

# FORMATION PHP:

## INTRODUCTION À SYMFONY

COURS 4



1. Introduction et QCM
2. Révision cours précédent
3. Introduction à Symfony
4. Qu'est-ce que Symfony ?
5. Installation de PHP 8, Composer, Symfony CLI, et VS Code.
6. Types de projets Symfony
7. Composants principaux de Symfony
8. Architecture d'un projet Symfony
9. Création d'une page "Hello Symfony"
10. Routage
11. Les contrôleurs dans Symfony
12. Moteur de template Twig
13. Objet Request et composant HttpFoundation
14. Formulaires Symfony
15. Base de données et Doctrine
16. Services Symfony
17. Sessions et cookies
18. Stockage et gestion

19. Protection CSRF
20. Prévention des attaques CSRF avec Symfony
21. Composant Security de Symfony
22. Génération de CRUD dans Symfony
23. Exercice
24. QCM

# RÉVISION

## Que signifie "ACID" dans le contexte des transactions SQL ?

- Atomicité, Cohérence, Isolation, Durabilité

## Le Factory Pattern ?

- Centraliser la création des objets.
- Réduire le couplage entre le client et les classes concrètes.
- Gérer dynamiquement plusieurs types d'objets.

### Exemple : Système de Paiement

- Nous voulons gérer plusieurs moyens de paiement (Stripe, PayPal, BankTransfer...).
- Une Factory centralise l'instanciation en fonction du type choisi.

```
// Interface commune
interface Payment {
    public function handlePayment(): string;
}

// Classes concrètes
class StripePayment implements Payment {
    public function handlePayment(): string {
        return "Processing payment with Stripe";
    }
}

class PayPalPayment implements Payment {
    public function handlePayment(): string {
        return "Processing payment with PayPal";
    }
}

// Factory
class PaymentFactory {
    public static function create(string $type):
    Payment {
        return match ($type) {
            'stripe' => new StripePayment(),
            'paypal' => new PayPalPayment(),
            default => throw new
            InvalidArgumentException("Invalid payment type")
        };
    }
}

// Utilisation
$payment = PaymentFactory::create('stripe');
echo $payment->handlePayment();
```

# INTRODUCTION À SYMFONY

## Qu'est-ce que Symfony ?

- Symfony est un Framework PHP open-source conçu pour développer des applications web de manière structurée et maintenable.
- Il est basé sur l'architecture MVC (Modèle-Vue-Contrôleur).
- Utilisé par de nombreuses entreprises et projets pour sa flexibilité et sa robustesse.

## Histoire de Symfony

- Création : Symfony a été créé en 2005 par la société française SensioLabs, sous l'impulsion de Fabien Potencier.
- Objectif initial : Fournir un Framework PHP structuré, extensible et adapté aux projets d'envergure.
- Dernière version : Symfony 7.1.8 (novembre 2023, <https://symfony.com/releases>), offrant des améliorations sur la performance, le développement moderne, et la compatibilité avec PHP 8+.

# INTRODUCTION À SYMFONY

Majoritairement utilisé :

- En Europe (France, Allemagne) et Amérique latine.
- Moins populaire aux États-Unis que Laravel, mais reconnu pour les grands projets d'entreprise.

Type de projets :

- Applications web complexes (portails d'entreprise, plateformes e-commerce).
- APIs robustes (ex. : microservices).

Pourquoi Symfony est si connu ?

- Robustesse : Utilisé par des géants comme Spotify, BlaBlaCar, Drupal.
- Réutilisation des composants : Les composants Symfony (HTTPFoundation, EventDispatcher) sont utilisés dans des CMS (WordPress, Laravel).
- Support à long terme (LTS) : Versions stables adaptées aux entreprises.

Symfony :

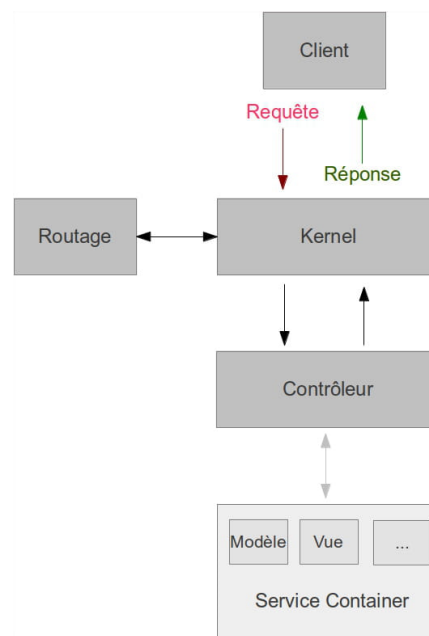
- Préféré pour les projets complexes en Europe, représente environ 10% des projets PHP dans le monde.
- <https://symfony.com/stats/downloads> (voir statistique de téléchargement)

Laravel : Plus populaire globalement (~30%? des projets PHP).

# INTRODUCTION À SYMFONY

## Pourquoi utiliser Symfony ?

- Modularité : Symfony est composé de composants indépendants que vous pouvez utiliser selon vos besoins.
- Communauté active : Un support important et une documentation riche.
- Compatibilité avec les standards : Respecte les bonnes pratiques de développement et les standards PSR (PHP Standards Recommendations).
- Flexibilité et extensibilité : S'adapte aux projets de toutes tailles, des petites applications aux grands systèmes complexes.



# MISE EN PLACE DE L'ENVIRONNEMENT DE DÉVELOPPEMENT SYMFONY

## PHP 8 :

- Nous avons PHP 8 dans notre XAMPP
- Si non → Téléchargez et installez depuis [php.net](https://php.net). Vérifiez avec `php -v`.

## Composer :

- Téléchargez et installez [Composer](https://getcomposer.org) depuis [getcomposer.org](https://getcomposer.org). Pendant l'installation, spécifiez le chemin vers php.exe dans XAMPP (généralement `C:\xampp\php\php.exe`).
- Vérifiez (ouvrez un terminal CMD ou PowerShell) avec `composer -v`.

```
C:\Users\behna>composer --version
Composer version 2.8.3 2024-11-17 13:13:04
PHP version 8.2.12 (C:\xampp\php\php.exe)
Run the "diagnose" command to get more detailed diagnostics output.

C:\Users\behna>php -v
PHP 8.2.12 (cli) (built: Oct 24 2023 21:15:15) (ZTS Visual C++ 2019 x64)
Copyright (c) The PHP Group
Zend Engine v4.2.12, Copyright (c) Zend Technologies
```

# MISE EN PLACE DE L'ENVIRONNEMENT DE DÉVELOPPEMENT SYMFONY

## Symfony CLI :

- Installez depuis <https://symfony.com/download>
- Installez Scoop
- Ouvrir PowerShell (se positionner sur la racine C:\>)
  - Passer les deux commandes suivantes
    - `Set-ExecutionPolicy -ExecutionPolicy RemoteSigned -Scope CurrentUser`
    - `Invoke-RestMethod -Uri https://get.scoop.sh | Invoke-Expression`

```
PS C:\> Set-ExecutionPolicy -ExecutionPolicy RemoteSigned -Scope CurrentUser
PS C:\> Invoke-RestMethod -Uri https://get.scoop.sh | Invoke-Expression
Initializing...
Downloading...
Creating shim...
Adding ~\scoop\shims to your path.
Scoop was installed successfully!
Type 'scoop help' for instructions.
```

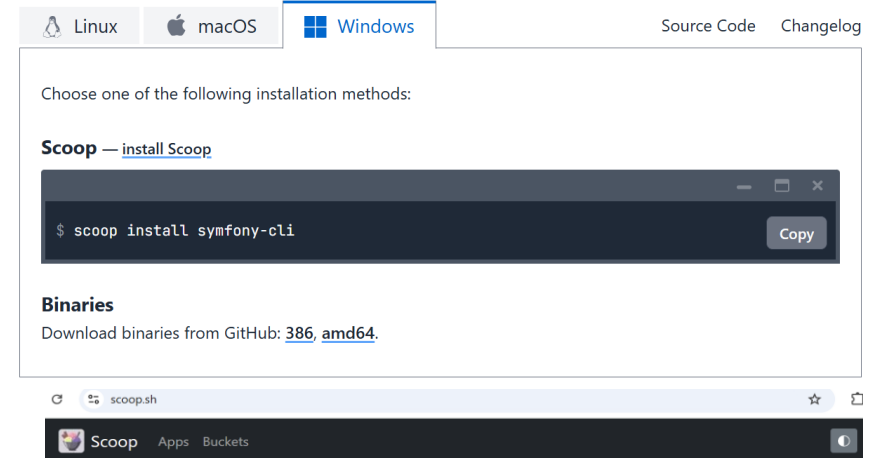
- Installer CLI avec la commande suivante : `scoop install symfony-cli`
- Vérifiez avec `symfony -v`.

```
PS C:\> scoop install symfony-cli
Installing 'symfony-cli' (5.10.4) [64bit] from 'main' bucket
symfony-cli_windows_amd64.zip (5,7 MB) [=====] 100%
Checking hash of symfony-cli_windows_amd64.zip ... ok.
Extracting symfony-cli_windows_amd64.zip ... done.
Linking ~\scoop\apps\symfony-cli\current => ~\scoop\apps\symfony-cli\5.10.4
Creating shim for 'symfony'.
'symfony-cli' (5.10.4) was installed successfully!
PS C:\> symfony -v
Symfony CLI version 5.10.4 (c) 2021-2024 Fabien Potencier (2024-10-11T14:05:11Z - stable)
Symfony CLI helps developers manage projects, from local code to remote infrastructure
```

## Download Symfony

### Step 1. Install Symfony CLI

The Symfony CLI is a developer tool to help you build, run, and manage your Symfony applications directly from your terminal. It's Open-Source, works on macOS, Windows, and Linux, and you only have to install it once in your system. Learn more about Symfony CLI.



Scoop  
A command-line installer for Windows

Search an app

### Quickstart

Open a [PowerShell terminal](#) (version 5.1 or later) and from the PS C:\> prompt, run:

```
Set-ExecutionPolicy -ExecutionPolicy RemoteSigned -Scope CurrentUser
Invoke-RestMethod -Uri https://get.scoop.sh | Invoke-Expression
```

For advanced installation options, check out the [Installer's Readme](#).

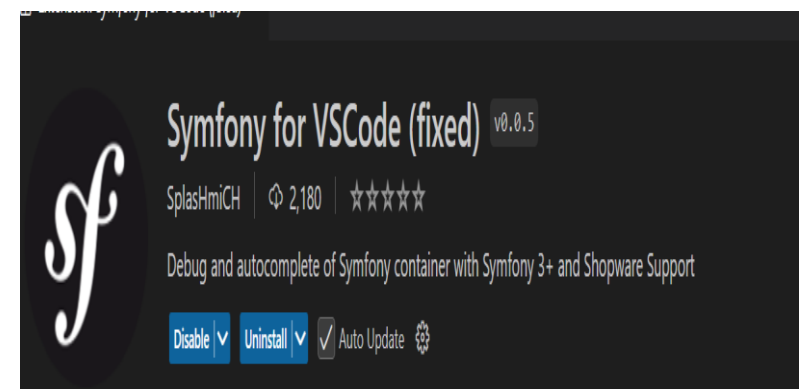
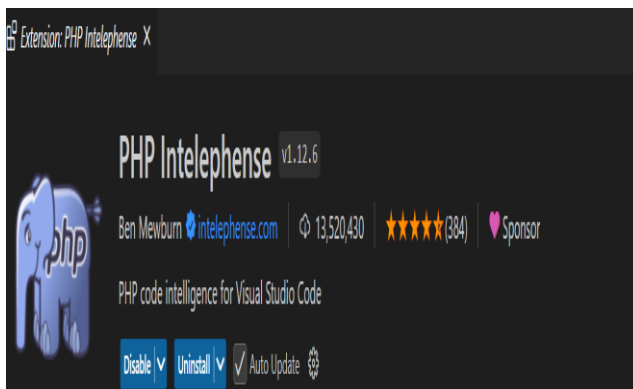


# MISE EN PLACE DE L'ENVIRONNEMENT DE DÉVELOPPEMENT SYMFONY

## Visual Studio Code (VS Code) :

- Téléchargez sur [code.visualstudio.com](https://code.visualstudio.com).
- Extensions VS Code :
  - PHP Intelephense: Analyse et auto-complétion avancée pour PHP.
  - Symfony Support: Intégration et navigation spécifiques pour Symfony,
  - Twig Language: Coloration syntaxique et auto-complétion pour Twig.

Ces extensions optimisent le développement PHP/Symfony avec un IDE léger comme VS Code.



# ÉTAPES POUR LANCER UNE PAGE SYMFONY

## Créer un projet Symfony :

- Avant de démarrer le serveur, créez un projet dans le dossier htdocs (ou un autre répertoire si configuré).

- `cd C:\xampp\htdocs`
- `composer require symfony/maker-bundle --dev`
- `symfony new my_project --webapp`
- `cd my_project`
- `symfony server:start`
- Accéder à votre page Symfony: <http://127.0.0.1:8000>

```
PS C:\xampp\htdocs> cd C:\xampp\htdocs
PS C:\xampp\htdocs> symfony new learnProject --webapp
* Creating a new Symfony project with Composer
* Setting up the project under Git version control
  (running git init C:\xampp\htdocs\learnProject)

Author identity unknown

*** Please tell me who you are.

Run

  git config --global user.email "you@example.com"
  git config --global user.name "Your Name"

to set your account's default identity.
Omit --global to set the identity only in this repository.

fatal: unable to auto-detect email address (got 'behna@BEHNASS.(none)')
exit status 128
PS C:\xampp\htdocs> cd .\learnProject\
```

```
PS C:\xampp\htdocs> cd .\learnProject\
PS C:\xampp\htdocs\learnProject> symfony server:start







[WARNING] run "symfony.exe server:ca:install" first if you want to run the web server with TLS support, or use "--p1
2" or "--no-tls" to avoid this warning


Following Web Server log file (C:\Users\behna\.symfony5\log\118095e5da8e7256cc74212ac1ba387abbbbed407.log)
Following PHP-CGI log file (C:\Users\behna\.symfony5\log\118095e5da8e7256cc74212ac1ba387abbbbed407\79ca75f9e90b4126a5955
a33ea6a41ec5e854698.log)

[WARNING] The local web server is optimized for local development and MUST never be used in a production setup.


[OK] Web server listening
The Web server is using PHP CGI 8.2.12
http://127.0.0.1:8000


[Web Server ] Nov 19 15:33:48 |DEBUG| PHP Reloading PHP versions
[Web Server ] Nov 19 15:33:48 |DEBUG| PHP Using PHP version 8.2.12 (from default version in $PATH)
[Web Server ] Nov 19 15:33:48 |INFO| PHP listening path="C:\xampp\php\php-cgi.exe" php="8.2.12" port=61632
```


 127.0.0.1:8000





# Welcome to Symfony 7

 You are using Symfony **7.1.8** version


 Your application is ready at:  
C:\xampp\htdocs\learnProject\

 You are seeing this page because the homepage URL is not configured and debug mode is enabled.


**NEXT STEP**  [Create your first page](#) to replace this placeholder page.

 **Learn**

- [Read Symfony Docs](#)
- [Watch Symfony Screencast](#)
- [Read Symfony Book](#)

 **Community & Support**

- [Symfony Support](#)
- [Join the Symfony Community](#)
- [Contribute to Symfony](#)

 **Stay Updated**

- [Symfony Blog](#)
- [Follow Symfony](#)
- [Attend Symfony Even](#)

# LES TYPES DE PROJETS SYMFONY

## Projet Web App (--webapp) :

- Inclut une configuration préconfigurée pour des applications web complètes.
- Contient des composants comme Twig, Doctrine, Webpack Encore, etc.
- Idéal pour la majorité des projets web.

## Projet Skeleton (par défaut, sans option) :

- Structure minimale avec uniquement les composants de base.
- Léger et adapté aux projets où vous ajoutez manuellement les dépendances nécessaires.
- `symfony new my_project`

## Projet Microservice (--micro) :

- Pré-configuré pour développer des microservices.
- Inclut des outils pour les API et des composants spécifiques (ex. API Platform).
- `symfony new my_microservice --micro`

## Projet CLI ou Service Backend (structure personnalisée) :

- Vous pouvez aussi démarrer avec le skeleton et n'ajouter que les composants CLI ou nécessaires à votre service backend.

Astuce : Utilisez `--version=<version>` si vous avez besoin d'une version spécifique de Symfony.

Pour notre cours, on utilise principalement le type Web App car il couvre les besoins en développement web complet. 😊

# LES COMPOSANTS PRINCIPAUX DE SYMFONY

## Kernel

- Cœur de l'application Symfony.
- Charge les Bundles.
- Configure les services et les routes.
- Gère le cycle requête/réponse HTTP.

## Bundles

- Extensions ou modules réutilisables.
- Exemples :
  - FrameworkBundle (noyau de Symfony).
  - TwigBundle (moteur de templates).
  - SecurityBundle (gestion de sécurité).

## Conteneur de services

- Gère les dépendances (objets/services).
- Principe :
  - Services définis dans « services.yaml ».
  - Injectés automatiquement dans les classes.

## Console

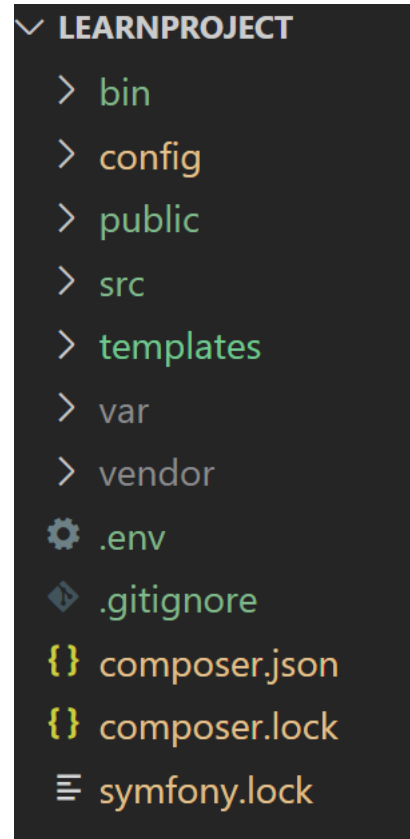
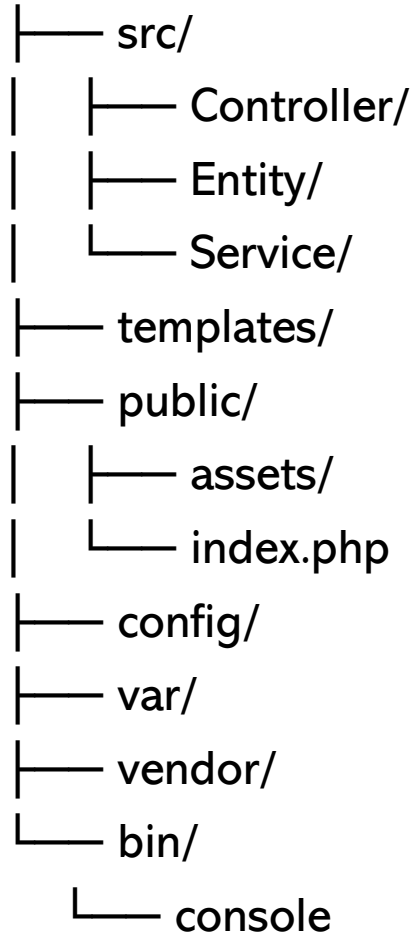
- Interface CLI pour gérer Symfony.
- Commandes utiles :
  - « php bin/console make:controller » : Crée un contrôleur.php

## Configurations

- Gérer le comportement de l'application.
- Fichiers clés :
  - config/packages/\*.yaml : Config des bundles.
  - config/routes.yaml : Définition des routes.

# ARCHITECTURE D'UN PROJET SYMFONY

my\_project/



Présentation des dossiers principaux:

- **src/** : Contient le code principal de votre application (contrôleurs, entités, services, etc.).
- **templates/** : Regroupe les fichiers Twig pour la vue (templates HTML).
- **public/** : Point d'entrée de l'application, héberge les fichiers accessibles au navigateur (CSS, JS, images, index.php).
- **config/** : Stocke les configurations de l'application (routes, services, bundles).
- **var/** : Contient des fichiers temporaires, logs, et le cache.
- **vendor/** : Géré par Composer, contient toutes les dépendances installées.
- **bin/** : Contient les scripts exécutables (console Symfony).

# CRÉER UNE PAGE "HELLO SYMFONY"

Créer un contrôleur :

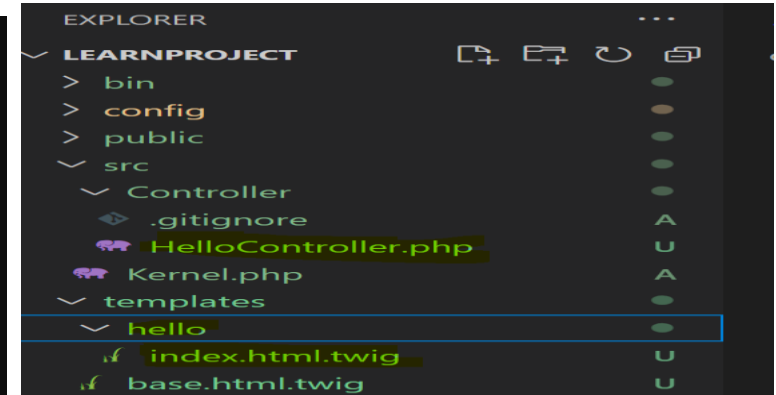
- Exécutez la commande : `php bin/console make:controller HelloController`

```
PS C:\xampp\htdocs\learnProject> php bin/console make:controller HelloController
created: src/Controller/HelloController.php
created: templates/hello/index.html.twig
```

Success!

Next: Open your new controller class and add some pages!

```
PS C:\xampp\htdocs\learnProject>
```



- Afficher la page : `http://127.0.0.1:8000/hello`

127.0.0.1:8000/hello

## Hello HelloController!

This friendly message is coming from:

- Your controller at `C:/xampp/htdocs/learnProject/src/Controller/HelloController.php`
- Your template at `C:/xampp/htdocs/learnProject/templates/hello/index.html.twig`

```
<?php
namespace App\Controller;

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

class HelloController extends AbstractController
{
    #[Route('/hello', name: 'app_hello')]
    public function index(): Response
    {
        return $this->render('hello/index.html.twig', [
            'controller_name' =>
                'HelloController',
        ]);
    }
}
```

# ROUTAGE

Symfony utilise un **système de routage** pour associer une URL à une action d'un contrôleur spécifique.

Le routage permet de structurer l'application et d'orienter les utilisateurs vers les bonnes pages.

- Les routes sont définies avec des annotations dans les contrôleurs ou dans des fichiers YAML ou XML.
- Par défaut, Symfony utilise les annotations de route pour associer une URL à une méthode du contrôleur.



```
#[Route('/hello/{name}', name: 'hello_page')]
public function hello(string $name = 'Symfony'):
    Response
    {
        return $this->render('hello.html.twig',
            ['name' => $name]);
    }
```

- Le paramètre {name} est dynamique et peut être passé dans l'URL.
- Cette route correspond à `http://127.0.0.1:8000/hello/{name}` où {name} est un paramètre optionnel.



# ROUTAGE

## Routage dans un fichier **YAML**:

- Les routes peuvent aussi être définies dans un fichier YAML, situé dans le dossier config/routes.yaml.
- Voici un exemple de définition de route dans YAML :



```
hello_page:  
  path: /hello/{name}  
  controller: App\Controller\HelloController::hello
```

- Le path : Définir l'URL à laquelle cette route répond.
- controller : Lien vers la méthode du contrôleur qui gère cette route.

# ROUTAGE

## Routage dans un fichier YAML:

- Les routes peuvent aussi être définies dans un fichier YAML, situé dans le dossier `config/routes.yaml`.
- Voici un exemple de définition de route dans YAML :

```
hello_page:
  path: /hello/{name}
  controller: App\Controller\HelloController::hello
```

```
namespace App\Controller;

use
Symfony\Bundle\FrameworkBundle\Controller\AbstractC
ontroller;
use Symfony\Component\HttpFoundation\Response;

class HelloController extends AbstractController
{
    public function hello(string $name): Response
    {
        return $this->render('hello.html.twig',
        ['name' => $name]);
    }
}
```

- Le path : Définir l'URL à laquelle cette route répond.
- controller : Lien vers la méthode du contrôleur qui gère cette route.

# ROUTAGE

## Redirection avec `redirectToRoute` :

- Pour rediriger l'utilisateur vers une autre route, vous pouvez utiliser la méthode `redirectToRoute()` dans votre contrôleur :
- Voici un exemple :

```
public function redirectToHello(): Response
{
    return $this->redirectToRoute('hello_page', ['name' => 'Symfony']);
}
```

- `redirectToRoute('hello_page')` redirige vers la route nommée `hello_page`.
- Le paramètre `['name' => 'Symfony']` remplace le paramètre dynamique `{name}` dans l'URL.

Le routage est fondamental pour naviguer dans une application. Il permet de définir comment les utilisateurs interagissent avec votre site.

Le fichier YAML offre une alternative aux annotations pour gérer les routes de manière plus structurée.

# ROUTAGE

## Pourquoi utiliser YAML pour les routes plutôt que des annotations ?

- Centralisation : Avec YAML, toutes les routes sont définies dans un fichier central (config/routes.yaml), ce qui facilite la gestion et l'organisation des routes pour des projets plus complexes.
- Séparation de la logique et de la configuration : En utilisant YAML, vous séparez la configuration des routes du code métier dans le contrôleur, ce qui peut rendre le projet plus clair.
- Préférences d'équipe : Certains développeurs préfèrent YAML pour une meilleure lisibilité ou pour respecter des conventions de l'équipe (par exemple, dans de grands projets).

## Annotations : La syntaxe

- Les annotations de route commencent effectivement par `#[` (depuis Symfony 6). Cela indique une annotation PHP et sert à associer la méthode du contrôleur à une route.
  - `#[Route('/hello/{name}', name: 'hello_page')]`

## Limiter les méthodes HTTP

- Vous pouvez restreindre une route à des méthodes HTTP spécifiques (GET, POST, etc.) pour assurer le bon comportement de votre API ou application web.

```
#[Route('/contact', name: 'contact', methods: ['POST'])]  
public function contact(Request $request): Response { ... }
```

# CONTRÔLEUR

Les contrôleurs sont responsables du traitement des requêtes et du retour des réponses dans Symfony. En utilisant la classe `AbstractController`, vous bénéficiez de nombreuses méthodes utiles pour simplifier le développement.

- `hello()` : Méthode qui reçoit un paramètre dynamique `name` et retourne une vue Twig avec ce paramètre.
- `$this->render()` : Permet de générer un fichier Twig (vue HTML) et de l'envoyer en réponse.

```
namespace App\Controller;

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

class HelloController extends AbstractController
{
    #[Route('/hello/{name}', name: 'hello_page')]
    public function hello(string $name = 'Symfony'): Response
    {
        return $this->render('hello.html.twig', [
            'name' => $name,
        ]);
    }
}
```

# VALIDATION ET PARAMÈTRES SUPPLÉMENTAIRES DANS LES ROUTES

## Validation avec des expressions régulières

- Vous pouvez restreindre les valeurs des paramètres dans la route en utilisant l'option requirements.



```
app_show_user:  
  path: /user/{id}  
  controller: App\Controller\UserController::show  
  requirements:  
    id: \d+ # Le paramètre 'id' doit être un entier
```



```
#[Route('/user/{id}', name: 'user_show', requirements: ['id' => '\d+'])]  
public function show(int $id): Response  
{  
    return new Response("Utilisateur ID : $id");  
}
```

# CONTRÔLEUR

Vous pouvez aussi retourner différents types de réponses dans Symfony :

- HTML via `render()`.
- JSON via `JsonResponse()` pour les API.
- Redirection via `redirectToRoute()` pour les redirections.

Les contrôleurs sont essentiels pour gérer la logique de votre application et structurer vos réponses.

Symfony rend cette tâche simple et cohérente grâce à des méthodes bien définies.

```
namespace App\Controller;

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

class HelloController extends AbstractController
{
    #[Route('/hello/{name}', name: 'hello_page')]
    public function hello(string $name = 'Symfony'): Response
    {
        // Retourner du HTML avec une vue Twig
        return $this->render('hello.html.twig', [
            'name' => $name,
        ]);
    }

    #[Route('/json', name: 'json_example')]
    public function jsonExample(): JsonResponse
    {
        // Retourner du JSON
        return new JsonResponse(['message' => 'Hello, Symfony!']);
    }

    #[Route('/redirect', name: 'redirect_example')]
    public function redirectExample(): Response
    {
        // Retourner une redirection vers une autre route
        return $this->redirectToRoute('hello_page', ['name' => 'Symfony']);
    }
}
```

# CONTRÔLEUR

Ajouter une nouvelle méthode dans le contrôleur et sa vue Twig associée :

- Vous pouvez ajouter une méthode dans le contrôleur et créer le fichier Twig associé manuellement.

Créer le fichier Twig associé :

Lorsque vous appelez la méthode `render()` dans le contrôleur, Symfony cherche un fichier Twig à l'emplacement spécifié. Par exemple, pour return `$this->render('hello/accueil.html.twig');`, il cherchera un fichier `accueil.html.twig` dans le répertoire `templates/hello/`.

Création manuelle du fichier Twig :

- Créez le répertoire `templates/hello/` s'il n'existe pas.
- Créez le fichier `accueil.html.twig` dans ce répertoire.
- Créez un fichier `liste_articles.html.twig` dans `templates/blog/`
- Vous pouvez voir les pages sur les liens suivants:
  - <http://127.0.0.1:8000/accueil>
  - <http://127.0.0.1:8000/blog>

```
// Dans HelloController.php
#[Route('/accueil', name: 'accueil')]
public function accueil(): Response
{
    return $this->render('hello/accueil.html.twig');
}

#[Route('/blog', name: 'liste_articles')]
public function listeArticles(): Response
{
    return $this->render('blog/liste.html.twig');
}
```



# CONTRÔLEUR

Principales commandes CLI pour travailler avec les contrôleurs dans Symfony:

- `php bin/console make:controller`
  - Crée un nouveau contrôleur avec un fichier Twig associé.
- `php bin/console make:controller --no-twig`
  - Crée un contrôleur sans générer de fichier Twig (utile si vous n'en avez pas besoin).
- `php bin/console debug:router`
  - Affiche la liste de toutes les routes enregistrées dans votre application, y compris celles définies dans les contrôleurs.

```
PS C:\xampp\htdocs\learnProject> php bin/console debug:router
```

Name	Method	Scheme	Host	Path
_preview_error	ANY	ANY	ANY	/_error/{code}.{_format}
app_hello	ANY	ANY	ANY	/hello
accueil	ANY	ANY	ANY	/accueil
liste_articles	ANY	ANY	ANY	/blog

# CONTRÔLEUR

```
PS C:\xampp\htdocs\learnProject> php bin/console
Symfony 7.1.8 (env: dev, debug: true)

Usage:
  command [options] [arguments]

Options:
  -h, --help                Display help for the given command. When no command is given display help for the list command
  -q, --quiet               Do not output any message
  -V, --version             Display this application version
                          Force (or disable --no-ansi) ANSI output
  -n, --no-interaction     Do not ask any interactive question
  -e, --env=ENV            The Environment name. [default: "dev"]
                          Switch off debug mode.
                          Enables profiling (requires debug).
  -v|vv|vvv, --verbose    Increase the verbosity of messages: 1 for normal output, 2 for more verbose output and 3 for
debug

Available commands:
  about                    Display information about the current project
  completion              Dump the shell completion script
  help                    Display help for a command
  list                    List commands
  assets:install          Install bundle's web assets under a public directory
  cache
  cache:clear             Clear the cache
  cache:pool:clear        Clear cache pools
  cache:pool:delete       Delete an item from a cache pool
  cache:pool:invalidate-tags Invalidate cache tags for all or a specific pool
  cache:pool:list         List available cache pools
  cache:pool:prune        Prune cache pools
  cache:warmup            Warm up an empty cache
  config:dump-reference   Dump the default configuration for an extension
  debug
  debug:autowiring        List classes/interfaces you can use for autowiring
  debug:config            Dump the current configuration for an extension
  debug:container         Display current services for an application
  debug:dotenv            List all dotenv files with variables and values
  debug:event-dispatcher  Display configured listeners for an application
  debug:router            Display current routes for an application
  debug:twig              Show a list of twig functions, filters, globals and tests
  lint
  lint:container          Ensure that arguments injected into services match type declarations
  lint:twig               Lint a Twig template and outputs encountered errors
  lint:yaml               Lint a YAML file and outputs encountered errors
  make
  make:auth               Create a Guard authenticator of different flavors
  make:command            Create a new console command class
  make:controller         Create a new controller class
  make:crud               Create CRUD for Doctrine entity class
  make:docker:database    Add a database container to your compose.yaml file
  make:entity             Create or update a Doctrine entity class, and optionally an API Platform resource
  make:fixtures            Create a new class to load Doctrine fixtures
  make:form               Create a new form class
  make:listener            [make:subscriber] Creates a new event subscriber class or a new event listener class
  make:message            Create a new message and handler
  make:messenger-middleware Create a new messenger middleware
  make:migration           Create a new migration based on database changes
  make:registration-form   Create a new registration form system
  make:reset-password      Create controller, entity, and repositories for use with symfonycasts/reset-password-bundle
  make:schedule            Create a scheduler component
  make:security:custom     Create a custom security authenticator.
  make:security:form-login Generate the code needed for the form_login authenticator
  make:serializer:encoder Create a new serializer encoder class
  make:serializer:normalizer Create a new serializer normalizer class
  make:stimulus-controller Create a new Stimulus controller
  make:test               [make:unit-test][make:functional-test] Create a new test class
  make:twig-component     Create a twig (or live) component
  make:twig-extension     Create a new Twig extension with its runtime class
  make:user               Create a new security user class
  make:validator           Create a new validator and constraint class
  make:voter              Create a new security voter class
  make:webhook             Create a new Webhook
  router
  router:match            Help debug routes by simulating a path info match
  secrets
  secrets:decrypt-to-local Decrypt all secrets and stores them in the local vault
  secrets:encrypt-from-local Encrypt all local secrets to the vault
  secrets:generate-keys   Generate new encryption keys
  secrets:list            List all secrets
  secrets:remove          Remove a secret from the vault
  secrets:reveal          Reveal the value of a secret
  secrets:set             Set a secret in the vault
```

# CONTRÔLEUR

## Exception avec AbstractController

```
use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;
use Symfony\Component\HttpFoundation\Exception\NotFoundHttpException;

#[Route('/user/{id}', name: 'user_show')]
public function show(int $id): Response
{
    if (!$user = $this->getUserById($id)) {
        throw $this->createNotFoundException("Utilisateur $id non trouvé !");
    }

    return new Response("Utilisateur : $user->name");
}
```

- `createNotFoundException()` : Utilisé pour renvoyer une erreur 404.
- Exceptions Symfony intégrées :
  - `AccessDeniedException` (403)
  - `NotFoundHttpException` (404)
  - `HttpException` (personnalisable).

Les exceptions améliorent la gestion des erreurs et fournissent des retours clairs à l'utilisateur.

# MOTEUR DE TEMPLATE TWIG

Nous allons aborder Twig, le moteur de templates de Symfony, avec un focus sur son utilisation pour afficher des vues dynamiques et réutilisables.

- Twig est un moteur de templates flexible et sécurisé utilisé pour générer des vues HTML dynamiques dans Symfony.
- Il permet d'intégrer des données depuis les contrôleurs dans des fichiers `.twig` (vues).

## Syntaxe de Base

- Variables : On insère les variables dans Twig avec la syntaxe `{{ variable }}`
  - `<h1>{{ title }}</h1>`
  - Si `title` est une variable passée depuis le contrôleur (par exemple, 'title' => 'Bienvenue'), elle sera affichée ici sous forme de texte.
- Contrôle de flux : Utilisation des boucles et des conditions.

```
{% if user %}
```

```
<p>Hello, {{ user.name }}!</p>
```

```
{% else %}
```

```
<p>Guest</p>
```

```
{% endif %}
```

- Ici, on vérifie si la variable `user` existe. Si oui, on affiche son nom. Sinon, on affiche "Guest"

# MOTEUR DE TEMPLATE TWIG

Boucles: Pour itérer sur des collections (comme des tableaux ou des objets).

```
<ul>
  {% for post in posts %}
    <li>{{ post.title }}</li>
  {% endfor %}
</ul>
```

- Cela permet de parcourir un tableau de posts et d'afficher chaque titre de post dans une liste.

## Héritage de Templates

- Twig permet de définir une base de template (layout) et d'hériter de celui-ci pour réutiliser des structures communes. Cela permet de centraliser la structure HTML (en-tête, pied de page, etc.) et de ne la définir qu'une seule fois.

{# base.html.twig #}

```
<html>
  <head><title>{% block title %}My App{% endblock %}</title></head>
  <body>{% block body %}{% endblock %}</body>
</html>
```

- Ici, {`% block title %`} et {`% block body %`} définissent des blocs qui peuvent être remplis ou modifiés par des templates enfants. Par défaut, le titre sera "My App" et le corps sera vide, mais les enfants peuvent redéfinir ces blocs.

# MOTEUR DE TEMPLATE TWIG

## Utilisation des Filtres et FonctionsExtension d'un Template

- Dans un autre fichier (par exemple `home.html.twig`), vous pouvez étendre ce template de base pour réutiliser sa structure tout en personnalisant certains blocs.

```
{# home.html.twig #}
```

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

```
{% block title %}Home{% endblock %}
```

```
{% block body %}<h1>Welcome to my website</h1>{% endblock %}
```

- `{% extends 'base.html.twig' %}` indique que ce fichier étend le template `base.html.twig`, et réutilise sa structure HTML.
- Le bloc `{% block title %}` est redéfini pour afficher "Home" comme titre de la page.
- Le bloc `{% block body %}` est remplacé par un message personnalisé.
- Les filtres permettent de manipuler les données avant de les afficher. Par exemple, pour formater une date.
- `<p>{{ post.date | date("Y-m-d") }}</p>`
  - Le filtre `| date("Y-m-d")` transforme la variable `post.date` en une chaîne au format année-mois-jour (ex : 2024-11-20).
- Les fonctions permettent de faire des actions comme inclure un autre fichier Twig.
- `{% include 'header.html.twig' %}`
  - Cette ligne inclut le fichier `header.html.twig` dans votre template, permettant ainsi de réutiliser des parties du code comme l'en-tête sur plusieurs pages.

# MOTEUR DE TEMPLATE TWIG

## Variable app, linting:

- Variable app : Elle permet d'accéder à des services ou des données dans Twig, par exemple :
  - `<p>{{ app.request.uri }}</p>`
  - L'utilisateur connecté (`app.user`)
- Linting : Utilisez `php bin/console lint:twig` pour vérifier la syntaxe des fichiers Twig.

# L'OBJET REQUEST DU COMPOSANT HTTPFOUNDATION

L'objet Request est un élément clé du composant HttpFoundation de Symfony. Il permet d'accéder à toutes les informations liées à une requête HTTP envoyée par le client : URL, paramètres, en-têtes, cookies, fichiers, etc.

- Exemple simple : Accéder aux paramètres GET



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

#[Route('/search', name: 'search')]
public function search(Request $request): Response
{
    $query = $request->query->get('q'); // Récupère ?q=valeur dans l'URL
    return new Response("Vous avez cherché : $query");
}
```



# L'OBJET REQUEST DU COMPOSANT HTTPFOUNDATION

- Accéder aux paramètres POST



```
public function submit(Request $request): Response
{
    $data = $request->request->get('data'); // Récupère 'data' depuis un formulaire POST
    return new Response("Données envoyées : $data");
}
```

- Requête GET : `$request->query->get('paramètre')`
- Requête POST : `$request->request->get('paramètre')`
- Cookies : `$request->cookies->get('nom_cookie')`
- Fichiers : `$request->files->get('nom_fichier')`

# FORMULAIRES

## Créer un formulaire basique

- Exécutez cette commande pour installer les paquets requis :
  - `composer require symfony/form symfony/validator symfony/orm-pack`
- `php bin/console make:form ContactType`
- `php bin/console make:controller ContactController`
- `php bin/console make:entity`

```
use Symfony\Component\Form\AbstractType;
use Symfony\Component\Form\FormBuilderInterface;
use Symfony\Component\Form\Extension\Core\Type\TextType;
use Symfony\Component\Form\Extension\Core\Type\EmailType;

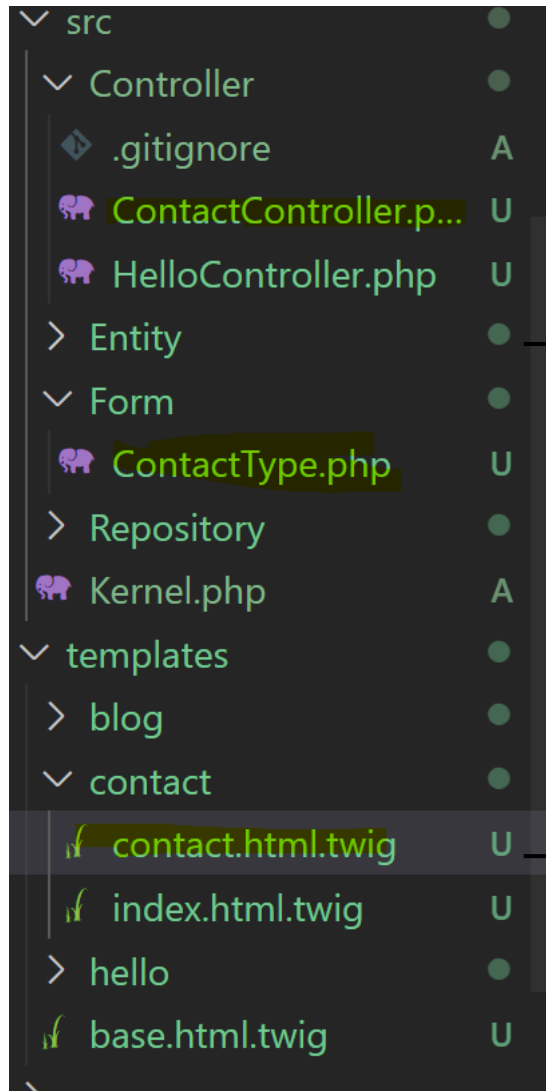
class ContactType extends AbstractType
{
    public function buildForm(FormBuilderInterface $builder, array $options): void
    {
        $builder
            ->add('name', TextType::class, ['label' => 'Nom'])
            ->add('email', EmailType::class, ['label' => 'Email']);
    }
}
```

```
#[Route('/contact', name: 'contact')]
public function contact(Request $request): Response
{
    $form = $this->createForm(ContactType::class);

    // Traiter le formulaire (soumission, validation, etc.)
    $form->handleRequest($request);
    if ($form->isSubmitted() && $form->isValid()) {
        // Traiter les données
    }

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

# FORMULAIRES



```
namespace App\Entity;

use Doctrine\ORM\Mapping as ORM;

#[ORM\Entity(repositoryClass: ContactRepository::class)]
class Contact
{
    #[ORM\Id]
    #[ORM\GeneratedValue]
    #[ORM\Column(type: 'integer')]
    private $id;

    #[ORM\Column(type: 'string', length: 255)]
    private ?string $name = null;

    #[ORM\Column(type: 'string', length: 255)]
    private ?string $email = null;

    #[ORM\Column(type: 'text')]
    private ?string $message = null;

    // Getter et setter pour chaque propriété...
}
```

← → ↻ ⓘ 127.0.0.1:8000/contact

Nom

Email

```
{{ form_start(form) }}
    {{ form_row(form.name) }}
    {{ form_row(form.email) }}
    <button type="submit">Envoyer</button>
{{ form_end(form) }}
```

# VALIDATION DES FORMULAIRES



```
use Symfony\Component\Validator\Constraints as Assert;

class Contact
{
    #[Assert\NotBlank(message: "Le nom ne peut pas être vide.")]
    #[Assert\Length(min: 3, max: 50, minMessage: "Le nom doit contenir au moins 3 caractères.")]
    private ?string $name;

    #[Assert\NotBlank(message: "L'email est requis.")]
    #[Assert\Email(message: "Veuillez fournir un email valide.")]
    private ?string $email;
}
```

# VALIDATION DES FORMULAIRES

```
use Symfony\Component\Form\Extension\Core\Type\TextType;
use Symfony\Component\Validator\Constraints\NotBlank;

$builder->add('name', TextType::class, [
    'constraints' => [
        new NotBlank(['message' => 'Le nom est obligatoire.']),
    ],
]);
```

## Liste des contraintes courantes :

- NotBlank : Assure que le champ n'est pas vide.
- Length : Définit une longueur minimale et/ou maximale.
- Email : Valide un format d'email.
- Regex : Valide via une expression régulière.
- Unique : Vérifie l'unicité d'une valeur.
- Entity : Vérifie l'unicité dans une base de données.
- Choice : Valide que la valeur fait partie d'une liste définie.
- GreaterThan / LessThan : Valide une plage numérique.

# BDD, ENTITÉS ET RELATIONS

Gestion des entités et de la base de données avec Doctrine.

Nous allons :

- Configurer la base de données
- Créer les entités avec Doctrine
- Faire les migrations pour appliquer les entités à la base de données

## Configuration de la base de données

- Avant de commencer à travailler avec Doctrine et les entités, il est nécessaire de configurer la connexion à la base de données dans Symfony.
- Dans le fichier `.env` ou `.env.local`, vous devez spécifier votre configuration de base de données.
- Exemple pour une base de données MySQL :  
`DATABASE_URL="mysql://username:password@127.0.0.1:3306/nom_de_votre_base"`

# BDD, ENTITÉS ET RELATIONS

## Créer une entité avec Doctrine

- Pour générer une entité en ligne de commande, vous utilisez la commande suivante :
  - `php bin/console make:entity`
  - Par exemple, pour créer une entité Contact, vous allez suivre les étapes guidées pour définir les propriétés de l'entité. Si vous voulez ajouter des champs comme name, email et message, la commande vous guidera à travers le processus pour les définir comme propriétés.
- `src/Entity/Contact.php` :

```
namespace App\Entity;

use Doctrine\ORM\Mapping as ORM;

/**
 * @ORM\Entity(repositoryClass=ContactRepository::class)
 */
class Contact
{
    /**
     * @ORM\Id
     * @ORM\GeneratedValue
     * @ORM\Column(type="integer")
     */
    private $id;

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

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

    /**
     * @ORM\Column(type="text")
     */
    private $message;




    // Getters et Setters...
}
```

# BDD, ENTITÉS ET RELATIONS

## Création des migrations

- Générer la migration : `php bin/console make:migration`
  - Cette commande génère une nouvelle migration en comparant la structure actuelle de vos entités Doctrine avec celle de la base de données.
  - Elle crée un fichier de migration dans le dossier `src/Migrations/` contenant les instructions SQL nécessaires pour synchroniser la base de données avec les entités (ajout, suppression ou modification de colonnes, etc.).
  - Si vous avez modifié une entité en ajoutant une nouvelle propriété, cette commande va générer une migration pour créer la colonne correspondante dans la base de données.
- Exécuter la migration pour appliquer les changements à la base de données: `php bin/console doctrine:migrations:migrate`
  - Cette commande exécute toutes les migrations en attente (celles générées par `make:migration`) pour appliquer les changements dans la base de données.
  - Elle applique les migrations à la base de données en exécutant les requêtes SQL contenues dans les fichiers de migration.
  - Une fois qu'une migration a été générée, vous devez exécuter cette commande pour appliquer les changements à la base de données.

```
PS C:\xampp\htdocs\learnProject> php bin/console doctrine:migrations:migrate
WARNING! You are about to execute a migration in database "symfony_db_abbs" that could result in schema ch
ta loss. Are you sure you wish to continue? (yes/no) [yes]:
> yes
[notice] Migrating up to DoctrineMigrations\Version20241120195026
[notice] finished in 58.2ms, used 12M memory, 1 migrations executed, 1 sql queries
[OK] Successfully migrated to version: DoctrineMigrations\Version20241120195026
```

Table	Action	Lignes	Type	Interclassement	Taille	Perte
<input type="checkbox"/> contact	★      	0	InnoDB	utf8mb4_unicode_ci	16,0 kio	-
<input type="checkbox"/> doctrine_migration_versions	★      	1	InnoDB	utf8_unicode_ci	16,0 kio	-
2 tables	Somme	1	InnoDB	utf8mb4_general_ci	32,0 kio	0 0



# BDD, ENTITÉS ET RELATIONS

## Manipulation des données avec Doctrine

- Doctrine fournit un accès simple à la base de données en utilisant l'Entity Manager. Par exemple, pour enregistrer un nouvel objet Contact dans la base de données, vous pourriez utiliser un contrôleur comme suit :

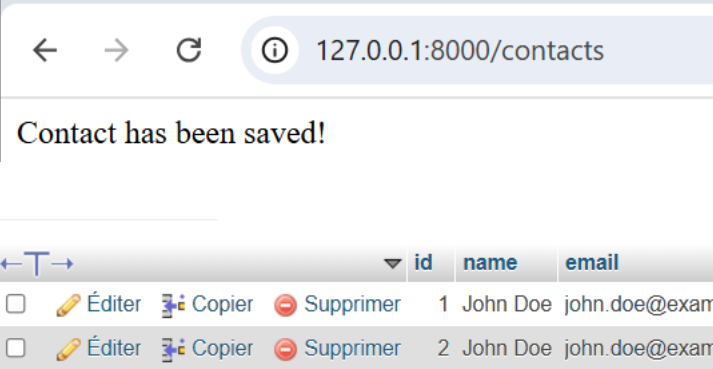
```
namespace App\Controller;

use App\Entity>Contact;
use Symfony\Component\HttpFoundation\Request;
use Symfony\Component\HttpFoundation\Response;
use Symfony\Component\Routing\Annotation\Route;
use Doctrine\ORM\EntityManagerInterface;
use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;

class ContactController extends AbstractController
{
    /**
     * @Route("/contact", name="contact")
     */
    public function index(Request $request, EntityManagerInterface $em): Response
    {
        // Créer un nouvel objet Contact
        $contact = new Contact();
        $contact->setName("John Doe");
        $contact->setEmail("john.doe@example.com");
        $contact->setMessage("Hello, this is a test message.");

        // Persister l'objet Contact dans la base de données
        $em->persist($contact);
        $em->flush();

        return new Response('Contact has been saved!');
    }
}
```



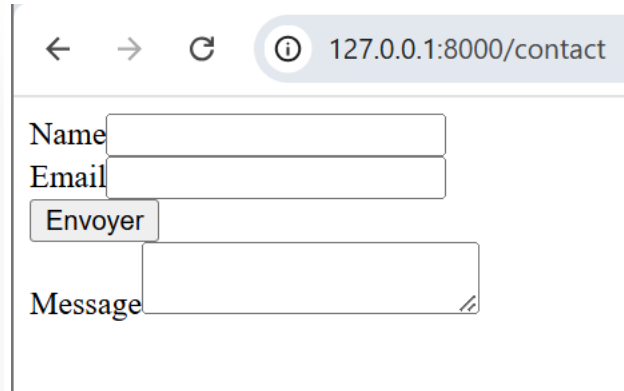
The screenshot shows a web browser at the address 127.0.0.1:8000/contacts. The page displays a message "Contact has been saved!". Below the message is a table with two rows of contact data. Each row has a checkbox, an edit icon, a copy icon, and a delete icon. The table columns are id, name, email, message, and stop.

	id	name	email	message	stop
<input type="checkbox"/> Éditer Copier Supprimer	1	John Doe	john.doe@example.com	Hello, this is a test message.	NULL
<input type="checkbox"/> Éditer Copier Supprimer	2	John Doe	john.doe@example.com	Hello, this is a test message.	NULL

# GESTION DES ENTITÉS AVEC LES FORMULAIRES

Une fois l'entité **Contact** créée et enregistrée dans la base de données, vous pouvez l'utiliser dans un formulaire pour collecter des données utilisateur.

```
private $entityManager;  
  
// Injection de l'EntityManagerInterface  
public function __construct(EntityManagerInterface $entityManager)  
{  
    $this->entityManager = $entityManager;  
}  
  
#[Route('/contact', name: 'contact')]  
public function contact(Request $request): Response  
{  
    $contact = new Contact();  
  
    // Créez le formulaire  
    $form = $this->createForm(ContactType::class, $contact);  
  
    // Traitez la soumission du formulaire  
    $form->handleRequest($request);  
  
    if ($form->isSubmitted() && $form->isValid()) {  
        // Enregistrez l'objet Contact dans la base de données  
  
        $this->entityManager->persist($contact);  
        $this->entityManager->flush();  
  
        return $this->redirectToRoute('contact');  
    }  
  
    return $this->render('contact/contact.html.twig', [  
        'form' => $form->createView(),  
    ]);  
}
```



← → ↻ ⓘ 127.0.0.1:8000/contact

Name

Email

Message

# GÉNÉRATION D'ENTITÉ DEPUIS LES TABLES EN BDD

Symfony peut générer des entités à partir de tables existantes en base de données à l'aide de la commande


- `php bin/console make:entity --regenerate App\Entity`

Cela synchronise les entités avec la structure de votre base de données.

# SERVICE

Un service est une classe qui exécute une logique métier ou une fonctionnalité. Il est généralement utilisé pour centraliser des actions réutilisables.

Les services sont injectés dans vos contrôleurs via l'injection de dépendances ou le conteneur de services.



```
// Service
class MyService {
    public function processData() {
        // logique ici
    }
}

// Injection dans le contrôleur
public function __construct(MyService $myService) {
    $this->myService = $myService;
}
```

# SERVICE

Créez un fichier CalculatorService.php dans le dossier src/Service :

```
// src/Service/CalculatorService.php
namespace App\Service;

class CalculatorService
{
    public function add(int $a, int $b): int
    {
        return $a + $b;
    }
}
```

Symfony enregistre automatiquement les services, mais vous pouvez explicitement configurer le service dans le fichier config/services.yaml

# config/services.yaml

services:

App\Service\CalculatorService: ~

Injection du service dans un contrôleur

```
// src/Controller/CalculatorController.php

namespace App\Controller;

use App\Service\CalculatorService;
use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;
use Symfony\Component\HttpFoundation\Response;

class CalculatorController extends AbstractController
{
    private CalculatorService $calculatorService;

    public function __construct(CalculatorService $calculatorService)
    {
        $this->calculatorService = $calculatorService;
    }

    public function index(): Response
    {
        $result = $this->calculatorService->add(3, 4); // Utilisation du service

        return new Response('Result: ' . $result);
    }
}
```

# SESSION & COOKIE

Symfony fournit une API pour gérer les sessions via le service Session.

```
// Stocker dans la session
```

```
$this->get('session')->set('username', 'John');
```

```
// Récupérer depuis la session
```

```
$username = $this->get('session')->get('username');
```

Symfony vous permet de créer, récupérer et supprimer des cookies via le service Response.

```
// Créer un cookie
```

```
$response = new Response();
```

```
$response->headers->setCookie(new Cookie('name', 'value', time() + 3600));
```

```
$response->send();
```

# COMPOSANT SYMFONY: SECURITY-CSRF

## Qu'est-ce que le CSRF ?

- Le CSRF (Cross-Site Request Forgery) est une attaque qui permet à un attaquant d'effectuer des actions non autorisées sur un site web en utilisant l'identité d'un utilisateur authentifié,
- Il protège vos formulaires en générant et validant des tokens uniques pour chaque requête, afin d'éviter les attaques CSRF.
- `composer require symfony/security-csrf`
- Ajoute automatiquement un token CSRF à vos formulaires, validé lors de la soumission pour garantir que la requête provient bien du formulaire légitime.
- Protection contre les attaques CSRF.
- Simplifie l'intégration dans les formulaires Symfony.

# INTRODUCTION AU COMPOSANT SECURITY

- Le composant Security de Symfony gère l'authentification, l'autorisation et les contrôles d'accès pour protéger les ressources de votre application.
- Il inclut des mécanismes comme les firewalls, les authentifiers et les encoders.
- Contrôle d'accès
  - Le contrôle d'accès en Symfony permet de définir qui peut accéder à quoi, en configurant des règles dans le fichier security.yaml.
  - On utilise des firewalls pour définir les zones protégées et des règles d'accès pour restreindre les actions.
- Messages d'erreur
  - Symfony gère les messages d'erreur via le composant Security, notamment lors de l'authentification échouée.
  - Ces messages peuvent être personnalisés dans la configuration ou directement dans les contrôleurs.
    - `$this->addFlash('error', 'Invalid username or password.');`
- Limiter les tentatives de connexion
  - Symfony permet de limiter les tentatives de connexion pour éviter les attaques par force brute.
  - Vous pouvez activer cette fonctionnalité via la configuration dans security.yaml.

```
security:
    firewalls:
        login:
            pattern: ^/login$
            form_login:
                max_attempts: 5
```



# COMMANDE POUR GÉNÉRER UN CRUD DANS SYMFONY

CRUD (Create, Read, Update, Delete) est un ensemble d'opérations de gestion de données essentielles dans une application web.

Symfony fournit une structure pour implémenter facilement ces actions avec les entités et les contrôleurs.

- Create : Créer une entité via un formulaire et la persister.
- Read : Lire et afficher les entités de la base.
- Update : Modifier les entités et les enregistrer.
- Delete : Supprimer une entité de la base de données.

# COMMANDE POUR GÉNÉRER UN CRUD DANS SYMFONY

→ Serveur : 127.0.0.1 » Base de données : symfony\_db\_abbs

StructureSQLRechercherRequêteExporterImporterOpérationsPrivilègesProcédures stockéesÉvènementsD

Filtres

Contenant le mot :

Table	Action
<input type="checkbox"/> article	★ Parcourir Structure Rechercher
<input type="checkbox"/> contact	★ Parcourir Structure Rechercher
<input type="checkbox"/> doctrine_migration_versions	★ Parcourir Structure Rechercher

3 tablesSomme

## Article index

Id	Title	Content	actions
1	billet first		<a href="#">show</a> <a href="#">edit</a>

[Create new](#)

classement	Taille	Perte
nb4_unicode_ci	16,0 kio	-
nb4_unicode_ci	16,0 kio	-
_unicode_ci	16,0 kio	-
mb4_general_ci	48,0 kio	0 o

```
PS C:\xampp\htdocs\learnProject> php bin/console doctrine:migrations:migrate --no-interaction
```

```
WARNING! You are about to execute a migration that could result in a loss. Are you sure you wish to continue? (yes/no) [yes]: > yes
```

```
[notice] Migrating up to DoctrineMigrations\Version20241120215337
[notice] finished in 26.6ms, used 12M memory, 1 migrations executed, 1 sql queries

[OK] Successfully migrated to version: DoctrineMigrations\Version20241120215337
```

```
PS C:\xampp\htdocs\learnProject>
```

☆ |

Symfony Docs

HTTP 500 Internal Server Error

Exception!

# EXERCICE : APPLICATION DE GESTION DE TÂCHES (TO-DO LIST)

1. Créer une application Symfony de gestion de tâches avec un header et un footer communs sur chaque page.
2. Utiliser Twig pour gérer la structure HTML et réutiliser les éléments.
3. Consignes :
  1. Création de l'entité Task : L'entité Task doit avoir les champs suivants :
    2. title : titre de la tâche.
    3. description : détails de la tâche.
    4. dueDate : date d'échéance.
    5. isCompleted : statut de la tâche.
4. Formulaire d'ajout/édition d'une tâche : Utilisez Symfony Forms pour créer un formulaire permettant d'ajouter ou modifier une tâche.
5. Affichage de la liste des tâches : Créez une page qui affiche toutes les tâches avec des options pour marquer une tâche comme terminée.
6. Filtrage et tri des tâches : Implémentez une fonctionnalité de filtrage par statut (toutes, complètes, en cours) et triez les tâches par date d'échéance.
7. Suppression d'une tâche : Implémentez une fonctionnalité permettant de supprimer une tâche.
8. Pagination : Ajoutez la pagination pour limiter le nombre de tâches affichées par page.
9. Création du Header et du Footer avec Twig :
10. Créez un fichier `base.html.twig` qui contiendra la structure commune du site (balise `<header>`, `<footer>`, et les liens de navigation).
11. Utilisez `block` et `extend` dans Twig pour réutiliser le header et footer dans chaque page.

# QCM

FIN

