Lors de notre stage nous avons développé un site Web coder de façon dynamique. Ce site avait pour but de gérer les différents projets des laboratoires de l’Université de Sherbrooke. Chaque projet étaient constitués d’un résumé en français en un anglais, de membre, et enfin de photo. Dans ces projets, on pouvait y trouver une liste de membre actif c’est-à-dire qu’ils travaillent encore sur le projet et une autre de membre inactif qui contient la liste des membres qui ont travaillés sur le projet mais qui désormais ne sont plus à l’Université ou qu’ils ont arrêtés.

Pour réaliser ce projet nous avons travaillés selon le la méthode MVC. Cette méthode consiste à séparer les différentes fonctionnalités :

* M : il correspond au modèle, c’est le modèle qui va manipuler les données de la base de donnée, c’est lui qui nous permet de récupérer toutes les données, les modifier, en ajouter ou encore d’en supprimer.
* V : il s’agit de la vue, cette partie est coder en HTML, c’est la partie qui permet d’afficher une page Web. Elle fait appelle au Controlleur qui lui dit ce qu’elle doit afficher et lui envoie des données récupéré auparavant dans le modèle.
* C : le Controlleur, comme dit ci-dessus, il demande au modèle les informations dont il a besoin pour ensuite choisir quelle page du site il va afficher et avec quelle donnée. Il peut aussi modifier le modèle.

**Les droits**

Notre site possédait plusieurs rôles, Administrateur, Propriétaire et Utilisateur. Chacun de ces rôles avait des fonctions différentes. L’Administrateur pouvait modifier les résumés des projets, ajouter des membres dans la base donnée, modifier certaines informations du profil d’un membre (nom, prénom, adresse mail, rôle, statut, cip). Le Propriétaire, lui avait la possibilité de modifier les résumés des projets, ajouter des participants à son projet, modifier les listes des membres actif ou inactif, changer les photos de son projet. Les Utilisateurs pouvaient simplement modifier leurs profils et naviguer sur le site web.

D’après l’architecture MVC nous n’avions pas le droit de faire des tests à l’intérieur des vues c’est pourquoi il y a certaine page qui possèdent plusieurs vues, elles correspondent chacune aux rôles.

**Les Controllers :**

Chaque controlleur correspond à un objet, dans notre projet nous avions un controller par page, ces controlleur étaient composés de plusieurs fonction, chaque fonction correspond à une fonctionnalité très précise, par exemple nous avions une fonction pour l’affichage de la page home, une autre pour l’affichage de la page homeEdit, etc.

**ConferencesController**

Controlleur permettant d’afficher la page conférence et réunion.

**ConstanteMessages**

Contient les messages du profil.

**ConstanteMessageFlash**

Contient tout les messages flash du site

Exemple :

**HomeController**

Ce controller est composé de 5 fonction.

La première fonction nous servait à afficher la page Home. Sur cette page plusieurs éléments étaient afficher. Le résumé dans la langue choisis par l’utilisateur, et les photos du projet. Ces éléments étaient récupérés dans la base de donnée en faisant appelle à une fonction du modèle Home puis initialisé dans une variable. Enfin pour afficher la bonne page en fonction du rôle on testait une variable de session qui nous renvoyait le rôle de la personne connecté. Ensuite en envoyait ces variables comme paramètre dans la page web.

La fonction changelanguage, nous permettait de changer le langage sur le page pour passer du français à l’anglais.

La fonction edithome fonctionne de la même manière que home, elle nous servait à afficher la page d’édition.

Contrairement aux autres fonctions, la fonction postEditHome, était initialisé en POST dans les routes et non pas en GET. Cette fonction avait pour but d’envoyer les modifications apportées par l’utilisateur lorsqu’il modifie le résumé de son projet ou ses photos puis de les enregistrer dans la base de donnée.

**Login Controller**

Pas besoin d’expliquer ce qu’est un login. La première fonction consiste simplement à empêcher une personne déjà connecté d’avoir accès à la page login. La fonction suivante récupère les informations des inputs et vérifie si le mot de passe correspond au pseudo rentré. Puis redirige vers la page home si les deux champs correspondent ou vers la page login si la personne c’est trompé. Enfin la dernière fonction permet de se déconnecter en supprimant ce qui se trouvait dans la variable de session.

**PasswordForgotController**

La fonction generateCode() génère un code de 20 caractères constituant le token.

sendMailResetForm() vérifie que l’utilisateur n’est pas connecté.

sendMailResetpassword() appel la fonction sendMail() ne transmettant le token et l’adresse mail du destinataire. Creer une instance de LostPassword.

SendMail() envoie le mail au destinataire, ce mail contient un lien finissant par le hash permettant d’identifier l’utilisateur via ce lien car le token est relié au cip dans l’instance de LostPassword créé précédemment.

PasswordChanged(), l’instance lostPassword n’est valable que 5 min après ce délai le lien devient inactif et il fut recommencer la démarche.

Changepassword() change le mot de passe de l’utilisateur si les deux mot de passe rentré dans le formulaire son identique.

**Profile Controller**

Sur notre site Web nous avions un système de connexion, une foi connecté l’utilisateur avait la possibilité de consulter son profil ou il retrouvait ses informations tels que son nom, son prénom, son adresse mail, son pseudo. La première fonction de ce controller, récupérait l’utilisateur grâce à une variable de session qui nous renvoyait son pseudo, ensuite on demandait d’afficher la page profile en lui envoyant comme paramètre l’utilisateur, son résumé et son statut. La deuxième fonction fonctionnait de la même manière que la précédente, sauf que cette fois-ci on affichait la page d’édition. La fonction suivante est utilisé seulement par l’administrateur, elle permet de modifier les informations d’un autre utilisateur. Pour y arriver cette fois on ne récupère pas l’utilisateur actif à l’aide de son cip mais son id. Ensuite on récupère ce qui se trouve dans les champs inputs à l’aide des librairies FormFactory et DynamicForm, puis chaque variable est envoyée dans l’utilisateur.

**Projects Controller**

Notre projet portait sur la gestion de projet, cette partie était donc importante. Trois fonctions composaient ce controller, d’abord on affichait la liste de tous les projets présents dans la base de donnée. Pour y arriver on créait une liste de projet qui récupérait tous les projets existant, tout ç était géré dans le modèle. La deuxième fonction faisait juste appelle au bon fichier HTML pour afficher la page d’ajout de projet. Enfin dans la dernière fonction on gérait l’ajout d’un projet, pour commencer seul les propriétaires avaient accès à cette page. Pour la création d’un projet deux paramètre étaient nécessaire, un propriétaire et un nom pour le projet. Pour le premier paramètre on récupérait le propriétaire qui était connecté, pour le deuxième, le nom était entré dans un input qu’on récupérait dans une variable. Enfin on faisait appelle à une fonction du modèle Project puis on sauvegardait dans la base de donnée, un propriétaire possédait une liste de projet dans laquelle il était présent, nous avons donc ajouter le projet créer à sa liste.

**PublicationController**

Controlleur permettant d’afficher la page publication.

**RegisterController**

registerForm() appel la vue en lui envoyant la liste des status existants si l’utilisateur est l’administrateur.

registerFormValidation() crée un nouvelle utilisateurs avec les champ remplis dans le formulaire.

registerRole() enregistre l’utilisateur avec son rôle indiqué.

**TeamController**

team() va charger le vue correcte selon le status de l’utilisateur dans l’équipe.

pastTeam() va charger la vue correct selon le status de l’utilisateur dans l’ancienne équipe.

EditTeam() envoie à la vue la liste des utilisateurs selon leur rôle dans l’équipe. (active, inactive et non présent dans le projet)

postAddUserTeam() va ajouter l’utilisateur sélectionné à l’équipe.

PostDesactivateUserTeam() va enlever l’utilisateur sélectionné de l’équipe.

PostActivateUserTeam() va activer un utilisateur dans l’équipe.