SF

ADELAIDE DONOVAN
CRETE JONATHAN
BALL PHILLIPE

GITHUB

https://github.com/adelaidedonovan

https://github.com/ball-philippe

https://github.com/CrtJonathan

PROJET PERSONNALISE ENCADRE

PRESENTATION

En classe de troisième, l'élève est amené à vivre une séquence d'observation en milieu professionnel.

D'une durée de 3 à 5 jour consécutive ou non, individuellement ou collectivement, le stage de 3e est obligatoire. Conventionné et non rétribué, il se déroule durant l'année hors vacances scolaires. Ce stage donne à l'élève l'occasion de découvrir le monde économique et professionnel, de se confronter aux réalités concrètes du travail et de préciser son projet d'orientation

CONTEXTE INITIAL

Le principal du collège "Les Capucins à Melun", M. Fortin, est à l'origine de la demande (janvier/février 2018).

Il souhaite faciliter l'accès à la liste des entreprises locales qui accueillent, ou qui ont accueilli dans le passé, des élèves de 3eme.

Pour cela il a sollicité 2 étudiantes de BTS SIO, Angélique Cottencin et Salomé Thomas, dans le cadre de leur stage de seconde année SLAM, afin de produire une solution logiciel métier, de type application web.

ADELAIDE DONOVAN
CRETE JONATHAN
BALL PHILLIPE

LA MISSION

Dans son état actuel, hormis un bug sur l'interface d'accueil (révélé par Firefox), l'application répond bien à la demande initiale, mais, comme toute application, elle est destinée à évoluer (correction de bug, évolution fonctionnelle, optimisation,).

Afin de préparer le terrain des futures évolutions, un refactoring s'impose.

L'architecture actuelle a été conçue en PHP, sans Framework qui aurait permis d'intégrer un cadre d'expertise technique fort.

Votre mission consiste à **démarrer le portage de la solution actuelle et plus encore vers Symfony**.

FORMALITE DE LIVRAISON

- Date limite : Vendredi 30 mars 2018 - 23h59
  REPOUSSÉE à **Lundi 02 mars 2018 - 23h59**
- Rapport envoyé par mail : ocapuozzo@gmail.com (qui centralise)
- Format du rapport : **stage3app- nom(s)-auteur(s).zip**
  (pas plus de 2 par groupe, avec explications sur la répartition du travail réalisé)
- Hébergement du projet : github.com (ou autre en mode public)

SF

ADELAIDE DONOVAN
CRETE JONATHAN
BALL PHILLIPE

CE QUI EST ATTENDU

Une **solution opérationnelle**, fonctionnant sous symfony4 et MysSql, (dans la limite de la couverture fonctionnelle définie ci-dessus) ainsi que la production d'un rapport constitué en deux parties : Analyse et Implémentation

- Rapport d'analyse
  - Diagramme UML d'analyse des entités du domaine
  - Schéma Relationnel (diagramme de préférence)
  - Diagramme complet des cas d'utilisation. Les cas d'utilisation **implémentés** seront mis en valeur par une couleur de fond spécifique.
- Rapport d'implémentation
  - Schéma organisationnel de la structure du projet (arborescence des dossiers)
  - Extraits d'implémentation de cas d'utilisation, contenant des illustrations : copies écran IHM (UI), extrait code source, …

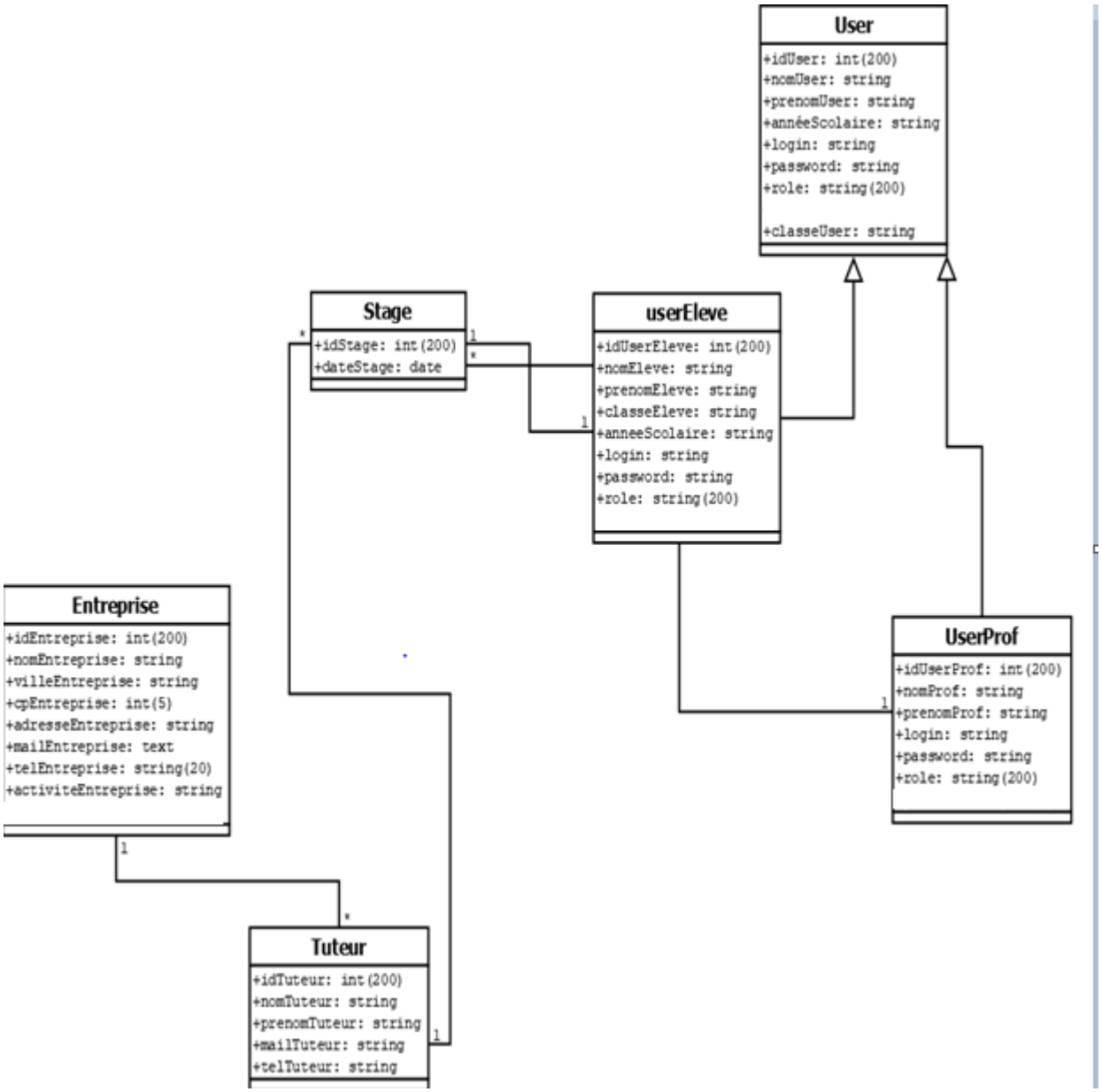
Choix\contraintes techniques :

- Framework coté serveur : Symfony 4 (Php)
- Template HTML 5up : GoodMoovie
- Utilisation du logiciel Laragon ou phpMyAdmin pour administrer une base de données mySql.
- Respect de la norme W3C
- Apprentissage du langage de template twig
- Hébergement du projet : github.com (ou autre en mode public)

SF

ADELAIDE DONOVAN
CRETE JONATHAN
BALL PHILLIPE

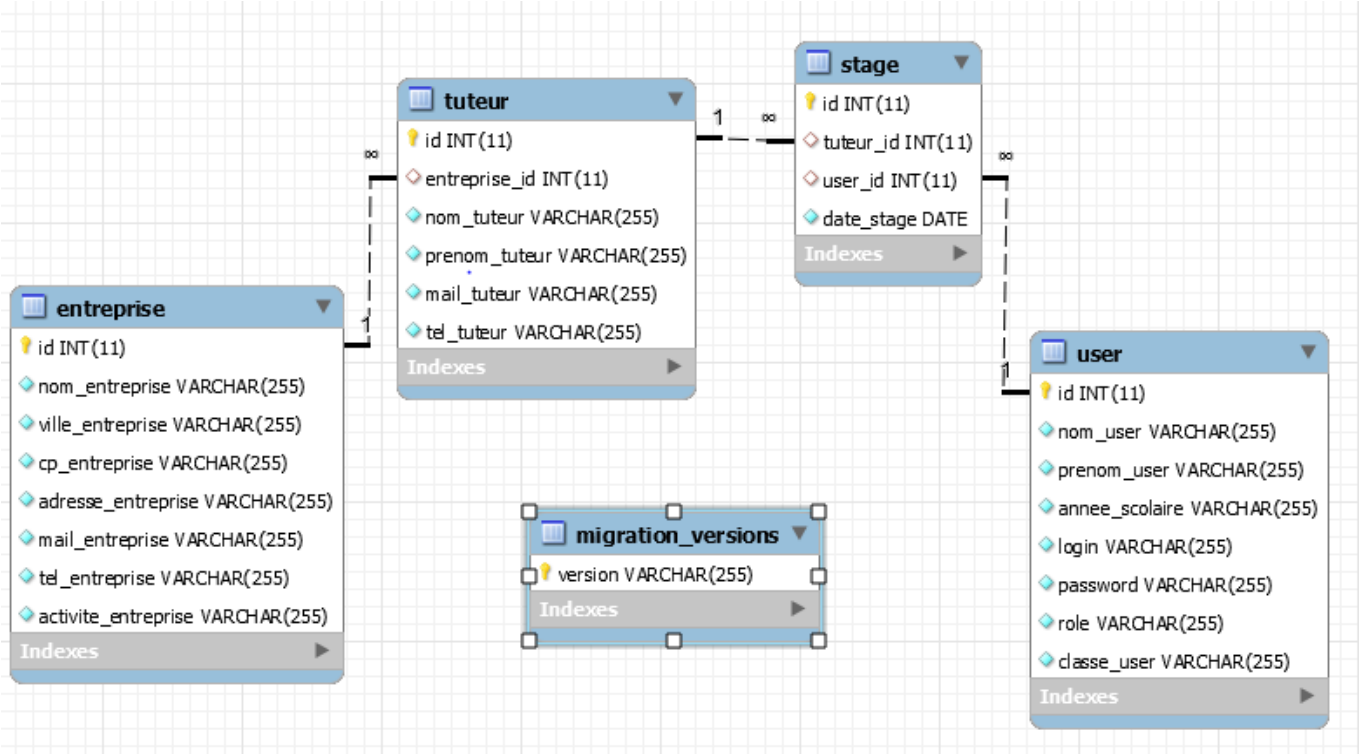
DIAGRAMME UML D'ANALYSE DES ENTITES DU DOMAINE



SF

ADELAIDE DONOVAN
CRETE JONATHAN
BALL PHILLIPE

SCHEMA RELATIONNEL

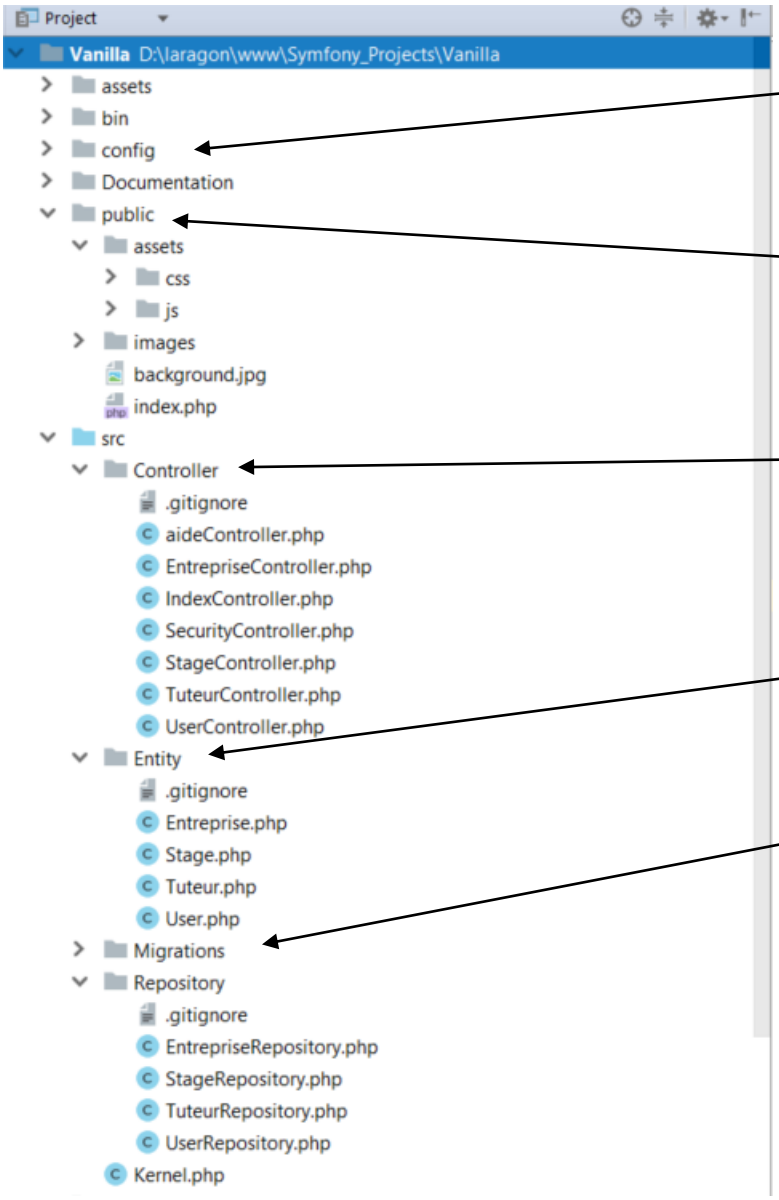


Ici nous trouvons le schéma relationnel du projet SYMFONY notons que la table « migration_versions » est automatiquement générer lors de la création de la base de données avec doctrine elle contient le numéro des versions de fichier de migration sous SYMFONY.

SF

ADELAIDE DONOVAN
CRETE JONATHAN
BALL PHILLIPE

ARBORESCENCE DES DOSSIERS



Fichiers de configuration

Composants web en accès public (css/js/images)

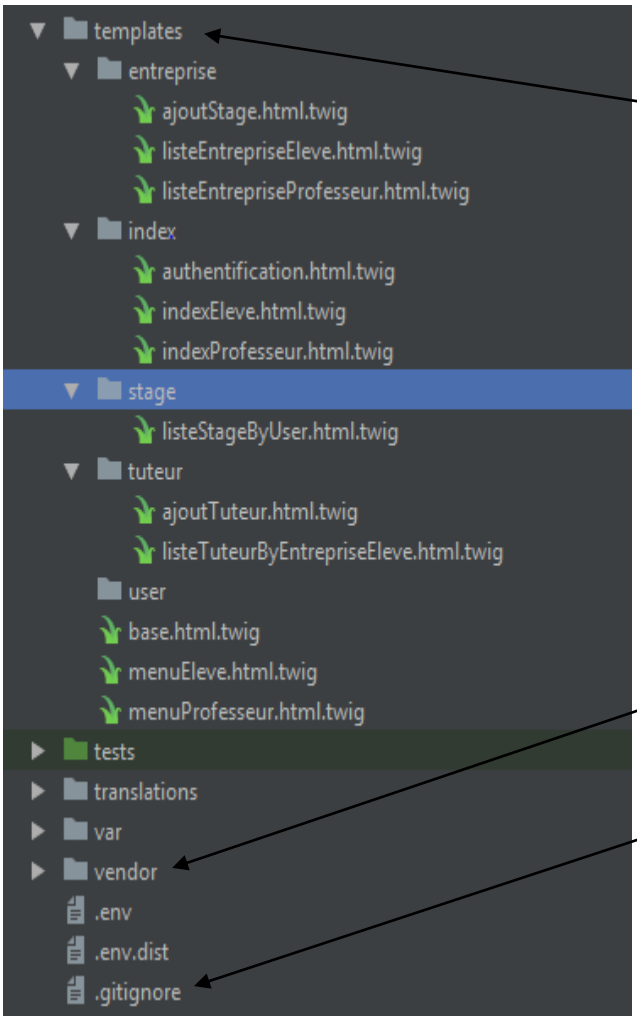
Répertoire contenant les différents contrôleurs

Répertoire contenant les différentes entités

Répertoire contenant les différents fichiers de migrations BDD

SF

ADELAIDE DONOVAN
CRETE JONATHAN
BALL PHILLIPE



Répertoire contenant les différentes vues :

- templates/entreprise : Vues associées au méthode d'action du Controller Entreprise
- templates/index : Vues dites communes tant aux élèves que aux professeurs
- templates/stage : Vues associées au méthode d'action du Controller Stage
- templates/tuteur : Vues associées au méthode d'action du Controller Tuteur
- templates : Vues contenant la structure et les menue des autres vue du templates

/vendor : Répertoire contenant les composants tiers

.gitignore : Fichier énumérant les fichiers qui doivent être ignorés par Git lors d'un commit

SF

ADELAIDE DONOVAN
CRETE JONATHAN
BALL PHILLIPE

EXTRAIT D'IMPLEMENTATION

Voici un extrait d'une entité crée dans le cadre de notre mission en l'occurrence l'entité Entreprise

```php
class Entreprise
{
    /**
     * @ORM\Id()
     * @ORM\GeneratedValue()
     * @ORM\Column(type="integer")
     */
    private $id;

    public function getId()
    {
        return $this->id;
    }


    /**
     * @ORM\Column(type="string")
     */
    private $nomEntreprise;

    /**
     * @ORM\Column(type="string")
     */
    private $villeEntreprise;
```

Notre Entité « Entreprise » contient des attributs et des méthodes permettant de récupérer ou de modifier « get & set » la valeur de ses attributs lors d'une création d'une entreprise.
Nos entités vont nous servir de modèles pour générer nos table dans notre base de donnée (phpMyAdmin, Laragon, etc.…) via la commande « php bin/console migrations:migrate »
Notons que nous devons d'abord crée un fichier de migration via la commande « php bin/console migrations : diff)
Pourquoi un fichier de migration ? cela nous permet de revenir à une version antérieure de la base de donnée si besoin

SF

ADELAIDE DONOVAN
CRETE JONATHAN
BALL PHILLIPE

```php
class EntrepriseController extends Controller
{



    /**
     * @Route("/listeEntrepriseEleve", name="listeEntrepriseEleve")
     */
    public function listeEntrepriseEleve()
    {
        $listeEntreprise = $this->getDoctrine()
            ->getRepository( persistentObject: Entreprise::class)
            ->findAll();
        return $this->render( view: 'entreprise/listeEntrepriseEleve.html.twig', compact( varname: 'listeEntreprise'));
```

Nous trouvons ci-dessus un de Controller que nous avons crée via la ligne de commande «bin/console make:controller Entreprise » le Controller Entreprise contient plusieurs méthode renvoyant vers plusieurs vue (TWIG), le Controller Entreprise est associées a l'entité Entreprise, toute les méthode utilisant cet entité sont dans le Controller Entreprise.

Les méthode dites d'action que nous avons développée utilisent doctrine. Doctrine est un ORM (object-relational mapping) visant à ce que l'utilisateur puisse manipuler ses tables de données en ayant l'illusion d'utiliser des objets.

la méthode « listeEntrepriseEleve » interroge et récupère tout les Object de l'entité Entreprise, ensuite fait Appel la méthode héritée (render) en lui passant un tableau d'entreprise « liteEntreprise » dans le nom de la vue 'listeEntreprise.html.twig'

SF

ADELAIDE DONOVAN
CRETE JONATHAN
BALL PHILLIPE

```php
/**
 * @ORM\OneToMany(targetEntity="App\Entity\Tuteur", mappedBy="entreprise")
 */
private $tuteurs;

public function __construct()
{
    $this->tuteurs = new ArrayCollection();
}
/**
 * @return Collection|Tuteur[]
 */

public function getTuteurs()
{
    return $this->tuteurs;
}
```

Une relation bidirectionnelle a deux extrémités : le côté propriétaire (partie active de la Relation) et le côté inverse qui le référence.

Dans notre exemple ci-dessus nous avons créé une liaison dite « bidirectionnelle » (navigable dans les 2 sens) entre entité afin de manipuler différentes données via doctrine).

La liaison « OneToMany » référence son Côté propriétaire par l'usage de l'attribut mappedBy.

Afin qu'elle soit effective, nous avons créé dans l'entité Tuteur la liaison « ManyToOne » elle est le coté propriétaire de cette liaison bidirectionnelle.

Le côté propriétaire d'une relation détermine les opérations de mises à jour dans la base de Données.

SF

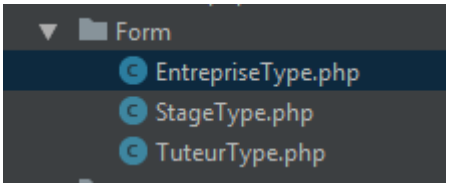
ADELAIDE DONOVAN
CRETE JONATHAN
BALL PHILLIPE

```php
class EntrepriseType extends AbstractType
{
    public function buildForm(FormBuilderInterface $builder, array $options)
    {
        $builder
            ->add( child: 'nomEntreprise')
            ->add( child: 'villeEntreprise')
            ->add( child: 'cpEntreprise')
            ->add( child: 'adresseEntreprise')
            ->add( child: 'mailEntreprise')
            ->add( child: 'telEntreprise')
            ->add( child: 'activiteEntreprise')
            ->add( child: 'Enregistrer', type: SubmitType::class);
        ;
    }

    public function configureOptions(OptionsResolver $resolver)
    {
        $resolver->setDefaults([
            'data_class' => Entreprise::class,
        ]);
    }
```

Après avoir créé les entités, les Controller, effectuer les liaisons entre entité, nous avons effectué une des opération CRUD en l'occurrence CREATE sur l'entité entreprise.

Doctrine s'occupe de générer le formulaire qui prend en compte les attribut de l'entité demander ainsi que les liaison faite dans cette entité par la commande « php bin/console make:form ».



Le formulaire est créé dans le dossier Form que crée symfony.

SF

ADELAIDE DONOVAN
CRETE JONATHAN
BALL PHILLIPE

```php
/**
 * @Route("/ajoutEntreprise", name="ajoutEntreprise")
 */
public function ajoutEntreprise(Request $request)
{
    $item = new Entreprise();
    $form=$this->createForm( type: EntrepriseType::class,$item);


    // Par défaut, le formulaire renvoie une demande POST au même contrôleur qui la restitue.
    if ($request->isMethod( method: 'POST')) {

        $form->submit($request->request->get($form->getName()));
        if ($form->isSubmitted() && $form->isValid()) {
            if ($form->isSubmitted() && $form->isValid()) {
                $item = $form->getData();
                $em = $this->getDoctrine()->getManager();
                $em->persist($item);
                $em->flush();
                return $this->redirectToRoute( route: 'listeEntrepriseEleve');
            }
            return $this->redirectToRoute( route: 'listeEntrepriseEleve');
        }
    }
    return $this->render( view: 'entreprise/ajoutStage.html.twig', array(
        'form' => $form->createView(),
    ));
}
```

Dans notre méthode ajoutEntreprise dans le Controller entreprise nous avons créé une nouvelle Object Entreprise vide, puis crée le formulaire et générer les attributs de l'entité que l'on va utiliser pour notre formulaire avec doctrine dans le formulaire EntrepriseType
Notre formulaire a présent crée est rendu sur la vue « ajoutStage.html.twig » dans le dossier entreprise

```
{{ form_start(form) }}
{{ form_widget(form) }}
{{ form_end(form) }}
```

Apres avoir récupérer les donnée de l'utilisateur notre méthode sauvegarde ses données dans la table lié a notre entité (Table Entreprise) avec « flush » et redirige l'utilisateur vers la route « listeEntrepriseEleve » (affichage de la liste des entreprise pour les élève dans la vue listeEntrepriseEleve.twig.hml)

```
<table class="table table-hover table-warning">
    <thead>
    <tr>
        <th>nom Entreprise</th>
        <th>Ville</th>
        <th>Code Postal</th>
        <th>Adresse</th>
        <th>email</th>
        <th>Téléphone</th>
        <th>Activité</th>
        <th>liste des tuteurs</th>


    </tr>
    </thead>
    <tbody>
    {% for i in listeEntreprise %}
        <tr>
            <td>{{  i.nomEntreprise }}</td>
            <td>{{  i.villeEntreprise}}</td>
            <td>{{  i.cpEntreprise }}</td>
            <td>{{  i.adresseEntreprise }}</td>
            <td>{{  i.mailEntreprise }}</td>
            <td>{{  i.telEntreprise }}</td>
            <td>{{  i.activiteEntreprise }}</td>
            <td><a href="{{ path('listeTuteurByEntreprise',

        </tr>
    {% endfor %}
```

Pour finir nous trouverons un exemple de vue(TWIG) renvoyer par la méthode listeEntrepriseEleve du Controller Entreprise précédemment vue, notre vue récupère le tableau envoyer par la méthode hérité Render par-delà sa référence « listeEntreprise » notons la syntaxe de la lecture du tableau « {% %} » elle permettent de définir un bloc ou l'on vas utiliser dans le cas présent la boucle for afin de parcourir notre tableau « listeEntreprise ».

ADELAIDE DONOVAN
CRETE JONATHAN
BALL PHILLIPE

FUTURES EVOLUTIONS DU PROJET SYMFONY

- Ajout future de l'application web sur un service d'hébergement.
- Opération CRUD (CREATE/READ/UPDATE/DELETE)
- Systèmes de session
- Fonctionnalité annexes (Google Maps)