



---

# Développement d'un site Web en Java

---

Laboratoire eXit – Université de Sherbrooke

**Stagiaire : Loïc Travaillé**

**Tuteur entreprise : Ruben Gonzalez-Rubio**

**Tuteur IUT : Hervé Gauvrit**

## Remerciements

Je tiens à remercier toutes les personnes qui ont contribué au succès de mon stage et à ce que ce dernier se passe dans les meilleures conditions possibles.

Tout d'abord, j'adresse mes remerciements à **Ruben Gonzalez-Rubio**, mon tuteur, professeur à l'Université de Sherbrooke, mais également directeur du laboratoire eXit, pour m'avoir donné la chance de pouvoir effectuer mon stage au sein de son laboratoire, mais également pour ses différents conseils et recommandations pendant mon séjour au Canada. Je le remercie également pour m'avoir proposé un logement à Sherbrooke.

Je tiens également à remercier **Julie Giguère**, commis aux affaires académiques à l'Université de Sherbrooke, pour m'avoir guidé dans les différentes démarches administratives avant mon arrivée et à mon arrivée.

De plus, je remercie **Claire Constantin**, professeur d'anglais à l'Université de Rennes 1, pour m'avoir informé de ce stage, m'avoir recommandé auprès de Ruben Gonzalez-Rubio et ainsi me permettre de vivre cette expérience.

Je tiens aussi à remercier **Françoise Grall** pour s'être déplacée à Sherbrooke pour ma soutenance et pour m'avoir donné des conseils relatifs à la rédaction de ce rapport de stage.

Je remercie également **Julien Beuve**, étudiant au département Génie Electrique et Informatique Industrielle, pour m'avoir accompagné dans cette aventure et pour avoir réalisé son stage avec moi.

Enfin, je remercie également **Quentin** et **Martin**, ainsi que **Vincent** et **Emilien**, pour leur présence, leur bonne humeur, mais également leur aide dans le cadre du stage ou des activités qu'on a pu faire tous ensemble au cours de ce séjour.

## Sommaire

I.	Présentation .....	6
a.	L'Université de Sherbrooke .....	6
b.	Le groupe eXit .....	7
II.	Présentation du Projet .....	8
a.	Problématique.....	8
b.	Cahier des charges .....	10
III.	Outils utilisés et méthode de travail employée .....	12
a.	Java .....	12
	Le langage orienté objet.....	12
	Les classes .....	12
	Les packages.....	12
	Les librairies.....	13
b.	Eclipse.....	13
c.	SVN .....	14
d.	Play Framework.....	14
e.	Le langage HTML .....	14
f.	Le langage CSS.....	15
g.	Le langage JavaScript.....	15
h.	Le langage SQL et Postgre SQL .....	16
i.	Le langage Scala .....	16
j.	JUnit.....	17
k.	La Méthode Scrum .....	17
	Répartition des rôles dans Scrum .....	18
	Les Evènements.....	18

Les Artéfacts .....	20
I. L'architecture MVC.....	20
IV. Réalisation du projet .....	22
a. Démarrage du projet.....	22
b. Les différents sprints réalisés .....	23
c. Le Template .....	25
d. La Page de l'Equipe de Recherche du Laboratoire / La Page Team .....	29
e. La Traduction du site .....	33
f. Le test unitaire relatif aux pages Team .....	33
g. Le fichier CSS .....	37
V. Conclusion Technique .....	39
a. Fonctionnement final .....	39
b. Modifications à apporter .....	39
VI. Conclusion générale .....	40
a. Au niveau professionnel.....	40
b. Au niveau personnel.....	40
VII. Annexes .....	41
a. L'intégralité du projet.....	41
b. Résumé en français du livre <i>Clean Code</i> .....	41

## Introduction

Pour l'obtention de mon Diplôme Universitaire de Technologie (DUT) en Génie Electrique et Informatique Industrielle (GEII), j'ai choisi d'effectuer mon stage de fin de deuxième année à l'étranger. Voulant orienter ma poursuite d'études ainsi que mon projet professionnel dans un milieu orienté vers l'informatique, j'ai choisi un stage au sein du laboratoire eXit, spécialisé en informatique, situé dans la faculté de Génie de l'Université de Sherbrooke, au Canada. Ce stage s'est déroulé du 9 avril au 23 juin, soit sur une période de 11 semaines. J'étais sous la responsabilité de M Ruben Gonzalez-Rubio, fondateur du laboratoire.

Pendant mon stage, nous étions quatre étudiants en 2<sup>ème</sup> année d'IUT : Julien et moi-même, du département Génie Electrique et Informatique Industrielle de l'IUT de Rennes, ainsi que Vincent et Emilien venant du département Informatique de l'IUT de Lannion, à travailler sur un unique projet, celui du développement d'un site Web sous Java. Deux autres étudiants, Quentin et Martin, en Licence Professionnelle Assistant et Conseiller Technique en Energie Electrique et Renouvelable (ACTEER) à l'Université de Rennes 1, étaient présents à l'Université de Sherbrooke pour eux, travailler au sein du laboratoire e-Tesc.

Ce stage m'aura parfaitement permis de finaliser mon choix d'études Post-DUT ainsi que mon projet professionnel. En effet, l'apprentissage de la programmation et de l'informatique ont été pour moi une immense motivation pour effectuer ce stage.

De plus, j'ai également voulu venir au Canada pour découvrir un autre pays, une nouvelle culture. Cependant, j'aurai aussi aimé perfectionner mon niveau d'anglais, mais malheureusement à Sherbrooke la plupart des gens parlent français, et non anglais. Christophe Colineaux, venu l'année dernière effectuer son stage au sein du laboratoire eXit m'avait beaucoup vanté les avantages de ce stage, et je dois avouer que je suis bien content d'avoir pu vivre cette expérience.

Ma mission au sein du laboratoire et pendant ces 11 semaines en compagnie de Julien, Emilien et Vincent, était de rendre dynamique le site Web du laboratoire e-Tesc, qui existait déjà sous forme statique, et également y ajouter quelques fonctionnalités que vous pourrez retrouver en détail dans la suite de ce rapport.

Je présenterai tout d'abord le cadre dans lequel j'ai effectué ce stage, c'est à dire l'Université de Sherbrooke ainsi que le laboratoire eXit. J'enchaînerai ensuite avec une présentation plus précise du projet, accompagnée d'une liste détaillée des différents outils dont nous avons eu besoin pendant ces 11 semaines de stage. Enfin, on rentrera dans le vif du sujet en compagnie d'une partie sur la réalisation du projet, les problèmes rencontrés, l'évolution du cahier des charges... Pour conclure, je dresserai un bilan global de ce stage et de ce qu'il a pu m'apporter.

## I. Présentation

### a. L'Université de Sherbrooke

Fondée en 1954, l'Université de Sherbrooke est une université francophone située à Sherbrooke, au Québec au Canada, dans la région de l'Estrie. L'Université de Sherbrooke est divisée en plusieurs campus, deux à Sherbrooke et un à Longueuil, en périphérie de Montréal. Les campus de Sherbrooke comportent à eux deux un total de 8 facultés qui sont les suivantes :

- Faculté de gestion
- Faculté de droit
- Faculté d'éducation
- Faculté de génie
- Faculté des lettres et sciences humaines
- Faculté des sciences
- Faculté des sciences de l'activité physique
- Faculté de médecine et sciences de la santé

Le laboratoire eXit dans lequel j'ai effectué mon stage se situe sur le campus principal de Sherbrooke, au sein de la faculté de Génie.

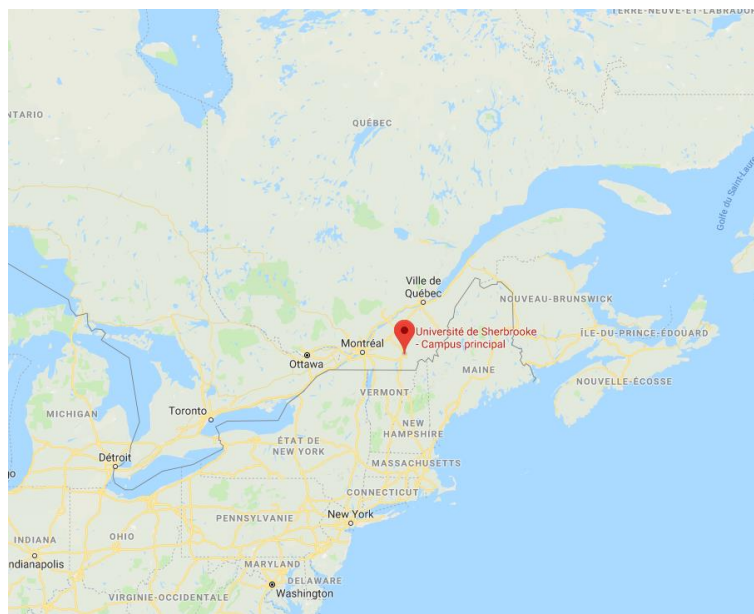


Figure 1 : Localisation de l'Université de Sherbrooke

Depuis quatre ans, l'Université de Sherbrooke se classe en tête des universités canadiennes. En 2014 et en 2015, selon le classement international établi par l'Université de Leiden aux Pays-Bas, l'Université de Sherbrooke est la première université francophone au Canada pour la pertinence de sa recherche. En 2016, l'Université est fréquentée par 40 500 étudiants provenant de 90 pays différents et son corps professoral se compose de 3 812 personnes.



*Figure 2 : La Faculté de Génie*

### b. Le groupe eXit

Le laboratoire eXit a été créé par mon tuteur, Ruben Gonzalez-Rubio, et est situé dans le plus grand bâtiment de la faculté de Génie. Ce laboratoire est axé dans la recherche en développement logiciel. Plusieurs de ces logiciels sont utilisés par l'Université de Sherbrooke.

Les membres de ce laboratoire sont continuellement renouvelés car beaucoup sont des doctorants ou des stagiaires. Le groupe accueille chaque année plusieurs étudiants français. Généralement, les stagiaires sont en charge d'actualiser les logiciels de l'Université ou rendre leur utilisation plus facile selon les contraintes techniques imposées.



## II. Présentation du Projet

Notre projet consistait principalement au développement d'un site Web permettant de consulter les informations du laboratoire e-Tesc, un laboratoire au sein de l'Université dans le domaine des énergies renouvelables et particulièrement des véhicules électriques. Les diverses informations que nous devions faire apparaître sur le site du laboratoire étaient notamment les projets en cours, l'équipe qui y travaille actuellement... Nous avons comme point de départ le site actuel du laboratoire e-Tesc.

### a. Problématique

Le site actuel du laboratoire e-Tesc était devenu totalement obsolète étant donné que la dernière modification sur ce dernier datait de juin 2016. Le site était statique, c'est-à-dire que toutes les informations y figurant étaient saisies « en dur » dans les différentes pages HTML, sans aucune liaison avec une quelconque base de données.

```

91      <tr>
92          <td><p><strong>Teaching Assistants</strong></p>
93          <p>- Serge A. Kodjo </p>
94          <p>(apedovi.kodjo@usherbrooke.ca)</p>
95          <p>&nbsp;</p>
96          <p>- Alexandre Tessier</p>
97          <p>(alexandre.tessier2@usherbrooke.ca)</p>
98      <p></p></td>
99      <td><p>&nbsp;</p>
100      <p><strong>MSc Students</strong></p>
101      <p>- Pascal-Andr   Fortin (pascal-andre.fortin@usherbrooke.ca)</p>
102      <p>- F  lix-Antoine Lebel (felix-antoine.lebel@usherbrooke.ca)</p>
103      <p>- Lara Reyes (lara.reyes.reyes@cta-brp-udes.com)<br />
104      - Nicolas Leroy (nicolas.leroy1@usherbrooke.ca)<br />
105      - Bastien Manetti (bastien.manetti@usherbrooke.ca)<br />
106      - David Joset (david.joset@usherbrooke.ca)</p>
107      <p>&nbsp;</p></td>
108  </tr>
109  <tr>
110      <td><p><strong>Lab. Technician</strong></p>
111      <p>- Denis Dufresne </p>
112      <p>(denis.dufresne@usherbrooke.ca)</p></td>
113      <td><p>&nbsp;</p>
114      <p><strong>BSc Students</strong></p>
115      <p>- Mateus Jacob (mateus.felix.jacob@usherbrooke.ca)</p>
116      <p>&nbsp;</p></td>
117  </tr>

```

Figure 3 : Un court extrait du code du Site Web du Laboratoire

De plus, beaucoup d'informations, comme par exemple les projets du laboratoire n'étaient plus à jour. Il fallait donc remédier à cela en ayant la possibilité de modifier ces informations directement depuis le site en utilisant un système de droits (administrateur, propriétaire d'un projet, utilisateur lambda).

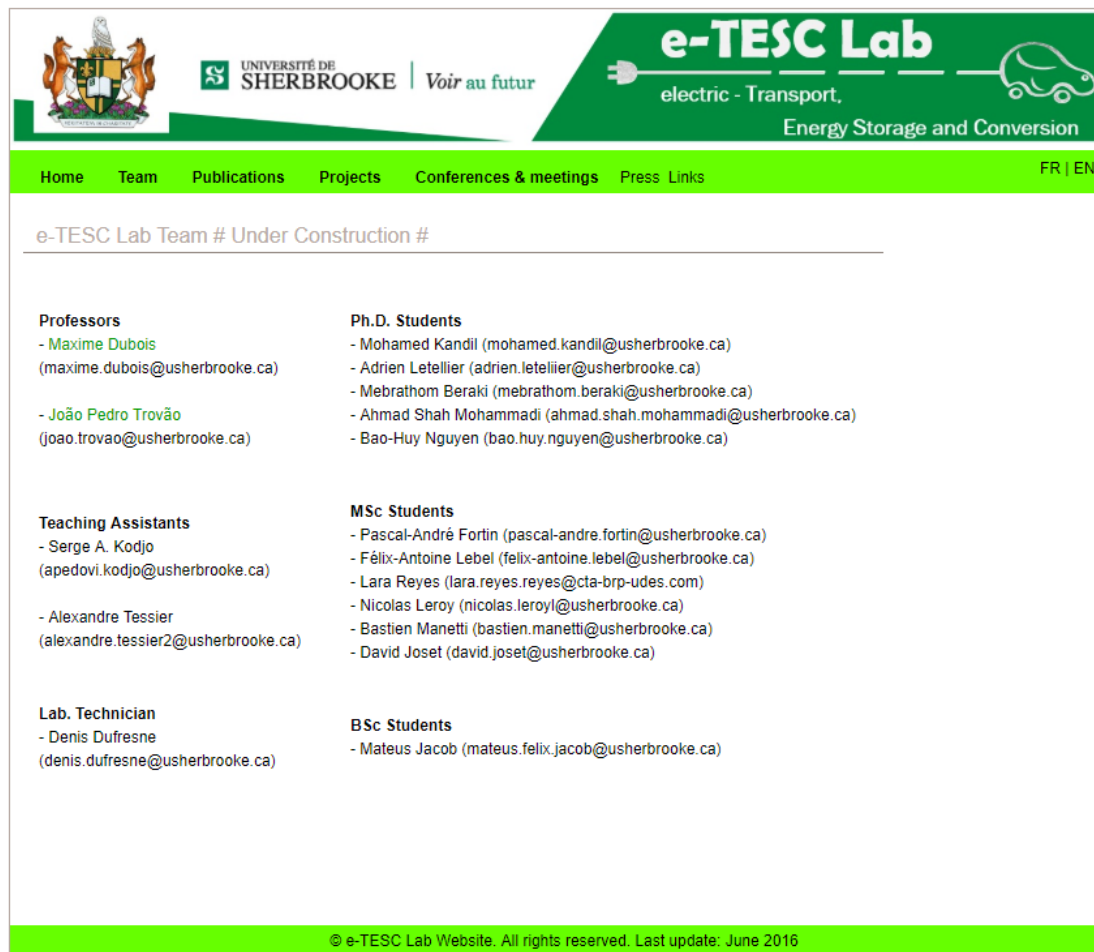
Voici ici un aperçu de la page d'accueil du site du laboratoire e-Tesc



Figure 4 : La Page d'Accueil du Site Web du Laboratoire

On peut également voir sur cette figure qu'en haut à droite, dans le menu, il y a deux boutons (EN et FR) pour avoir le site disponible en anglais et en français. Cependant, ces deux boutons ne fonctionnaient pas du tout sur le site actuel et nous avons pour mission de les faire fonctionner de sorte à avoir un site disponible en français et en anglais, pour qu'il soit compréhensible pour une plus grande population.

Aussi, on ne pouvait pas avoir beaucoup d'informations sur les membres de l'équipe, notamment une description d'eux, de leurs études... C'est pourquoi, il était important de créer un compte pour chaque membre de l'équipe, de sorte que cette dernière puisse modifier quand elle veut ses informations, et qu'elles soient consultables par tout le monde voulant avoir divers renseignements.



The screenshot shows the e-TESC Lab website. The header includes the Université de Sherbrooke logo and the text "Voir au futur". The main navigation bar is green with links: Home, Team, Publications, Projects, Conferences & meetings, Press Links, and FR | EN. Below the navigation bar, a message reads "e-TESC Lab Team # Under Construction #". The team is listed in two columns:

<b>Professors</b> - Maxime Dubois (maxime.dubois@usherbrooke.ca) - João Pedro Trovão (joao.trova@usherbrooke.ca)	<b>Ph.D. Students</b> - Mohamed Kandil (mohamed.kandil@usherbrooke.ca) - Adrien Letellier (adrien.letellier@usherbrooke.ca) - Mebrathom Beraki (mebrathom.beraki@usherbrooke.ca) - Ahmad Shah Mohammadi (ahmad.shah.mohammadi@usherbrooke.ca) - Bao-Huy Nguyen (bao.huy.nguyen@usherbrooke.ca)
<b>Teaching Assistants</b> - Serge A. Kodjo (apedovi.kodjo@usherbrooke.ca) - Alexandre Tessier (alexandre.tessier2@usherbrooke.ca)	<b>MSc Students</b> - Pascal-André Fortin (pascal-andre.fortin@usherbrooke.ca) - Félix-Antoine Lebel (felix-antoine.lebel@usherbrooke.ca) - Lara Reyes (lara.reyes.reyes@cta-brp-udes.com) - Nicolas Leroy (nicolas.leroy@usherbrooke.ca) - Bastien Manetti (bastien.manetti@usherbrooke.ca) - David Joset (david.joset@usherbrooke.ca)
<b>Lab. Technician</b> - Denis Dufresne (denis.dufresne@usherbrooke.ca)	<b>BSc Students</b> - Mateus Jacob (mateus.felix.jacob@usherbrooke.ca)

At the bottom, a green footer bar contains the text: "© e-TESC Lab Website. All rights reserved. Last update: June 2016".

Figure 5 : la page de présentation de l'équipe du laboratoire

## b. Cahier des charges

Notre projet consistait donc à développer un site Web dynamique pour consulter les informations du laboratoire.

Nous devons programmer grâce au langage JAVA principalement, mais nous devons également utiliser différents langages de programmation Web comme HTML, SCALA, JavaScript, ou encore CSS.

De plus, nous avons besoin d'une base de données SQL, gérée via pgAdmin, de Postgre SQL, un système de gestion de base de données relationnelle et objet.

Pour faire ce projet en équipe, nous avons travaillé selon la méthode Scrum et nous avons utilisé SVN, un logiciel de gestion de versions, pour travailler tous ensemble sur le même projet et en même temps.

Notre maître de stage nous a demandé également de travailler avec le Framework Play qui nous permettait de gérer au mieux la structure du site Web en langage Java, et également de travailler sous le logiciel Eclipse sur lequel nous avons déjà travaillé à l'IUT durant le module d'Informatique Embarquée au Semestre 4.

La dernière contrainte était qu'il fallait respecter l'architecture MVC, mais également les différentes « lois de programmation » explicitées dans le livre *Clean Code*, que nous avons lu et sur lequel nous avons fait un résumé en français pour pouvoir s'y référer lorsque cela était nécessaire. Vous pourrez d'ailleurs retrouver ce résumé en annexe à la fin de ce rapport.

### III. Outils utilisés et méthode de travail employée

#### a. Java

Java est un langage de programmation orienté objet créé par deux employés de Sun Microsystems. Sa première version date de 1995. Il est utilisé dans beaucoup de domaines (robotique mobile, informatique...) et est facilement transportable d'une plateforme à une autre (Windows, Linux...). Rachetée en 2009, la société Sun est désormais propriété de la société Oracle et donc Java aussi.



#### Le langage orienté objet

Un objet en programmation orientée objet représente une idée, un concept, par exemple un livre ou une personne. On définit dans cet objet des caractéristiques : le titre, l'auteur, le style du livre par exemple.

L'objet possède donc une caractéristique interne et doit pouvoir interagir avec d'autres objets, on définit donc son comportement à l'aide de méthodes/fonctions. Ces méthodes permettent de réaliser les fonctionnalités attendues du programme.

#### Les classes

Une classe est la description de données, appelées attributs, et d'opérations appelées méthodes. Il s'agit d'un modèle de définition pour des objets ayant le même ensemble d'opérations et le même ensemble d'attributs. A partir d'une classe, on peut créer un ou plusieurs objets par instantiation : chaque objet est une instance d'une seule classe. Dans cette classe on définit l'objet et toutes ses propriétés. Pour revenir à l'exemple ci-dessus, l'objet Livre peut être créé dans une classe.

#### Les packages

Les classes sont répertoriées dans des packages, qui sont différents selon le type de classe. Il n'y a aucune restriction, les packages sont là pour organiser les fichiers dans notre projet. Certaines méthodes définissent des manières de ranger ces packages selon le rôle des classes. Par exemple toutes les classes qui serviront à stocker des données telles que des utilisateurs, des rôles ou bien des tâches seront stockées dans un package.

## Les librairies

Une librairie est un ensemble de classes prédéfinies. Java en possède beaucoup qui lui sont directement intégrées. Mais le développeur a la possibilité d'importer des librairies additionnelles venant d'intervenants extérieurs ou même d'en créer soi-même. Les librairies permettent d'ajouter des fonctionnalités telles que l'affichage d'une erreur dans la console, ou bien faire des calculs mathématiques compliqués.

### b. Eclipse

Nous avons utilisé le logiciel libre et open source Eclipse afin de développer notre site web. Eclipse est un environnement de travail intégré (IDE) permettant d'écrire des programmes en Java principalement, ou d'autres langages à l'aide de plug-ins. Il a donc été très utile pour notre projet qui comprenait 5 langages distincts (HTML, CSS, Java, Scala et JavaScript).



Pour utiliser Eclipse dans le but de faire de la programmation JAVA, il faut au préalable installer le kit de développement Java (Java JDK 1.8).

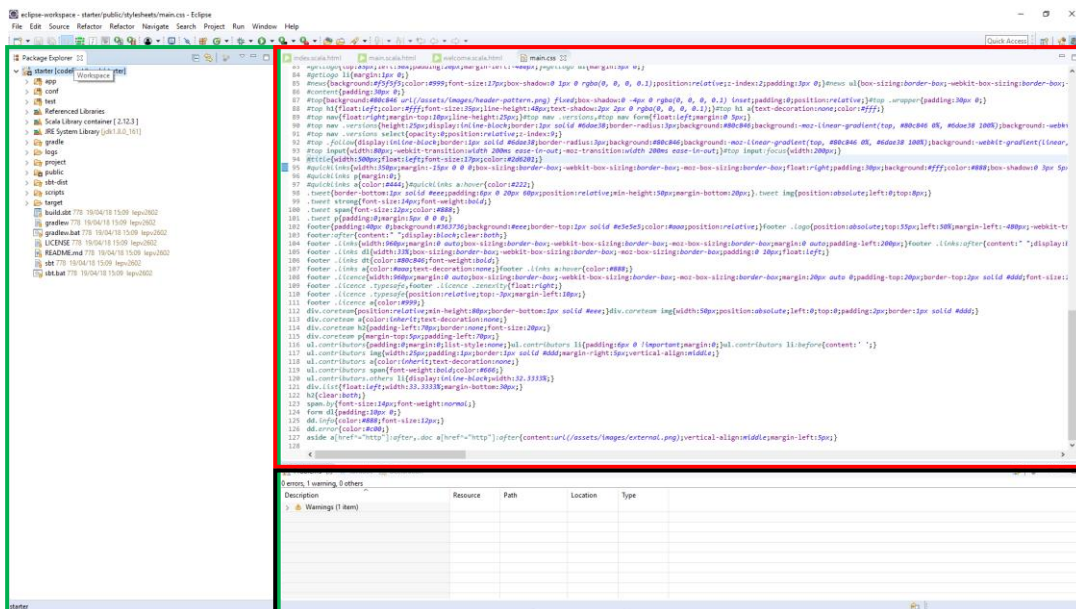


Figure 6 : Interface d'Eclipse

Encadré noir : Affichage de la console afin de déterminer les erreurs de notre Programme.

Encadré rouge : Editeur de programme, nous programmons dans cet encadré.

Encadré vert : Explorateur du projet.

### c. SVN

Subclipse SVN est un plug-in téléchargeable dans le marché d'Eclipse permettant de travailler en groupe autour d'un projet. Le projet est ainsi localisé sur un serveur (ici le serveur de l'Université), et est accessible aux membres de l'équipe. Chacun peut travailler sur une page différente, et ajoute sa nouvelle version du fichier au projet. Ainsi, le projet est toujours mis à jour par les différents développeurs. Nous utilisons également Tortoise pour transférer des fichiers hors projets, sur SVN, notamment des documentations techniques.



### d. Play Framework

Play Framework est un Framework Web Open Source qui permet d'écrire rapidement des applications Web en Java ou en Scala. Inspiré par d'autres Frameworks tels que Ruby on Rails ou Django, il vise à apporter un outil simple et productif sur la machine virtuelle Java. Du fait qu'il ne soit pas basé sur le moteur Java de Servlet, il offre un système plus simple et plus puissant pour développer une application Web en Java.



Avec l'aide de JUnit, il nous a été très utile pour la réalisation de tests unitaires sur notre application. Ces tests permettaient de vérifier le bon fonctionnement de notre site Web ainsi que trouver les potentielles erreurs et failles du site.

### e. Le langage HTML

L'*HyperText Markup Language*, généralement appelé HTML, est un langage de balise conçu pour représenter les pages Web. C'est un langage permettant d'écrire de l'hypertexte, d'où son nom, mais il permet également de structurer de manière sémantique et logique et de mettre en forme le contenu des pages, d'inclure des ressources multimédias comme des images, des formulaires de saisie... Il permet de créer des documents transportables sur tous types d'appareils (Windows, Linux...) comme Java. Il est souvent utilisé conjointement avec le langage de programmation JavaScript et le langage de description CSS.





#### f. Le langage CSS

Les feuilles de style en cascade, généralement appelées CSS, de l'anglais *Cascading Style Sheets*, forment un langage informatique qui décrit la présentation des documents HTML et XML. CSS est couramment utilisé dans la conception de sites Web. Le rendu d'un document est déterminé par les concepts de boîtes et de flux. Le moteur de rendu CSS établit une « structure de formatage » reflétant l'arbre logique du document. Chaque élément de cette structure génère une ou plusieurs zones dotées de propriétés d'affichage. L'affichage s'effectue alors à partir du flux des boîtes successivement générées pour chaque élément tel qu'il apparaît dans l'ordre linéaire de la structure de formatage.

**CSS**

#### g. Le langage JavaScript

JavaScript est un langage de programmation de scripts principalement employé dans les pages Web interactives mais aussi pour les serveurs. C'est un langage orienté objet à prototype, c'est-à-dire que les bases du langage et ses interfaces sont fournies par des objets qui ne sont pas des instances de classes mais qui sont chacun équipés de constructeurs permettant de créer leurs propriétés, et notamment une propriété de prototypage qui permet d'en créer des objets héritiers personnalisés. De ce fait, les fonctions sont des objets de première classe.

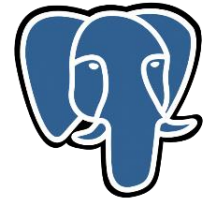


Du code JavaScript peut être intégré directement au sein des pages web, pour y être exécuté sur le poste client. C'est alors le navigateur web qui prend en charge l'exécution de ces programmes appelés scripts. Généralement, JavaScript sert à contrôler les données saisies dans des formulaires HTML, ou à interagir avec le document HTML via l'interface *Document Object Model*, fournie par le navigateur (on parle alors parfois de HTML dynamique ou DHTML). Il est aussi utilisé pour réaliser des applications dynamiques, des transitions, des animations ou manipuler des données réactives, à des fins ergonomiques ou cosmétiques.



## h. Le langage SQL et Postgre SQL

SQL, *Structured Query Language*, en français Langage de Requête Structurée, est un langage informatique normalisé servant à exploiter des bases de données relationnelles. La partie langage de manipulation des données de SQL permet de rechercher, d'ajouter, de modifier ou de supprimer des données dans les bases de données relationnelles.



PostgreSQL

PostgreSQL est un système de gestion de base de données relationnelles et objets. C'est un outil libre fondé sur une communauté mondiale de développeurs et d'entreprises, il n'est pas contrôlé par une seule entreprise. Il comporte différentes interfaces utilisateurs, comme psql (interface en lignes de commande), pgAdmin (interface graphique) ou encore phpPgAdmin (interface Web). Dans notre cas, nous avons utilisé pgAdmin qui est très facile d'utilisation via son interface graphique permettant de comprendre rapidement le fonctionnement de la base de données.

## i. Le langage Scala

Scala est un langage de programmation multi-paradigme conçu pour exprimer les modèles de programmation courants dans une forme concise et



Scala

élégante. Son nom vient de l'anglais *Scalable language* qui signifie langage qui peut être mis à l'échelle, langage adaptable. Il intègre les paradigmes de programmation orientée objet et de programmation fonctionnelle. Il est prévu pour être compilé en bytecode Java (exécutable sur la machine virtuelle Java). Si on souhaite l'utiliser exclusivement avec la machine virtuelle Java, il est alors possible d'utiliser les bibliothèques écrites en Java de façon complètement transparente. Ainsi, Scala bénéficie de la maturité et de la diversité des bibliothèques qui ont fait la force de Java depuis une dizaine d'années.

## j. JUnit

JUnit est un framework de test unitaire pour le langage Java. Créé par Kent Beck et Erich Gamma, JUnit est certainement le projet de la série des xUnit ayant le plus de succès. JUnit définit deux types de fichiers de tests. Les TestCase (cas de test) sont des classes contenant un certain nombre de méthodes de tests. Ils servent généralement à tester le bon fonctionnement d'une classe. Une TestSuite permet d'exécuter un certain nombre de TestCase déjà définis. Dans un TestCase, il n'y a pas de main, chaque test est indépendant. De plus, JUnit est intégré par défaut dans les environnements de développement intégré Java, notamment Eclipse.

# JUnit

## k. La Méthode Scrum

Scrum est un schéma d'organisation de développement de produits complexes. Plus simplement, c'est un cadre méthodologique dédié à la gestion de projet ayant pour but d'améliorer la productivité de l'équipe. Il est défini par ses créateurs comme un « cadre de travail holistique itératif qui se concentre sur les buts communs en livrant de manière productive et créative des produits de la plus grande valeur possible ».

Il s'appuie sur le découpage d'un projet en boîtes de temps, nommées « sprints ». Ces derniers peuvent durer entre quelques heures et un mois, mais idéalement ils durent deux semaines. Chaque sprint commence par une estimation de la difficulté de la tâche suivie d'une planification opérationnelle, et se termine par une démonstration de ce qui a été achevé. Avant de démarrer un nouveau sprint, l'équipe réalise une rétrospective. Cette technique analyse le déroulement du sprint achevé, afin d'améliorer ses pratiques. Le flot de travail de l'équipe de développement est facilité par son auto-organisation.

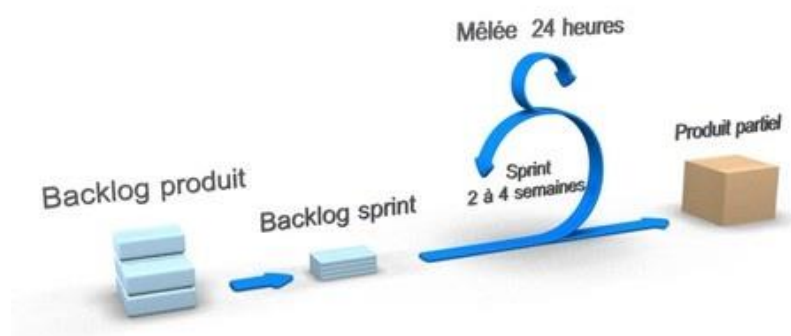


Figure 7 : Schéma de la méthode Scrum

## Répartition des rôles dans Scrum

Scrum définit trois rôles : le propriétaire du produit (*product owner*), le maître de mêlée (*scrum master*) et l'équipe de développement

Propriétaire du produit : c'est le représentant des clients et des utilisateurs. Il doit maximiser la valeur du produit et du travail de l'équipe de développement. Il est seul à orienter l'activité de l'équipe de développement. De ce fait, cet acteur se charge de différents rôles et responsabilités. Il explicite les éléments du carnet du produit, il définit l'ordre dans lequel les fonctionnalités seront développées, il prend les décisions importantes concernant l'orientation du projet et il s'assure que le carnet du produit soit visible et compris par l'équipe de développement.

Maître de mêlée : c'est le responsable de la compréhension, de l'adhésion et de la mise en œuvre du framework. En tant que facilitateur, il aide l'équipe à déterminer quelles interactions avec l'extérieur lui sont utiles, et lesquelles sont freinantes. Il aide alors à maximiser la valeur produite par l'équipe. Ses attributions sont nombreuses, comme communiquer la vision et les objectifs à l'équipe, apprendre au propriétaire du produit à rédiger les composantes du carnet du produit, faciliter les rituels de scrum ou encore coacher l'équipe de développement.

Equipe de développement : constituée de 3 à 9 personnes, elle a pour responsabilité de livrer à chaque fin d'itération une nouvelle version de l'application enrichie de nouvelles fonctionnalités et respectant le niveau de qualité nécessaire pour être livrée. Elle est autoorganisée et choisit la façon d'accomplir son travail, sans que ce soit imposé par une personne externe. Il n'y a pas non plus de notion de hiérarchie interne : toutes les décisions sont prises ensemble. Ce mode d'organisation a pour objectif d'augmenter l'efficacité de travail de l'équipe. Elle est pluridisciplinaire et comporte toutes les compétences pour réaliser son projet, sans faire appel à des personnes externes à celle-ci.

## Les Evènements

### Le Sprint

Le sprint est une période d'un mois au maximum, au bout de laquelle l'équipe délivre un incrément du produit, potentiellement livrable. Une fois la durée choisie, elle reste constante pendant toute la durée du développement. Un nouveau sprint démarre dès la fin du précédent. Chaque sprint possède un but et on lui associe une liste d'éléments du carnet du produit à réaliser.

Pour réaliser un sprint Scrum, on utilise un Scrum Board qui se divise en 4 colonnes distinctes :

- To Do : les tâches qui n'ont pas encore commencé
- In Progress : les tâches qui sont en cours de développement (une par membre d'équipe)
- Test : les tâches qui ont été développées mais qui attendent d'être testées et vérifiées par un autre membre de l'équipe de développement pour confirmer son bon fonctionnement
- Done : les tâches finies et fonctionnelles

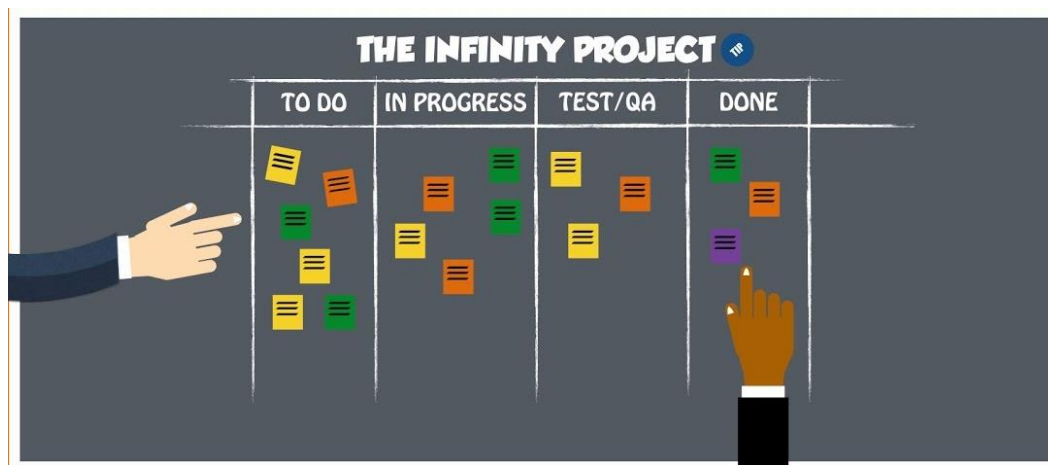


Figure 8 : Scrum Board, schéma d'un sprint

Durant un sprint, l'objet du sprint ne peut être modifié, la composition de l'équipe reste constante, la qualité n'est pas négociable, la liste d'éléments est sujette à négociations entre le propriétaire du produit et l'équipe de développement. Si l'objet du sprint devient obsolète pendant celui-ci, le propriétaire du produit peut décider de l'annuler. Dans ce cas, les développements terminés sont revus par le propriétaire du produit, qui peut décider de les accepter ou non. Les éléments du sprint n'étant pas acceptés sont de nouveau estimés et remis dans le carnet du produit. Un nouveau sprint démarre alors.

## Les Artéfacts

### Le Carnet du Produit (*Product Backlog*)

C'est une liste ordonnée de tout ce qui pourrait être requis dans le produit et l'unique source de besoins pour tous les changements à effectuer sur le produit. Il évolue constamment au cours de la vie du produit et n'est « jamais fini ». Chaque élément de ce dernier représente une fonctionnalité, un besoin, une amélioration ou un correctif, auquel sont associées une description, une estimation de l'effort nécessaire à la réalisation de l'élément et une grandeur permettant d'ordonner les éléments entre eux. L'activité d'affinage du carnet du produit et de ses éléments est effectuée conjointement par le propriétaire du produit et par l'équipe de réalisation. C'est notamment elle seule qui a le mot final sur les estimations des éléments du carnet du produit.

### Le Carnet de Sprint (*Sprint Backlog*)

En début de sprint, un but est décidé. Pour atteindre cet objectif, l'équipe de développement choisit lors de la réunion de planification de sprint quels éléments du carnet de produit seront réalisés. Ces éléments sont alors groupés dans un carnet de sprint. Chaque équipe met à jour régulièrement le carnet de sprint au cours de son activité, afin que celui-ci donne une vision la plus précise possible de ce que l'équipe prévoit de réaliser pour atteindre l'objectif du sprint. Le carnet de sprint est sous la responsabilité de l'équipe et elle seule a le pouvoir de le modifier en cours d'itération.

### L'Incrément du Produit

L'incrément de produit est l'ensemble des éléments du carnet de produit finis pendant ce sprint, et aussi ceux finis pendant les sprints précédents. Les éléments sont finis lorsqu'ils remplissent la définition de fini. Cet incrément doit être utilisable, qu'il soit publié ou non.

## I. L'architecture MVC

L'architecture MVC, ou architecture Modèle-Vue-Contrôleur, est un motif d'architecture logicielle destiné aux interfaces graphiques très populaire pour les applications Web. Le motif est composé de trois types de modules ayant trois responsabilités différentes : les modèles, les vues et les contrôleurs.

- Un modèle (*Model*) contient les données à afficher. Il est lié à la base de données et permet d'en extraire diverses informations, ou alors de lui en transmettre. Sans lui, l'application est vide de données, c'est la base de l'application.
- Une vue (*View*) contient la présentation de l'interface graphique. C'est ce que l'on transmet à l'utilisateur, le visage de l'application en quelque sorte.
- Un contrôleur (*Controller*) contient la logique concernant les actions effectuées par l'utilisateur. En fonction des actions de l'utilisateur, il modifie les données du modèle ou de la vue.

L'architecture MVC peut se résumer via le schéma suivant qui donne les différentes relations et interactions entre le modèle, la vue et le contrôleur.

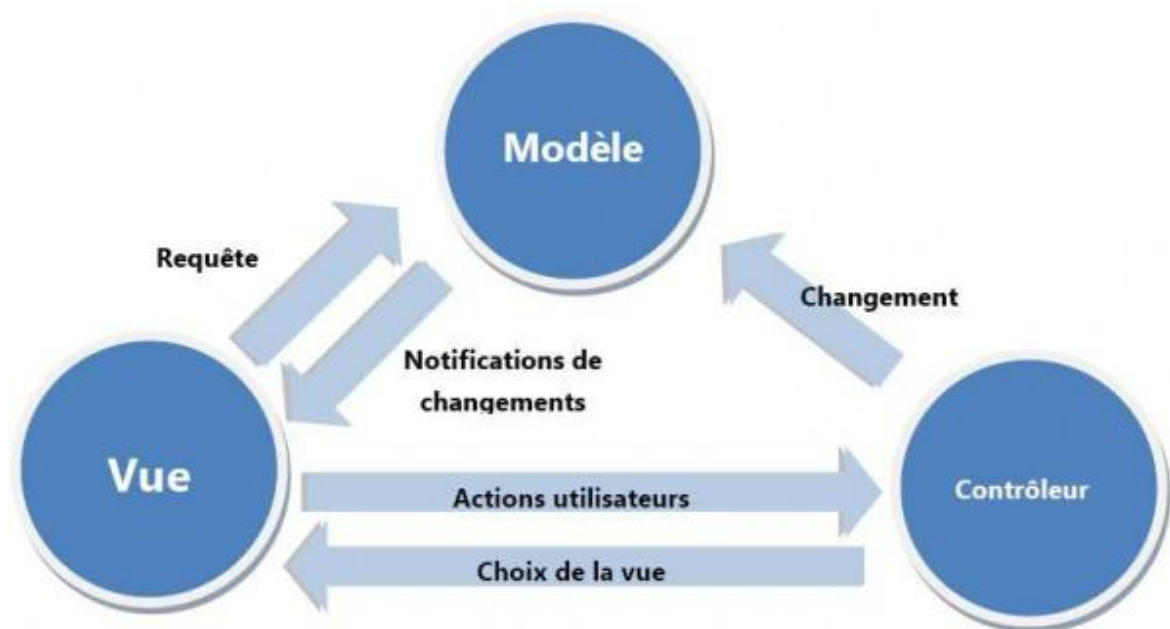


Figure 9 : Schéma de l'architecture MVC

## IV. Réalisation du projet

### a. Démarrage du projet

Dans le but de réaliser notre projet de la meilleure des façons possibles, il a fallu lire de la documentation pour mieux connaître les attentes de notre tuteur, mais également pour effectuer un travail propre et structuré. Les trois premières semaines de notre stage ont été consacrées à se documenter au maximum sur les langages que nous allions utiliser, ainsi qu'à l'exploration de programmes fournis pour comprendre leur fonctionnement. Nous avons d'abord lu le livre Clean Code de Robert Cecil Martin, un livre écrit en anglais expliquant comment réaliser un code sain et facile de compréhension. Nous en avons profité pour faire un résumé en français, que vous trouverez en annexe.

Ensuite, il a fallu préparer les ordinateurs pour nous permettre de développer de la meilleure manière. Contrairement à ce que nous pouvions penser, il ne s'agissait totalement pas d'une partie de plaisir. Installer Eclipse Jee Oxygen, ainsi que tous les plugins fut compliqué selon les différentes versions de Windows, sans compter les soucis avec les diverses versions de Java. Nous utilisons donc Eclipse Jee Oxygen 3A dans sa version 4.7.3a, avec les plugins suivants :

- Subclipse 4.2.3, pour importer et exporter des projets via SVN
- EclEmma Java Code Coverage 3.0.1 pour effectuer les différents tests et repérer aisément les erreurs éventuelles
- Scala IDE 4.7 pour développer en Scala

De plus, nous avons utilisé la version 1.8 de Java, version la plus stable actuellement, ainsi que sbt 1.1.4 pour lancer l'application sur nos ordinateurs. Pour la gestion du dossier SVN qui était mis à notre disposition, nous avons également installé Tortoise.

Pour finir, nous nous sommes rendus sur le site de Play Framework, nous avons téléchargé deux projets et nous avons essayé de comprendre comment ils fonctionnaient. C'est à ce moment qu'on a appris le principe et le fonctionnement de l'architecture MVC. On a également pu découvrir les formulaires de saisie comme par exemple ceux pour l'inscription sur un site, avec des zones de saisie.

A la suite de ce travail de préparation, nous avons reçu le premier carnet de sprint de notre tuteur et nous nous sommes rapidement mis au travail sur le premier sprint. Nous avons listé les tâches, puis nous les avons réparties selon la méthode Scrum. Nous avons à notre disposition un tableau et des post-it, de façon à travailler selon la méthode Scrum.

### b. Les différents sprints réalisés

Nous avons réalisé un total de 3 sprints, d'environ 2 semaines chacun. En voici les carnets de sprint élaborés en début de sprint, après chaque réunion avec notre tuteur, c'est-à-dire la liste des différentes tâches.



Figure 10 : Carnets des Sprints réalisés

Le premier sprint n'était pas tellement comparable aux deux autres, car le sujet n'était pas le site du laboratoire e-Tesc mais un site pour afficher les CV des professeurs de l'Université de Sherbrooke. L'objectif était de créer un site avec un système de connexion et de découvrir le Framework Play ainsi que le fonctionnement du site grâce à ce Framework.

Je me suis personnellement occupé de créer le Template du site, autrement dit les informations de l'onglet, l'en-tête et le pied de page. J'ai également créé un petit fichier CSS pour rendre le site plus agréable aux yeux de l'utilisateur. Emilien s'est occupé de créer et de configurer toute la base de données et également du système de connexion au site. Julien quant à lui s'est chargé de créer la page d'accueil du site, dans laquelle apparaissait un seul CV en statique. Vincent a créé le système pour récupérer son mot de passe lorsqu'on l'a perdu, plus ou moins identique à ceux utilisés dans de nombreux site.



Le deuxième sprint quant à lui s'était axé sur le site du laboratoire e-Tesc, de sorte à le rendre dynamique. Il a donc fallu adapter la page d'accueil du premier sprint, créer la page d'affichage de l'équipe de recherche, créer la page pour consulter ou modifier son profil ou consulter celui d'un membre de l'équipe de recherche, modifier le Template selon le rôle du compte sur lequel on est connecté (administrateur, propriétaire de projets ou utilisateur), mais également les différents boutons sur le menu. De plus, il a fallu trouver un système pour avoir un site disponible en français et en anglais et ajouter la possibilité qu'un administrateur puisse créer des comptes.

Je me suis chargé de modifier le Template, de créer la page d'affichage de l'équipe de recherche et de traduire la partie statique du site, tandis que Julien s'est occupé de modifier la page d'accueil, que Vincent a travaillé sur la page d'affichage et de modification du profil et qu'Emilien s'est occupé de la page de création de comptes.

Le troisième sprint venait en complément du second. Il apportait plusieurs améliorations à ce dernier, et également plusieurs correctifs sur des erreurs effectuées lors de la réalisation du second sprint. Tout d'abord, nous avons dû réorganiser toutes les vues pour respecter l'architecture MVC. Ensuite, il a fallu ajouter, sur toutes les pages dans lesquelles il y avait une zone de saisie de texte (un résumé de projet, une description d'utilisateur...), un champ de saisie pour un texte en français. De cette façon, on avait un texte en français et un texte en anglais dans la base de données, base de données qu'il a fallu modifier en conséquence pour accueillir ce nouveau champ. De plus, on a ajouté la page d'affichage de l'ancienne équipe de recherche étant donné qu'on avait remarqué sur le site actuel que certaines personnes n'étaient plus là mais qu'il pouvait être utile de les intégrer au site si jamais quelqu'un souhaitait avoir des renseignements sur le travail que ces personnes avaient effectué. Enfin, nous avons ajouté une page listant tous les projets du laboratoire de sorte à pouvoir consulter un projet ou un autre.

Je me suis occupé de réorganiser les vues et d'afficher l'ancienne équipe de recherche. Vincent s'est occupé de séparer les pages de consultation et de modification de profil. Julien s'est occupé de la page d'affichage des projets et de la modification de la page d'accueil. Emilien s'est occupé de modifier la base de données et a également ajouté quelques images au site.

### c. Le Template

Le Template se divise en 4 parties : les paramètres de l'onglet, la bannière, l'en-tête et le pied de page. Par la suite j'appellerai l'en-tête « *Header* » et le pied de page « *Footer* ». Voici un aperçu du Template avec la page d'accueil :

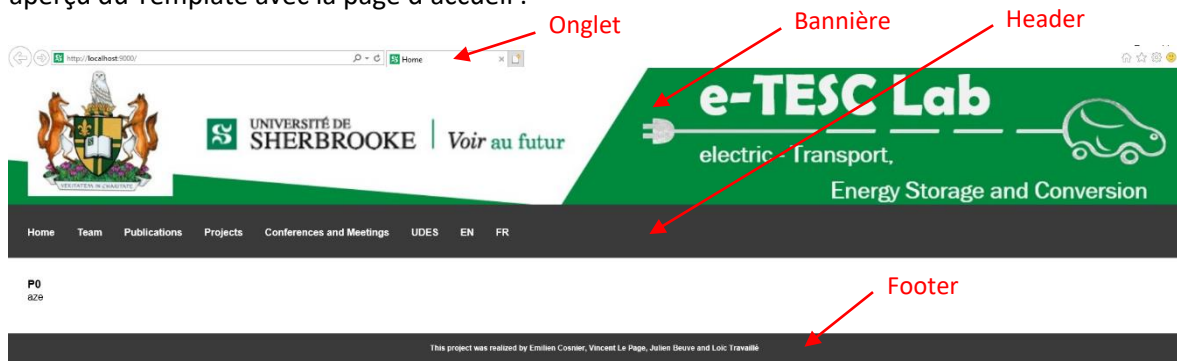


Figure 11 : Page d'accueil du site réalisé

Lors du premier sprint, le Template ne comprenait pas de bannière. Il y avait seulement un Header avec un menu comportant des boutons, adaptés au premier sprint backlog, permettant de naviguer au travers du site et un Footer fixe, tel qu'il est sur l'aperçu ci-dessus. A chaque appui sur un bouton du Header, on renvoyait vers une route permettant d'accéder au contrôleur qui affiche la page souhaitée. De plus, le Header variait en fonction d'un paramètre : la variable de session permettant de savoir si quelqu'un était connecté sur le site ou non. Si quelqu'un était connecté, on affichait le bouton *Logout*, sinon on affichait le bouton *Login*. On avait donc le rendu suivant :



Figure 12 : Aperçu du Header du premier sprint



Figure 13 : Code de la vue du Header du premier sprint

Le Template était en fait un ensemble de 3 fichiers, le Header que vous avez vu au-dessus, le Footer et un fichier qui regroupait ce Header et ce Footer, le contenu de la page à afficher et qui définissait l'onglet. Dans ce fichier, qu'on a appelé Template, on définissait l'icône de l'onglet, le titre de l'onglet mais également les fichiers utiles au site, c'est-à-dire les fichiers CSS et JavaScript, via la route `Assets.versioned(file)` permettant d'ajouter au site et prédéfinie par Play.

```
1 @*Template*@
2
3 @(title: String)(content: Html)
4
5 <!DOCTYPE html>
6 <html lang="en">
7 <head>
8   <title>@title</title>
9   <link rel="stylesheet" media="screen" href="@routes.Assets.versioned("stylesheets/main.css")">
10  <link rel="shortcut icon" type="image/png" href="@routes.Assets.versioned("images/icon.png")">
11  <script src="@routes.Assets.versioned("javascripts/hello.js")" type="text/javascript"></script>
12 </head>
13
14 <body>
15   @Header()
16   <div class="content">
17     @content
18   </div>
19   @Footer()
20 </body>
21 </html>
```

Récupération du titre de l'onglet et du contenu à afficher

Définition des fichiers utiles au site (CSS, JavaScript) et de l'icône et du titre de l'onglet

Appel du Header, du contenu et du Footer

Figure 14 : Code du Template du premier Sprint

Chaque contrôleur qui servait à afficher une page utilisait une méthode d'appel du Template avec deux paramètres, le titre de la page ainsi que le contenu de la page à afficher. Par exemple, pour la page d'accueil on avait le code suivant :

```
6 public class HomeController extends Controller {
7
8   public Result home() {
9     return ok(views.html.Template.render("Home", views.html.Home.render()));
10   }
11 }
12
13 }
```

Figure 15 : Code du contrôleur de la Page d'Accueil

La méthode `ok(request)` utilisée ci-dessus est une méthode Scala permettant d'afficher la page Web dans l'application utilisée.

Au deuxième sprint, nous sommes rattachés au site du laboratoire e-Tesc. Nous avons mis en place le système de rôles et donc le Header variait beaucoup en fonction du rôle de la personne connectée (administrateur, propriétaire ou utilisateur). Nous avons choisi de modifier la couleur du Header en fonction de ce rôle. De plus, les boutons du menu n'étaient pas tout à fait les mêmes pour tous. L'administrateur était le seul à pouvoir créer un compte pour quelqu'un d'autre notamment, il avait donc un bouton en plus que les autres utilisateurs. On récupérait donc la variable de session pour adapter le Header. Via le fichier CSS, on modifiait la couleur du Header selon cette même variable de session. De plus, j'avais adapté le menu pour afficher les boutons permettant d'avoir le site en français et en anglais.

Voici un aperçu des 4 différents Headers selon les 4 possibilités, dans l'ordre des droits décroissants (administrateur, propriétaire, utilisateur, non-connecté) ainsi que le code rédigé :



Figure 16 : Aperçus du Header selon les différents rôles du deuxième sprint

```

3  @if(session("admin")!=null) {
4      <header class="admin">
5  }
6
7  @if(session("owner")!=null) {
8      <header class="owner">
9  }
10
11 @if(session("owner")==null && session("admin")==null)
12 {
13     <header>
14 }
15
16 <div class="menu">
17     <a href=@routes.HomeController.home() id="lhome"></a>
18     <a href=@routes.TeamController.team() id="lteam"></a>
19     <a href=@routes.PublicationsController.publications id="lpublications"></a>
20     <a href=@routes.ProjectsController.projects id="lprojects"></a>
21     <a href=@routes.ConferencesController.conferences id="lconf"></a>
22     <a href="https://www.usherbrooke.ca">UDES</a>
23     <a href=@routes.HomeController.changeLanguage("english") class="Language" id="en">EN</a>
24     <a href=@routes.HomeController.changeLanguage("french") class="Language" id="fr">FR</a>
25 </div>
26
27 <div class="user">
28     @if(session("user")==null){
29         <a href=@routes.LoginController.loginForm id="llogin"></a>
30     }
31
32     @if(session("admin")!=null) {
33         <a href=@routes.RegisterController.registerForm id="lregister"></a>
34         <a href=@routes.ProfileController.profile(session.get("name")) id="lprofile"></a>
35         <a href=@routes.LoginController.disconnection id="llogout"></a>
36     }
37
38     @if(session("owner")!=null) {
39         <a href=@routes.ProfileController.profile(session.get("name")) id="lprofile"></a>
40         <a href=@routes.LoginController.disconnection id="llogout"></a>
41     }
42
43     @if(session("admin")==null && session("owner")==null && session("user")!=null) {
44         <a href=@routes.ProfileController.profile(session.get("name")) id="lprofile"></a>
45         <a href=@routes.LoginController.disconnection id="llogout"></a>
46     }
47 </div>
48 </header>

```

Test de la variable de session pour la couleur du Header

Test de la variable de session pour l'affichage des différents boutons

Figure 17 : Code du Header du deuxième sprint

Les messages flash de connexion étaient toujours là, même si j'ai choisi ici de ne pas vous les faire apparaître dans le code étant donné qu'ils sont identiques à ceux du premier sprint.

On a également introduit la traduction au site c'est pourquoi il y a beaucoup moins de texte en dur sur le code précédent. La traduction se faisant dans un fichier JavaScript, il a fallu ajouter une ligne supplémentaire dans le fichier Template pour importer ce fichier JavaScript et l'utiliser, mais nous verrons ça en détail dans la partie consacrée à la traduction que vous pourrez trouver dans la suite de ce rapport.

Lors de la réunion de fin du deuxième sprint, nous nous sommes aperçus que nous ne respections pas parfaitement l'architecture MVC car, comme vous avez pu le voir dans les codes précédents, nous faisons des tests dans les vues pour savoir qui était connecté. De ce fait, et pour une compréhension accrue du projet, nous avons « dupliqué » le Header en 4 Headers différents, un par rôle, de même pour les Templates. De ce fait, nous ne testions plus la variable de session dans les vues mais nous la testions dans les contrôleurs. Cependant, nous étions toujours obligés de faire un test pour afficher le message flash de connexion réussie ou échouée. Nous avons donc choisi, pour mieux se retrouver dans les fichiers du projet, de créer un dossier regroupant tous les Headers et un dossier regroupant tous les Templates. De plus, nous avons fait le choix de ne plus changer la couleur du Header en fonction du rôle du compte sur lequel nous sommes connecté mais d'afficher un message pour encore mieux savoir le rôle du compte. De plus, l'ajout de la bannière a été effectué lors de ce sprint, via une simple ligne de code. Voici donc le code du HeaderAdmin et celui du Header lorsque l'on n'est pas connecté au site :

```
1 @*Header for Admin*@
2
3 @<header class="admin">
4     
5     <div class="link admin">
6         <div class="menu">
7             <a href="@routes.HomeController.home()" id="mhome"></a>
8             <a href="@routes.TeamController.team()" id="mteam"></a>
9             <a href="@routes.PublicationsController.publications()" id="mpublications"></a>
10            <a href="@routes.ProjectsController.projects()" id="mprojects"></a>
11            <a href="@routes.ConferencesController.conferences()" id="mconferences"></a>
12            <a href="https://www.usherbrooke.ca">UDES</a>
13            <a href="@routes.HomeController.changeLanguage("english")" class="Language" id="en">EN</a>
14            <a href="@routes.HomeController.changeLanguage("french")" class="Language" id="fr">FR</a>
15        </div>
16    </div>
17    <div class="user">
18        <a href="@routes.RegisterController.registerForm()" id="mregister"></a>
19        <a href="@routes.ProfileController.profile(session.get("name"))" id="mprofile"></a>
20        <a href="@routes.LoginController.disconnection()" id="mlogout"></a>
21    </div>
22 </div>
23 </header>
24 <div>
25     <p class="role texte" id="Ladminmessage">
26 </div>
```

Ajout de la bannière au site

Figure 18 : Code du HeaderAdmin du troisième sprint

```
1 @*Header for Not Logged*@
2
3 @<header>
4     
5     <div class="link">
6         <div class="menu">
7             <a href="@routes.HomeController.home()" id="mhome"></a>
8             <a href="@routes.TeamController.team()" id="mteam"></a>
9             <a href="@routes.PublicationsController.publications()" id="mpublications"></a>
10            <a href="@routes.ProjectsController.projects()" id="mprojects"></a>
11            <a href="@routes.ConferencesController.conferences()" id="mconferences"></a>
12            <a href="https://www.usherbrooke.ca">UDES</a>
13            <a href="@routes.HomeController.changeLanguage("english")" class="Language" id="en">EN</a>
14            <a href="@routes.HomeController.changeLanguage("french")" class="Language" id="fr">FR</a>
15        </div>
16    </div>
17 </header>
```

Figure 19 : Code du Header non connecté du troisième sprint

Le fichier Template ayant été lui aussi dupliqué en 4 fichiers distincts, *TemplateAdmin* fait appel au *HeaderAdmin*, *TemplateOwner* fait appel au *HeaderOwner*, *TemplateUser* fait appel au *HeaderUser* et *Template* fait appel au *Header* (non connecté).

Dans ces fichiers, le seul test effectué était celui concernant la langue du site. Il aurait fallu doubler le nombre de pages et tester la variable de session dans les contrôleurs, mais, contrairement aux tests de rôle, cela ne gênait en rien la compréhension du fonctionnement du site donc nous pouvions laisser le test de la variable de langue tel quel. Cependant, comme nous ajoutons directement le contenu dans le Template, il a fallu modifier tous les contrôleurs pour renvoyer vers le bon Template selon la variable de session. Nous avons choisi d'utiliser un *switch* variant selon la variable de session. De ce fait, nous avons un code qui ressemblait à celui-ci (page de l'équipe de recherche du laboratoire) :

```
public Result team()
{
    session("project", "P0");
    Project ptest = Project.findById("P0");
    List<Users> listActive = ProjectUserParticipation.getProjectsUserActive(ptest);

    Result page=null;

    if(session("user")!=null) {
        switch(session("user")) {
            case "admin":    page= ok(views.html.templates.TemplateAdmin.render("Team",views.html.teams.TeamGestion.render(listActive,Status.find.all())));
                           break;
            case "owner":    page= ok(views.html.templates.TemplateOwner.render("Team",views.html.teams.TeamGestion.render(listActive,Status.find.all())));
                           break;
            case "user":     page= ok(views.html.templates.TemplateUser.render("Team",views.html.teams.Team.render(listActive,Status.find.all())));
                           break;
            default :       page= ok(views.html.templates.Template.render("Team",views.html.teams.Team.render(listActive,Status.find.all())));
                           break;
        }
    } else {
        page= ok(views.html.templates.Template.render("Team",views.html.teams.Team.render(listActive,Status.find.all())));
    }
    return page;
}
```

Test de la variable de session

Template différent selon le rôle

Figure 20 : Code du contrôleur de la page de l'équipe de recherche du laboratoire

Grâce aux diverses évolutions du Template, j'ai pu réussir à développer un Template parfaitement fonctionnel et respectant le *product backlog*.

#### d. La Page de l'Équipe de Recherche du Laboratoire / La Page Team

A partir du deuxième sprint, j'ai développé la page affichant les membres de l'équipe de recherche du laboratoire. Ecrite en statique sur le site de base du laboratoire, je devais faire en sorte que cette liste de personne soit liée à la base de données. Emilien a créé dans la base de données une table *project\_user\_participation* qui comportait plusieurs éléments dont voici la liste :

- *id* (clé primaire de la table, incrémente à chaque élément ajouté)
- *user\_participationcip* (identifiant d'un utilisateur)
- *project\_participationname* (nom du projet)
- *activity* (1 si la personne est dans le projet, -1 sinon)

Il a également écrit une partie du modèle qui permettait d'écrire dans la base de données ou d'en extraire des informations clés.

Dans le contrôleur chargé de l'affichage et du traitement de la page Team, je venais récupérer une liste d'utilisateurs grâce à une méthode du modèle que j'avais développée. J'envoyais ensuite cette liste et également la liste de tous les statuts (Professeurs, Doctorants ; Techniciens...) dans la page de façon à l'afficher. Voici donc le code du contrôleur en question, suivi du code du modèle utilisé pour récupérer la liste des participants du projets :

```
public Result team() {
    session("project", "P0");
    Project ptest = Project.find.byId("P0");
    List<Users> = listTeam = ProjectUserParticipation.getProjectsUser(ptest);
    return ok(views.html.Template.render("Team", views.html.Team.render(listTeam, Status.find.all())));
}
```

Figure 21 : Code du contrôleur de la page Team

```
public static List<Users> getProjectsUser(Project project) {
    List<ProjectUserParticipation> listParticipations = ProjectUserParticipation.find.all();
    ArrayList<Users> listUsers = new ArrayList<Users>();

    for (int i = 0; i < listParticipations.size(); i++) {
        if (project.equals(listParticipations.get(i).getProjectPaticipation())) {
            listUsers.add(listParticipations.get(i).getUserPaticipation());
        }
    }
    return listUsers;
}
```

Figure 22 : Code de la méthode du modèle récupérant la liste des participants du projet

Ce deuxième code teste via la méthode *equals()* si l'identifiant d'un utilisateur quelconque est identique à celui d'un participant du projet et permet donc de recouper ces listes de sorte à obtenir une liste d'utilisateurs et non pas une liste de participations. A noter que les méthodes *Status.find.all()* et *ProjectUserParticipation.find.all()* fonctionnent de la même manière et permettent de récupérer tous les éléments contenus dans leurs tables respectives (*Status* et *ProjectUserParticipation*).

```
1 @*Team Page*@
2
3 @(listTeam : List<Users>, listStatus : List<Status>)
4
5 @defining(play.core.PlayVersion.current) { version =>
6   <h1 id="lteampage"></h1>
7   <div id="teamList">
8     @for(status <- listStatus) {
9       <h2>@status.getName()</h2>
10      <div class="teamElement">
11        @for(membre <- listTeam) {
12          @if(status==membre.getStatus()) {
13            <div class="userTeam">
14              <a href=@routes.ProfileController.profile(membre.getCip())><h3>@membre.getFirstName() @membre.getLastName()</h3></a>
15            </div>
16          }
17        }
18      </div>
19    }
20  </div>
21
22  @if(session("owner")!= null) {
23    <a href=@routes.TeamController.editTeam><button type="submit" id="Ledit"></button></a>
24  }
25 }
```

Figure 23 : Code de la vue de la page Team

Dans ce code HTML ci-dessus, il me suffisait de parcourir la liste des statuts via une boucle for et pour chacun tester si le statut de chaque utilisateur récupéré était égal au statut cherché. On avait donc une boucle for dans laquelle il y avait un if. Si le statut d'une personne était égal au statut parcouru, on affichait son nom sous forme de lien pour accéder à son profil. De plus, j'avais ajouté un bouton en bas de la page permettant au propriétaire du projet d'accéder à la page de modification de l'équipe de recherche. A noter qu'un des soucis majeurs lors de la réalisation de cette page était les espaces dans les noms des statuts. Nous avons donc pris la décision de supprimer les espaces pour nous simplifier la tâche. Voici donc le rendu que j'ai obtenu de la page seule, sans Header et sans Footer :

## Team

**Professors • MScStudents • BScStudents • Ph.D.Students • Lab.Technicians • TeachingAssistants**

[ouiii ouiiii](#)

Figure 24 : Vue de la page Team

A noter que je n'étais pas connecté en tant que propriétaire lorsque j'ai fait cette capture d'écran, c'est pourquoi on ne voit pas le bouton pour modifier l'équipe. Il n'y a également qu'une seule personne dans le projet, un Professeur nommé ouiii car on était encore en plein développement et que c'était une page de test.

En ce qui concerne la page de modification de l'équipe du laboratoire, j'avais tout simplement choisi d'afficher la liste de tous les utilisateurs du site, et lorsque l'on cliquait sur leur nom on était redirigé vers leur profil et on pouvait alors modifier leur statut et leur rôle, mais également les retirer du projet.

```
public Result editTeam() {  
    if(session("owner")!=null) {  
        return ok(views.html.Template.render("Edit Team",views.html.EditTeam.render(Users.find.all())));  
    } else {  
        flash(ConstantsMessagesFlash.ERROR,ConstantsMessagesFlash.REGISTER_FORM_NOT_ADMIN);  
        return redirect(routes.TeamController.team());  
    }  
}
```

Figure 25 : Code du contrôleur de la page de modification de l'équipe du laboratoire

On peut noter que si un utilisateur n'étant pas propriétaire tentait d'accéder à cette page en tapant /EditTeam dans l'URL, le test de la variable de session permettait de le renvoyer sur la page Team en lui indiquant, via un message d'erreur, qu'il n'avait pas accès à cette page.



Lors de la réunion de fin du deuxième sprint, notre tuteur nous a dit qu'il serait bien de séparer les personnes qui ont travaillé sur le projet et celles qui travaillent dessus actuellement. J'ai donc choisi de faire une page avec les membres actifs, et une page avec les anciens membres du laboratoire. Il y avait donc la page Team, avec les membres actifs, et la page Past Team avec les anciens membres. Emilien avait modifié le modèle de sorte que la valeur *activity* de la table *ProjectUserParticipation* puisse avoir 3 valeurs : 1 si la personne est active dans le projet, 0 si elle est ancienne au projet et -1 pour tout le reste. Le contrôleur de la page Team ne fait donc plus appel à la méthode *getProjectsUser* du modèle *ProjectUserParticipation* qui récupérait toutes les participations différentes de -1, mais fait maintenant appel à la méthode *getProjectsUserActive*, qui récupère les participations ayant une activité à 1, de la même façon que le contrôleur de la page Past Team fait appel à la méthode *getProjectsUserInactive* qui récupère les participations ayant une activité à 0. Voici donc les deux méthodes utilisées ici, extraites du modèle *ProjectUserParticipation* :

```
public static List<Users> getProjectsUserInactive(Project project) {  
    List<ProjectUserParticipation> listParticipations = ProjectUserParticipation.find.all();  
    ArrayList<Users> listInactive = new ArrayList<Users>();  
  
    for(int i=0 ; i < listParticipations.size() ; i++) {  
        if(project.equals(listParticipations.get(i).getProjectParticipation())) {  
            if(listParticipations.get(i).getActivity()==0) {  
                listInactive.add(listParticipations.get(i).getUserParticipation());  
            }  
        }  
    }  
    return listInactive;  
}
```

Figure 26 : Code de la méthode du modèle récupérant la liste des participants inactifs du projet

```
public static List<Users> getProjectsUserActive(Project project) {  
    List<ProjectUserParticipation> listParticipations = ProjectUserParticipation.find.all();  
    ArrayList<Users> listActive = new ArrayList<Users>();  
  
    for (int i = 0; i < listParticipations.size(); i++) {  
        if (project.equals(listParticipations.get(i).getProjectParticipation())) {  
            if (listParticipations.get(i).getActivity() == 1) {  
                listActive.add(listParticipations.get(i).getUserParticipation());  
            }  
        }  
    }  
    return listActive;  
}
```

Figure 27 : Code de la méthode du modèle récupérant la liste des participants actifs du projet

On récupère donc deux listes, l'une affichée sur la page Team et l'autre sur la page Past Team, en utilisant le même principe que le contrôleur et la vue du deuxième sprint. Emilien avait repris la page EditTeam que j'avais créé pour faire en sorte d'ajouter des personnes au projet directement sur cette page et non pas en relayant vers leur profil.

Grâce à ces différentes évolutions, j'ai pu parvenir à créer deux pages totalement fonctionnelles pour l'affichage des membres actuels du laboratoire et des anciens membres.

#### e. Les tests unitaires relatifs aux pages Team

Il est important de tester son programme via des tests unitaires. Ils permettent de vérifier une faille qu'on aurait potentiellement oubliée, mais ils sont principalement là pour vérifier que tout fonctionne parfaitement et comme on le souhaite. J'ai donc développé plusieurs tests permettant de vérifier si les pages Team fonctionnent (Team, PastTeam et EditTeam). Voici le code Java des tests effectués :

```
@Before
public void before()
{
    app = fakeApplication();
    request = new Http.RequestBuilder();
    start(app);
}

@Test
public void renderTeam()
{
    request = new Http.RequestBuilder().method("GET").uri(routes.TeamController.team().url());
    Result result = route(app,request);
    assertEquals("This is not the page", "/Team", request.uri());
    assertEquals(OK, result.status());
}

@Test
public void renderPastTeam()
{
    request = new Http.RequestBuilder().method("GET").uri(routes.TeamController.pastTeam().url());
    Result result = route(app,request);
    assertEquals("This is not the page", "/PastTeam", request.uri());
    assertEquals(OK, result.status());
}

@Test
public void testToGoEditTeamNotOwner()
{
    request = new Http.RequestBuilder().method("GET").uri(routes.TeamController.editTeam().url());
    Result result = route(app,request);
    assertEquals("This is not the page", "/Team", result.redirectLocation().get());
    assertEquals(303, result.status());
}

@After
public void teardown()
{
    stop(app);
}
```

Initialisation d'une fausse application

Test de l'existence de la page Team et d'un accès possible pour tous

Test de l'existence de la page PastTeam et d'un accès possible pour tous

Test de l'existence de la page EditTeam et d'un accès impossible lorsque l'on n'est pas connecté

Arrêt de la fausse application

Figure 28 : Code des tests des pages Team

Les méthodes `assertEquals()` sont définies par défaut dans JUnit, c'est pour cela que nous les utilisons. Elles sont très pratiques et retournent `True` or `False`. Si le résultat est `True`, le test est validé et réussi. Si le résultat est `False`, c'est un échec, il faut donc corriger le problème.

Dans les codes ci-dessus on peut voir OK et 303 qui diffèrent entre les deux premiers tests et le troisième. OK correspond à un accès possible à la page. 303 correspond à une redirection à la suite d'un problème, ici le manque de droits.

Il suffit ensuite d'exécuter les tests dans l'invite de commandes de Windows, et on obtient le résultat suivant :

```
[info] Test projetPropre.TeamTest.renderPostTeam started
[info] application - Creating pool for datasource 'default'
[warn] application - Application.conf @ file:/D:/ProjetsS20/JAVA/eclipse-workspace/ProjetPropre/target/scala-2.12/classes/application.conf: 364: applyEvolutions.default is deprecated, use play.evolutions.db.default.autoApply instead
[info] application - ApplicationTimer demo: Starting application at 2018-06-14T15:07:16.994Z
[warn] o.h.v.m.ParameterMessageInterpolator - MW000184: ParameterMessageInterpolator has been chosen, El interpolation will not be supported
[info] application - Creating pool for datasource 'default'
[warn] application - Application.conf @ file:/D:/ProjetsS20/JAVA/eclipse-workspace/ProjetPropre/target/scala-2.12/classes/application.conf: 364: applyEvolutions.default is deprecated, use play.evolutions.db.default.autoApply instead
[info] application - ApplicationTimer demo: Starting application at 2018-06-14T15:07:18.552Z
[warn] o.h.v.m.ParameterMessageInterpolator - MW000184: ParameterMessageInterpolator has been chosen, El interpolation will not be supported
[info] play.api.Play - Stopping current application
[info] application - ApplicationTimer demo: Stopping application at 2018-06-14T15:07:18.788Z after 2s.
[info] application - Shutting down connection pool.
[info] application - ApplicationTimer demo: Stopping application at 2018-06-14T15:07:19.517Z after 1s.
[info] application - Shutting down connection pool.
[info] Test projetPropre.TeamTest.testOfOdditTeamOwner started
[info] application - Creating pool for datasource 'default'
[warn] application - Application.conf @ file:/D:/ProjetsS20/JAVA/eclipse-workspace/ProjetPropre/target/scala-2.12/classes/application.conf: 364: applyEvolutions.default is deprecated, use play.evolutions.db.default.autoApply instead
[info] application - ApplicationTimer demo: Starting application at 2018-06-14T15:07:19.738Z
[warn] o.h.v.m.ParameterMessageInterpolator - MW000184: ParameterMessageInterpolator has been chosen, El interpolation will not be supported
[info] application - Creating pool for datasource 'default'
[warn] application - Application.conf @ file:/D:/ProjetsS20/JAVA/eclipse-workspace/ProjetPropre/target/scala-2.12/classes/application.conf: 364: applyEvolutions.default is deprecated, use play.evolutions.db.default.autoApply instead
[info] application - ApplicationTimer demo: Starting application at 2018-06-14T15:07:20.026Z
[warn] o.h.v.m.ParameterMessageInterpolator - MW000184: ParameterMessageInterpolator has been chosen, El interpolation will not be supported
[info] play.api.Play - Stopping current application
[info] application - ApplicationTimer demo: Stopping application at 2018-06-14T15:07:20.128Z after 1s.
[info] application - Shutting down connection pool.
[info] application - ApplicationTimer demo: Stopping application at 2018-06-14T15:07:20.218Z after 0s.
[info] application - Shutting down connection pool.
[info] Test projetPropre.TeamTest.renderTeam started
[info] application - Creating pool for datasource 'default'
[warn] application - Application.conf @ file:/D:/ProjetsS20/JAVA/eclipse-workspace/ProjetPropre/target/scala-2.12/classes/application.conf: 364: applyEvolutions.default is deprecated, use play.evolutions.db.default.autoApply instead
[info] application - ApplicationTimer demo: Starting application at 2018-06-14T15:07:20.402Z
[warn] o.h.v.m.ParameterMessageInterpolator - MW000184: ParameterMessageInterpolator has been chosen, El interpolation will not be supported
[info] application - Creating pool for datasource 'default'
[warn] application - Application.conf @ file:/D:/ProjetsS20/JAVA/eclipse-workspace/ProjetPropre/target/scala-2.12/classes/application.conf: 364: applyEvolutions.default is deprecated, use play.evolutions.db.default.autoApply instead
[info] application - ApplicationTimer demo: Starting application at 2018-06-14T15:07:20.621Z
[warn] o.h.v.m.ParameterMessageInterpolator - MW000184: ParameterMessageInterpolator has been chosen, El interpolation will not be supported
[info] application - ApplicationTimer demo: Stopping application at 2018-06-14T15:07:20.693Z after 0s.
[info] application - Shutting down connection pool.
[info] application - ApplicationTimer demo: Stopping application at 2018-06-14T15:07:20.752Z after 0s.
[info] application - Shutting down connection pool.
[info] Test projetPropre.TeamTest.renderTeam started
[info] application - Creating pool for datasource 'default'
[warn] application - Application.conf @ file:/D:/ProjetsS20/JAVA/eclipse-workspace/ProjetPropre/target/scala-2.12/classes/application.conf: 364: applyEvolutions.default is deprecated, use play.evolutions.db.default.autoApply instead
[info] application - ApplicationTimer demo: Starting application at 2018-06-14T15:07:20.802Z
[warn] o.h.v.m.ParameterMessageInterpolator - MW000184: ParameterMessageInterpolator has been chosen, El interpolation will not be supported
[info] application - Creating pool for datasource 'default'
[warn] application - Application.conf @ file:/D:/ProjetsS20/JAVA/eclipse-workspace/ProjetPropre/target/scala-2.12/classes/application.conf: 364: applyEvolutions.default is deprecated, use play.evolutions.db.default.autoApply instead
[info] application - ApplicationTimer demo: Starting application at 2018-06-14T15:07:20.821Z
[warn] o.h.v.m.ParameterMessageInterpolator - MW000184: ParameterMessageInterpolator has been chosen, El interpolation will not be supported
[info] application - ApplicationTimer demo: Stopping application at 2018-06-14T15:07:20.893Z after 0s.
[info] application - Shutting down connection pool.
[info] application - ApplicationTimer demo: Stopping application at 2018-06-14T15:07:20.952Z after 0s.
[info] application - Shutting down connection pool.
```

Figure 29 : résultat des tests des pages Team

Tous les tests ont été effectués avec succès. Les pages Team fonctionnent donc comme nous le souhaitons, avec les redirections nécessaires pour les personnes n'ayant pas accès à la page `EditTeam`.

## f. La Traduction du site

La traduction du site se faisait en deux parties. Nous avons un script qui traduisait tous les textes « en dur » sur le site, c'est-à-dire le menu du header, les titres des pages, les boutons etc..., mais nous avons également fait en sorte de pouvoir accueillir des informations en français et en anglais dans la base de données, notamment les résumés des projets, les résumés des utilisateurs etc... Je me suis personnellement occupé du script qui traduisait les textes « en dur ». Le fonctionnement était simple. Un fichier JavaScript comprenait deux méthodes, la méthode `english()` et la méthode `french()`. Ces méthodes étaient appelées dans les 4 fichiers HTML des Templates. L'utilisateur pouvait changer la langue à tout moment en cliquant sur les boutons FR ou EN situés en haut de la page, dans le menu du Header.

```

1  @*Header for Not Logged*@
2
3  <header>
4      
5      <div class="Link">
6          <div class="menu">
7              <a href="@routes.HomeController.home()" id="mhome"></a>
8              <a href="@routes.TeamController.team()" id="mteam"></a>
9              <a href="@routes.PublicationsController.publications()" id="mpublications"></a>
10             <a href="@routes.ProjectsController.projects()" id="mprojects"></a>
11             <a href="@routes.ConferencesController.conferences()" id="mconferences"></a>
12             <a href="https://www.usherbrooke.ca">UDES</a>
13             <a href="@routes.HomeController.changeLanguage("english")" class="Language" id="en">EN</a>
14             <a href="@routes.HomeController.changeLanguage("french")" class="Language" id="fr">FR</a>
15         </div>
16     </div>
17 </header>
18 <div class="messages">
19     @if(flash.containsKey("success")) {
20         <p class="success texte">@flash.get("success")</p>
21     }
22     @if(flash.containsKey("error")) {
23         <p class="error texte">@flash.get("error")</p>
24     }
25 </div>

```

Figure 30 : Code du Header permettant la traduction

A chaque clic sur un des boutons, nous étions renvoyés vers la méthode `HomeController.changeLanguage(string)`. Cette méthode récupérait le string (*french* ou *english*) pour le placer dans la variable de session *language*. Voici le code du contrôleur en question :

```

public Result changeLanguage(String language) {
    if(session("project") == null) {
        session("project", "P0");
    }
    session("language", language);
    return redirect(routes.HomeController.home());
}

```

Figure 31 : Code du contrôleur utilisé pour la traduction

Il suffisait ensuite de récupérer cette variable de session pour choisir la méthode du script à utiliser. Par exemple, pour le Template utilisé lorsque l'on n'est pas connecté, le code HTML était le suivant :

```

1  @*Template for Not Logged*@
2
3  @(title: String)(content: Html)
4
5  <!DOCTYPE html>
6  <html lang="en">
7      <head>
8          <title>@title</title>
9          <link rel="stylesheet" media="screen" href="@routes.Assets.versioned("stylesheets/main.css")">
10         <link rel="shortcut icon" type="image/png" href="@routes.Assets.versioned("images/icon.png")">
11         <script src="@routes.Assets.versioned("javascripts/hello.js")" type="text/javascript"></script>
12         <script src="@routes.Assets.versioned("javascripts/dictionary.js")" type="text/javascript"></script>
13     </head>
14
15     <body>
16         @headers.Header()
17         <div class="content">
18             @content
19         </div>
20         @Footer()
21     </body>
22     @if(session.get("language")=="french"){
23         <script type="text/javascript">
24             french()
25         </script>
26     }else{
27         <script type="text/javascript">
28             english()
29         </script>
30     }
31 </html>

```

Figure 32 : Code du Template permettant la traduction

Pour ensuite savoir à quoi correspondait chaque élément, chaque bouton, le contenu à afficher dedans, j'ai choisi de mettre « des identifiants » aux différents éléments. Par exemple, dans le Header précédent, on peut voir, dans le menu, que chaque élément a un identifiant attribué grâce à l'attribut id dans sa balise. Par exemple, pour le bouton Home on a *id*= « *mhome* », pour le bouton Team on a *id*= « *mteam* », etc... La suite est simple, il suffisait de développer le fichier JavaScript de sorte qu'il mette du texte pour la balise ayant l'attribut id correspondant. Voici un court extrait de la méthode *french()* ainsi qu'un court extrait de la méthode *english()* du fichier JavaScript :

```
1 function french()
2 {
3     //MENU
4     document.getElementById("mhome").innerHTML = "Accueil";
5     document.getElementById("mteam").innerHTML = "Equipe";
6     document.getElementById("mpublications").innerHTML = "Publications";
7     document.getElementById("mprojects").innerHTML = "Projets";
8     document.getElementById("mconferences").innerHTML = "Conférences et Réunions";
9
10    if(document.getElementById("mregister")!=null)
11    {
12        document.getElementById("mregister").innerHTML = "Créer un compte";
13    }
14
295 function english()
296 {
297     //MENU
298     document.getElementById("mhome").innerHTML = "Home";
299     document.getElementById("mteam").innerHTML = "Team";
300     document.getElementById("mpublications").innerHTML = "Publications";
301     document.getElementById("mprojects").innerHTML = "Projects";
302     document.getElementById("mconferences").innerHTML = "Conferences and Meetings";
303
304     if(document.getElementById("mregister")!=null)
305     {
306         document.getElementById("mregister").innerHTML = "Create account";
307     }
308 }
```

Figure 33 : Extraits du Script de traduction

Via la méthode *getElementById* on récupérait l'élément souhaité et on modifiait son contenu via la méthode *innerHTML*. Cependant, il fallait, pour les éléments n'apparaissant pas sur toutes les pages, tester s'ils étaient présents ou non. C'est pourquoi je vérifiais si l'élément était nul ou non via un *if*. On peut voir au-dessus que j'effectue ce test sur le bouton Register du menu par exemple, car seul l'administrateur y a accès et donc il apparaît uniquement sur le HeaderAdmin mais il n'apparaît pas sur les autres Headers. Les autres boutons comme le bouton Home, le bouton Team, apparaissant sur toutes les pages, je n'ai pas besoin d'effectuer le test de leur existence ou non. Le problème d'un fichier JavaScript est que dès qu'il rencontre une erreur (un élément qui n'existe pas, des guillemets oubliés dans son code...), il s'arrête à l'endroit où il est et donc il faut faire extrêmement attention lors de son écriture. Pour faire ce travail de traduction il faut être méthodique pour ne pas reprendre le même id pour deux éléments différents aussi, sinon cela crée un conflit qui peut être compliqué à corriger sur un script de 600 lignes comme le nôtre.

Voici un aperçu de la page d'accueil du site, dans les deux langues différentes.

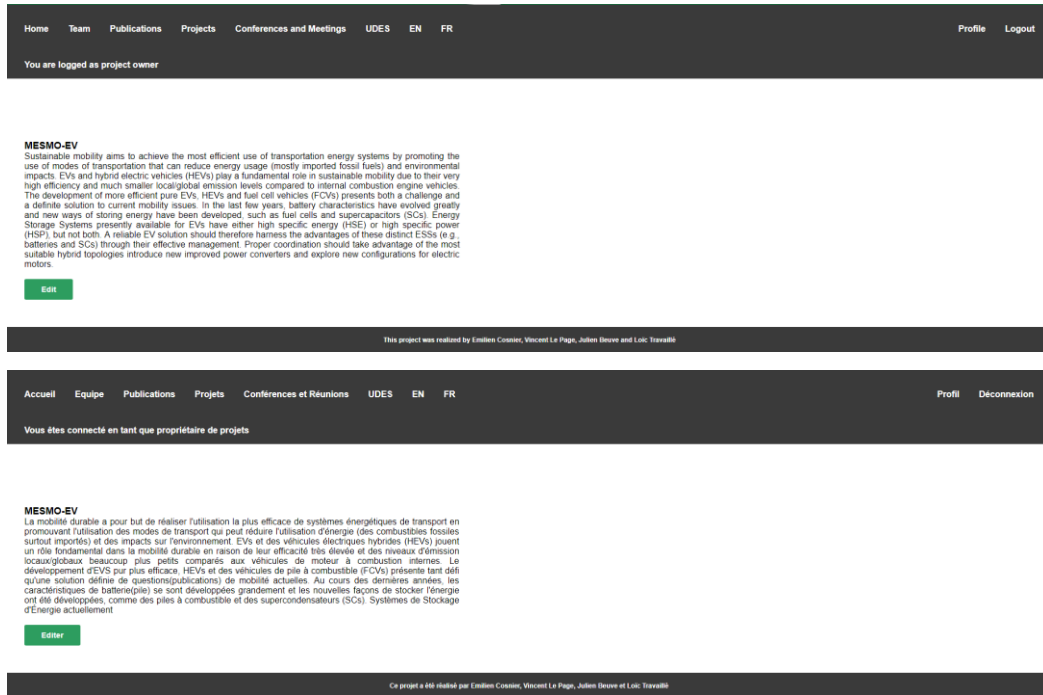


Figure 34 : Aperçu de la page d'accueil du site en anglais et en français

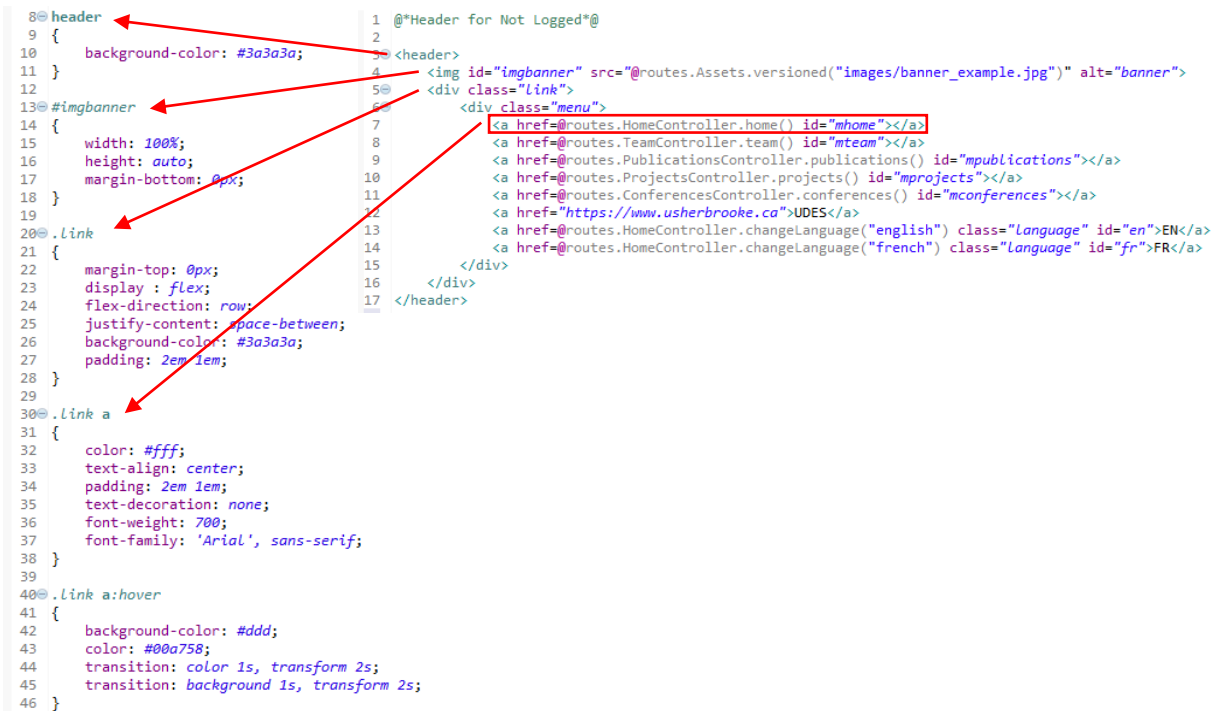
Grâce à ce script, on a pu obtenir un site parfaitement traduit en anglais et en français, mais cependant la traduction pour les résumés des projets ou des utilisateurs ne se faisait pas automatiquement, comme la traduction des boutons. Il fallait saisir des informations en français et en anglais si on souhaitait avoir les deux langues disponibles.

### g. Le fichier CSS

Lors du premier sprint, j'ai programmé une ébauche de fichier CSS pour mettre en forme le site et le rendre plus élégant. Sans fichier CSS, il est beaucoup plus dur de mettre de la couleur, de changer les polices etc...J'ai notamment ajouté la mise en forme du menu, en modifiant la couleur de fond, la disposition des boutons, la police, mais aussi la transition lorsque l'on survole le bouton etc.. J'ai aussi fait en sorte que les titres des pages (hors page d'accueil) soient plus gros, soient en vert...

L'objectif était également de respecter plus ou moins les couleurs de l'Université, avec le gris et le vert notamment. La police est identique à celle utilisée sur le menu du site de l'Université également. Le fichier CSS rentre plus dans un respect de la charte graphique de l'Université. Cependant, nous aurions pu utiliser des logiciels qui auraient développé le fichier CSS à notre place, comme Bootstrap par exemple, mais ce n'était pas le but recherché.

Voici un court extrait du code CSS sur la première image, et le fichier HTML qui lui correspond sur la seconde :



```
8 header
9 {
10   background-color: #3a3a3a;
11 }
12
13 #imgbanner
14 {
15   width: 100%;
16   height: auto;
17   margin-bottom: 0px;
18 }
19
20 .link
21 {
22   margin-top: 0px;
23   display : flex;
24   flex-direction: row;
25   justify-content: space-between;
26   background-color: #3a3a3a;
27   padding: 2em 1em;
28 }
29
30 .link a
31 {
32   color: #fff;
33   text-align: center;
34   padding: 2em 1em;
35   text-decoration: none;
36   font-weight: 700;
37   font-family: 'Arial', sans-serif;
38 }
39
40 .link a:hover
41 {
42   background-color: #ddd;
43   color: #00a758;
44   transition: color 1s, transform 2s;
45   transition: background 1s, transform 2s;
46 }
```

```
1 @*Header for Not Logged*@
2
3 <header>
4   
5   <div class="link">
6     <div class="menu">
7       <a href="@routes.HomeController.home()" id="mhome"></a>
8       <a href="@routes.TeamController.team()" id="mteam"></a>
9       <a href="@routes.PublicationsController.publications()" id="mpublications"></a>
10      <a href="@routes.ProjectsController.projects()" id="mprojects"></a>
11      <a href="@routes.ConferencesController.conferences()" id="mconferences"></a>
12      <a href="https://www.usherbrooke.ca">UDES</a>
13      <a href="@routes.HomeController.changeLanguage("english")" class="Language" id="en">EN</a>
14      <a href="@routes.HomeController.changeLanguage("french")" class="Language" id="fr">FR</a>
15    </div>
16  </div>
17 </header>
```

Figure 35 :Extrait du Code CSS avec son association HTML

Prenons par exemple la ligne 7 dans le code HTML (encadrée en rouge). Son contenu est soumis à plusieurs personnalisations : a, menu, link et header, du plus petit au plus gros. Dans le fichier CSS, on peut voir « la classe mère » header, et la division link, puis le format de texte a. Le contenu est donc personnalisé par toutes ses parties, et étant donné qu'il s'agit d'un bouton, on peut personnaliser le fait qu'il soit survolé par la souris via la fonction :hover.

## V. Conclusion Technique

### a. Fonctionnement final

A l'heure actuel, notre site Web est parfaitement fonctionnel. La création des utilisateurs ainsi que leurs rôles et statuts marche comme nous le souhaitions.

La page d'affichage de l'équipe de recherche fonctionne, celle des anciens membres aussi. L'ajout de projets est possible depuis le site, tout comme la modification et la consultation de ces derniers.

Pendant les jours restants jusqu'à la fin du stage, nous allons nous consacrer à la résolution de quelques soucis mineurs encore existants, mais nous allons également tenter de développer un système pour ajouter des images relatives aux projets depuis le site Web.

### b. Modifications à apporter

Si nous avions eu plus de temps pour réaliser ce projet, nous aurions pu nous charger de créer les pages Publications et Conferences and Meetings, présentes actuellement sur le site du laboratoire e-Tesc. La page Publications aurait pu contenir des articles rédigés par le laboratoire sur l'avancement des projets en cours par exemple. La page Conferences and Meetings, elle, aurait pu contenir des résumés des différentes réunions et conférences effectuées par le laboratoire notamment.

De plus, nous aurions pu faire un site beaucoup plus propre d'un point de vue esthétique, mais cela n'était pas le cœur du sujet de stage. On aurait également pu perfectionner notre système d'ajout d'images au site, que nous étions en train de réaliser après le troisième sprint.

Enfin, il aurait pu être nécessaire et intéressant de créer un système de cookies qui aurait permis à l'utilisateur de notre site Web de ne pas avoir à se connecter à chaque fois.



## VI. Conclusion générale

Pour établir un bilan de ces 11 semaines de stage, ce fut une excellente expérience pour moi, aussi bien sur le plan professionnel que personnel.

### a. Au niveau professionnel

A l'IUT nous avons beaucoup appris à programmer en C ou en VHDL. Grâce à la deuxième année, j'avais pu approfondir mes connaissances en Java, que j'avais découvert au lycée. De plus, je connaissais un peu le langage HTML et le langage CSS, mais je ne m'en étais que très peu servi et j'ai donc pu apprendre de nouvelles choses sur ces deux langages. En ce qui concerne le JavaScript et le SQL, ce sont deux langages que j'ai totalement découvert pendant ce stage étant donné qu'ils étaient tout nouveau pour moi. De plus, la programmation en elle-même était beaucoup plus compliquée que celle apprise à l'IUT.

J'ai pu découvrir une quantité très importante d'outils, mais j'ai également pu assimiler beaucoup de notions tout au long du stage. Toutes ces connaissances me serviront sans aucun doute à l'avenir, mais également la méthode Scrum pour tout travail en équipe sur un projet.

Je suis extrêmement ravi d'avoir effectué ce stage car il m'a permis de confirmer mon choix de poursuite d'études et de projet professionnel. L'année prochaine je pourrai apprendre de nouveaux langages et concepts de programmation en école d'ingénieurs.

### b. Au niveau personnel

Cette expérience était également très enrichissante étant donné qu'elle s'est passée dans un pays étranger. J'ai beaucoup apprécié parler avec les Canadiens, même si cela se faisait souvent en français. J'ai cependant pu parler en anglais avec certaines personnes parfois.

Je tiens une fois de plus à remercier fortement toutes les personnes qui m'ont permis d'effectuer ce stage ainsi que celles que j'ai pu côtoyer durant ces 11 semaines.

J'ai par ailleurs eu l'occasion de découvrir les villes de Québec, Toronto, Montréal, et même New York, qui ont été très enrichissantes d'un point de vue linguistique par notamment.

Je vous remercie pour avoir lu l'intégralité de ce rapport.

## VII. Annexes

### a. L'intégralité du projet

Vous pouvez retrouver l'intégralité du projet tel qu'il est actuellement en cliquant sur ce lien : <http://www.mediafire.com/file/5jwi5pxxc0lrwmt/projetPropre.zip/file>

Cependant, vous ne pourrez pas lancer le site sur votre machine car vous n'aurez pas accès à la base de données et vous aurez besoin d'Eclipse pour ouvrir tous les fichiers.

A noter que le code peut comporter quelques erreurs que nous sommes en train de corriger.

### b. Résumé en français du livre *Clean Code*

#### Chapitre 1 : Clean Code

Avoir un code propre est important. Si celui-ci n'est pas lisible ou trop complexe il sera soit inutilisable soit source d'erreurs lors des mises à jour de l'application. Le code se doit d'être bien agencé. Si une équipe commence un projet sans se soucier de la lisibilité du code de production, sa courbe de productivité sera fortement décroissante avec le temps.

Règle des scouts : "laisser un point de campement aussi propre que possible" Il en va de même avec le code.

#### Chapitre 2 : Convention de nommage

Les noms des variables doivent être significatifs et parler d'eux-mêmes. Dans l'absolu ils doivent être uniques dans chacun des fichiers dans lesquels ces variables se trouvent.

#### Chapitre 3 : Les fonctions

Elles doivent être petites. Elles doivent faire une seule chose. Il n'y a qu'un seul niveau d'abstraction dans chaque fonction. Il faut utiliser des noms descriptifs L'utilisation des switches, du fait de leur nature, font plusieurs choses. Il faut donc les intégrer dans une fonction de bas niveau afin d'éviter qu'elle soit répétée.

Le meilleur cas c'est de n'avoir aucun argument, au maximum il ne faut pas dépasser trois sauf dans des cas très spéciaux. Les fonctions qui ne possèdent qu'un argument servent en général à poser une question (par exemple vérifier si un fichier existe) ou alors pour transformer un argument. L'utilisation d'arguments booléens n'est pas très pratique, car la fonction réalise alors deux choses : l'une quand la condition est vraie et une autre quand elle fausse. Les fonctions qui possèdent deux arguments sont plus compliquées à comprendre que celles qui ne possèdent qu'un argument. On peut les utiliser lorsque l'on définit un point par exemple Il faut préférer l'utilisation des exceptions plutôt que des retours d'erreurs Ne pas se répéter Structurer son programme

#### Chapitre 4 : Les commentaires

L'utilisation des commentaires est due au manque de clarté du programme, le programme doit pouvoir renvoyer votre intention rien que par le nom de ses variables, classes ou fonctions. Dans certain cas il y a des commentaires légaux, ils sont nécessaires (copyright et autorship). Les commentaires ont d'autres utilisations tels que la clarification, l'expression des dangers et conséquences. Les mauvaises utilisations de commentaires sont les commentaires redondants, ceux qui servent à meubler votre code. Les commentaires sont trompeurs quand ils ne sont pas à jour, lorsque l'on met en commentaire des lignes de code que l'on a écrit.

#### Chapitre 5 : Le formatage

Il est important d'avoir un code structuré pour qu'il puisse être aisément modifié et compris.

- 1) Le formatage vertical : le code ne doit pas dépasser un certain nombre de lignes qui dépend du type de fichier manipulé, dans la limite de 500 lignes maximum. Plus le fichier sera petit et concis, plus il sera facile de le lire.
- 2) Les noms : Les noms des fichiers doivent être écrit en langage de haut niveau, et décriront parfaitement la fonction des fichiers. Au sein des fichiers les noms des données seront fonction de leur importance dans la hiérarchie du fichier. Plus on sera bas dans la hiérarchie plus on pourra se permettre de nommer les variables avec des noms "abstraits".
- 3) La séparation des concepts : Chaque concept au sein d'un fichier sera séparé d'un blanc afin de faciliter la lecture et de "chapitrer" le code . Ceci permettra de distinguer plus rapidement les différents concepts d'un fichier.

- 4) La distance verticale : Les variables doivent être déclarées le plus près possible de l'endroit où elles seront utilisées (cas particulier des variables d'instance, celles-ci devront toutes être déclarées au même endroit, en Java en haut de la classe). Si une fonction ou méthode en appelle une autre elles devront également être proches.
- 5) L'ordre vertical : Les fonctions les plus importantes devront être placées le haut possible dans le code source.
- 6) Le formatage horizontal : Les lignes ne doivent pas être trop chargées, privilégier des instructions courtes, quitte à devoir séparer l'instruction en deux parties.
- 7) L'alignement : Les données doivent toutes avoir le même alignement. Important : se mettre d'accord sur les normes de présentation de code lorsque l'on travaille en équipe.

## Chapitre 6

Tout d'abord les données doivent être abstraites c'est à dire qu'un étranger lisant le code ne doit pas savoir précisément à quelle utilisation correspond chaque variable pour qu'il ne puisse pas utiliser les accesseurs simplement.

Polymorphisme ou procédurale ?

Polymorphisme : Square rectangle and circle with calcul méthode inside. Ajouter des classes est donc plus facile mais les fonctions sont toutes modifiées.

Procédurale et structure de données : Square rectangle circle then méthode calcul. Ajouter des fonctions est donc plus simple mais la classe est plus lourde.

Loi de Démeter : Un module ne doit pas connaître les liens internes des objets manipulés donc les accesseurs des objets vont à l'inverse de ce principe

"Ne parlez qu'à vos amis immédiats" Objet A ne peut pas faire appel à B pour connaître C

Une méthode d'un objet peut appeler :

- Son objet

- Les paramètres de la méthode

- Les objets créés ou instanciés par la méthode

- Les objets membres de son objet

Avantage -> les objets sont moins dépendants de la structure interne des autres objets

Désavantage -> grand nombre de petites méthodes

DTO -> classe avec variable publique et sans fonction

Beans -> Classe avec variable privée et accesseurs

### Chapitre 7 : Les exceptions

Dans chaque programme, il y a des exceptions qui apparaissent à la suite d'entrées anormales ou les appareils peuvent échouer. Le but d'une exception est de permettre au code de continuer à faire ce pourquoi il a été créé. Cependant ces exceptions doivent être claires. Dans certains codes la gestion des erreurs est tellement dispersée qu'il est impossible de voir d'où provient l'erreur.

L'utilisation de la commande `try{} catch(){} finally{}`, permet de clarifier le code et de séparer la gestion des erreurs et l'arrêt du périphérique. Grâce à cette séparation on a la possibilité de lire et comprendre indépendamment chaque partie. Lorsque l'on essaye d'accéder à un fichier inexistant, le test ne génère pas d'exception; cependant si l'on cherche un fichier invalide, on obtient une exception. La signature de chaque méthode énumérerait toutes les exceptions qu'il pourrait transmettre à son appelant. De plus, ces exceptions faisaient partie du type de la méthode. À l'époque, nous pensions que les exceptions vérifiées étaient une excellente idée; et oui, elles peuvent donner un certain avantage. Cependant, il est clair maintenant qu'elles ne sont pas nécessaires à la production de logiciel robuste.

Si l'une des fonctions d'un niveau très bas est modifiée de telle sorte qu'elle doit lancer une exception, on doit ajouter une clause `throws` à cette dernière. Mais cela signifie que chaque fonction qui appelle notre fonction doit ajouter une clause `throws`. A cause de cela l'encapsulation est interrompue, car chaque fonction doit connaître les détails de cette exception de bas niveau.

On appelle le modèle de cas spécial [Fowler] lorsque l'on crée une classe ou configure un objet afin qu'il gère un cas particulier pour nous.

Ne pas retourner la valeur NULL : si on veut retourner une valeur NULL, il faut plutôt penser à retourner un objet SPECIAL CASE ou lancer une exception.

Conclusion : Le code propre est lisible, mais il doit aussi être robuste. Ce ne sont pas des objectifs contradictoires. Nous pouvons écrire du code propre robuste si nous considérons que la gestion des erreurs est une préoccupation distincte.

## Chapitre 8

1) En utilisant un code tiers : Il est difficile de comprendre un code tiers, mais également de l'intégrer à notre code à nous. Il faut donc créer des tests pour comprendre comment fonctionne le code tiers. Ces tests précis et expérimentaux peuvent être réutilisés si une nouvelle version du code tiers sort, pour savoir s'il y a des différences importantes. Ils permettent de savoir si le package que nous utilisons fonctionne comme on le souhaite. Même une fois intégré, il n'y a aucune garantie que le code tiers reste compatible avec nos attentes. L'auteur peut à tout moment changer son code, pour ajouter de nouvelles fonctionnalités, fixer des bugs... A chaque mise à jour il y a de nouveaux risques.

2) En utilisant un code qui n'existe pas encore : On peut faire quelque chose de propre et facilement compréhensible, en utilisant des termes simples et qui montrent parfaitement à quoi correspond chaque élément.

## Chapitre 9 : Les tests unitaires

Lois du TDD (Test Driven Development) :

Première loi. Vous ne devez pas écrire un code de production tant que vous n'avez pas écrit un test unitaire d'échec.

Deuxième loi. Vous devez uniquement écrire le test unitaire suffisant pour échouer. L'impossibilité de compiler est un échec.

Troisième loi. Vous devez uniquement écrire le code de production suffisant pour réussir le test d'échec courant.

Le code de test est aussi important que le code de production. La chose la plus importante dans un test : la lisibilité. Il est primordial de rendre le code lisible et plaisant à lire. Faire des petits tests est important. Une seule fonctionnalité par test.

F.I.R.S.T :

Fast : Lancement des tests rapide

Independent : Pas de test dépendant à d'autres tests.

Repeatable (reproductible): Le test doit être exécutable dans tous les domaines. Donc non-spécifique.

Self-validating : Doit se valider lui-même en retournant True ou False.

### Chapitre 10 : Les Classes

L'écriture d'une classe commence par la définition d'une liste de variables lui étant propres. En premier les constantes, puis les variables de classe (static) et enfin les variables d'instance. La plupart des méthodes et variables d'instance se devront d'être déclarées en private afin de les protéger le plus possible (sauf cas particulier, exemple test devant faire appel à certaines fonctions).

Une classe doit être petite. Tout comme les fonctions, les classes doivent aller à l'essentiel. Les classes doivent avoir peu de variables d'instance, et au moins une d'entre elle doit être utilisée dans chaque méthode d'instance. Une classe respectant ce principe est dite unie. Si une classe est trop grande est que certaines ressources sont similaires à une autre, il faut essayer de séparer cette classe en deux.

### Chapitre 12

D'après Kent Beck, il y a 4 règles pour réaliser un "design" simple.

Règle 1 Faire tous les tests : Un système qui ne peut être vérifié ne devrait pas être déployé. Si l'on veut rendre nos classes testables, il faut que nos classes soient petites et uniques. Donc si on s'assure de rendre nos programmes testables, on pourra créer des bons designs

Règle 2 : La duplication est le principal ennemi d'un système bien conçu. Cela représente du travail supplémentaire, un risque supplémentaire, et de la complexité inutile.

Règle 3 : Il est facile d'écrire un code que nous comprenons puisqu'au moment où nous l'écrivons nous comprenons le problème. À mesure que les systèmes deviennent plus complexes, ils prennent plus de temps pour un développeur à comprendre. Le code doit donc exprimer l'intention de son programmeur. Pour ça il faut choisir des noms expressifs pour chaque fonction, méthode et variable. Les tests aussi doivent être expressifs, car le but premier d'un test est d'agir comme de la documentation. Grâce aux tests une personne devrait être capable de comprendre rapidement le rôle d'une classe.

Règle 4 : L'objectif est de garder le système global petit, tout en gardant les fonctions.

### Chapitre 13 : La concurrence

La concurrence c'est les threads. Faire des programmes qui s'exécutent en même temps pour que le résultat soit plus rapide. Elle améliore les performances lorsqu'il est possible de partager les temps d'attente. Avec un thread la structure du programme est très différente. Cependant cela entraîne des surcoûts en termes de performances et de code. C'est également très complexe à mettre en œuvre. 2 thread empruntent 2 chemins différents (un int en possède 12 870). La quasi-totalité de ces chemins entraînent un résultat correct cependant d'autre ne le font pas. Alors pour prévenir ce problème, le code de concurrence doit être séparé de tout autre code. On utilise "synchronized" sur les zones critiques pour être sûr que cela fonctionne bien.

### Chapitre 15 : Au cœur de JUnit

Les tests sont effectués dans l'ordre du début jusqu'à la fin du programme. Toutes les conditions doivent être encapsulées Des noms de variable clairs et non ambigus doivent être choisis, même dans les tests JUnit.

### Conclusion du livre

Seule l'expérience pourra vous prouver ce qui est écrit ici. Ce livre permet aux développeurs d'adhérer à de bons principes et modèles.