Développement web

c’est la partie développement Front-end partie développement back-End
- La partie Front-End : la partie client autrement notre interface (ce qu’il voit le client)
- La partie back-End : la partie serveur où se passent le traitement

Historique et évolution du Web :

Web 1.0 (les années 1990) : sites web statique (vitrine)
Web 2.0 (années 2000) : introduit une interactivité homme machine
Web 3.0 (années 2010, conceptuel) :associé à l'idée d'un Web sémantique
Web 4.0 (conceptuel, vision future) :extension du Web 3.0, avec une utilisation plus poussée de l'IA

Un Framework :

c’est une façon de simplicité, il facilite l’implémentation du code source, accélérer le procès du code source

Environnement de travail :

1- Xamp avec la version de PHP 8.1

2- l’installation du composer : C’est un gestionnaire de dépendance

Environnement de travail :

symfony new FirstPromfony/skeleton:"^5.4" FirstProject

Structure du projet : ject --version=5.4 –webapp

Composer create-project sy

- Bin : dossier qui contient des fichiers exécutable
- config : on trouve la configuration
- Migration : dans ce fichier on peut voir l’historique de la base de données Lechnowa
- Public : il contient les fichier publique da l’application : css , image ,
- Src : le dossier source : code source : on trouve déjà le Controller, entity

Controller

Création d'un controller:

```
php bin/console make:controller
```

Dans notre Controller on va trouver une méthode générer par défaut :
1- Chaque méthode doit retourner une réponse :
2- 80% Chaque méthode doit avoir une route--> si elle va être exécutée par le navigateur
Si on a un problème d'exécution d'une route : il faut lancer la commande
- composer require doctrine/annotations

Lancer le serveur :

```
symfony server:start
```

création BD + Génération ENTITY

1- config dans le fichier .env :

```
DATABASE_URL="mysql://root:@127.0.0.1:3306/3A61"
```

2- php bin/console doctrine:database:create

1- php bin/console make:entity

→ entity/student.php

→ repository/StudentRepository

2- php bin/console make:migration

→ migrations/version20231610092705

3- php bin/console doctrine:migrations:migrate

→ entity/student.php

- Par défaut il a créé l'id, puisque chaque table dans la BD doit avoir une clé primaire
- Par forcément le nom est ID, on peut le changer par exemple REF

#[ORM\Id] spécifie la clé primaire: primary KEY

#[ORM\GeneratedValue] -> auto-incrémentée l'ID

#[ORM\Column] -> colonne

→ repository/StudentRepository

Pour chaque entité, il existe un Repository

C’est une classe PHP qui contient des méthodes prêtes pour la récupération de données

Il existe 3 façons pour récupérer les objets :

- les méthodes de récupération de base : findAll(), findBy(), find($id)
- les méthodes magiques : findByX(), findOneByX()
- les méthodes de récupération personnalisées : DQL/QueryBuilde

Fichier de migration

2- php bin/console make:migration : créer un fichier de migration

Created ou ? Sous le dossier migration

→ migrations/version20231610092705

version20231610092705. **php ==> 2023/16/10 09:27 05 : versions**

3- php bin/console doctrine:migrations:migrate: lancer une migration

--------- Contenu du fichier migration----------

-> Code SQL pour créer la table students

Chaque fichier de migration possède trois méthodes:

- **La méthode getDescription() :permet de décrire la migration**
- **La méthode up(): est exécutée lorsqu'on applique la migration en utilisant la commande php bin/console doctrine:migrations:migrate**
- **La méthode down() :est exécutée lorsqu'on annule la migration en utilisant la commande php bin/console doctrine:migrations:migrate prev**

manipulation du fichier de migration

doctrine:migrations:current	Afficher la version actuelle
doctrine:migrations:execute version	Exécuter une seule version de migration
doctrine:migrations:generate	Créer un fichier de migration vide
doctrine:migrations:latest	Afficher la dernière version migration
doctrine:migrations:migrate	Exécuter toutes les versions de migrations non exécutées
doctrine:migrations:status	Afficher l'état d'un ensemble de migrations
doctrine:migrations:up-to-date	Nous indiquer si le schéma est à jour
doctrine:migrations:version version -- add/delete	Ajouter ou supprimer manuellement les versions de migration de la table des versions.
doctrine:migrations:sync-metadata-storage	Vérifier si le stockage des métadonnées est à jour
doctrine:migrations:list	Afficher la liste de toutes les migrations disponibles et leur état
doctrine:migrations:migrate next	Executer la méthode up de la premiere migration générée et non exécutée (une seule)
doctrine:migrations:migrate prev	Exécuter la méthode down de la dernière migration

La méthode AFFICHE :

1- création du route
2- Injecter le repository dans la méthode
3- Récupérer les données
4- Envoyer les données vers le fichier html

```
#[Route('/fetch', name: 'fetch')]
public function fetch(StudentRepository $repo):Response
{
    $result=$repo->findAll();
    return $this->render('student/test.html.twig',[
        'response' =>$result
    ]);
}
```

La méthode AJOUT :(statique avec les SET)

```
#[Route('/add', name: 'add')]
    public function add(ManagerRegistry $mr):Response{

        $s= new Student ();    // création instance
        $s->setName('samar');  // remplir l'objet $S
        $s->setEmail('samar@gmail.com');
        $s->setAge('20');

        $em=$mr->getManager();
        $em->persist($s);       //preparation
        $em->flush();           // l'exécution
        return $this->redirectToRoute('fetch');    //redirection
    }
```

La méthode REMOVE:(doit être paramétré)

```
#[Route('/remove/{id}', name: 'remove')]

    public function remove($id,ManagerRegistry $mr, StudentRepository $repo)
:Response
    {
        $student=$repo->find($id);    // récupération de l'objet à supprimer

        $em=$mr->getManager();
        $em->remove($student);        //LA suppression
        $em->flush();
        return new Response('removed');
    }
```

La méthode UPDATE :(statique avec les SET)

```
#[Route('/update{id}', name: 'update')]
    public function update(ManagerRegistry $mr,Request
$req,$id,StudentRepository):Response{

        $s= $repo->find($id); ();   // récupération objet
        $form=$this->createForm(StudentType::class,$s);
        $form->handleRequest($req);
        if($form->isSubmitted())
        {
        $em=$mr->getManager();
        $em->persist($s);       //preparation
        $em->flush();           // l'exécution
        return $this->redirectToRoute('fetch');   //redirection
    }
```

return $this->render('student/update.html.twig',[

'f'->$form->createView()

]);

ENTITY CLASSROOM:

Créer l’entité classroom: ref, name , createdAt

* **Changement nom clé primaire :**

1- #[ORM\Column(name:'ref')]

→ dans la base de données ref mais dans l'entité reste ID :

2- private ?int $ref = null; public function getRef(): ?int

php bin/console make:migration

php bin/console doctrine : migrations :migrate

JOINTURE

La jointure dans le code source : c’est une propriété dans la classe

1- php bin/console make :entity Student
2- New property : classroom
3- Type : relation
4- What class related : Classroom
5- ManyToOne
6- YES
7- e.g. $classroom->getStudents : Oui : puisque bidirectional

Dans Student.php

```
    #[ORM\ManyToOne(inversedBy: 'students')]
    private ?Classroom $classroom = null;
```

- (inversedBy: 'students')] : indique le nom de la propriété dans l'entité Classroom

- Dans student ManyToOne

 alors dans classroom il faut qu'on trouve OneToMany

- Et au lieu de inversedBy, on trouve mappedBy

ERROR :

ERREUR: Column name "id" referenced for relation from App\Entity\Student towards App\Entity\Classroom does not exist.

Par défaut la clé primaire dans une entity pour symfony c'est le ID

Le PROB : que n'a pas trouvé ID dans classroom il y'a ref
Il faut l'informer de ce changement :

SOLUTION

```
    #[ORM\ManyToOne(inversedBy: 'students')]
    #[ORM\JoinColumn(name:'nc_id',referencedColumnName:'ref')]
    private ?Classroom $classroom = null;
```

name :indique le nom de la colonne dans la table de la base

referencedColumnName : le name elle fait référence à la colonne ref.

NB : cette erreur si seulement on a changé la clé primaire

FORM :

*** création BD :**

```
1- php bin/console make:form FormName
```

*** L'envoi du formulaire à la page TWIG :**

```
2- return $this->render('formation/.html.twig',
['formA' => $form]);
```

*** Afficher la totalité du formulaire avec la méthode form()**

```
3- {{ form(nomDuFormulaire) }}
```

StudentType.php

```
class StudentType extends AbstractType
{
    public function buildForm(FormBuilderInterface $builder, array
$options): void
    {
        $builder
            ->add('name')
            ->add('email')
            ->add('age')
            ->add('classroom')
            ->add('save',SubmitType::class)   // on a ajouter button SAVE
        ;
    }
```

La méthode AJOUT :(Dynamique avec FORM)

```
#[Route('/addF', name: 'addF')]
public function addF(ManagerRegistry $mr, Request $req):Response{
  $s= new Student ();                    // 1- instance
  $form=$this->createForm(StudentType::class,$s); //2-  création du formulaire
  $form->handleRequest($req);     // 3- analyse request et récupérer les données
      if($form->isSubmitted()){
       $em=$mr->getManager();              //4-persist+flush
       $em->persist($s);
       $em->flush();
       return $this->redirectToRoute('fetch');
      }
      return $this->render('student/add.html.twig',[
       'f'=>$form->createView()
      ]);
```

* La méthode **handleRequest()** de la classe Form permet de récupérer les données saisies dans les inputs du formulaire

** La méthode **isSubmitted()** de la classe Form permet de savoir si on est effectivement en méthode POST

Error : Symfony\Component\Form\FormRenderer::renderBlock(): Argument #1 ($view) must be of type Symfony\Component\Form\FormView, Symfony\Component\Form\Form given, called in

→ L'affichage de formulaire si et seulement si je vais l'appelle de la méthode createView()

```
'f'=>$form->createView()
```

Error :

Object of class App\Entity\Classroom could not be converted to string

→ AJOUTER toString() dans Classroom

La méthode Update:Avec formulaire

```
  #[Route('/update/{id}', name: 'update')]
    public function update($id,StudentRepository $repo, ManagerRegistry $mr, Request
$req):Response
    {
    $s=$repo->find($id) ;//  1 - recuperation
    $form=$this->createForm(StudentType::class,$s) ;                    // 2-
        $form->handleRequest($req);
    if($form->isSubmitted())  {
        $em=$mr->getManager();  //3- persist+flush
        $em->persist($s);
        $em->flush();
       return $this->redirectToRoute('fetch');
    }
    return $this->renderForm('student/update.html.twig',[
        'f'=>$form
    ]);
```

La méthode renderForm est utilisée lorsque vous souhaitez générer un formulaire dans une vue Twig et gérer sa soumission automatiquement

Avec renderForm ; on utilise pas createView()

DQL/QUERYBUILDER

symfony il vous donne la main pour implémenter ou créer une requête personnalisé avec les 2 approches que j'ai citées : DQL et Query Builder

DQL :

- Langage de requêtes pour le modèle objet, et non pour le schéma relationnel.
- SQL utilise des ***noms de table et des noms de colonnes*** dans la requête alors que DQL manipule des ***objets***.
- **DQL** permet d'écrire des requêtes sous forme de chaînes de caractères.

```
#[Route('/dql', name: 'dql')]
public function dqlStudent (-1-EntityManagerInterface $em): Response
{
$req=$em->-2-createQuery("select -5- s from -3- App\Entity\Student -4- s");      //select * from student
-6- $result=$req->getResult();
-7- dd($result);
-8- return $this->render('student/searchStudent.html.twig',[
    "students"=>$result
    ]);
        }
```

-1- EntityManagerInterface : service qui nous permet de donner une instance $em ou on a une méthode qui s'appelle createQuery: qui nous permet de créer une requête DQL

-2- $req=$em->createQuery(''); -> prend une requête sous forme de chaîne de caractère

-3- je vais manipuler la base de donner en tant qu'orienté objet, au lieu de student je vais indiquer le chemin vers notre entity : App\Entity\Student

-4- je vais donner alias : pour faciliter la manipulation

-5- à la place de *, je vais mentionner l'alias s CAD tous les champs

-6- on demande à symfony de donner le résultat de cette requête là

-7- pour voir les informations concernant $result avec dump

-8- afficher la liste des étudiants dans fichier TWIG et après on va faire la recherche par nom par exemple

Je vais implémenter une méthode qui afficher les noms des étudiants ordonnées ascendant

```
    #[Route('/dql3', name: 'joindql')]
    public function dqljoin(EntityManagerInterface $em)
    {
$req=$em->createQuery("select s.name from App\Entity\Student s Order By s.name DESC");
        $result=$req->getResult();
        dd($result);
    }
```

QUERYBUILDER

Je vais sélectionner la liste des étudiants à travers QueryBuilder :

---------- implémenter la méthode dans le repository ----------

```
public function listEtudiantQB() {
  $req=$this->createQueryBuilder('s')  ;
  $preresult=$req->getQuery();
  $result=$preresult->getResult();
  return $result;
}
```

---------- Appeler dans le controlleur ----------

```
#[Route('/qb', name: 'qb')]
    public function qb(StudentRepository $repo){
      $result=$repo->listEtudiantQB();
      dd($result);
    }
```

----------Selectionner des champs ----------

```
public function listEtudiantQB()
{
  $req=$this->createQueryBuilder('s')
  ->select('s.name');
  $preresult=$req->getQuery();
  $result=$preresult->getDQL();
  return $result;
}
```

```
    public function findByAbbe($abbe)
    {
        return $this->createQueryBuilder('s')
            ->andWhere('s.abbe = :abbe')
            ->setParameter('abbe', $abbe)
            ->getQuery()
            ->getResult();
    }
```

find by critere

```
 public function studentsByAbbe(StudentRepository $studentRepository, $abbe):
Response
    {
```

Bonne chance à tous <3

vos Questions ?