Développement web côté serveur avec Symfony

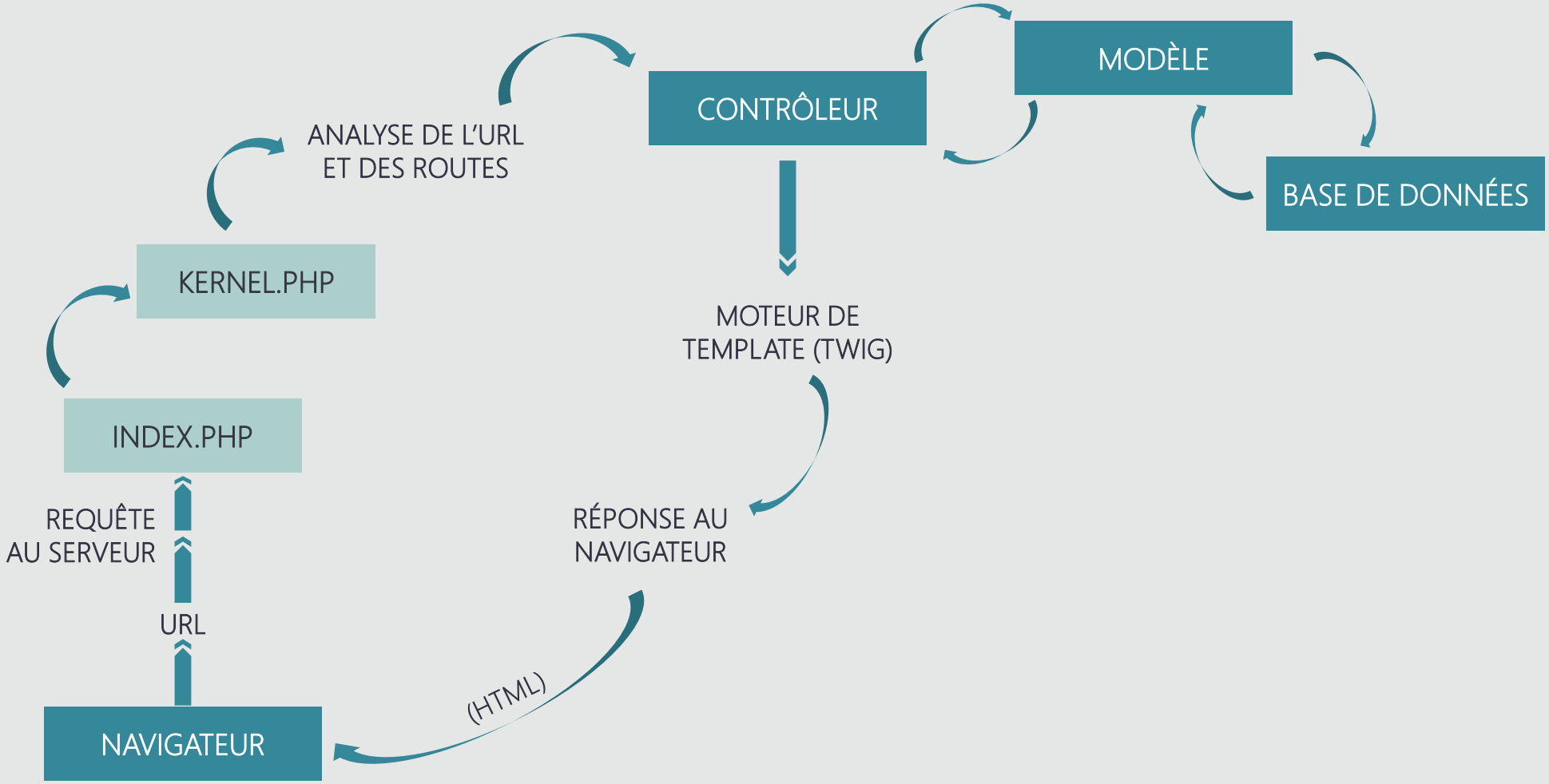
Module 05 - Données et Doctrine

Objectifs

- Comprendre l'utilité d'un ORM
- Savoir créer, modifier, sauvegarder et supprimer des données
- Savoir récupérer des données

Données et Doctrine

Le parcours d'une requête HTTP sous Symfony



Données et Doctrine

Le problème avec les données

- Codes répétitifs
- Les tableaux sont peu fiables
- Différents SGBD

Données et Doctrine

La solution

- Utiliser des objets
- Automatiser les tâches répétitives
- Abstraire la communication avec le SGBD

Configuration de Doctrine

- Configuration :
  - `/.env`
  - `/.env.local`
  - `/config/packages/doctrime.yaml`

1. Paramètres de base de données dans `.env.local`
2. `php bin/console doctrine:database:create`
3. Modifier l'interclassement au besoin dans phpMyAdmin

Données et Doctrine

Les entités

- Une classe PHP représentant les données
- Une classe -> une table
- Une propriété de classe -> champ d'une table

- On indique à Doctrine quelles propriétés doivent être sauvegardées et de quelle manière

Données et Doctrine

Les entités

Une classe PHP

```php
class Book
{
    private $title;
    private $author;
    private $pages;
    private $datePublished;

    public function getTitle()
    {
        return $this->title;
    }

    public function setTitle($title): void
    {
        $this->title = $title;
    }

    public function getPages()
    {
        return $this->pages;
    }

    // autres getters et setters
```

Données et Doctrine

Une entité Doctrine

```php
<?php

namespace App\Entity;

use App\Repository\BookRepository;
use Doctrine\ORM\Mapping as ORM;

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

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

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

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

Données et Doctrine

Générer des entités

1. Créer l'entité avec `php bin/console make:entity`
2. Ajouter et configurer les champs de la classe

La même commande permet également de *modifier* une entité !

Données et Doctrine

L'EntityManager de Doctrine

- EntityManager : gestionnaire d'entités
- Pour les C, U et D du CRUD
- 2 méthodes pour le récupérer depuis un contrôleur :
  - $this->getDoctrine()->getManager()
  - En utilisant l'injection de dépendances
- Un EntityManager pour toutes les entités

Données et Doctrine

Récupérer l'EntityManager

```php
//...
use Doctrine\ORM\EntityManagerInterface;

class MainController extends AbstractController
{
    /**
     * @Route("/", name="main_home")
     */
    public function show(EntityManagerInterface $entityManager)
    {

    }
```

```php
/**
 * @Route("/", name="main_home")
 */
public function show()
{
    $entityManager = $this->getDoctrine()->getManager();
}
```

Les méthodes principales de l'EntityManager

- `$em->persist($objet)`
- `$em->remove($objet)`
- `$em->flush()`
- `$em->createQuery($dql)`
- `$em->getRepository($className)`

Démonstration

Les Repository de Doctrine

- Repository : dépôt
- Pour le R du CRUD
- 3 méthodes pour le récupérer depuis un contrôleur :
  - En utilisant l'injection de dépendances
  - $this->getDoctrine()->getRepository(Objet::class)
  - $em->getRepository(Objet::class)
- Un Repository par entité

Les méthodes principales du Repository

- `$repo->findAll()`
- `$repo->find($id)`
- `$repo->findOneByTitle($title)`
- `$repo->findOneBy(["name" => "lorem"])`
- `$repo->findBy([], ["price" => "DESC"], 30, 0)`
- `$repo->count()`

Démonstration

Les requêtes personnalisées

- Les Repository sont modifiables
- On y ajoute les requêtes complexes
- 2 "langages" possibles :
  - DQL
  - QueryBuilder
- Une méthode par requête

- *Les requêtes se font aux classes PHP et non aux tables !*

Données et Doctrine

Le DQL

- Doctrine Query Language
- Très inspiré du SQL

Données et Doctrine

Un exemple de DQL

```php
// dans un ProductRepository

public function findWellRatedProducts()
{
    $entityManager = $this->getEntityManager();
    $dql = "SELECT p FROM App\Entity\Product p
            WHERE p.rating > 8
            AND p.active = true";
    $query = $entityManager->createQuery($dql);
    $query->setMaxResults(20);
    return $query->getResult();
}
```

Données et Doctrine

Le QueryBuilder

- Classe de Doctrine
- Génère du DQL grâce à un enchaînement d'appels à des méthodes

Données et Doctrine

Un exemple de QueryBuilder

```php
public function findWellRatedProducts()
{
    $queryBuilder = $this->createQueryBuilder('p');
    $queryBuilder
        ->andWhere('p.rating > 8')
        ->andWhere('p.active = true');
    $queryBuilder->setMaxResults(20);
    $query = $queryBuilder->getQuery();
    return $query->getResult();
}
```

```php
public function findWellRatedProducts()
{
    $entityManager = $this->getEntityManager();
    $dql = "SELECT p FROM App\Entity\Product p
            WHERE p.rating > 8
            AND p.active = true";
    $query = $entityManager->createQuery($dql);
    $query->setMaxResults(20);
    return $query->getResult();
}
```

Données et Doctrine

Requêtes dynamiques

```php
public function findWellRatedProducts($minRating = null)
{
    $queryBuilder = $this->createQueryBuilder('p');
    $queryBuilder->andWhere('p.active = true');

    if ($minRating){
        $queryBuilder
            ->andWhere('p.rating > minRating')
            ->setParameter('minRating', $minRating);
    }

    $queryBuilder->setMaxResults(20);
    $query = $queryBuilder->getQuery();
    return $query->getResult();
}
```

```php
public function findWellRatedProducts($minRating = null)
{
    $entityManager = $this->getEntityManager();
    $dql = "SELECT p FROM App\Entity\Product p
            WHERE p.active = true";
    if ($minRating){
        $dql .= " AND p.rating > :minRating";
    }
    $query = $entityManager->createQuery($dql);

    if ($minRating){
        $query->setParameter("minRating", $minRating);
    }

    $query->setMaxResults(20);
    return $query->getResult();
}
```

Démonstration

Conclusion

- Vous comprenez l'utilité d'un ORM
- Vous savez faire un CRUD sur les données avec Doctrine
- Vous connaissez les bases du DQL et du QueryBuilder pour réaliser des requêtes complexes