Sylvain ALLAIN
Aliou DIEBATE

Projet Symfony 5

Gestion des Sessions & Stagiaires

SOMMAIRE

Introduction

Pour la réalisation de ce projet, nous nous baserons sur le design pattern MVC et sur le framework Symfony5.

Le modèle MVC décrit une manière d'architecturer une application informatique en la décomposant en trois sous-parties :

- La partie Modèle ;
- La partie Vue ;
- La partie Contrôleur ;

La partie **Modèle** rassemble les accès aux données. Il peut s'agir d'un ensemble de fonctions ou de classes.
La partie **Vue** s'occupe des interactions avec l'utilisateur : présentation, saisie et validation des données.
La partie **Contrôleur** gère la sécurité et les conditions de l'application. Elle fait le lien entre l'utilisateur et le reste de l'application.

Rappel des consignes

1- Réaliser le MCD / MLD de l'application

2 - Réaliser l'application de gestion en respectant le design pattern MVC : on devra pouvoir afficher les sessions de formation disponibles, le programme de chaque session (module + catégorie) ainsi que les personnes inscrites à chaque session. On pourra également afficher la liste des stagiaires (détails ainsi que les sessions auxquelles se sont inscrits les stagiaires).

3 - Enfin, une interface permettra d'ajouter des stagiaires et des sessions (on admettra que les modules + catégories ont été saisis en amont) et on donnera la possibilité de programmer les différents modules dans les sessions ajoutées précédemment.

Sylvain ALLAIN
Aliou DIEBATE

I. Réalisation du cahier des charges

1) Brainstorming

Nous avons commencé par lire et noter les consignes du document.

A partir de cela, nous avons réfléchis sur les différentes étapes et tâches à réaliser afin de parvenir à la réalisation de ce projet.

Nous nous sommes accordés sur 4 grandes étapes de développement :

1. Spécification fonctionnelle
2. Spécification technique
3. Réalisation
4. Phase de Tests à chaque étape de développement

Nous avons donc réalisé un diagramme de Gantt afin de pouvoir se projeter dans le temps.

2) Organisation et Projection

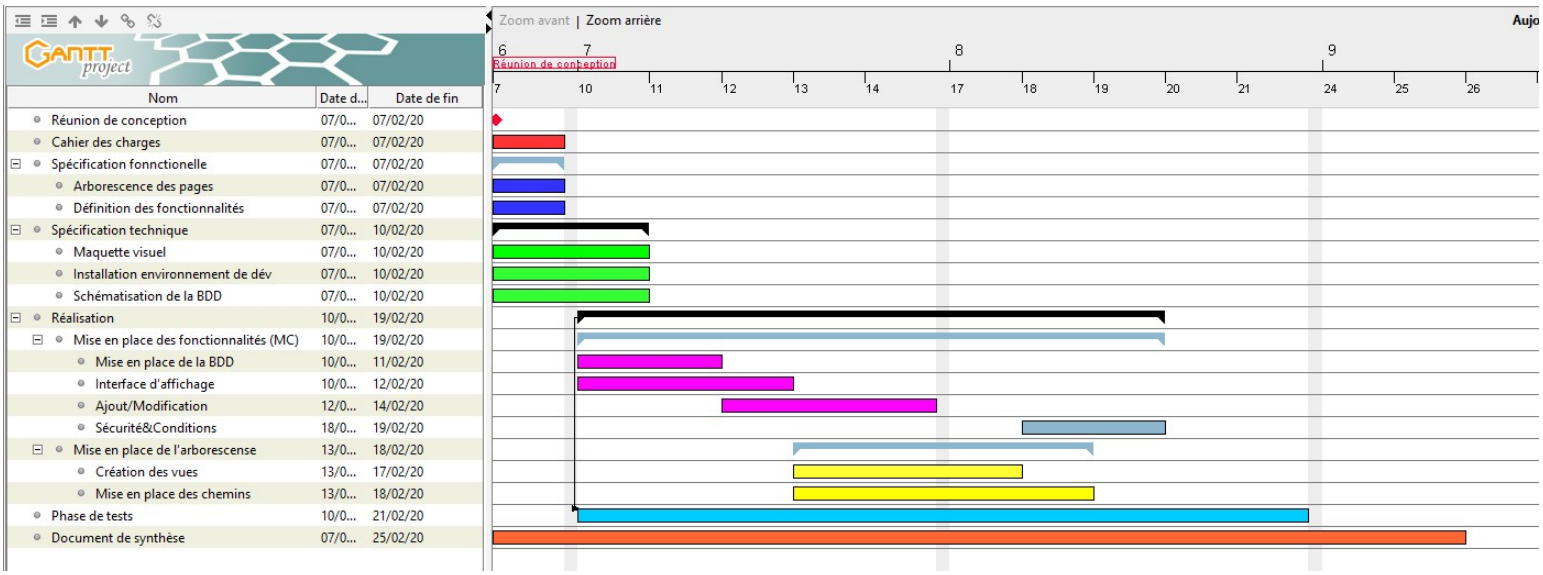


Diagramme de Gantt du Projet

Faire un diagramme de Gantt nous a permis de bien organiser les différentes étapes du projet et de pouvoir se projeter dans le temps.

Cela a été source de discussion, et nous a permis de voir comment chacun se projetait sur le sujet.

Nous avons aussi pu voir quelle tâche pouvait être répartis entre nous (MCD et Maquettage visuelle par exemple).

Nous avons aussi fait le choix de travailler chacun sur un ensemble fonctionnalités :

- Session (Controller, Vue, Forms...) & CRUD associé

- Stagiaire (Controller, Vue, Forms...) & CRUD associé

L'utilisation de Git & Github nous permettra de pouvoir travailler en parallèle sur le même projet, tout en faisant attention à ne pas modifier les fichiers de l'autre afin de ne pas créer de problème de versionning (merge).

II. Spécification fonctionnelle

1) Définition des fonctionnalités

1.1) Accueil

On souhaite avoir une page d'accueil.

Cette page aura un titre, logo Elan et un lien vers la liste des sessions et la liste des stagiaires

Un header avec un lien vers l'accueil sera présent sur chaque page afin de permettre un retour à la cette page depuis n'importe quelle page du site.

1.2) Sessions

a) Liste des sessions

Ici sera listé les sessions existantes ainsi que les informations de temps et de places.

Un bouton mènera vers un formulaire afin de créer une session

Liens :

- Voir une session
- Éditer une session
- Supprimer un session

b) Voir les infos d'une session

Cette page aura 2 tableaux :

- La liste des modules de la session.
- La liste des stagiaires inscrit à cette session. Lien vers infos d'un stagiaire

Possibilité d'ajouter un stagiaire et/ou un module à la session.

c) Ajout de session

Cette page comportera un formulaire avec les infos nécessaire à l'ajout d'une session.

Elle devra remplir certaines conditions comme les champs obligatoire et une date de fin ultérieur à la date de début.

1.3) Stagiaires

a) Liste des stagiaires

Ici sera listé les stagiaires existants dont la date de naissance et le nombre de sessions auquel il est inscrit.

Un bouton permettra de créer un stagiaire.

Liens :

- Voir un stagiaire
- Editer un stagiaire
- Supprimer un stagiaire

b) Voir les infos d'un stagiaire

Ici sera affiché toutes les infos d'un stagiaire

Une liste affichera les sessions auquel il est inscrit. En cliquant sur un intitulé on pourra afficher les infos d'une session

c) Ajout d'un stagiaire

Cette page comportera un formulaire avec les infos nécessaire à l'ajout d'un stagiaire.

Elle devra remplir certaines conditions comme les champs obligatoires par exemple.

2) Arborescence des pages

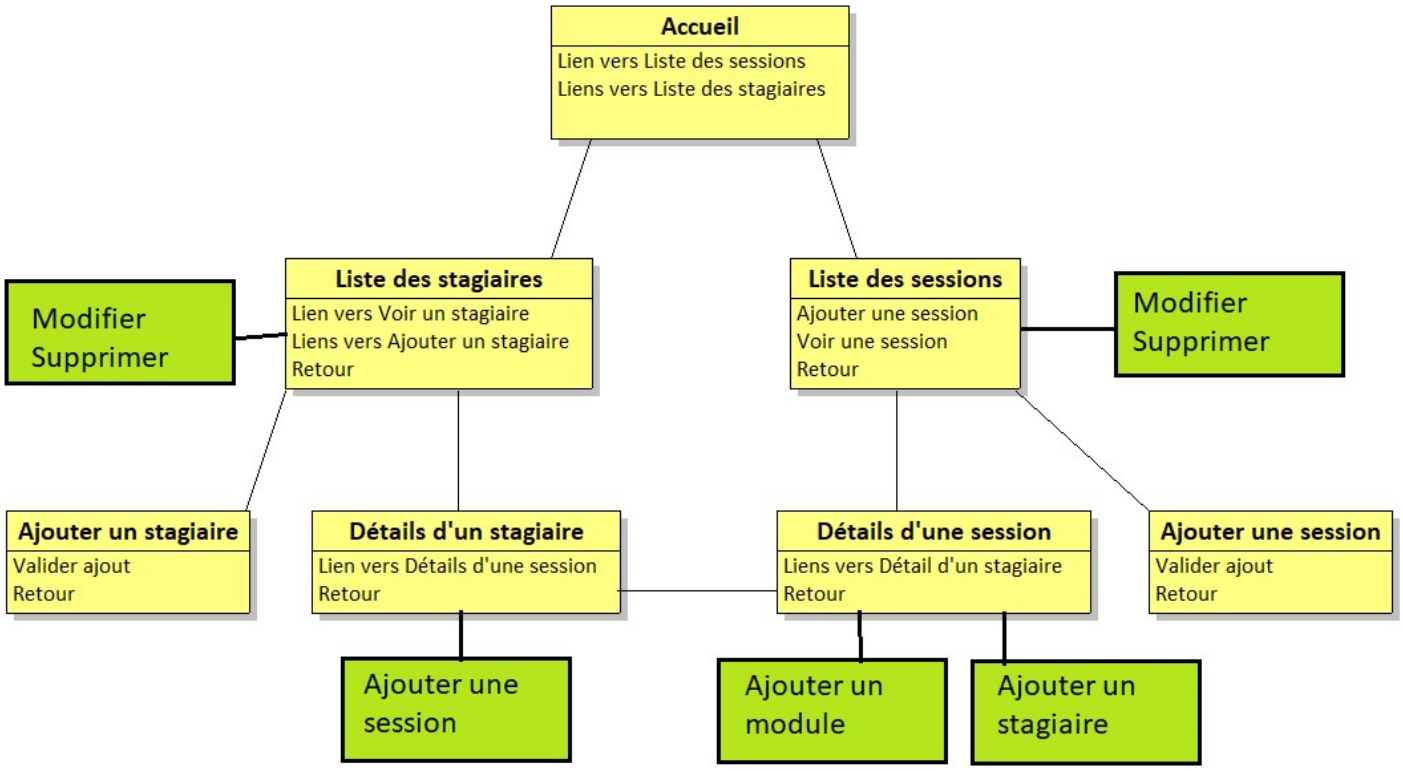


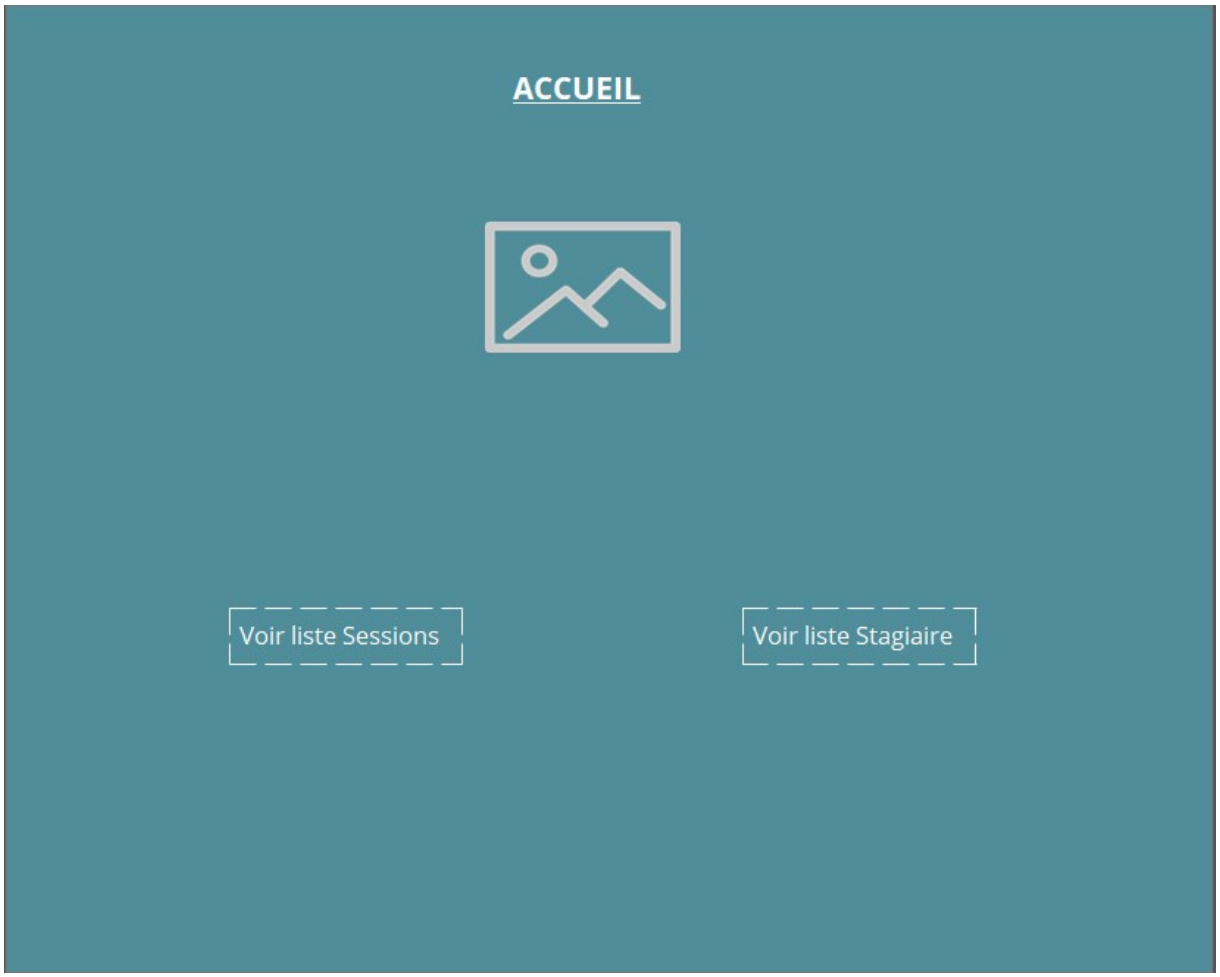
Schéma représentant l'arborescence du Projet

En jaune, les pages prévus à la base du projet.

En vert, les pages qui se sont rajouté en cours de projet.

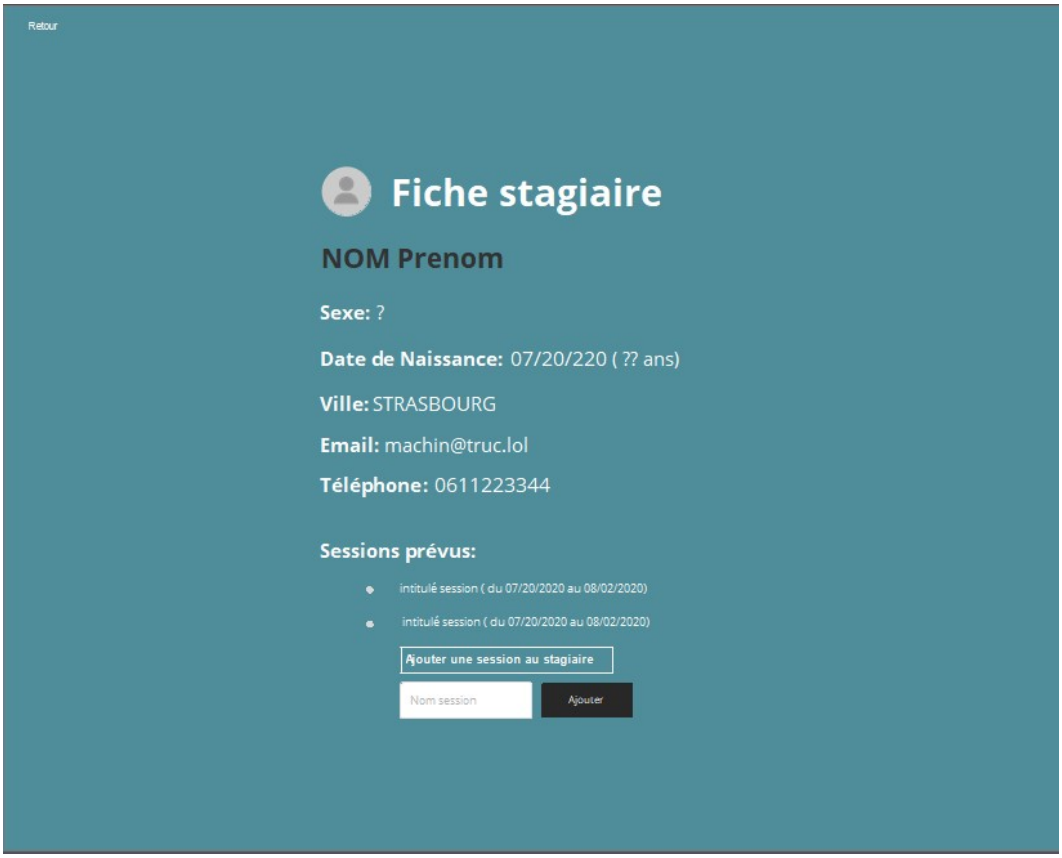
III. Spécification technique

1) Maquettage visuel



Représentation du maquettage visuelle de la page d'accueil

Représentation du maquettage visuelle de la liste des stagiaires



Représentation du maquettage visuelle de la page d'information d'un stagiaire

2) Environnement de développement

Front-end : **HTML5, CSS3 & framework JQuery**

- **HTML5** : Langage de balisage qui sert à l'écriture de texte et d'hypertexte indispensable à la mise en forme d'une page Web

- **CSS3** : Feuille de style en cascade. Permet la mise en forme du langage HTML.

Back-end : **PHP & Symfony5**

- **PHP** : PHP est un langage back-end procédural.

- **Symfony5** : Symfony est un Framework de PHP. Repose sur le pattern design MVC.

- **Twig** : Générateur de template intégré à Symfony

- **Doctrine** : ORM (Objet Relation Mapping) de Symfony, permet de manipuler les données d'une base de données dans des objets (Programmation Orienté Objet)

Outil de Gestion de projet :

- **Github** : Service d'hébergement et de gestion de développement

- **GanttProject :** Outil de planification, diagramme de Gantt

- **JustInMind** : Outils de maquettage visuel

- **Jmerise** : Outil de modélisation du MCD/MLD

- **Visual Studio Code** : Logiciel de développement

- **Laragon** : Outil de serveur local et base de données local.

3) Schématisation de la base de données

3.1) Modèle Logique de Données MLD

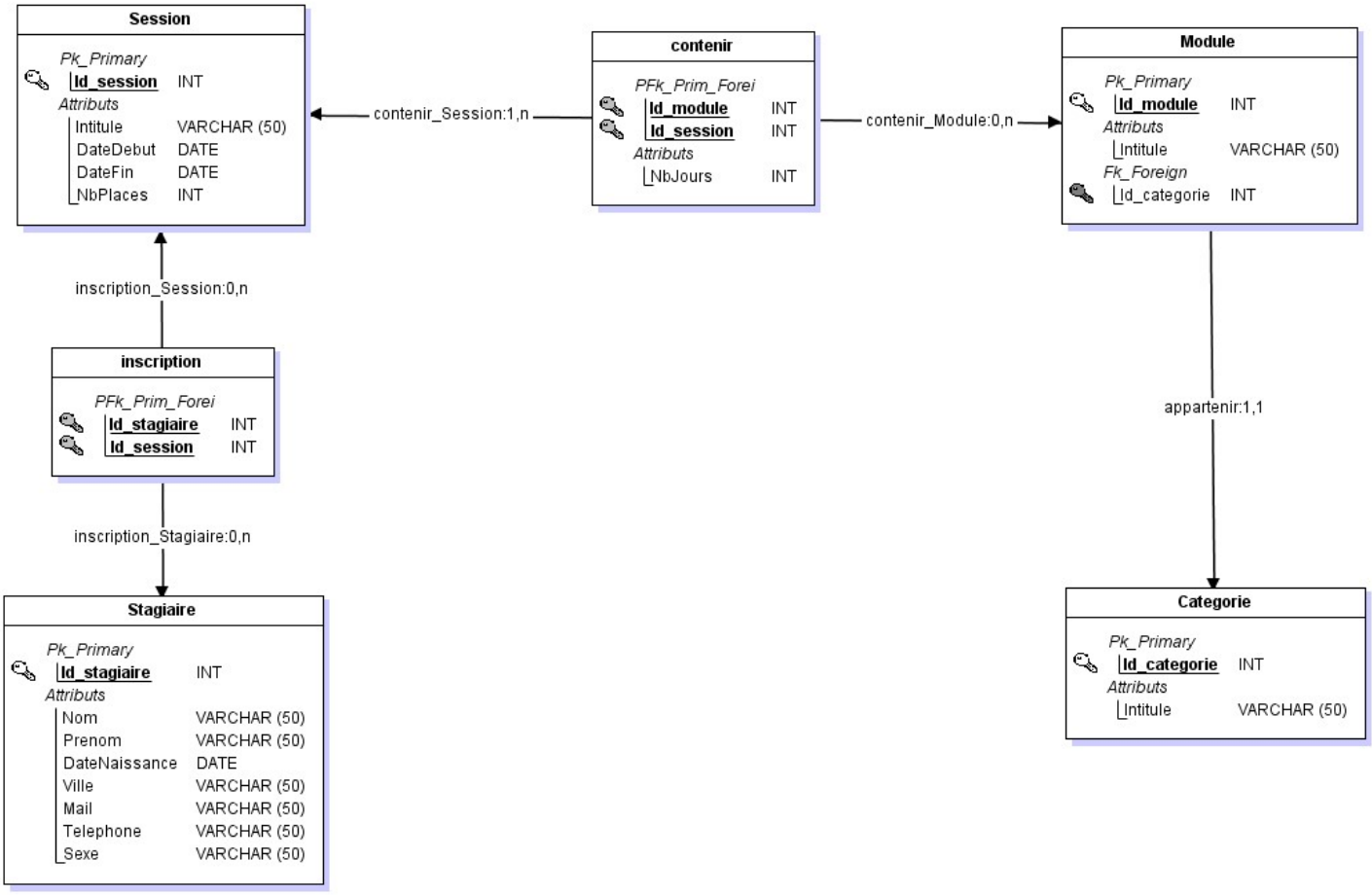


Schéma représentant la base de données

Lors de la réalisation du MLD, nous nous sommes rendus compte de certaines spécificités du projet

- Le fait que le nombres de jours d'une session n'était pas figé et devait donc se situer dans une table associative
- Les catégories allaient être une entité à part.
- Il y a une relation ManyToMany entre Stagiaire et Session.
- Il y a une relation ManyToOne entre Module et Catégorie.
- Il y a une relation ManyToMany entre Session et Module avec un champ intermédiaire « NbJours » dans la table Contenir

3.2) Script SQL

Script SQL du projet

```
CREATE DATABASE IF NOT EXISTS `formation` ;

USE `formation`;

-- Listage de la structure de la table formation. categorie

CREATE TABLE IF NOT EXISTS `categorie` (

  `id` int(11) NOT NULL AUTO_INCREMENT,

  `intitule` varchar(50) COLLATE utf8mb4_unicode_ci NOT NULL,

  `one_to_many` varchar(255) COLLATE utf8mb4_unicode_ci NOT NULL,

  PRIMARY KEY (`id`)

) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_unicode_ci;

-- Listage de la structure de la table formation. contenir

CREATE TABLE IF NOT EXISTS `contenir` (

  `id` int(11) NOT NULL AUTO_INCREMENT,

  `module_id` int(11) DEFAULT NULL,

  `session_id` int(11) NOT NULL,

  `nb_jours` int(11) NOT NULL,

  PRIMARY KEY (`id`),

  KEY `IDX_3C914DFDAFC2B591` (`module_id`),

  KEY `IDX_3C914DFD613FECDF` (`session_id`),

  CONSTRAINT `FK_3C914DFD613FECDF` FOREIGN KEY (`session_id`) REFERENCES `sess
ion` (`id`),

  CONSTRAINT `FK_3C914DFDAFC2B591` FOREIGN KEY (`module_id`) REFERENCES `modul
e` (`id`)

) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_unicode_ci;

-- Listage de la structure de la table formation. migration_versions

CREATE TABLE IF NOT EXISTS `migration_versions` (

  `version` varchar(14) COLLATE utf8mb4_unicode_ci NOT NULL,

  `executed_at` datetime NOT NULL COMMENT '(DC2Type:datetime_immutable)',

  PRIMARY KEY (`version`)

) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_unicode_ci;
```

```
-- Listage de la structure de la table formation. module
CREATE TABLE IF NOT EXISTS `module` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `categorie_id` int(11) NOT NULL,
  `intitule` varchar(255) COLLATE utf8mb4_unicode_ci NOT NULL,
  PRIMARY KEY (`id`),
  KEY `IDX_C242628BCF5E72D` (`categorie_id`),
  CONSTRAINT `FK_C242628BCF5E72D` FOREIGN KEY (`categorie_id`) REFERENCES `cat
egorie` (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_unicode_ci;
-- Listage de la structure de la table formation. session
CREATE TABLE IF NOT EXISTS `session` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `intitule` varchar(255) COLLATE utf8mb4_unicode_ci NOT NULL,
  `date_debut` date NOT NULL,
  `date_fin` date NOT NULL,
  `nb_places` int(11) NOT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_unicode_ci;
-- Listage de la structure de la table formation. stagiaire
CREATE TABLE IF NOT EXISTS `stagiaire` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `nom` varchar(50) COLLATE utf8mb4_unicode_ci NOT NULL,
  `prenom` varchar(50) COLLATE utf8mb4_unicode_ci NOT NULL,
  `date_naissance` date NOT NULL,
  `ville` varchar(255) COLLATE utf8mb4_unicode_ci DEFAULT NULL,
  `mail` varchar(255) COLLATE utf8mb4_unicode_ci NOT NULL,
  `telephone` varchar(255) COLLATE utf8mb4_unicode_ci DEFAULT NULL,
  `sexe` varchar(255) COLLATE utf8mb4_unicode_ci DEFAULT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_unicode_ci;
```

```
-- Listage de la structure de la table formation. stagiaire_session

CREATE TABLE IF NOT EXISTS `stagiaire_session` (

  `stagiaire_id` int(11) NOT NULL,

  `session_id` int(11) NOT NULL,

  PRIMARY KEY (`stagiaire_id`,`session_id`),

  KEY `IDX_D32D02D4BBA93DD6` (`stagiaire_id`),

  KEY `IDX_D32D02D4613FECDF` (`session_id`),

  CONSTRAINT `FK_D32D02D4613FECDF` FOREIGN KEY (`session_id`) REFERENCES `sess
ion` (`id`) ON DELETE CASCADE,

  CONSTRAINT `FK_D32D02D4BBA93DD6` FOREIGN KEY (`stagiaire_id`) REFERENCES `st
agiaire` (`id`) ON DELETE CASCADE

) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_unicode_ci;
```

IV) Réalisation

1) Mise en place des fonctionnalités

1.1) Mise en place de la BDD

À l'aide du framework Symfony, nous avons créé les entités du MLD et les relations (ManyTo Many, OneToMany) qui ont permis de créer notre base de données.

Nous avons remarqué que Symfony avait du mal à gérer les relation ManyToMany avec un champ dans la table associative, comme c'est le cas de la table Contenir entre Session et Module.

Nous avons donc dû faire 2 relations OneToMany :

- De Session vers Contenir
- De Module vers Contenir

1.2) Interface d'affichage

L'application doit permettre l'affichage de la liste des sessions et celle des stagiaires.

Pour cela nous avons commencé par créer les fichiers SessionController et StagiaireController afin de définir les actions et les chemins.

Dans les fichiers StagiaireRepository et SessionsRepository, nous avons effectué une requête DQL (Doctrine Query Langage) qui récupère les données de l'entité concerné.

1.3) Interface d'ajout, modification et de suppression

Les illustrations présentent dans cette partie du document sont celle qui ont été utilisés pour Stagiaire. Les mêmes méthodes ont été utilisés pour Session.

a) Ajout Session/Stagiaire par formulaire

Via le terminal, nous avons créé le formulaire lié à l'entité correspondante permettant l'affichage et l'ajout d'un stagiaire et d'une session en instanciant un objet qui récupère les données du formulaire.

Il crée la donnée et l'ajoute à la BDD (persist & flush)

```
    /**
    * @Route("/new", name="form_stagiaire")
    */
    public function newForm(Request $request){

        //On instancie un nouvel objet Stagiaire
        $newStagiaire = new Stagiaire();
        //On crée le formulaire avec cet objet vide
        $form = $this->createForm(StagiaireFormType::class, $newStagiaire);

        //On récupère les données du formulaire
        $form->handleRequest($request);
        // Si le formulaire est soumis et valide
        if ($form->isSubmitted() && $form->isValid()) {

            $newStagiaire = $form->getData();

            //Grâce à Doctrine, on instancie l'entité, on crée la donnnée (per
sist)
            //et on met à jours la BDD (flush)
            $entityManager = $this->getDoctrine()->getManager();
            $entityManager->persist($newStagiaire);
            $entityManager->flush();
            //On redirige vers la liste des stagiaires avec un message de vali
dation
            $this-
>addFlash("success", "Le stagiaire a bien été ajouté à la liste !");
            return $this->redirectToRoute('stagiaire_index');
```

```
        }

        //On envoie le formulaire vers la vue
        return $this->render('stagiaire/stagiaireForm.html.twig', [
        'stagiaire_form' => $form->createView()
        ]);
    }
```

b) Modification Session/Stagiaire par formulaire

En reprenant le formulaire de création de stagiaire, crée précédemment, on assigne cette fois un objet de l'entité Stagiaire déjà existant.

Cela permet d'afficher le formulaire avec les données correspondante à ce qu'on veut modifier.

Il n'y a plus qu'à actualiser la BDD. (flush)

c) Suppression Session/Stagiaire

```
    /**
    * @Route("/delete/{id}", name="remove_one_stagiaire")
    */
    public function removeOnestagiaire(Stagiaire $stagiaire, EntityManagerInte
rface $entityManager){
        //On appelle la fonction remove propre à Symfony.
        //On lui passe l'objet Stagiaire courant
        //On actualise la BDD
        $entityManager->remove($stagiaire);
        $entityManager->flush();

    //On redirige vers la liste des stagiaires
    return $this->redirectToRoute("stagiaire_index");
 }
```

d) Ajout/Suppression d'un stagiaire à une session et ajout d'une session à un stagiaire

Il y a une petite nuance ici, puisqu'on ne va pas supprimer de stagiaire ou de session mais on vient **supprimer le lien qu'il y a entre un stagiaire et une session**, donc une entrée de la table stagiaire-session

```
    /**
    * @Route("/{id}/removesession/{id_session}", name="remove_one_session_from
_stagiaire")
    */
    public function removeOneSessionFromStagiaire(Stagiaire $stagiaire, Reques
t $request){
        //On récupère l'ID de la session concerné passé en GET
        $id = $request->attributes->get('id_session');
        //Grâce à l'EntityManager de Symfony, on récupère l'objet Session à pa
rtir de son ID
        $entityManager = $this->getDoctrine()->getManager();
        $session = $this->getDoctrine()->getRepository(Session::class)-
>find($id);
        //On supprime l'entrée Session du tableau lié à ce stagiaire
        $stagiaire->removeSession($session);
        $entityManager->flush();

        //On redirige vers la page du stagiaire avec un joli message
        $this->addFlash("success", "La session a bien été supprimé !");
    return $this-
>redirectToRoute("showOne_stagiaire", array('id' => $stagiaire->getId()));
 }
```

2) Mise en place des chemins

En accord avec le maquettage visuel et l'arborescence du projet, nous avons donc mis en place un bouton "Accueil" qui sera présent sur chaque page ainsi que des boutons "Retour" qui permettent une navigation fluide entre les différentes pages du projet.

3) Sécurité et Validation

Durant les phases de tests, nous nous sommes rendus compte qu'il fallait mettre certaines conditions à l'utilisation du site.

Voici les situations rencontrées ainsi que les solutions que nous avons pu y apporter :

- Ne pas pouvoir ajouter un stagiaire à une session pleine
    - Si place restante est égale à 0, on enlève le bouton "Ajouter un stagiaire" dans la.

```
        {% if (session.nbPlaces) - (session.stagiaires|length) > 0 %}
        <a class="submit" href="{{ path('ajout_stagiaire', {'id':session.id})
}}">Ajouter un stagiaire à la session</a>
        {% else %}
        <p style="color:red">La session est pleine</p>
        {% endif %}
```

    - La session n'est pas proposée à la liste des sessions dans « Ajouter une session » à un stagiaire. + La session n'est pas proposée si le stagiaire y es déjà inscrit
        - Pour résoudre cette solution nous avons dû créer notre propre formulaire. Dans le contrôleur, nous filtrons les données qui seront proposé dans la liste déroulante.
        - Ici la condition posée répond à deux contraintes :
            - La session n'est pas proposée si le stagiaire y es déjà inscrit OU si le nombre de place est inférieur ou égale à 0.

```
  foreach($sessions as $key => $session){
          if($stagiaire->getSessions()->contains($session) ||
              $session->getNbPlaces() - count($session->getStagiaires()) <= 0)
              {
                  unset($sessions[$key]);
              }
    }
```

- Date de fin de session après la date de début de session
    - Rajout d'une condition dans l'entité Session
        - Nous avons d'abords tenté de créer des contraintes dans la Query Builder sans y parvenir. Nous avons finalement opté pour cette solution : Mettre la contrainte au niveau de l'entité.
        - Cependant il s'agirais aujourd'hui d'une mauvaise pratique. La contrainte devrait se faire au niveau de la vue et du contrôleur.

```
use Symfony\Component\Validator\Constraints as Assert;
    /**
     *    @Assert\Expression(
     *      "this.getDateDebut() <= this.getDateFin()",
```

```
     *    message="La date de fin ne doit pas être antérieure à la date début"
     *      )
     */
```

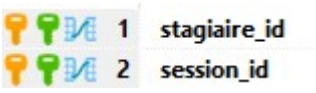
- Ne pas pouvoir ajouter 2 fois le même stagiaire
    - Adresse mail unique
        - Pour rendre un utilisateur unique, nous avons choisis de rendre son adresse mail unique.
        - Car 2 utilisateurs pour avoir le même nom, le même prénom ou les 2.
        - L'idéal serait de rendre unique l'association Nom+Prénom+DateDeNaissance.

```
use Symfony\Bridge\Doctrine\Validator\Constraints\UniqueEntity;

/**
 * @UniqueEntity(fields={"mail"}, message="Cet email est déjà utilisé")
 */
```

- Ne pas pouvoir ajouter 2 fois le même nom de session
    - Rendre unique le champ « Intitulé » dans l'entité Session
    - Reprends le même principe que celui vus ci-dessus avec l'adresse mail.

- Ne pas pouvoir ajouter à une session un stagiaire déjà inscrit
    - Par défaut la relation ManyToMany entre Session et Stagiaire a rendu l'association des deux unique.



Représentation de la table Stagiaire-Session dans la BDD

    - Ne pas le proposer dans la liste déroulante
        - Pour filtrer la liste déroulante nous avons d'abords tenté de passer par le QueryBuilder de Symfony. Nous avons eu quelques difficultés et il s'est avéré plus simple de créer notre propre formulaire qui permettra de filtrer les options de la liste déroulante

- SessionController :

```
     /**
    * @Route("/newStagiaire/{id}", name="ajout_stagiaire")
    */
    public function AjoutStagiaireForm(Session $session, Request $request, Ent
ityManagerInterface $em){


    // Liste déroulante adapté
        $stagiaires = $em->getRepository(Stagiaire::class)->findAll();

        foreach($stagiaires as $key => $stagiaire){
            if($session->getStagiaires()->contains($stagiaire)){
                unset($stagiaires[$key]);
            }
        }

        if($stagiaire_id = $request->request->get('stagiaire')){
            $stagiaire = $em->getRepository(Stagiaire::class)-
>findOneBy(['id' => $stagiaire_id]);

            $session->addStagiaire($stagiaire);

            $em->flush();
            $this->addFlash("success", "Stagiaire bien ajouté !");
            return $this-
>redirectToRoute("show_one_session", array('id' => $session->getId()));
        }

        return $this->render('session/ajoutStagiaireForm.html.twig', [
            'stagiairesDispo' => $stagiaires,
            'session' => $session
        ]);
```

- Au niveau de la vue

```
<form action="{{ path('ajout_stagiaire', {'id':session.id}) }}" method="post">
   <div>
      <select name="stagiaire">
        {% for stagiaire in stagiairesDispo %}
          <option value="{{ stagiaire.id }}">{{ stagiaire.fullName }}</option>
        {% endfor %}
       </select>
       <div>
          <input class="submit" type="submit" value="Ajouter">
       </div>
```

```
    </div>
</form>
```

- Ne pas pouvoir ajouter 2 fois un module à une session
    - Un des exemples les plus intéressants.
    - Ici nous avons dû modifier le schéma de la BDD en utilisant des clés d'index. C'est-à-dire que la relation entre les champs doit être unique dans la table Contenir (entre Session et Module).

```
/**
 * @ORM\Table(name="contenir",
 *              uniqueConstraints={
 *                  @ORM\UniqueConstraint(name="contenir_unique",
 *                      columns={"module_id", "session_id"})
 *              })
 *
 */
```

Représentation de la Index Key (en rouge) dans la BDD HeidiSQL

V) Améliorations possible

1) Plus de sécurités et de contraintes

Certains éléments de sécurité et de validation n'ont pas pu être traités.
En effet il aurait été intéressant que :

- Un stagiaire ne puisse pas être inscrit à 2 session en même temps.

- La durée d'une session en jours (différence entre date de fin et date de début) ne soit pas inférieur au nombre de jours totale des modules.

2) Amélioration esthétique et ergonomique

Par simplicité, les formulaires sont tous sur une page dédiée.

On aurait pu imaginer que certains formulaire simple (l'ajout d'un module à une session par exemple) soit présent sur la page concernée.

3) Ajout de fonctionnalités

A terme, on pourrait imaginer l'ajout de fonctionnalité comme :

- CRUD Modules permettant la création et la gestion de modules.

- CRUD Catégories permettant la création et la gestion de catégories.

- CRUD Formateur permettant la création de formateurs et de gérer la répartition des formateurs sur différents modules auxquelles ils sont formés.

Conclusion

Après le temps nécessaire à la prise en main de Symfony, on peut se rendre compte de la puissance et des possibilités que propose ce framework.

Parfois Symfony donne l'impression qu'une simple condition deviens compliqué à mettre en place.

Cependant nous avons eu parfois des difficultés à trouver les informations pertinentes pour différentes raisons (problème de versions de Symfony, documentations imprécises, pas toujours les mots clé pertinents…).

Le soutiens des formateurs a été une aide précieuse. Sans cet accompagnement, nous n'aurons pas pu résoudre une bonne partie des situations rencontré dans « Sécurité et Validation ».