

Compte rendu projet tutoré

Présentation générale	1
Modèle de données	4
Structure de l'application	6
Utilisation de l'application	15
Perspective d'évolution et erreurs	23
Autres aspects techniques	24

1)Présentation générale

Au cours de cette troisième année à ISIS, nous devons réaliser un projet tutoré où nous avons le choix des sujets. Ainsi, nous avons décidé de travailler sur une application web permettant la gestion des stages à ISIS. Notre objectif était de produire un logiciel le plus complet possible afin que l'école puisse l'utiliser les prochaines années.

Durant le premier semestre, nous avons donc commencé par des recherches sur les blockchains (cf. rapport du 1er semestre). Ensuite, à partir du second semestre, nous avons commencé notre application.

Le maître d'ouvrage nous a imposé quelques fonctions que nous devons obligatoirement programmer dans notre application : Tout d'abord, notre application doit afficher la liste des entreprises ayant déjà accueilli des stagiaires ISIS et les sujets précédents afin de faciliter la recherche de stage. De plus, elle doit permettre la saisie d'offres de stage, la signature de conventions de stage et la planification des soutenances. Nous n'avions aucune base des années précédentes, nous partions de 0.

Notre application a plusieurs cibles bien précises :

- La cible principale : **Les étudiants**. Elle doit leur permettre de trouver plus facilement un stage.
- **Les entreprises** : L'application doit leur permettre de proposer des offres de stages afin de trouver des stagiaires plus rapidement.
- **L'administration de ISIS** : Pour faire l'intermédiaire entre les entreprises et les étudiants et faciliter les démarches pour les stages (signature des papiers sur l'applications, remplir les conventions, etc)

Pour le développement, nous avons utilisé différents logiciels tels que MagicDraw pour le modèle de données et Netbeans ou IntelliJ IDEA pour l'implémentation du code de l'application. Pour mettre en commun le code source, nous avons utilisé GitHub.

Nous avons aussi décidé de mettre en place un système de login dans l'application afin que tout le monde puisse se créer un compte. Dans un premier temps, notre idée était de relier notre application avec les comptes universitaires. Cela aurait permis aux étudiants de ne pas créer de compte et d'utiliser leurs identifiants de l'INU Champollion pour se connecter. Malheureusement, l'administration de Champollion n'était pas d'accord pour nous donner l'accès.

Après avoir listé toutes les fonctions que nous voulions programmer dans l'application, nous avons réalisé que ce projet n'était pas faisable en 6 mois. En effet, pour développer une application complète (back-end & front-end fonctionnels), il nous

faudrait une année supplémentaire. Nous nous sommes donc concentrés sur les fonctions que nous pensions les plus essentielles qui sont la recherche de stage et l'archivage des différents stages effectués.

Ainsi, voici les fonctions actuelles de l'application :

Pour les entreprises :

- Créer un compte
- Proposer / supprimer une offre de stage
- Afficher toutes les offres de stages

Pour l'administration :

- Supprimer une offre de stage
- Afficher la liste de tous les étudiants
- Afficher la liste de toutes les entreprises

Pour les étudiants :

- Créer un compte
- Visualiser les différentes offres de stages.
- Obtenir les descriptifs de tous les stages par année
- Déposer des documents sur la plateforme

2)Modèle de données

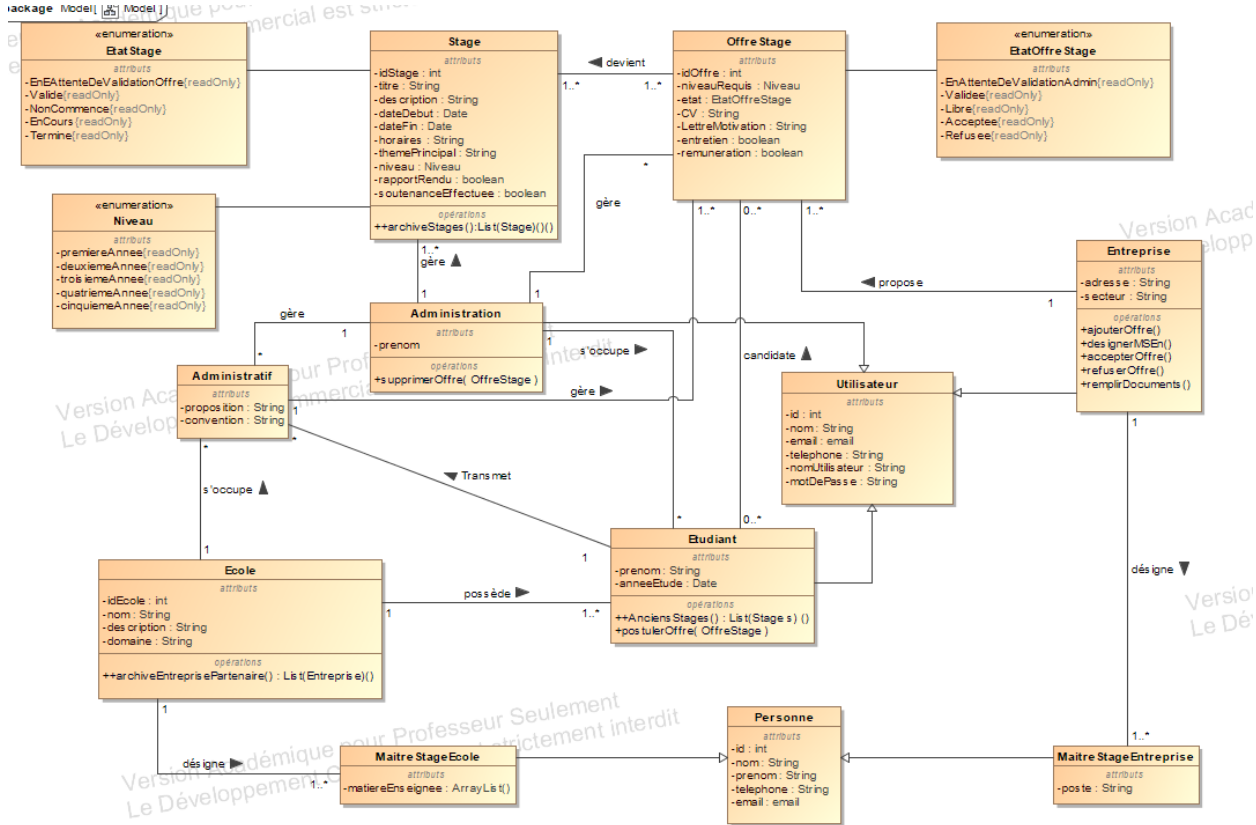


Figure 1 : Modèle conceptuel de donnée de notre application

Notre modèle de données contient 13 tables et celles-ci sont reliées par différentes relations bien précises.

Nous allons vous expliquer les principales relations. Tout d'abord, la relation entre OffreStage et stage est une relation OneToOne. En effet, un stage ne peut être défini que par une seule offre de stage et inversement.

La relation entre Ecole et Etudiant est quant à elle, une relation OneToMany. Un étudiant ne peut appartenir à une seule école mais une école contient plusieurs étudiants.

Ensuite, les tables MaitreStageEcole et MaitreStageEntreprise sont reliées à Personne par une relation d'héritage afin d'en hériter les attributs.

On peut remarquer sur notre MCD que les classes Étudiant et Entreprise sont, quant à elles, reliées à Utilisateur par une relation d'héritage.

On peut ainsi se demander pourquoi nous n'avons pas fait hérité Utilisateur de Personne. C'est ce que nous avons fait au début de notre projet. Malheureusement, ce n'était pas possible car cela allait poser un problème au niveau des entreprises et d'un point de vue sémantique. En effet, les entreprises sont des utilisateurs mais ne sont pas des personnes. Les personnes ont un prénom mais pas les entreprises. Après avoir compris cela nous avons donc dû laisser nos 2 tables utilisateur et personnes.

3) Structure de l'application

Nous allons maintenant expliquer la structure de notre application. Commençons par vous expliquer les contenus de nos entités.

A) Nos entités :

```
@Getter @Setter @NoArgsConstructor @RequiredArgsConstructor @ToString
@Entity
public class OffreStage {
    @Id @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Integer id;

    @NotNull
    private String titre;

    @NotNull
    private String description;

    @NotNull
    private LocalDate dateDebut;

    @NotNull
    private LocalDate dateFin;

    private String horaires;

    // @NotNull
    private String niveauRequis;

    // @NotNull
    private String CV;

    // @NotNull
    private String lettreMotivation;

    // @NotNull
    private EtatOffreStage etatOffre;

    @ManyToMany
    List<Etudiant> candidats = new LinkedList<>();

    @ManyToOne
    Entreprise proposeur;

    @OneToOne
    private Stage stage;
}
```

Voici notre classe OffreStage. Dans celle-ci, nous ne faisons que définir les attributs qui composeront notre offre de stage. De plus, sur l'image à gauche on remarque ligne 1 la présence d'un @Setter et d'un @Getter qui permettent de créer automatiquement les getters et les setters pour tous les attributs. La 2ème utilité de cette classe est la mise en place des cardinalités avec entreprise, stage et étudiant.

Figure 2 : La classe OffreStage

Ainsi, je ne présenterais pas les entités Ecole, Entreprise, etudiant, MaitreStageEcole, MaitreStageEntreprise, Rôle, Personne et Stage car ce sont des classes très similaires dans lesquelles on définit uniquement les attributs et les cardinalités.

```
package gestionStages.entity;

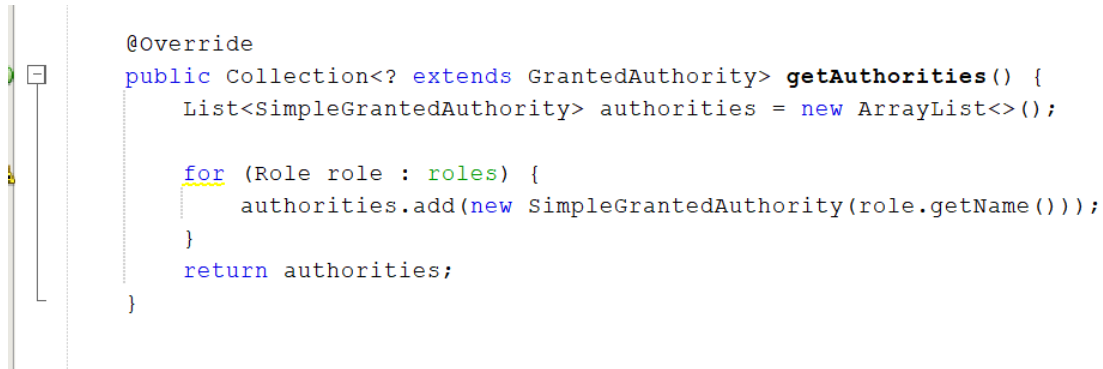
public enum EtatStage {
    ENATTENTEDEVALIDATIONOFFRE, VALIDE, NONCOMMENCE, ENCOURS, FINI, RAPPORT, SOUTENANCE
}

package gestionStages.entity;

public enum EtatOffreStage {
    ENATTENTEDEVALIDATIONADMIN, VALIDEE, AREFUSEE, LIBRE, ACCEPTEE, EREFUSEE
}
```

Figure 3 et 4 : Enumérations EtatStage et EtatOffreStage

Les classes EtatStage et EtatOffreStage sont des énumérations qui permettront de définir l'avancement d'un stage ou d'une offre de stage.



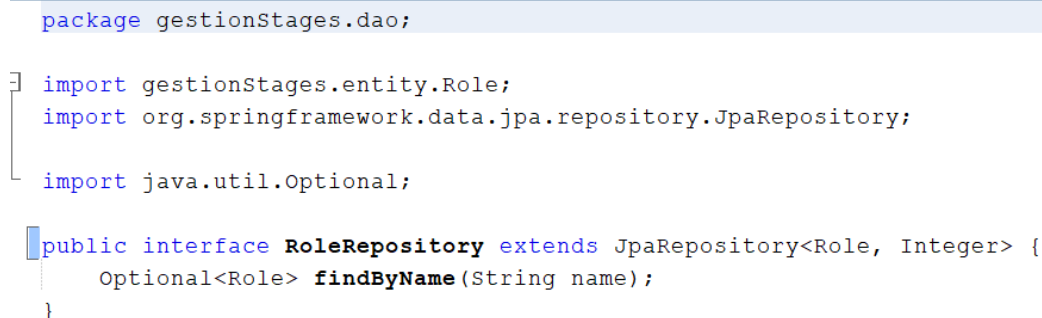
```
@Override
public Collection<? extends GrantedAuthority> getAuthorities() {
    List<SimpleGrantedAuthority> authorities = new ArrayList<>();

    for (Role role : roles) {
        authorities.add(new SimpleGrantedAuthority(role.getName()));
    }
    return authorities;
}
```

Figure 5 : Capture d'écran d'une fonction de notre classe Utilisateur

La classe Utilisateur, quant à elle, est un peu plus spéciale. En effet, elle permet tout comme OffreStage de définir les attributs et les cardinalités. Nous avons ajouté, comme vous pouvez le voir ci-dessus, une fonction qui définit les autorisations selon le rôle de l'utilisateur.

B) Nos repositories :



```
package gestionStages.dao;

import gestionStages.entity.Role;
import org.springframework.data.jpa.repository.JpaRepository;

import java.util.Optional;

public interface RoleRepository extends JpaRepository<Role, Integer> {
    Optional<Role> findByName(String name);
}
```

Figure 6 : capture d'écran de notre classe RoleRepository

Tous nos DAO sont des Repository qui étendent le JpaRepository qui contient un grand nombre de fonctions utilisables. Par exemple, notre repository de rôle ci-dessus permet de trouver un rôle via son nom. Tous nos DAO ont pratiquement la même forme.

C) Nos Contrôleurs

Nos controller contiennent toutes les fonctions que nous allons utiliser dans notre application. Nous en avons créé 11, nous n'allons pas expliquer le fonctionnement de tous les controllers mais nous allons expliquer les fonctions importantes de offresStagesController.

```

34 @Controller
35 @RequestMapping(path = "/offreStage")
36 public class OffreStageController {
37     @Autowired
38     private OffreStageRepository dao;
39
40     @Autowired
41     private EntrepriseRepository dao2;
42
43     @Autowired
44     private EtudiantRepository dao3;
45
46     @Autowired
47     private StageRepository dao4;
48
49     /**
50      * Affiche toutes les offres de stage dans la base
51      *
52      * @param model pour transmettre les informations à la vue
53      * @return le nom de la vue à afficher ('afficheOffreStage.html')
54      */
55     @GetMapping(path = "show")
56     public String afficheToutesLesOffresDeStage(Model model) {
57         model.addAttribute("offreStage", dao.findAll());
58         return "afficheOffreStage";
59     }
60

```

Figure 7 : Début de notre controller offre Stage (attributs + fonction 'show')

La première fonction nous permet de montrer la totalité des offres de stages présentes sur l'application.

Compte rendu projet FIE 3 - Gestion des stages

```
--
1
67 @GetMapping(path = "add")
68 public String montreLeFormulairePourAjout(@ModelAttribute("offreStage") OffreStage offreStage, Model model) {
69     model.addAttribute("entreprises", dao2.findAll());
70     return "formulaireOffreStage";
71 }
72
73 /**
74  * Appelé par 'formulaireOffreStage.html', méthode POST
75  *
76  * @param offreStage Une offre de stage initialisée avec les valeurs saisies dans le formulaire
77  * @param redirectInfo pour transmettre des paramètres lors de la redirection
78  * @return une redirection vers l'affichage de la liste des offres
79  */
80 @PostMapping(path = "save")
81 public String ajouteOffrePuisMontreLaListe(OffreStage offreStage, Stage stage, @AuthenticationPrincipal Entreprise user,
82     RedirectAttributes redirectInfo, Model model) {
83     String message;
84     try {
85         // cf. https://www.baeldung.com/spring-data-crud-repository-save
86         model.addAttribute("entreprises", dao2.findAll());
87         offreStage.setProposeur(user);
88         dao.save(offreStage);
89
90         // If(offreStage.getEtatOffre()== etatOffre.VALIDEE){
91         //     stage.setTitre(offreStage.getTitre());
92         //     stage.setDescription(offreStage.getDescription());
93         //     stage.setDateDebut(offreStage.getDateDebut());
94         //     stage.setDateFin(offreStage.getDateFin());
95         //     stage.setEntrepriseAccueil(offreStage.getProposeur());
96         //     dao4.save(stage);
97         // }
98         // Le code de la catégorie a été initialisé par la BD au moment de l'insertion
99         message = "L'offre '" + offreStage.getTitre() + "' a été correctement enregistrée";
100     } catch (DataIntegrityViolationException e) {
101         // Les noms sont définis comme 'UNIQUE'
102         // En cas de doublon, JPA lève une exception de violation de contrainte d'intégrité
103         message = "Erreur : L'offre '" + offreStage.getTitre() + "' existe déjà";
104     }
105     // RedirectAttributes permet de transmettre des informations lors d'une redirection,
106     // Ici on transmet un message de succès ou d'erreur
107     // Ce message est accessible et affiché dans la vue 'afficheGalerie.html'
108     redirectInfo.addFlashAttribute("message", message);
109     return "redirect:show"; // POST-Redirect-GET : on se redirige vers l'affichage de la liste
110 }
111
```

Figure 8 : Suite de notre controller OffreStage

La seconde fonction du controller permet de retourner le formulaire d'offre stage. Puis la troisième fonction permet d'ajouter des offres de stage après avoir rempli le formulaire, puis de montrer la liste de toutes les offres.

Compte rendu projet FIE 3 - Gestion des stages

```
117 */
118 @GetMapping(path = "modifier")
119 public String montreLeFormulairePourModifier(@RequestParam("id") OffreStage offreStage, Integer id, Model model) {
120
121     model.addAttribute("offreStage", dao.findById(id));
122     return "modifierOffre";
123 }
124
125 /**
126  * Appelé par 'afficheOffreStage.html', méthode POST
127  *
128  * @param offreStage Une offre de stage initialisée avec les valeurs saisies dans le formulaire
129  * @param redirectInfo pour transmettre des paramètres lors de la redirection
130  * @return une redirection vers l'affichage de la liste des offres
131  */
132 @PatchMapping(path = "modifier")
133 public String modifierOffrePuisMontreLaListe(@RequestParam("id") OffreStage offreStage, Stage stage, Integer id,
134                                             EtatOffreStage etatOffre, RedirectAttributes redirectInfo, Model model) {
135     String message;
136     try {
137         // cf. https://www.baeldung.com/spring-data-crud-repository-save
138         model.addAttribute("offreStage", dao.findById(id));
139         dao.save(offreStage);
140         // if(offreStage.getEtatOffre() != etatOffre.VALIDEE){
141         //     stage.setTitre(offreStage.getTitre());
142         //     stage.setDescription(offreStage.getDescription());
143         //     stage.setDateDebut(offreStage.getDateDebut());
144         //     stage.setDateFin(offreStage.getDateFin());
145         //     stage.setEntrepriseAccueil(offreStage.getProposeur());
146         //     dao4.save(stage);
147         // }
148         // Le code de la catégorie a été initialisé par la BD au moment de l'insertion
149         message = "L'offre '" + offreStage.getTitre() + "' a été correctement enregistrée";
150     } catch (DataIntegrityViolationException e) {
151         // Les noms sont définis comme 'UNIQUE'
152         // En cas de doublon, JPA lève une exception de violation de contrainte d'intégrité
153         message = "Erreur : L'offre '" + offreStage.getTitre() + "' existe déjà";
154     }
155     // RedirectAttributes permet de transmettre des informations lors d'une redirection,
156     // Ici on transmet un message de succès ou d'erreur
157     // Ce message est accessible et affiché dans la vue 'afficheGalerie.html'
158     redirectInfo.addFlashAttribute("message", message);
159     return "redirect:show"; // POST-Redirect-GET : on se redirige vers l'affichage de la liste
160 }
161 }
```

Figure 9 : Suite 2 de notre controller OffreStage

La suite du controller est composée de 2 fonctions comme précédemment. La première permet uniquement de montrer le formulaire qui permettra de modifier l'offre. Puis la seconde la modifie puis montre la liste de toutes les offres.

```
193
194 @GetMapping(path = "delete")
195 public String supprimeUneOffrePuisMontreLaListe(@RequestParam("id") OffreStage offreStage, RedirectAttributes redirectInfo) {
196     String message = "L'offre '" + offreStage.getTitre() + "' a bien été supprimée";
197     try {
198         dao.delete(offreStage); // Ici on peut avoir une erreur
199     } catch (DataIntegrityViolationException e) {
200         // violation de contrainte d'intégrité si on essaie de supprimer une offre de stage
201         message = "Erreur : Impossible de supprimer l'offre '" + offreStage.getTitre() + "'";
202     }
203     // RedirectAttributes permet de transmettre des informations lors d'une redirection,
204     // Ici on transmet un message de succès ou d'erreur
205     // Ce message est accessible et affiché dans la vue 'afficheOffreStage.html'
206     redirectInfo.addFlashAttribute("message", message);
207     return "redirect:show"; // on se redirige vers l'affichage de la liste
208 }
```

Figure 10 : Fin de notre controller OffreStage

La dernière fonction de notre controller permet de supprimer totalement l'offre de stage sélectionnée.

D) Nos Services et Validator :

Ce code est utilisé principalement pour la mise en place d'un login. Nous avons utilisé le code envoyé par M. Bastide en le modifiant car le code mettait obligatoirement le compte créer en User ce qui ne nous convenait pas. En effet, nous avons besoin de 3 types de compte : les comptes administrations, les comptes étudiants et les comptes entreprises.

E) Notre service de stockage

Ce code est utilisé pour upload de fichiers. Il s'agit d'un code envoyé par M. Bastide. Seules les autorisations nous ont posé des problèmes sur ce fichier.

En effet, en mettant ensemble les deux codes (Login + Upload) les fichiers ne pouvaient pas s'uploader (erreur 403), nous n'avions pas les droits même si nous étions connectés en tant qu'administrateurs. Pour régler cela, nous avons dû contraindre le contrôleur d'upload de fichier en autorisant l'accès aux utilisateurs avec le rôle `ROLE_ETUDIANT` uniquement.

F) Nos pages HTML :

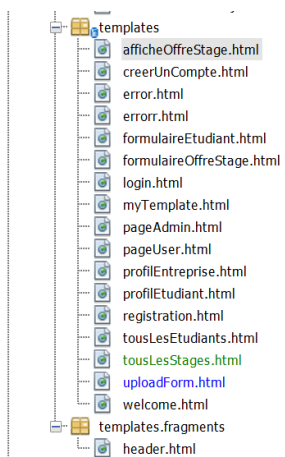


Figure 11 : Capture d'écran de nos différentes pages HTML

Voici la liste des différentes pages HTML que nous avons créées pour notre application. On peut remarquer que nous possédons un fichier du nom de `header.html` dans le dossier `templates.fragments` et une multitude de pages HTML dans le dossier `templates`.

Tout a été pensé pour que l'interface utilisateur soit agréable. Comme vous allez le voir dans le style (cf. Notre CSS) le design est épuré, l'utilisateur a très facilement accès à l'information qu'il cherche.



```

1 <!DOCTYPE HTML>
2 <html lang="fr" xmlns="http://www.w3.org/1999/xhtml" xmlns:th="http://www.thymeleaf.org">
3 <head>
4 <title>Header</title>
5 <meta charset="UTF-8" />
6 <meta name="viewport" content="width=device-width, initial-scale=1.0" />
7 <link rel="stylesheet" type="text/css" th:href="@{/css/style.css}" href="../../static/css/style.css"/>
8 </head>
9 <body>
10 <div th:fragment="header" class="container">
11 <div class="navbar">
12 <a href="/">Accueil</a>
13 <a href="/offrestage/show">Les offres de stage</a>
14 <a href="/login">Se connecter</a>
15 </div>
16 </div>
17 </body>
18 </html>
19

```

Figure 12 : Capture d'écran de headers.html

Voici le fichier headers.html. Nous avons fait le fragment de la ligne 10 à 16 pour ensuite pouvoir l'appeler directement dans les autres fichiers HTML.



```

1 <!DOCTYPE HTML>
2 <!--/*
3 Le contrôleur transmet deux informations à cette vue :
4 - offreStage : la liste des offres à afficher
5 - message : un éventuel message d'erreur ou d'information
6 */-->
7
8 <html lang="fr" xmlns="http://www.w3.org/1999/xhtml" xmlns:th="http://www.thymeleaf.org">
9 <head>
10 <title>Liste des offres de stage</title>
11 <meta charset="UTF-8" />
12 <meta name="viewport" content="width=device-width, initial-scale=1.0" />
13 <link rel="stylesheet" type="text/css" th:href="@{/css/style.css}" href="../../static/css/style.css"/>
14 </head>
15 <body>
16 <section th:insert="fragments/header :: header">Header</section>
17 <section id="titre">
18 <div class="container">
19 <div class="titre">
20 <h1>Les offres de stages</h1>
21 </div>
22 </div>
23 </section>
24 <section id="listeOffres">

```

Figure 13 : capture d'écran de la page HTML afficheOffreStage

Par exemple, dans notre page afficheOffreStage on peut voir à la ligne 10 la commande permettant d'insérer notre fragment dans une autre page HTML.

G) Notre CSS :

Pour notre CSS, nous sommes partis du principe que le site était destiné principalement aux étudiants, nous avons donc cherché à mettre en place un design simple et moderne. Ainsi nous avons décidé de partir sur le style neumorphisme afin de donner un effet de relief aux différents boutons ou données de l'application, nous avons choisi d'utiliser les couleurs du style (gris et bleu). Au niveau des polices, nous avons choisi une typo Google Fonts, la principale du style : Nunito Sans-serif

H) Nos scripts JS :

```

1  let boutonEtudiant = document.getElementById("jeSuisUnEtudiant");
2  let boutonEntreprise = document.getElementById("jeSuisUneEntreprise");
3
4  boutonEtudiant.addEventListener("click", formulaireNouveauCompte);
5  boutonEntreprise.addEventListener("click", formulaireNouveauCompte);
6
7
8  function formulaireNouveauCompte(event) {
9      if(event.target.id == "jeSuisUnEtudiant"){
10         document.location.href="/etudiant/creerUnCompte";
11     }
12     if(event.target.id == "jeSuisUneEntreprise"){
13         document.location.href="/entreprise/creerUnCompte";
14     }
15 }

```

Figure 14 : Script JS pour le bouton de création de compte

Notre premier script JS est utilisé pour le bouton utilisé pour le choix du type de compte. La fonction va s'appliquer si on appuie sur l'un des deux boutons et va nous rediriger vers le formulaire de création de compte du type qu'on a choisi (entreprise ou étudiant).

```

1  let boutonNom = document.getElementById("nom");
2  let boutonPrenom = document.getElementById("prenom");
3  let boutonAnneeEtude = document.getElementById("anneeEtude");
4
5  let nomActif = 0;
6  let prenomActif = 0;
7  let anneeActif = 0;
8
9
10 boutonNom.addEventListener("click", boutonNomClique);
11 boutonPrenom.addEventListener("click", boutonPrenomClique);
12 boutonAnneeEtude.addEventListener("click", boutonAnneeClique);
13
14 function boutonNomClique(){
15     if (!nomActif) {
16         boutonNom.style.boxShadow = "inset -3px -3px 6px rgba(255, 255, 255, 0.75), inset 3px 3px 6px rgba(0, 0, 0, 0.3)";
17         nomActif = 1;
18     }
19     else{
20         boutonNom.style.boxShadow = "-3px -3px 6px rgba(255, 255, 255, 0.75), 3px 3px 6px rgba(0, 0, 0, 0.3)";
21         nomActif = 0;
22     }
23 }
24
25 function boutonPrenomClique(){
26     if (!prenomActif) {
27         boutonPrenom.style.boxShadow = "inset -3px -3px 6px rgba(255, 255, 255, 0.75), inset 3px 3px 6px rgba(0, 0, 0, 0.3)";
28         prenomActif = 1;
29     }
30     else{
31         boutonPrenom.style.boxShadow = "-3px -3px 6px rgba(255, 255, 255, 0.75), 3px 3px 6px rgba(0, 0, 0, 0.3)";
32         prenomActif = 0;
33     }
34 }
35
36 function boutonAnneeClique(){
37     if (!anneeActif) {
38         boutonAnneeEtude.style.boxShadow = "inset -3px -3px 6px rgba(255, 255, 255, 0.75), inset 3px 3px 6px rgba(0, 0, 0, 0.3)";
39         anneeActif = 1;
40     }
41     else{
42         boutonAnneeEtude.style.boxShadow = "-3px -3px 6px rgba(255, 255, 255, 0.75), 3px 3px 6px rgba(0, 0, 0, 0.3)";
43         anneeActif = 0;

```

Figure 15 : script JS pour les boutons filtres du formulaire où sont noté tous les étudiants

Cette fonction ne nous sert pas encore, nous n'avons pas eu le temps mais nous le laissons si ce projet est repris dans les prochaines années.

4) Utilisation de l'application

Lors de l'ouverture de l'application, on arrive directement sur la page d'accueil qui nous permet d'accéder à toutes les offres de stage de notre application sans se connecter.



Figure 16 : Page d'accueil de l'application

Ensuite, on peut accéder à toutes les offres sans être connecté mais on ne peut pas postuler à un stage. Ainsi, pour pouvoir accéder pleinement à l'application, il faut cliquer sur l'onglet "se connecter".



Figure 17 : Page des offres de stage.

Il faut, par la suite, créer un compte ou s'identifier si le compte est déjà créé.

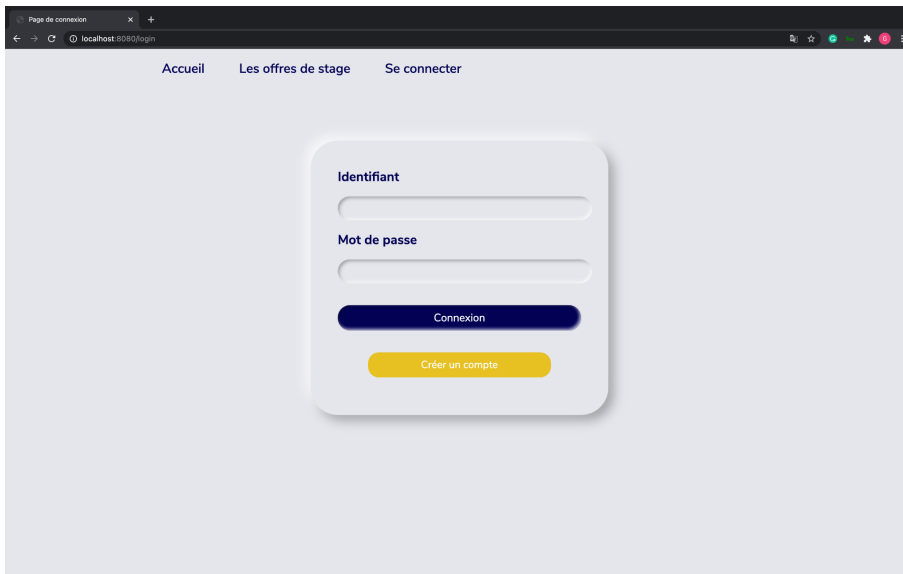
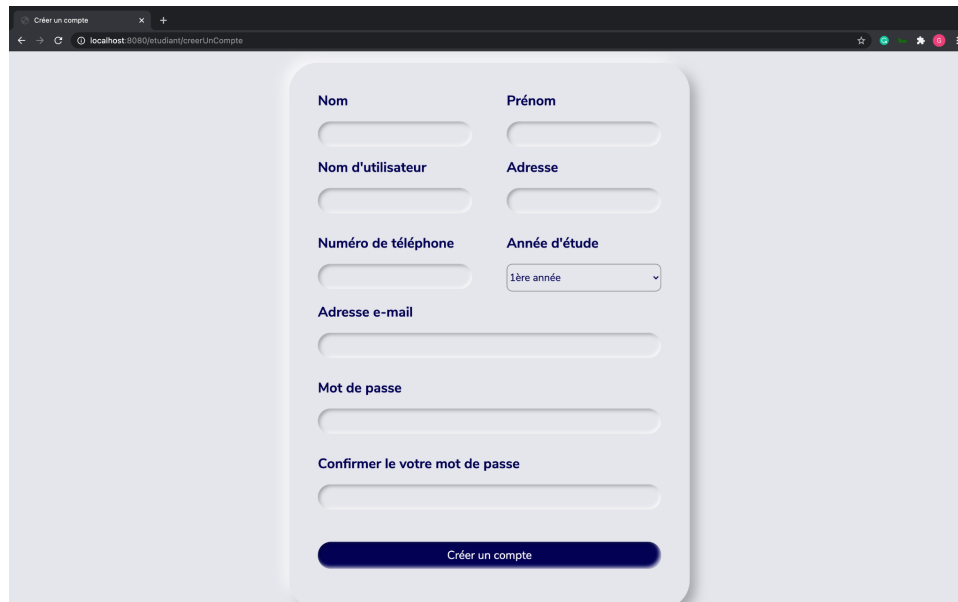


Figure 18 : Page de connexion

Pour créer le compte il faut d'abord choisir entre un compte entreprise et un compte étudiant, puis remplir le formulaire qui correspond à chacun des comptes.

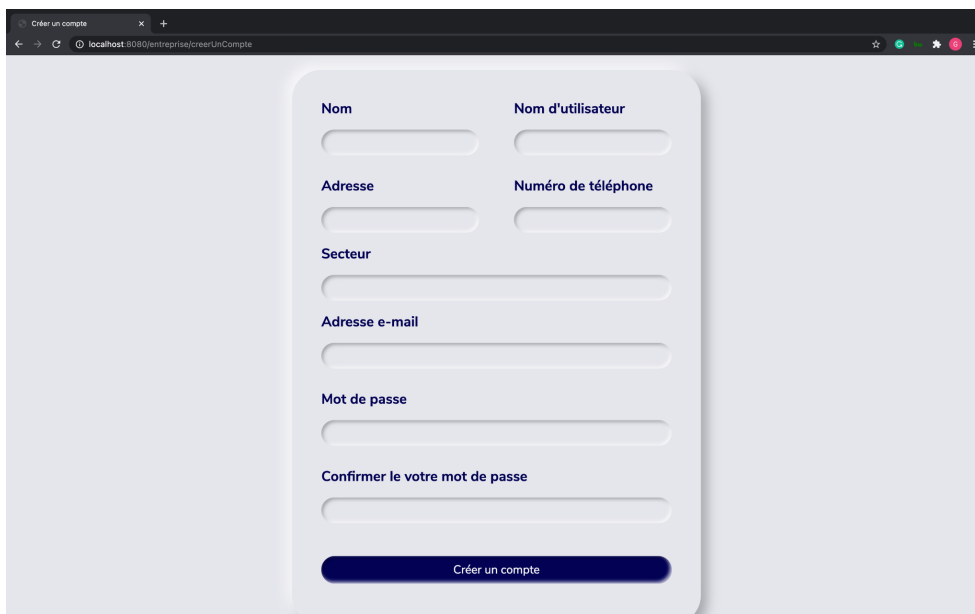


Figure 19 : choix entre compte entreprise et compte étudiant



The screenshot shows a web browser window with the address bar displaying 'localhost:8080/etudiant/creerUnCompte'. The page features a light blue background with a central white rounded rectangle containing the registration form. The form includes the following fields: 'Nom' and 'Prénom' (top row), 'Nom d'utilisateur' and 'Adresse' (second row), 'Numéro de téléphone' and 'Année d'étude' (third row, with a dropdown menu showing '1ère année'), 'Adresse e-mail' (fourth row), 'Mot de passe' (fifth row), and 'Confirmer le votre mot de passe' (sixth row). A dark blue button labeled 'Créer un compte' is positioned at the bottom of the form.

Figure 20 : page de création de compte pour les étudiants



The screenshot shows a web browser window with the address bar displaying 'localhost:8080/entreprise/creerUnCompte'. The page features a light blue background with a central white rounded rectangle containing the registration form. The form includes the following fields: 'Nom' and 'Nom d'utilisateur' (top row), 'Adresse' and 'Numéro de téléphone' (second row), 'Secteur' (third row), 'Adresse e-mail' (fourth row), 'Mot de passe' (fifth row), and 'Confirmer le votre mot de passe' (sixth row). A dark blue button labeled 'Créer un compte' is positioned at the bottom of the form.

Figure 21 : page de création de compte pour les entreprises

Puis, après la création de compte, selon le type de compte, les pages accessibles seront différentes selon le rôle de la personne inscrite. En effet, les étudiants auront accès à modifier leur profils, afficher les offres de stage et auront un bouton postuler à côté de ces dernières. Les entreprises, quant à elles, pourront modifier leur profils et leur offres, afficher les offres de stage et ajouter une offre de stage.

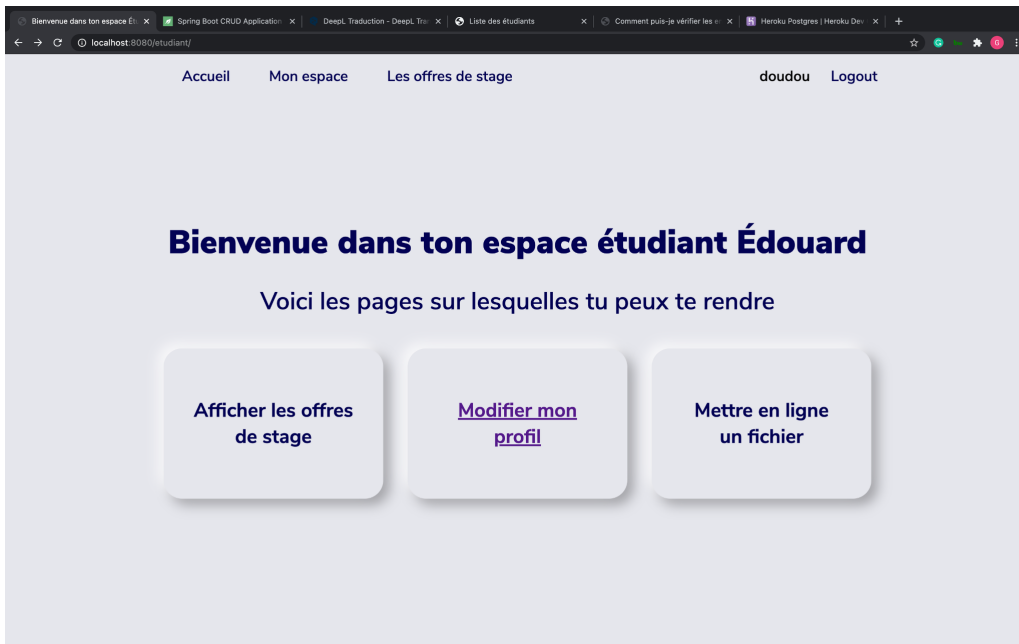


Figure 22 : Page d'accueil après s'être connecté ou inscrit comme étudiant

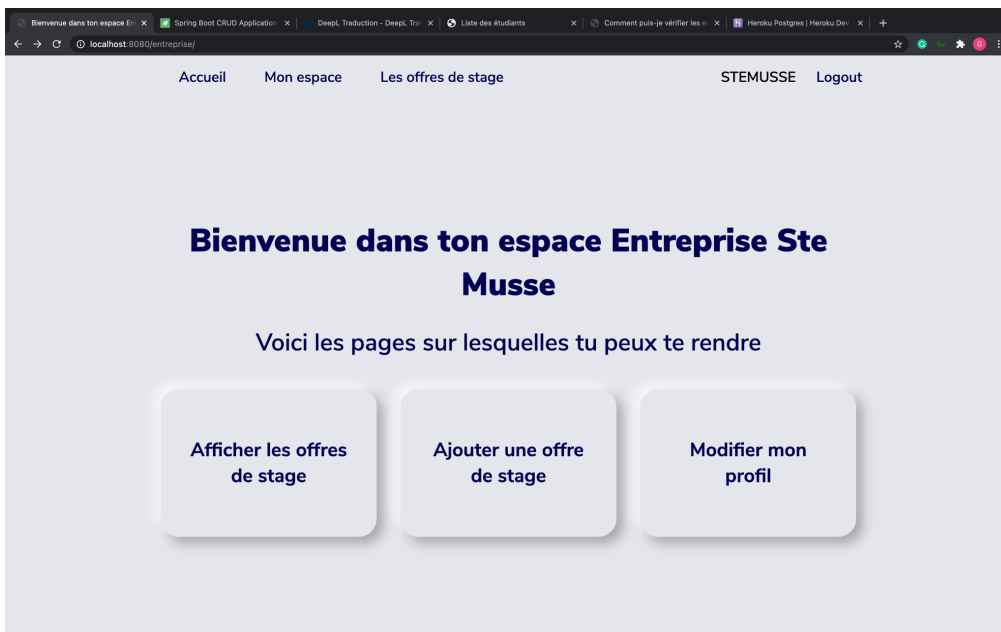


Figure 23 : Page d'accueil après s'être connecté ou inscrit comme entreprise

De plus, il y a une troisième page d'accueil différentes mais celle-ci est pour l'administration. Il n'y a pas de possibilité de créer des comptes administration actuellement. Pour avoir cette dernière page, il faut se connecter avec les logins administrateur défini dans application.properties. Ce rôle permet de visualiser et de supprimer n'importe quelle offre

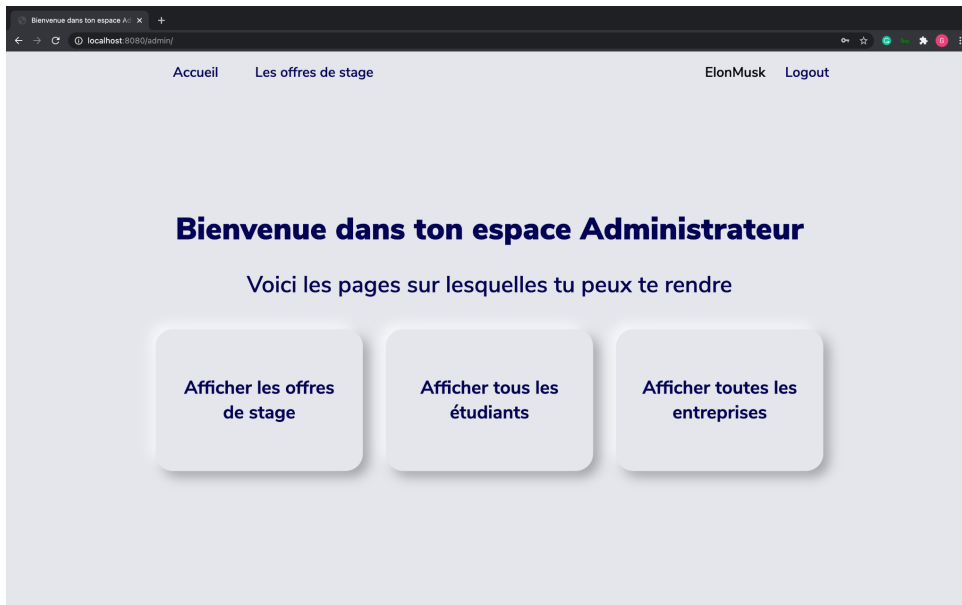


Figure 24 : Page d'accueil après s'être connecté comme administrateur

Les différentes pages accessible avec les rôles :

Rôle Administrateur :



Figure 25: Page des offres avec le rôle administrateur avec l'option suppression.

Compte rendu projet FIE 3 - Gestion des stages

Accueil Les offres de stage ElonMusk Logout

Ajouter une offre

Entreprise: **ElonMusk** Libellé de l'offre

Libellé

Description de l'offre

Description

Date de début Date de fin

jj/mm/aaaa jj/mm/aaaa

Etat:

LIBRE

Enregistrer

Figure 26: Page d'ajout d'offres avec le rôle administrateur

Accueil Mon espace Les offres de stage ElonMusk Logout

Toutes les entreprises

Nom Secteur Année d'étude Recherche un étudiant

AP-HP	hôpital	0660006600
PIERRE FABRE	laboratoire	0660006600

Figure 27 : Formulaire contenant toutes les entreprises

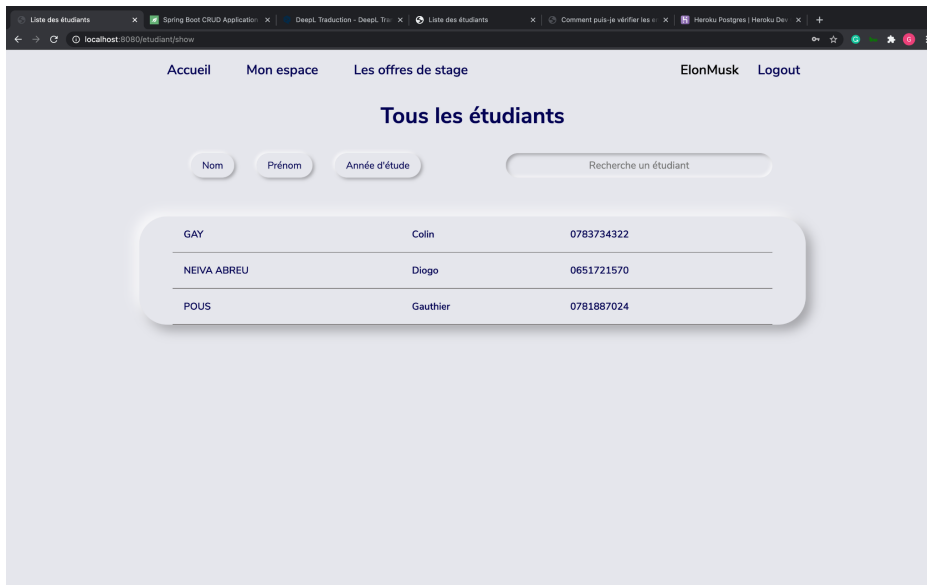


Figure 28 : Formulaire contenant tous les étudiants

Rôle Étudiant :



Figure 29: Page des offres des stages avec le bouton postuler (qui ne fonctionne pas encore) pour le rôle étudiant



Figure 30 : Page permettant l'upload de fichier en ligne

Role entreprise :

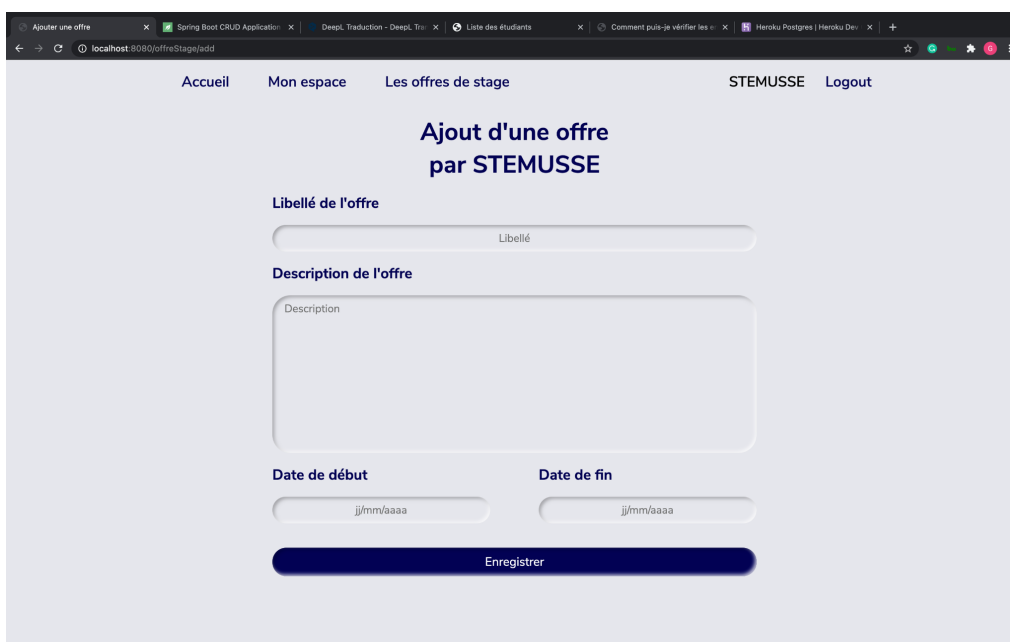


Figure 31 : Page permettant d'ajouter un stage pour le rôle entreprise.

Nous avons rencontré un problème de dernière minute, pour l'option modifierProfil, il nous faut des donn

5) Perspective d'évolution et erreurs

Comme expliqué dans notre présentation générale, notre version actuelle de l'application n'est pas optimale. En effet, nous n'avons pas eu assez de temps pour la développer entièrement. Nous nous sommes concentrés sur les parties que nous pensions indispensables à notre projet.

Ainsi, voici une liste d'amélioration à apporter pour rendre l'application beaucoup plus utile :

- Possibilité de modifier les offres et les comptes utilisateur
- Effectuer une fonction postuler qui permet d'envoyer les différents document à l'entreprise via son E-mail
- mettre en place un canal de discussion (entre l'étudiant, l'entreprise et l'école) afin de faciliter les échanges. Cela permettrait de remplir les propositions et les conventions de stage plus rapidement ainsi que de répondre aux interrogations des étudiants facilement.
- mettre en place tout le côté administration de l'application (pouvoir communiquer facilement avec les élèves et les entreprises, pouvoir voir rapidement les stages de tous les élèves).
- mise en place de filtres (principaux : entreprise, offres, conseils, mon compte, dans offres : localisation, année, type, durée du stage, secteur) pour pouvoir trouver un stage qui nous intéresse sans difficulté.
- voir les historiques des entreprises dans lesquelles les anciens étudiants ont effectué leurs stages
- possibilité de signer les documents via l'application (signature électronique)
- correction des bugs mineurs
- relier les comptes de l'application avec les adresses mail universitaire de l'INU champollion

6)Autres aspects techniques

Pour pouvoir rendre notre application utilisable, nous avons dû utiliser de nombreuses fonctions compliquées à coder. Nous avons donc dû faire appel à M. Bastide afin de mettre en place la gestion et l'authentification des utilisateurs, mais aussi pour l'upload de fichiers en ligne (ce qui va nous permettre d'archiver les propositions, les conventions,...).

Ainsi, pour mettre en place la gestion des utilisateurs, nous avons utilisé cet exemple : <https://github.com/bastide/gestionutilisateurs> et nous l'avons adapté afin de l'utiliser dans notre code.

Pour l'upload de fichier en ligne nous avons utilisé et adapté cet exemple : <https://github.com/bastide/uploading-files>

Note de fin : ce compte-rendu a été rédigé en même temps que nous finissions le code, il est possible que certaines parties de notre rapport soient différentes de notre code final mais les changements seront légers.