Symfony

C. BENSARI

Plan

- Introduction
- Un framework PHP ?
- Quelques composants de Symfony
- Symfony – Caractéristiques
- Symfony – installation
- Symfony – architecture
- Symfony – architecture MVC

Introduction

- Symfony est un **framework open-source** destiné pour les applications web PHP
- Populaire et possède une grande communauté qui participe à son évolution
- Un ensemble d'outils élégants permettant la création d'applications web plus rapidement
- Inspiré par les frameworks Ruby on Rails, Django et Spring Framework
- Sponsorisé par **SensioLabs** et développé par **Fabian POTENCIER** en 2005

Un framework PHP ?

- Un framework est une collection de classes prêtes à être utilisées afin de développer une application web
- Symfony est un framework **full-stack** il contient un ensemble de composants (classes) PHP réutilisables
- Symfony est conçu pour optimiser le développement d'applications web
- Symfony fonctionne avec une configuration flexible en utilisant **YAML**, **XML** ou les **annotations**
- De nouvelles caractéristiques sont ajoutées à chaque nouvelle release de Symfony

Symfony - caractéristiques

- Le système Model-View-Controller
- Framework PHP performant
- Un routage flexible des URI
- Un code réutilisable et maintenable
- Gestion de la session
- Logging des erreurs
- Des classes de base de données complètes avec un support sur différentes plateformes
- Une communauté très active
- Un ensemble de composants découplés et réutilisables
- Assure une standardisation et une inter-opérabilité d'applications
- Sécurise l'application contre différentes attaques
- Le moteur de templating **Twig**

Symfony - Installation

- Deux moyens pour installer Symfony :
  - **Symfony installer** : installation à partir d'un fichier exécutable (nécessite Xampp, Wamp, ..)
    - Commande de création d'une application web PHP :

      symfony new HelloWorld
  - **Composer** : Utilisation de l'outil composer (ligne de commande) pour installer **Symfony**
    - Commande de création d'une application web PHP :

    composer create-project symfony/website-skeleton HelloWorld ``4.4.*``

    ou

    composer create-project symfony/skeleton HelloWorld

    Et aussi, **dans le dossier du projet** :

    composer require symfony/web-server-bundle --dev **(serveur embarqué de symfony)**
    - Pour exécuter l'application, lancer la commande (dans le répertoire du projet créé):

      php bin/console server:run (ctrl + C pour arrêter le serveur)
    - Pour tester l'application, sur le navigateur :

      http://localhost:8000

Symfony - Architecture

- Symfony est un ensemble de composants et de paquets de haute qualité
  - Les composants sont un ensemble de classes fournissant un corps d'une fonctionnalité unique. Exemple: **CachComponent** est un composant fournissant des fonctionnalités de cash qui peuvent être ajoutées à n'importe quelle application
  - Une application Symfony est elle même un paquet contenant une collection de sous paquets. Un paquet peut utiliser des composants Symfony ainsi que des composants tiers pour fournir des fonctionnalités. Exemple: **FrameworkBundle, DoctrineBundle** et **FrameworkExtraBundle**

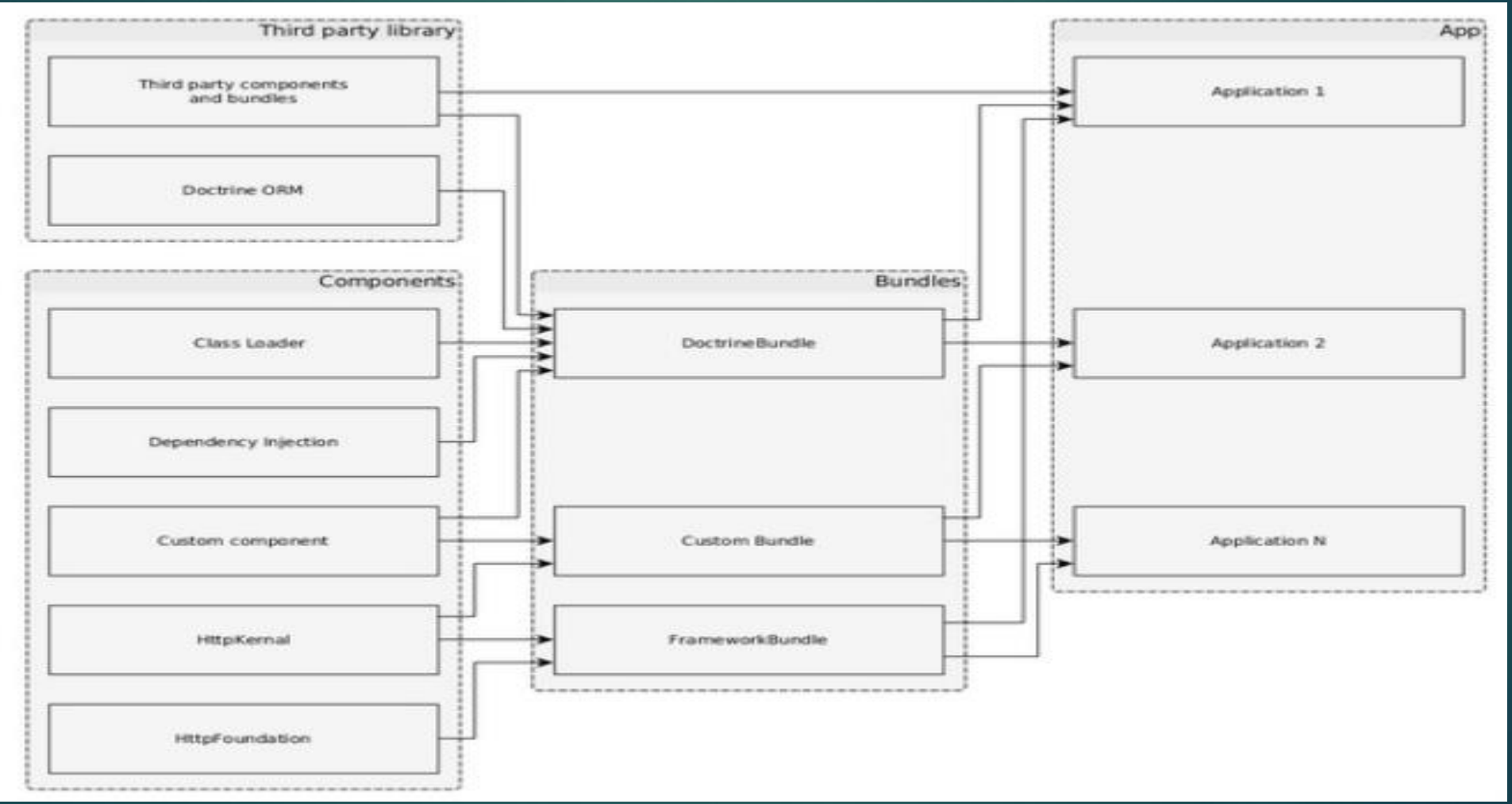
Quelques composants de Symfony

- HttpFoundation : composant permettant de bien manipuler les requêtes et réponses HTTP
- Validator : composant permettant de valider les données
- Kernel : c'est le composant cœur de Symfony. Il permet la gestion des environnements , des routes et a la possibilité de manipuler des requêtes HTTP
- FileSystem : composant fournissant un système de commande de base pour la création de fichiers et dossiers, etc
- Console : composant fournissant différentes options de création de commandes qui peuvent être exécutées dans un terminal

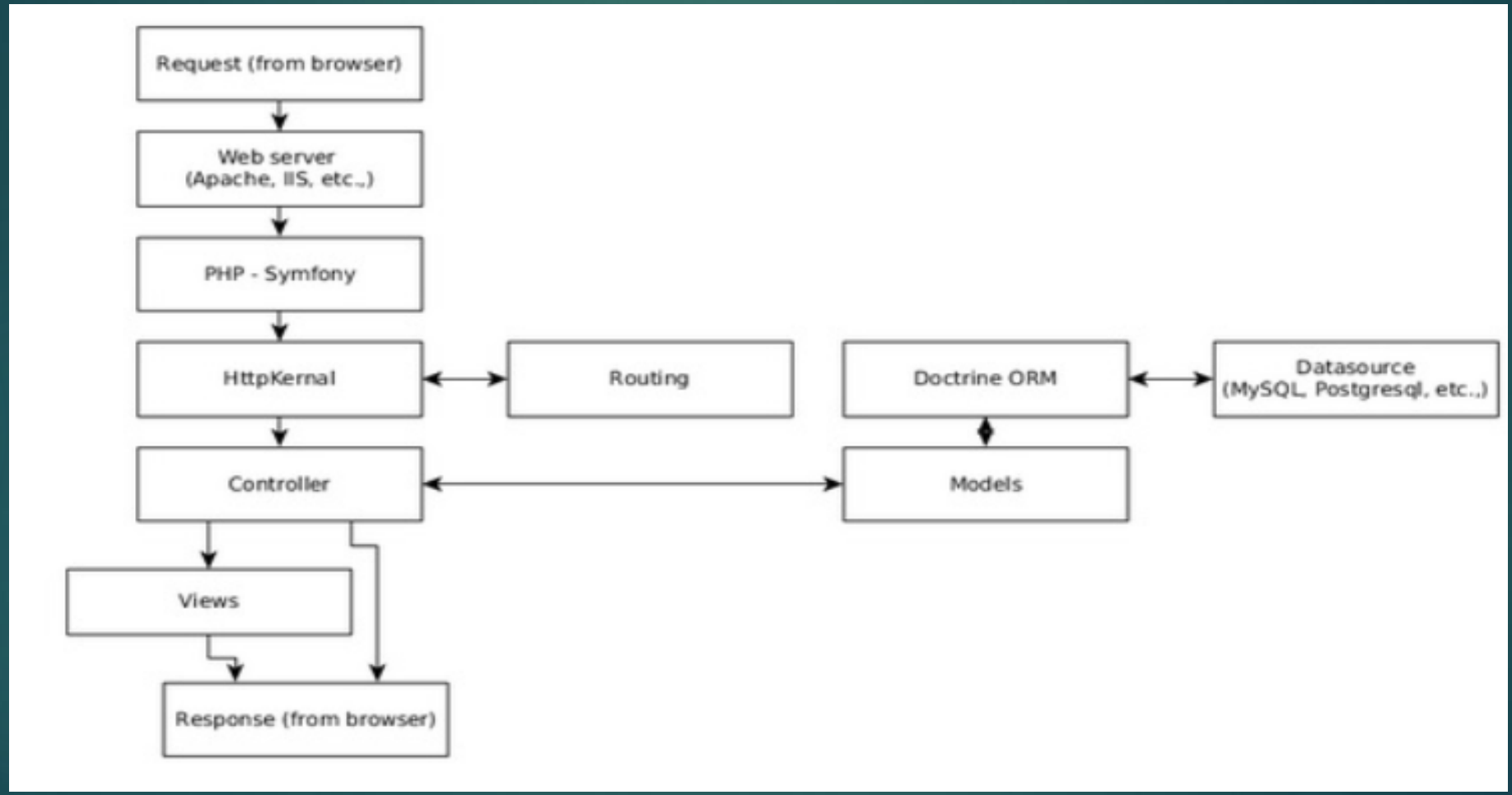
Quelques composants de Symfony

- EventDispatcher : fournit un système d'actions basées sur des evenements sur des objets PHP (Event et EventListener)
- DependencyInjection : fournit des mécanismes efficaces pour créer des objets avec leurs dépendances
- Serializer : permet la sérialisation d'objet vers différents formats (JSON, XML, …) et la dé-sérialisation vers les objets originaux sans perte de données
- Translation : fournit des options pour internationaliser une application
- Debug : fournit différentes options pour activer le debugging en PHP
- PHPUnitBridge : fournit des options pour améliorer PHPUnit (Tests unitaires)
- Routing : composant permettant de faire un mapping entre les requêtes HTTP et un ensemble de variables de configuration prédéfinies
- Security : fournit un système de sécurité complet pour les applications : Authentification, Certification, …

Symfony - Architecture



Symfony - workflow



Symfony - Architecture MVC

- MVC (Model View Controller) est un pattern avec une architecture en couches très utilisée pour les applications web
  - Model : couche qui représente la structure des différentes entités métiers (les données)
  - View : couche qui se charge d'afficher (données) à l'utilisateur selon la situation
  - Controller : Manipule les requêtes de l'utilisateur en interagissant avec la couche Model est fournit ensuite une Vue avec les données nécessaires
- Le framework web de Symfony fournit des fonctionnalités de haut niveau nécessaires pour la mise en place d'une application d'entreprise

Symfony - Controller

- **Controller** est le responsable de la manipulation de toute requête HTTP arrivant à une application Symfony
- Il lit l'information depuis la requête , crée ensuite et retourne un objet **Response** (réponse HTTP) au client
- Pour accéder à la requête dans un controller, utiliser la fonction getRequest() du controller.
- Pour créer une requête, utiliser la fonction statique Request::create(url, method, params)
- Pour créer une réponse , utiliser new Response(donnees, status, headers)

Symfony - Controller

```php
$request = Request::create(
    '/student',
    'GET',
    array('name' => 'student1')
);
```

Création d'une requête HTTP

```php
$response = new Response(json_encode(array('name' => $name)));
$response->headers->set('Content-Type', 'application/json');
```

Création d'une réponse HTTP

```php
$request = $this->getRequest();
```

Récupération d'une requête HTTP depuis le controleur

Symfony - Routing

- Le **Routing** est un mécanisme de correspondance entre des URI et leurs programmes (controller) correspondant
- Le composant **HttpKernel** intercepte la requête du navigateur (URI) et détermine la route correspondante à l'URI de la requête
- Les correspondances sont exprimées dans un fichier de configuration YAML, XML ou dans un Controller à travers des **annotations** « **@Route** »
- Une URI/URL se compose généralement de trois segments :
  - Machine (host) : www.mon_appli.com
  - Chemin (Path) : /articles/detail_article
  - Requête (request segment) : ?id_article=100

Symfony – Routing
Configuration avec Annotations

- Dans cette classe, deux Routes sont définies avec l'annotation **@Route** qui se trouvent au-dessus des méthodes
- La fonction *homeAction* est exécutée si le navigateur envoi une URI se terminant par **/student/home**
- La fonction *aboutAction* est exécutée si l'URI se termine avec **/student/about**

```php
namespace AppBundle\Controller;

use Symfony\Bundle\FrameworkBundle\Controller\Controller;
use Sensio\Bundle\FrameworkExtraBundle\Configuration\Route;

class StudentController extends Controller {
    /**
     * @Route("/student/home")
     */
    public function homeAction() {
        // ...
    }

    /**
     * @Route("/student/about")
     */
    public function aboutAction() {
    }
}
```

Symfony – Routing
Configuration avec Annotations

- Cas de pages avec pagination.
  - Exemples :
    - student/2
    - student/3
    - student/4
- Utilisation des *wildcards* formats
- Dans cet exemple, le \d+ représente une expression régulière qui signifie : « nombre décimal avec n'importe quel nombre de chiffre »

```
namespace AppBundle\Controller;

use Symfony\Bundle\FrameworkBundle\Controller\Controller;
use Sensio\Bundle\FrameworkExtraBundle\Configuration\Route;

class StudentController extends Controller {
   /**
      * @Route("/student/{page}", name = "student_about", requirements = {"page": "\d+"})
   */
   public function aboutAction($page) {
      // ...
   }
}
```

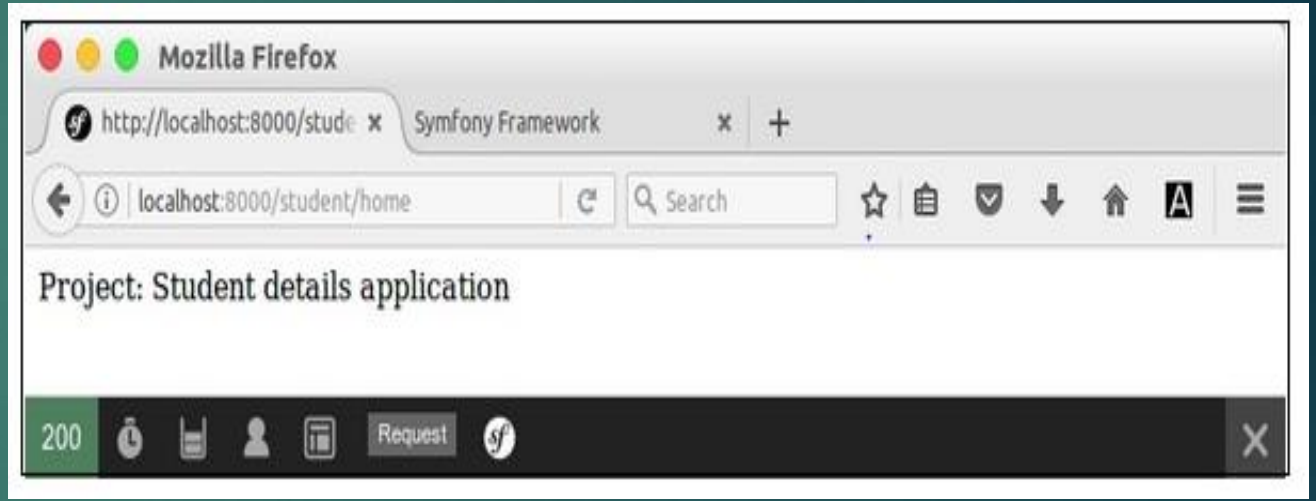
Exemple d'utilisation d'une route

```php
<?php
namespace AppBundle\Controller;

use Sensio\Bundle\FrameworkExtraBundle\Configuration\Route;
use Symfony\Component\HttpFoundation\Response;
use Symfony\Bundle\FrameworkBundle\Controller\Controller;

class StudentController  {
   /**
      * @Route("/student/home")
   */

   public function homeAction() {
      $name = 'Student details application';
      return new Response(
         '<html><body>Project: '.$name.'</body></html>'
      );
   }
}
```



Symfony – View Engine - TWIG

- La couche View de Syfmony se charge de la séparation entre la logique applicative et la logique de présentation
- Si le Controller aura besoin d'afficher du HTML, CSS, il transfert la tâche au moteur de vue (View Engine)
- Par défaut, les fichiers Html, se trouvent dans le dossier « templates »
- Par défaut, Symfony utilise le moteur de temlplating « TWIG »
- Twig est un langage de templating puissant qui permet d'écrire des templates lisible de manière facile
- Syntaxe générale de twig :
  - {{ ….. }} affiche une variable ou le résultat d'une expression dans le template (page)
  - {% …. %} un mot clé (TAG) permettant le contrôle de la logique. Utilisé généralement pour l'appel de fonctions
  - {# …. #} : mettre un commentaire mono ou multi-lignes

Symfony – View Engine - TWIG
Exemple d'utilisation

- La méthode « render » fournie par « Controller » permet d'afficher un template en l'envoyant comme un objet « Response »
- Elle peut prendre deux paramètres, le chemin vers le template et les données à passer sous forme d'un tableau

```php
<?php
namespace AppBundle\Controller;

use Sensio\Bundle\FrameworkExtraBundle\Configuration\Route;
use Symfony\Component\HttpFoundation\Response;
use Symfony\Bundle\FrameworkBundle\Controller\Controller;

class StudentController extends Controller {
    /**
     * @Route("/student/home")
     */
    public function homeAction() {
        return $this->render('student/home.html.twig');
    }
}
```

Symfony – View Engine - TWIG
notion de Layout

- Un Layout représente la partie commune de plusieurs vues. Exemple, un footer ou un header
- Avec TWIG il est possible d'utiliser la notion d 'héritage de templates
- L'héritage de template permet de construire un template de base qui contient les élements communs d'un site web définis par des « blocks »

Symfony – View Engine - TWIG
exemple d'utilisation de Layout

base.html.twig

```
<!DOCTYPE html>
<html>
   <head>
      <meta charset = "UTF-8">
      <title>{% block title %}Parent template Layout{% endblock %}</title>
   </head>
</html>
```

index.html.twig

```
{% extends 'base.html.twig' %}
{% block title %}Child template Layout{% endblock %}
```

Symfony - Model

- La couche **Model** joue un rôle important dans le framework **Symfony** et dans la plupart des frameworks web. Elle représente les entités du métier.
- Les entités sont créées par les utilisateurs (form) de l'application, récupérées à partir d'une base de données, modifiées par les utilisateurs puis **persistées** (sauvegardées) dans la base de données
- Les données de les classes de la partie **Model** sont aussi affichées pour les utilisateurs à travers les **Vues**
- La principale problématique des classes du Model c'est qu'ils sont en format **objet** tandis que la base données est au format **relationnel** (tables et relations)
- Nous avons besoin de faire **correspondre** (**mapping**) ces classes à nos tables. Ceci est fait grâce au **Bundle Doctrine** de Symfony
- **DoctrineBundle** intègre Symfony avec un **ORM** (**Object Relational Mapping**) qui se charge de faire le correspondance d'une manière transparente pour le développeur. Le mapping (correspondance classe ⇔ table) est réalisé grâce à un **ORM**

Utilisation de doctrine

- Pour installer doctrine (cas d'un projet symfony skeleton) :

  composer require doctrine maker

- Pour créer la base de données (sans tables) :
  - Ouvrir le fichier « .env » qui se trouve à la racine du projet et modifier la propriété DATABASE_URL avec les bons paramètres (user_name, password, host, db_name)

```
DATABASE_URL=mysql://root:@127.0.0.1:3306/db_hello_world?serverVersion=5.7
```

  - Lancer la commande suivante (toujours se positionner sur la racine du projet) :

    php bin/console doctrine:database:create

```
L570@DESKTOP-4A87S6G MINGW64 /f/Devloppement/php/symfony/HelloWorld
$ php bin/console doctrine:database:create
Created database `db_hello_world` for connection named default
```

Utilisation de doctrine

- Pour créer une table dans la base de donnée, il faut passer par les étapes suivantes :
  - Création de la classe « Entity » associée à la table

php bin/console make:entity Person

```
L57O@DESKTOP-4A87S6G MINGW64 /f/Devloppement/php/symfony/HelloWorld
$ php bin/console make:entity Person

 created: src/Entity/Person.php
 created: src/Repository/PersonRepository.php

 Entity generated! Now let's add some fields!
 You can always add more fields later manually or by re-running this command.

 New property name (press <return> to stop adding fields):
```

  - Après avoir créé l'Entity « Person » symfony propose d'ajouter des attributs à l'entity avec leur nom, type, longueur et si ils sont nullables ou pas
  - Une fois la saisie terminée, la classe « Person » est créée avec ses attributs et leurs getters et setters

Utilisation de doctrine

Entity

La classe contient les informations suivantes :

- le namespace **App\Entity** : dossier où va se trouver notre classe Person
- Utilisation du bundle Doctrine et de son composant **Mapping** (utilisation de l'alias **ORM)**
- La classe est annotée avec **@ORM\Entity (@Mapping\Entity())**
- L'annotation possède l'attribut **repositoryClass** qui indique la classe se chargeant de la communication (select) avec la base de données pour l'entité
- l'attribut id de l classe Person est annoté avec **@ORM/Id().** Ceci veut dire que cet attribut représentera l'identifiant de l'entité Person (**PRIMARY KEY)**
- **@ORM/GeneratedValue** précise la stratégie utilisée pour générer la clé primaire (id)
- **@ORM/Column** proposes des attributs pour donner des informations sur la colonne (nom, type, nulllable, longueur,..)

Person.php

```php
namespace App\Entity;

use Doctrine\ORM\Mapping as ORM;

/**
 * @ORM\Entity(repositoryClass="App\Repository\PersonRepository")
 */
class Person
{
    /**
     * @ORM\Id()
     * @ORM\GeneratedValue()
     * @ORM\Column(type="integer")
     */
    private $id;

    /**
     * @ORM\Column(type="string", length=50, nullable=true)
     */
    private $firstName;

    /**
     * @ORM\Column(type="text", nullable=true)
     */
    private $lastName;

    /**
     * @ORM\Column(type="integer")
     */
    private $age;
```

Utilisation de doctrine
Repository

- La classe **PersonRepository** hérite de la classe **ServiceEntityRepository** de **Doctrine**
- La classe fournit des méthodes toutes prêtes permettant d'effectuer quelques requêtes vers la base de données (find, findOneBy, findAll, findBy
- La classe propose des méthodes exemples utilisant des fonctions du composant **ServiceEntityRepository**

PersonRepository.php

```php
namespace App\Repository;

use App\Entity\Person;
use Doctrine\Bundle\DoctrineBundle\Repository\ServiceEntityRepository;
use Doctrine\Common\Persistence\ManagerRegistry;

/**
 * @method Person|null find($id, $lockMode = null, $lockVersion = null)
 * @method Person|null findOneBy(array $criteria, array $orderBy = null)
 * @method Person[]    findAll()
 * @method Person[]    findBy(array $criteria, array $orderBy = null, $limit = null, $offset = null)
 */
class PersonRepository extends ServiceEntityRepository
{
    public function __construct(ManagerRegistry $registry)
    {
        parent::__construct($registry, Person::class);
    }

    // /**
    //  * @return Person[] Returns an array of Person objects
    //  */
    /*
    public function findByExampleField($value)
    {
        return $this->createQueryBuilder('p')
            ->andWhere('p.exampleField = :val')
            ->setParameter('val', $value)
            ->orderBy('p.id', 'ASC')
            ->setMaxResults(10)
            ->getQuery()
            ->getResult()
```

Utilisation de doctrine
Migrations

- Afin de transformer (migrer) une classe de type Entity en une table dans une base de donnée relationnelle, il faut passer par les deux étapes suivantes :
  - Exécution de la commande permettant la génération de classes de type « Migration ». Les classes migrations possèderont des méthodes permettant la création, modification ou suppression de la table dans/depuis la base de données. Les classes générées peuvent être modifiées au besoin

php bin/console doctrine:migrations:diff

```
L57O@DESKTOP-4A87S6G MINGW64 /f/Devloppement/php/symfony/HelloWorld
$ php bin/console doctrine:migrations:diff
Generated new migration class to "F:\Devloppement\php\symfony\HelloWorld/src/Mi
rations/Version20191120193836.php"

To run just this migration for testing purposes, you can use migrations:execute
--up 20191120193836

To revert the migration you can use migrations:execute --down 20191120193836
```

Utilisation de doctrine
Migrations

Dans cet exemple :

- La méthode **up** permet la création de la table dans la base de données
- La méthode **down** permet la suppression de la table depuis la base de données

```php
final class Version20191120193836 extends AbstractMigration
{
    public function getDescription() : string
    {
        return '';
    }

    public function up(Schema $schema) : void
    {
        // this up() migration is auto-generated, please modify it to your needs
        $this->abortIf($this->connection->getDatabasePlatform()->getName() !== 'mysql',
                                                'Migration can only be executed safely on \'mysql\'.');

        $this->addSql('CREATE TABLE person (
                id INT AUTO_INCREMENT NOT NULL,
                first_name VARCHAR(50) DEFAULT NULL,
                last_name LONGTEXT DEFAULT NULL,
                age INT NOT NULL,
                mail_address VARCHAR(100) NOT NULL,
                PRIMARY KEY(id)) DEFAULT CHARACTER SET utf8mb4 COLLATE `utf8mb4_unicode_ci` ENGINE = InnoDB');
    }

    public function down(Schema $schema) : void
    {
        // this down() migration is auto-generated, please modify it to your needs
        $this->abortIf($this->connection->getDatabasePlatform()->getName() !== 'mysql',
            'Migration can only be executed safely on \'mysql\'.');

        $this->addSql('DROP TABLE person');
    }
}
```

Utilisation de doctrine
Migrer les migrations

- Exécution de la commande effectuant la migration de la classe vers une table de la base de données :

php bin/console doctrine:migrations:migrate

```
WARNING! You are about to execute a database migration that could result in sche
ma changes and data loss. Are you sure you wish to continue? (y/n)y
Migrating up to 20191120193836 from 0

  ++ migrating 20191120193836

     -> CREATE TABLE person (id INT AUTO_INCREMENT NOT NULL, first_name VARCHAR(
50) DEFAULT NULL, last_name LONGTEXT DEFAULT NULL, age INT NOT NULL, mail_addres
s VARCHAR(100) NOT NULL, PRIMARY KEY(id)) DEFAULT CHARACTER SET utf8mb4 COLLATE
 `utf8mb4_unicode_ci` ENGINE = InnoDB

  ++ migrated (took 1225.4ms, used 14M memory)

  ------------------------

  ++ finished in 1326.9ms
  ++ used 14M memory
  ++ 1 migrations executed
  ++ 1 sql queries
```