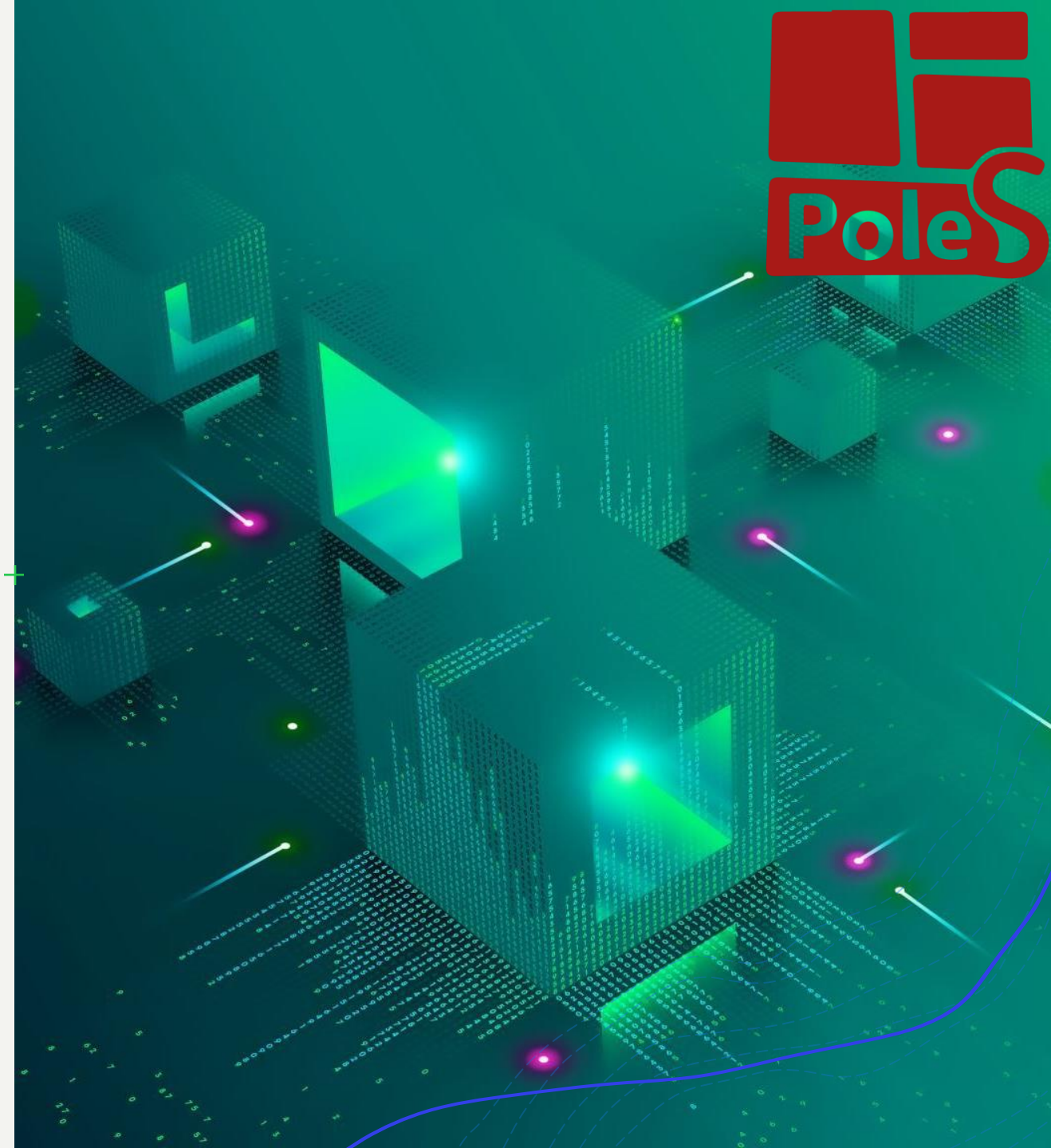
Symfony

Framework PHP



Histoire de Symfony

+ Création : 18 octobre 2005

+ Créateur : Fabien Potencier (Lead dev chez sensioLabs)

+ Développé par : Symfony SAS

+ Ecrit en : PHP

+ Type : Framework

Qu'est-ce qu'un framework ?

- Un framework est une structure qui améliore un langage de programmation (comme PHP avec Symfony ou JavaScript avec React) en fournissant une architecture prédéfinie, des bibliothèques intégrées et des fonctions utiles. Il facilite le développement d'applications en offrant des solutions prêtes à l'emploi pour des tâches courantes, facilitant ainsi le processus de codage et augmentant l'efficacité des développeurs.

- Attention : un framework IMPOSE son cadre de travail, par exemple bootstrap (framework front-end) impose l'utilisation de grilles de 12 colonnes pour l'écran (entre autres)

Pourquoi utiliser Symfony?

+ Symfony est fourni avec l'architecture MVC (model-vue-controleur) pour une meilleure organisation et facilitation du code.
+ Symfony rassemble des composants utiles et essentiels pour simplifier le développement :
+ Doctrine ORM : Facilite l'intéraction avec la BDD en reliant les objets PHP aux tables de la BDD
+ Gestion des routes : nous permet de définir et de gérer les URL de notre application
+ Gestion des sessions : gère la session utilisateur de manière sécurisée
+ Composant d'authentification : gère l'authentification et les accès aux pages
+ Composant de mail : pour simplier l'envoi de mails depuis l'application
+ Faker : génère de fausses données pour les tests

Creation projet Symfony

+Pour créer notre projet Symfony, nous utiliserons composer:

composer create-project symfony/skeleton nom_du_projet
puis : composer require webapp

+(La commande --webapp signifie que les dépendances les plus utilisées dans les applications Symfony seront automatiquement incluses.)

Lignes de commandes

+ Nous allons commencer par récupérer les lignes de commandes de symfony grâce à scoop avec : scoop install symfony-cli.
+ Une fois fait, nous aurons accès aux lignes de commandes symfony (cela facilitera le développement de l'application)

+ Installer le maker-bundle qui nous permettra de créer directement nos éléments (controleur, repository, template, entity etc) avec :
composer require symfony/maker-bundle –dev

+ Par exemple pour créer un controleur, avec symfony-cli nous aurons juste à faire (symfony console make:controller), alors que sans nous devrons manuellement créer nos controleurs et nos templates

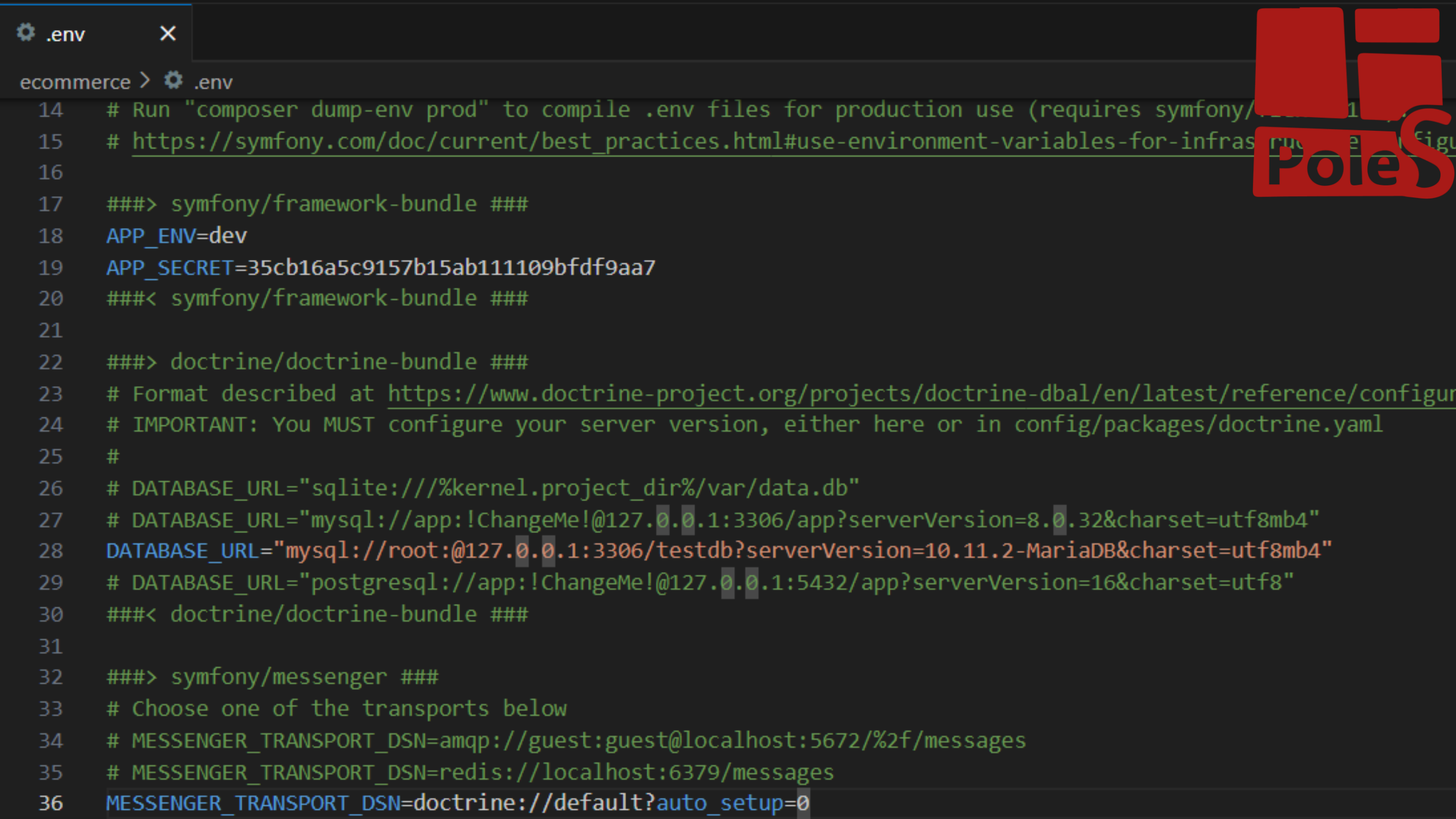
Creation BDD et installation ORM

- Avec Symfony, nous pouvons créer et gérer notre base de données directement via des lignes de commande, sans avoir besoin de passer par phpMyAdmin.
- Pour cela, nous devons installer le pack ORM (Object-Relational Mapping) qui facilite la gestion de la base de données en permettant de créer et de manipuler des entités à l'aide d'un modèle objet.
- Nous pouvons installer ce pack en utilisant Composer avec la commande suivante : composer require symfony/orm-pack
- Ce pack inclut les outils nécessaires pour configurer Doctrine ORM, qui est la bibliothèque utilisée pour l'interaction avec la base de données dans Symfony.
- Avec Doctrine, nous pourrons facilement créer les tables de la base de données, effectuer des migrations, et gérer les données en utilisant des objets PHP plutôt que des requêtes SQL brutes.

.env

+ Après avoir installé ORM, nous accederons au fichier .env qui contiendra les informations de connexion à la BDD (par defaut sur postgresql, mais nous travaillerons sur mysql, alors nous commenterons postgresql et decommenterons la ligne du dessus

+ Nous remplirons les informations de notre BDD pour avoir le resultat suivant :


.env
ecommerce > .env
14 # Run "composer dump-env prod" to compile .env files for production use (requires symfony/
15 # https://symfony.com/doc/current/best_practices.html#use-environment-variables-for-infras
16
17 ###> symfony/framework-bundle ###
18 APP_ENV=dev
19 APP_SECRET=35cb16a5c9157b15ab111109bfdf9aa7
20 ###< symfony/framework-bundle ###
21
22 ###> doctrine/doctrine-bundle ###
23 # Format described at https://www.doctrine-project.org/projects/doctrine-dbal/en/latest/reference/configur
24 # IMPORTANT: You MUST configure your server version, either here or in config/packages/doctrine.yaml
25 #
26 # DATABASE_URL="sqlite:///%kernel.project_dir%/var/data.db"
27 # DATABASE_URL="mysql://app:!ChangeMe!@127.0.0.1:3306/app?serverVersion=8.0.32&charset=utf8mb4"
28 DATABASE_URL="mysql://root:@127.0.0.1:3306/testdb?serverVersion=10.11.2-MariaDB&charset=utf8mb4"
29 # DATABASE_URL="postgresql://app:!ChangeMe!@127.0.0.1:5432/app?serverVersion=16&charset=utf8"
30 ###< doctrine/doctrine-bundle ###
31
32 ###> symfony/messenger ###
33 # Choose one of the transports below
34 # MESSENGER_TRANSPORT_DSN=amqp://guest:guest@localhost:5672/%2f/messages
35 # MESSENGER_TRANSPORT_DSN=redis://localhost:6379/messages
36 MESSENGER_TRANSPORT_DSN=doctrine://default?auto_setup=0

Créer la bdd

+ Nous avons maintenant nos informations à la BDD dans .env, il faut maintenant dire à symfony de créer notre bdd avec la commande : symfony console doctrine:database:create

+ Avant d'envoyer dans mysql notre BDD, nous allons créer une entité avec des informations (user par exemple), une entité représentera la table de la bdd dans notre application sous forme de classe d'objet

Creation entité

+ Pour créer cette entité, la commande est :
  symfony console make:entity user

+ Nous pourrons ensuite remplir l'entité des informations dont nous avons besoin sur l'utilisateur (qui seront converties en colonnes de table une fois dans la bdd)

+ Maintenant les informations remplies, nous verrons que symfony nous a créé un fichier repository et un fichier entity

Migrations

+ Les migrations permettent de mettre à jour notre BDD sans avoir besoin d'y accéder directement. Avec les commandes :

```
symfony console make:migration
symfony console doctrine:migrations:migrate
```

+ Cela enverra une mise à jour sur notre base de données sans la modifier manuellement (par exemple en modifiant sur phpmyadmin)

Modifications dans entités

+ Nous pouvons modifier la base de données depuis symfony directement, mais comment faire les modifications?

+ Pour cela, nous allons utiliser les entités. Les entités sont des classes PHP qui représentent les tables de notre base de données, et elles sont liées à ces tables via l'ORM (Object-Relational Mapping). L'ORM permet de faire correspondre les objets en PHP aux enregistrements dans la base de données de manière transparente.

Exemple de modification

+ Nous avons créé notre BDD et déjà fait une migration, mais nous avons oublié par exemple que "telephone" dans la table user pouvait être null, inutile d'aller sur phpmyadmin pour le modifier, nous pouvons directement le faire dans l'entité correspondante.

```
+    #[ORM\Column(nullable: true)]
+    private ?string $telephone = null;
```

+ Il nous suffit maintenant de refaire une migration pour que la BDD soit mise à jour

Creation controller/template

+ Qui dit Model (entity/repository) dit Controller et View, et pour ça, Symfony nous offre les deux en un, il nous suffit simplement de créer un controller pour qu'un template nous soit offert avec via Twig (moteur de template par defaut de Symfony)

+ La commande sera : symfony console make:controller

+ Entrez le nom de votre Controller, un Controller et un Template seront créés

Services / Injections de dépendances

+ Un **service** dans Symfony permet de gérer une tâche spécifique, comme l'envoi d'emails de confirmation, par exemple.
+ L'**injection de dépendances** consiste à fournir à une classe les services ou objets dont elle a besoin sans qu'elle ait à les instancier directement. Ces dépendances sont injectées via le constructeur ou un setter.

Exemple:

- Dans notre dossier Service, nous avons une classe EmailService qui contient une méthode sendEmail()

- Nous déclarerons ce service dans services.yaml : App\Service\EmailService: ~ (~ signifie que nous laissons Symfony gérer la configuration automatiquement)

- Puis nous injecterons EmailService directement dans le controleur via le constructeur, Symfony gèrera automatiquement la dépendance et fournira une instance de EmailService

Mais au fait, c'est quoi le constructeur ?

+Le constructeur est une méthode magique (les méthodes magiques commencent toujours par __) utilisée pour initialiser des propriétés lors de la création d'un objet.

Par exemple, si nous avons besoin d'une propriété qui stocke la date d'aujourd'hui et dont la valeur ne changera pas, (une date de commande) nous pouvons l'initialiser dans le constructeur. Ainsi, cette propriété sera prête à l'emploi dès la création de l'objet.

Composants

+ Symfony nous fournit des composants essentiels pour le gain de temps lors du code :

+ Authenticator : C'est un composant pour gérer l'authentification des utilisateurs dans Symfony. Il regroupe plusieurs éléments clés :
+ Authentification de l'utilisateur : Détermine comment les utilisateurs se connectent à l'application en vérifiant leurs informations d'identification, comme les noms d'utilisateur et les mots de passe.
+ Gestion de la session utilisateur : Permet de maintenir l'état d'authentification d'un utilisateur à travers différentes requêtes HTTP, en utilisant des sessions ou des jetons.
+ Contrôle d'accès (Access Control) : Détermine quel type d'utilisateur peut accéder à quelles pages de l'application. Cela se configure en définissant des règles qui associent des rôles d'utilisateur aux chemins URL.

Authenticator

+ Authenticator dans Symfony gère l'authentification des utilisateurs en vérifiant les informations d'identification comme le nom d'utilisateur et le mot de passe. Une fois l'utilisateur authentifié, un jeton de session est attribué pour maintenir l'identité de l'utilisateur à travers les requêtes suivantes. Cela garantit que l'utilisateur reste connecté tout au long de sa session.
+ On l'utilise avec la commande : symfony console make:auth
+ deux choix s'offrent à nous : l'utiliser avec ou sans formulaire intégré (à chacun de choisir ses préférences)

Access Control

+Nous pouvons décider quel utilisateur aura accès à quelle page grâce à l'Access Control, il nous suffit de nous rendre dans le fichier security.yaml et d'ajouter les routes ainsi que les roles qui y auront accès

```yaml
# Easy way to control access for large sections of your
# Note: Only the *first* access control that matches wi
access_control:
    - { path: ^/admin, roles: ROLE_ADMIN }
    - { path: ^/profile, roles: ROLE_USER }
```