```

Symfony

Doctrine

Manipulation de base de données (CRUD)

Doctrine fournit des méthodes pour effectuer les opérations CRUD (Create, Read, Update, Delete) sur la base de données :

Create : Pour créer une nouvelle entrée dans la base de données, nous créons une nouvelle instance de l'entité, nous définissons ses propriétés, et nous utilisons l'EntityManager pour persister et flusher l'entité.

Read : Pour lire une entrée de la base de données, nous utilisons le repository de l'entité pour trouver l'entité par son identifiant ou par d'autres critères.

Update : Pour mettre à jour une entrée de la base de données, nous récupérons l'entité, nous modifions ses propriétés, et nous utilisons l'EntityManager pour flusher l'entité.

Delete : Pour supprimer une entrée de la base de données, nous récupérons l'entité, et nous utilisons l'EntityManager pour la supprimer et flusher.

Symfony

Doctrine

Manipulation de base de données (CRUD)

Voici des exemples de manipulation de base avec Doctrine :

Création d'une nouvelle entrée :

```
$product = new Product();
$product->setName('New product');
$entityManager->persist($product);
$entityManager->flush();
```

Lecture d'une entrée :

```
$product = $entityManager->getRepository(Product::class)->find(1);
```

Mise à jour d'une entrée :

```
$product->setName('Updated product');
$entityManager->flush();
```

Suppression d'une entrée :

```
$entityManager->remove($product);
$entityManager->flush();
```

Symfony

Formulaires

Les formulaires en Symfony

Dans Symfony, un formulaire est un **objet PHP** qui représente un **formulaire HTML**.

Voici un exemple de création de formulaire via Symfony :

```
$builder = Forms::createBuilder();
$form = $builder
        ->add('name', FormType::TEXT)
        ->add('email', FormType::EMAIL)
        ->add('message', FormType::TEXTAREA)
        ->add('send', FormType::SUBMIT)
        ->getForm();
```

Dans cet exemple, nous créons un formulaire avec trois champs : un champ texte pour le nom, un champ email pour l'adresse email et un champ texte multi-ligne pour le message.

Nous ajoutons également un bouton d'envoi pour soumettre le formulaire.

Symfony

Formulaires

Traitement des données du formulaire

Une fois que le formulaire est créé, nous devons **traiter les données** envoyées par l'utilisateur. Voici un exemple :

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

$request = Request::createFromGlobals();

$form->handleRequest($request);

if ($form->isSubmitted() && $form->isValid()) {

        $data = $form->getData();

        // Ici, nous pouvons traiter les données du formulaire

        // Par exemple, enregistrer les données dans une base de
        données

}
```

L'ÉCOLE ATYPIQUE

Symfony

La console

Découverte de la console Symfony

La console Symfony est un outil indispensable pour **faciliter le développement** d'applications web. **En ligne de commande** elle permet, d'automatiser des tâches répétitives et d'interagir avec les composants Symfony.

Exemples de commande Symfony :

- php bin/console list // Liste de toutes les commandes
- php bin/console make:controller NomDuController // Génère un contrôleur
- php bin/console make:entity // Créer une entité
- ..