

Université de Maroua  
Institut Supérieur du Sahel  
Département d'Informatique  
et des Télécommunications



The University of Maroua  
The Higher Institute of the Sahel  
Department of Computer Science and  
Telecommunications

INFORMATIQUE ET TELECOMMUNICATIONS

## **DEVELOPPEMENT D'UN GENERATEUR DE SQUELETTES D'APPLICATIONS WEB POUR ABLE – SARL**

**Mémoire présenté et soutenu en vue de l'obtention du Diplôme d'INGENIEUR DE  
CONCEPTION EN INFORMATIQUE option GENIE LOGICIEL**

par

**EWANE Jules-Alfred**  
*Licence en Informatique*  
**13Z572S**

sous la direction de

**Prof. Dr.-Ing. habil. KOLYANG**

devant le jury composé de :

Président :	Pr. DOKA YAMIGNO Serge
Examineur :	Pr. EMVUDU WONO Yves
Rapporteur :	Prof. Dr.-Ing. habil. KOLYANG
Invité :	M. NZEPA SATCHE Franck

*Année Académique 2014/ 2015*

## DEDICACE

*A*

*Mes chers parents,*

***Monsieur et Madame EFAWA ;***

*Pour leurs plus grandes leçons de vie, la crainte de Dieu et l'amour du travail,*

*Et en reconnaissance des innombrables sacrifices consentis à mon égard.*

## **REMERCIEMENTS**

La réalisation de ce modeste travail est le fruit d'efforts personnels encadrés de soutien multiple, provenant d'horizons divers, auxquels il serait légitime d'exprimer ma profonde gratitude. Il s'agit en particulier de :

- Prof. Dr.-Ing. habil. KOLYANG, pour les multiples corrections apportées à ce travail ;
- Dr. VIDEME BOSSOU Olivier, pour les efforts considérables qu'il a fournis pour mener à bien notre formation ;
- M. NZEPA SATCHE Franck, encadreur professionnel, pour le temps qu'il m'a consacré durant le déroulement de mon stage ;
- Ma fiancée DJOMO Phalone, pour l'encouragement constant et le soutien moral permanent dont elle fut à l'origine ;
- Mes camarades, pour les échanges fructueux pendant notre formation.

# TABLE DES MATIERES

DEDICACE.....	i
REMERCIEMENTS .....	ii
Liste des sigles et abréviations .....	viii
RESUME.....	ix
ABSTRACT .....	x
Liste des tableaux .....	xi
Liste des figures et illustrations.....	xii
INTRODUCTION GENERALE.....	1
CHAPITRE I : CONTEXTE ET PROBLEMATIQUE .....	3
I.1- Présentation de l'entreprise ABLE-SARL .....	3
I.1.1- Présentation .....	3
I.1.2- Organisation.....	4
I.2- Contexte et problématique.....	4
I.2.1- Contexte.....	4
I.2.2- Problématique .....	4
I.3- Méthodologie.....	5
I.4- Objectifs.....	5
I.4.1- Objectif général .....	5
I.4.2- Objectifs spécifiques.....	5
CHAPITRE II : GENERALITES .....	6
Introduction .....	6
II.1- Le Rapid Application Development.....	6
II.2- L'outil Grails .....	7
II.2.1- L'architecture Grails.....	8

II.2.2- Le langage Groovy .....	8
II.2.3- Création d'un projet avec Grails .....	10
II.2.3.1- Création des classes de domaine .....	10
II.2.3.2- Scaffolding .....	11
II.2.3.3- Running .....	12
II.2.4- Avantages de Grails .....	13
II.3- Les outils JUnit et Git.....	13
II.3.1- JUnit .....	13
II.3.2- Git.....	14
II.4- Le pattern MVC .....	15
II.4.1- Définition d'un pattern .....	15
II.4.2- Présentation du MVC .....	15
II.4.3- Principe du MVC .....	15
II.5- Construction d'un projet UML.....	16
II.6- Le Développement Agile .....	18
II.6.1- Les principes du Manifeste Agile.....	18
II.6.2- La modélisation agile (AM) .....	19
Conclusion.....	20
<b>CHAPITRE III : ANALYSE ET CONCEPTION .....</b>	<b>21</b>
Introduction .....	21
III.1- Analyse .....	22
III.1.1- Cahier des charges .....	22
III.1.1.1- A propos du générateur .....	22
III.1.1.2- A propos de l'application générée .....	24
III.1.2- Spécifications du système pour le générateur .....	25
III.1.2.1- Acteur du système .....	25
III.1.2.2- Cas d'utilisation .....	26

III.1.2.3- Description des cas d'utilisation .....	27
III.1.3- Spécification du système de l'application web résultante de la génération .....	32
III.1.3.1- Acteurs et cas d'utilisation.....	32
III.1.3.2- IHM de l'application web générée.....	33
III.2- Conception .....	35
III.2.1- Diagrammes des use cases .....	35
III.2.2- Diagramme de classes du générateur.....	38
III.2.2.1- Description des entités .....	38
III.2.2.2- Représentation du diagramme.....	38
Conclusion.....	39
CHAPITRE IV : IMPLEMENTATION ET IMPLANTATION .....	40
Introduction .....	40
IV.1- Implémentation du générateur .....	40
IV.1.1- Environnement de développement.....	40
IV.1.1.1- L'IDE NetBeans .....	40
IV.1.1.2- Le langage Java.....	40
IV.1.2- Présentation du projet .....	41
IV.1.2.1- L'arborescence.....	41
IV.1.2.2- Les classes « métier » .....	42
IV.1.2.3- Les classes Factoring .....	43
IV.1.2.4- Les classes LISTENER.....	44
IV.1.3- Persistances des objets .....	46
IV.1.4- Génération et exécution du projet.....	47
IV.1.4.1- La méthode contruireLeProjet de l'interface ServiceGrails .....	47
IV.1.4.2- Le fichier runGrails.bat.....	49
IV.1.4.3- Importation et exportation de projet .....	49

IV.1.5- Gestion des tests (JUnit) .....	50
IV.1.6- Gestion des versions avec Git en local .....	51
IV.2- Construction du squelette Grails.....	51
IV.2.1- Configurations effectuées sur Grails .....	52
IV.2.1.1- Connexion à une base de données .....	52
IV.2.1.2- Le mappeur Hibernate .....	52
IV.2.1.3- Installation du plugin Searchable.....	53
IV.2.2- Les contrôleurs.....	53
IV.2.2.1- Les contrôleurs générés et le Scaffolding .....	53
IV.2.2.2- MyOwn Controller.....	54
IV.2.2.3- UserController .....	54
IV.2.2.4- La recherche.....	54
IV.2.3- Construction de l'interface graphique .....	55
IV.2.3.1- Les pages GSP .....	55
IV.2.3.2- Le HTML et les codes CSS .....	57
IV.2.4- Sécurité de l'application .....	58
IV.2.4.1- Le domaine User .....	58
IV.2.4.2- Le contrôleur UserController.....	59
IV.2.4.3- Le filtrage.....	61
IV.2.4.4- Le cryptage .....	61
IV.2.4.5- L'interface de connexion .....	62
IV.3- Implantation au sein de l'entreprise.....	63
Conclusion.....	63
CHAPITRE V : RESULTATS ET COMMENTAIRES.....	64
CONCLUSION GENERALE ET PERSPECTIVES.....	68
ANNEXES .....	69
Annexe1 : Guide d'exploitation de Wable.....	69

Annexe 2 : Arborescence d'un projet Grails dans l'IDE <i>Grails-Tool-Suite</i> .....	77
Annexe 3 : Gestion simplifiée des collections avec Groovy.....	78
Annexe 4 : Interfaces par défaut de Grails .....	79
Annexe 5 : Code de chargement d'un projet WABLE.....	80
Annexe 6 : Code du fichier login.GSP .....	81
BIBLIOGRAPHIE ET WEBOGRAPHIE .....	82



## Liste des sigles et abréviations

N°	Symboles	Significations
1.	AGL	Atelier de Génie Logiciel
2.	BD	Base de données
3.	CRUD	Create-Read-Update-Delete
4.	CSS	Cascading Style Sheet
5.	GSP	Groovy server Pages
6.	HTML	HyperText Markup Language
7.	IDE	Integrated Development Environment
8.	<i>IHM</i>	Interface Homme Machine
9.	J2EE	Java 2 Enterprise Edition
10.	JAR	Java ARchive
11.	JSP	Java Server Pages
12.	JVM	Java Virtual Machine
13.	L4G	Langage de Quatrième Génération
14.	<i>MVC</i>	Modèle - Vue - Contrôleur
15.	OMT	Object Modeling Technique
16.	OO	Orienté Objet
17.	RAD	Rapid Application development
18.	SARL	Société A Responsabilité Limitée
19.	UML	Unified Modeling Language

## RESUME

L'essentiel de ce travail fut de mettre sur pied un générateur de squelettes d'applications web pour l'entreprise ABLE-SARL. En effet, la mise sur pied d'un logiciel nécessite généralement beaucoup de temps. Mais cela n'est pas toujours ainsi perçu par le client qui passe une commande pour un besoin éminent.

Nous avons étudié des outils et des techniques qui font allusion au Développement Rapide d'Applications. Cette étude nous a permis de concevoir puis d'implémenter un système de génération des applications web minimales dites squelettes d'application web.

Le programmeur utilise ce soft pour générer une application web à partir des informations qu'il indique. Ce logiciel, baptisé **WAbLe**, permet au programmeur d'entrer les entités et d'indiquer des relations qui permettront de générer le squelette d'une application web de type Java2EE. Ces entités sont indiquées manuellement ou à partir d'entités Java.

Le système pourra s'étendre à l'entrée des entités à partir des scripts de base de données ou encore à partir d'autres environnements de développement.

## ABSTRACT

Most of this work was to develop a web application skeletons generator for the company ABLE-SARL. Indeed, the development of software usually requires much time. But that is not always well perceived by the customer who needs the software early.

We studied the tools and techniques that allude to Rapid Application Development. This study allowed us to design and implement a system for generating web applications known as minimum web application skeletons.

The programmer uses this app to generate a web application based on the information he indicates. This software, called **Wable**, allows the programmer to enter the entities and to indicate relationships that will generate the skeleton of a Java2EE type of web application. These entities are set manually or from Java entities.

The system will extend to the entry of entities from database scripts or from other development environments.

## Liste des tableaux

Tableau 1 : Récapitulatif et description des exigences du générateur.....	24
Tableau 2 : Récapitulatif des cas d'utilisation du générateur.....	27
Tableau 3 : Description des cas d'utilisation de l'application générée, par type d'utilisateur.....	32

## Liste des figures et illustrations

Figure 1 : Logo de l'entreprise ABLE-SARL .....	3
Figure 2 : Exécution de la commande grails dans une invite Windows .....	7
Figure 3 : Contenu d'un dossier de projet grails .....	10
Figure 4 : Première page d'un projet sur grails .....	12
Figure 5 : Cycle de vie des états d'un fichier dans Git. ....	14
Figure 6 : Schématisation du MVC.....	16
Figure 7 : Scénario du cas d'utilisation « supprimer une entité dans un projet» .....	29
Figure 8 : Scénario du cas d'utilisation « modifier un attribut d'une entité».....	30
Figure 9 : Scénario du cas d'utilisation « établir une relation entre 02 entités» .....	30
Figure 10 : Maquette de l'interface principale de l'application générée.....	33
Figure 11 : Maquette de l'interface de connexion de l'application générée .....	34
Figure 12 : Diagramme des cas d'utilisation du générateur .....	36
Figure 13 : Diagramme des cas d'utilisation de l'application générée .....	37
Figure 14 : Diagramme de classes.....	38
Figure 15 : Version du java utilisée .....	41
Figure 16 : Arborescence du projet dans l'explorateur Windows.....	41
Figure 17 : Arborescence du projet dans l'IDE NetBeans .....	42
Figure 18 : Contenus des paquetages FACTORIES et LISTENERS .....	45
Figure 19 : Capture du fichier ServicesGrails.java .....	48
Figure 20 : Représentation du pont Grails dans l'import-export du projet .....	50
Figure 21 : JUnit dans la librairie du projet .....	50
Figure 22 : Invite montrant la version de Git utilisée .....	51
Figure 23 : Illustration de la recherche d'un objet dans l'application.....	54
Figure 24 : Disposition des blocs dans l'application web générée.....	57

Figure 25 : Maquette de l'interface de connexion.....	62
Figure 26 : Première interface de WAbLe.....	64
Figure 27 : Importation ou exportation d'un projet sur WAbLe .....	65
Figure 28 : Interface d'accueil de l'application web générée par WAbLe.....	65
Figure 29 : Interface de connexion de l'application web générée.....	66
Figure 30 : Interface de recherche d'un objet dans l'application générée.....	66

# INTRODUCTION GENERALE

Le génie logiciel est un domaine des sciences de l'ingénieur dont l'objet d'étude est la conception, la fabrication, et la maintenance des systèmes informatiques complexes. Il constitue un ensemble toujours dynamique et perpétuellement à la recherche de nouvelles solutions de mise sur pied de logiciels. Les nouvelles techniques et les nouveaux outils doivent à chaque fois apporter des solutions d'amélioration de performance, de robustesse, de simplicité et surtout de rapidité dans la création d'application. Une astuce serait de généraliser les techniques qui ont marché et qui ont été confirmées, on parle de design patterns ; pourquoi réinventer la roue ?

Une application web est un logiciel exploité via une navigation entre des pages web. La mise sur pied d'un générateur de squelettes d'application web permet d'éviter de construire à chaque fois une nouvelle charpente pour une nouvelle application. La rapidité est un défi majeur dans le génie logiciel. Ce générateur permettra d'exploiter un même squelette d'application web, le configurer pour en créer de nouvelles.

Au sein de l'entreprise ABLE-SARL où nous avons travaillé, ce projet, baptisé **Wable**, permet désormais d'implémenter avec plus de célérité des applications web.

Pour présenter notre travail de façon optimale, nous l'avons découpé en cinq chapitres. Contexte et la problématique, c'est le premier chapitre. Il s'agira de présenter le mobile ou encore la raison qui nous a motivés dans ce projet. On relèvera également des interrogations survenues.

Ensuite suivra le chapitre des généralités dans lequel nous présenterons quelques techniques et outils utilisés.

Comme tout projet informatique digne, nous allons dans le troisième chapitre modéliser le problème à résoudre, il s'agit de l'analyse et de la conception.

Le quatrième chapitre présente l'implémentation et l'implantation du produit. Il s'agira ici tout simplement de l'exécution du plan de conception et de l'implantation de l'application dans l'entreprise.

Le dernier chapitre des Résultats et Commentaires nous permettra d'évaluer notre travail, de le présenter à travers quelques captures et d'y ajouter quelques commentaires.



# CHAPITRE I : CONTEXTE ET PROBLEMATIQUE

## I.1- Présentation de l'entreprise ABLE-SARL

### I.1.1- Présentation

ABLE-SARL est une entreprise à responsabilité limitée. Il est mis sur pied par un groupe de deux jeunes camerounais en 2012. Son siège social se trouve à Dschang. Il a des sites éparés, parmi lesquels celui de Bamenda. Son sigle est « *The light of new technologies* » et son logo est représenté par la figure ci-dessous.



**Figure 1 : Logo de l'entreprise ABLE-SARL**

L'entreprise offre à ce jour, neuf services à savoir :

- + Ventes des consommables informatiques et de reprographie ;
- + Maintenance informatique et des systèmes de reprographie ;
- + Locations ;
- + Conception et production des supports de communication (flocage et autres...) ;
- + Carterie ;
- + Formations en maintenance informatique et des systèmes de reprographie ;
- + Reprographie (mini-imprimerie)
- + **Développement web et logiciels**
- + Installation des réseaux informatiques.

C'est dans l'avant dernier service que nous avons passé l'important de notre stage.

### I.1.2- Organisation

L'entreprise est répartie en cinq directions.

Une direction générale et quatre autres réparties telle suit :

- ❖ Direction de production ;
- ❖ Direction des finances ;
- ❖ Direction des ventes et locations ;
- ❖ Direction de marketing.

## I.2- Contexte et problématique

### I.2.1- Contexte

ABLE-SARL reçoit un nombre considérable de demandes en termes de production de logiciels. Au regard du non respect du délai souvent observé, l'entreprise a décidé de mettre sur pied un nouveau mécanisme pour les activités de son génie logiciel.

Cette initiative est pensée ; puisqu'on constate que des tâches similaires sont souvent effectuées dans le développement, pour chaque application. Pourquoi ne pas donc les généraliser ?

Dans la conception de chaque application, on manipule des entités. Et généralement, on voudra, dans les fonctionnalités, pouvoir **ajouter** une entité, la **supprimer**, la **modifier** ou l'**afficher** tout simplement.

La **recherche** et la **sécurité** sont également des fonctionnalités que l'on retrouve dans des applications.

### I.2.2- Problématique

Ayant recensé les mobiles de ce projet dans la partie précédente, on peut bien se demander *comment faciliter la construction d'applications en mettant sur pied un bagage qui permette d'avoir les prémices d'un logiciel intégrant les fonctionnalités citées ci-dessus ?*

A la réponse à cette question, plusieurs interrogations peuvent émerger :

- ✓ Quels outils peuvent permettre la conception et la mise en œuvre d'un tel système de génération ?
- ✓ Comment construire un tel générateur ?
- ✓ Comment motiver le programmeur à utiliser un tel générateur ?

### **I.3- Méthodologie**

Nous nous proposons de résoudre le problème en suivant les étapes suivantes :

- ✚ Brève étude comparative de quelques outils offrant les fonctionnalités CRUD ;
- ✚ Choix d'un outil ;
- ✚ Construction d'un squelette de base en utilisant l'outil choisi ;
- ✚ Ajout des fonctionnalités de recherche et de sécurité au squelette de base construit ;
- ✚ Analyse et conception d'un système de génération ;
- ✚ Liaison du squelette de base au générateur ;
- ✚ Implémentation et tests.

### **I.4- Objectifs**

#### **I.4.1- Objectif général**

L'objectif général de ce travail est de mettre sur pied un logiciel qui permettra de générer des squelettes d'applications web pour l'entreprise ABLE-SARL.

#### **I.4.2- Objectifs spécifiques**

L'objectif général sera atteint en quatre principales étapes :

- ❖ la première va consister à analyser, puis concevoir un système de génération d'applications web ;
- ❖ la deuxième sera celle de la conception de l'IHM attendue du squelette généré ;
- ❖ ensuite, il va s'agir d'implémenter le générateur ;
- ❖ et enfin, ce sera l'implantation du logiciel au sein de l'entreprise accompagnée d'un document permettant de guider le programmeur dans l'exploitation du logiciel.

## CHAPITRE II : GENERALITES

### Introduction

Dans l'accomplissement d'un travail scientifique, on s'appuie « toujours » sur des résultats d'autres recherches. Il devient donc très indispensable, dans un tel projet, de présenter les outils et techniques qui ont été utilisés pour parvenir à une bonne fin. Dans ce chapitre des généralités, nous présenterons la technique du *Robot Automatique de Développement*, puis l'outil *Grails* qui la met en valeur. Dans un souci de respect des principes du génie logiciel, pour éviter des erreurs de régression et pour mieux gérer les versions, nous utiliserons respectivement les outils *JUnit* et *Git* que nous présentons également dans ce chapitre. L'architecture *MVC*, le langage *UML* et la méthode de développement *Agile* feront l'objet des dernières présentations.

### II.1- Le Rapid Application Development

Le Rapid Application Development (RAD), francisé en « Robot Automatique de Développement », est une méthode de développement rapide de logiciels. Il se fait généralement au moyen des outils tels que des AGL (Atelier de Génie Logiciel) ou des L4G (Langages de Quatrième Génération).

Partant du résultat de la modélisation, ils doivent pouvoir générer une application avec des fonctionnalités basiques telles que l'ajout, la modification et la suppression d'une entité ; également l'affichage des données par liste. On parle généralement de CRUD (Create, Read, Update, Delete).

Ces outils sont des ensembles de programmes permettant la conception de programmes, d'applications ou de systèmes parfois très complexes. Ils sont généralement formés par des langages puissants et évolués, accompagnés d'utilitaires de création d'interfaces graphiques des programmes générés. Exemple : Visual Basic, Powerbuilder, Spring Roo, Grails.

Avec un L4G, on programme vite et c'est simple, mais le code généré est souvent lourd et très lent, sans véritable optimisation. De plus, on n'a que rarement accès aux entrailles de son programme, et s'il ne fonctionne pas, on peut mettre beaucoup de temps à diagnostiquer.

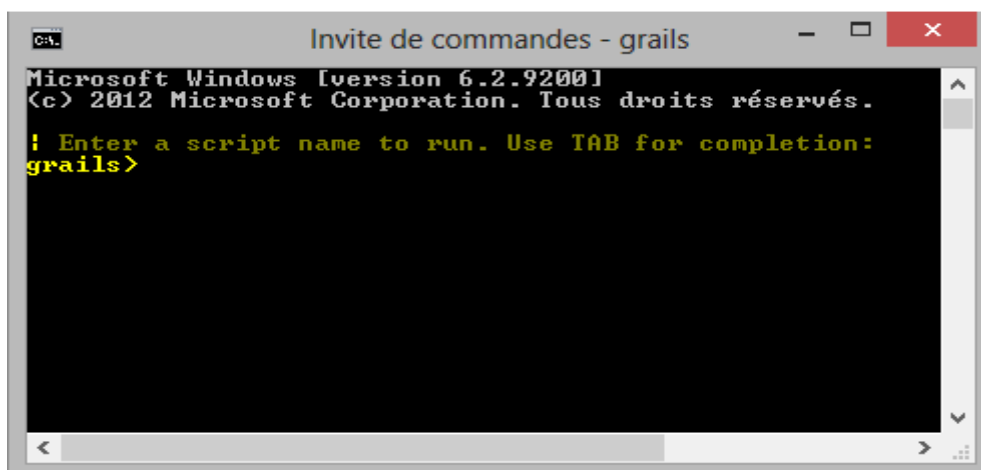
Il est donc nécessaire pour le programmeur de trouver l'outil qui répondra à ses attentes. Un outil qui soit compatible au langage utilisé, qui soit le plus « open source » possible et qui procure une architecture familière. Spring Roo et Grails sont des outils destinés à la plateforme Java et qui utilisent des architectures en MVC. Grails a particulièrement attiré notre attention.

## II.2- L'outil Grails

Dans le monde Java, Grails, c'est-à-dire Groovy on Rails, apporte une réponse à la concurrence des frameworks de développement rapide tels que Ruby on rails ou .NET. Il permet de créer rapidement des applications WEB. Il est initié en 2005 par Graeme Rocher. C'est un Framework open source de développement agile, édité par la société *SpringSource*.

Il s'installe comme le framework *maven* : on copie son dossier et on indique l'emplacement du *bin* dans la configuration du PATH. On peut l'utiliser en ligne de commandes ou alors via des IDE qui l'intègrent.

Sa commande principale est *grails*. Lorsque les configurations son bien faites, l'exécution de cette commande dans une invite de commandes Windows nous donne l'aperçu suivant :



**Figure 2 : Exécution de la commande grails dans une invite Windows**

### II.2.1- L'architecture Grails

Grails est un serveur d'application web basé sur le langage Groovy et le framework Spring. Comme il est déjà dit plus haut, il fonctionne suivant une architecture MVC. Cette architecture sera présentée dans la partie suivante parlant des patterns.

Une application Grails se décompose conséquemment en trois parties : les domaines, les contrôleurs et les vues.

- **Les domaines** (ou modèles) sont des classes servant à modéliser les données et à établir les relations entre elles. Cela permet d'établir le mapping entre les Objets et la base de données, grâce aux outils tels que *Hibernate*.

Il est possible d'effectuer des requêtes de base sur les domaines. On trouve sur chaque domaines des méthodes `get()`, `save()`, `list()`, `findByNom()`, etc... Ces méthodes n'ont jamais été définies par le développeur, mais elles existent grâce au framework.

Il est également possible de contrôler la validation des formulaires depuis le fichier qui définit le domaine.

- **Les contrôleurs.** Un contrôleur est une classe qui reçoit la requête de l'utilisateur et qui, en fonction de l'action demandée, va effectuer le traitement.

Pour gérer quelle est l'action et quel contrôleur est appelé, Grails se base sur le formatage de l'URL : *http://<...>/controller/action/*

- **Les vues** sont représenté par des fichiers GSP(Groovy Server Page). Elles sont similaires aux JSP. La différence vient du fait qu'on peut insérer du code Groovy.

On constate dans l'arborescence qu'il y a une vue dédiée à chaque contrôleur et une vue dédiée à une action du contrôleur (Voir Annexes).

### II.2.2- Le langage Groovy

Le projet Open Source Groovy est initié en 2003 par James Strchan et Bob McWhirter. C'est un Langage Orienté Objet (LOO) destiné à la plateforme Java. C'est une alternative inspirée de Python, Ruby et Smalltalk.

Il est intégré et compatible avec la JVM. On peut y utiliser des bibliothèques Java ; il peut aussi être directement utilisé dans des classes Java. Sa syntaxe est d'ailleurs proche de Java. La page de téléchargement <http://groovy.codehaus.org/Download> propose divers liens pour obtenir Groovy.

La syntaxe de Groovy est faite pour apporter plus de flexibilité au langage Java.

Voici les différentes simplifications que l'on peut effectuer sur un code Java que l'on souhaite transformer en Groovy :

- Les "getters" et les "setters" sont implémentés par défaut. Il n'est donc pas nécessaire de les implémenter.
- L'utilisation des setters est simplifiée. On peut écrire :  
objet.champ="nouvelle valeur";
- Comme la vérification des types s'effectue à l'exécution, il n'est pas nécessaire de déclarer le type lors d'une déclaration de variable.

On peut donc remplacer : ***String nom*** par ***def nom***, dans le premier cas on définit une variable ***nom*** de type ***String***, dans le second cas on définit simplement une variable ***nom***.

- Il n'est pas nécessaire d'écrire le mot ***return*** pour retourner un objet à la fin d'une méthode. C'est l'objet qui se trouve sur la dernière ligne de cette méthode qui est automatiquement retourné.

- La concaténation est simplifiée. Il est possible d'insérer des variables à l'intérieur des chaînes de caractères :

```
def sayHello(){  
    "Hello${nom}  
    ${prenom} !"  
}
```

- Les points virgule ne sont plus obligatoires.

GROOVY apporte également une gestion simplifiée des collections ! (voire Annexes).

## II.2.3- Création d'un projet avec Grails

La création d'un projet avec Grails se fait en trois étapes :

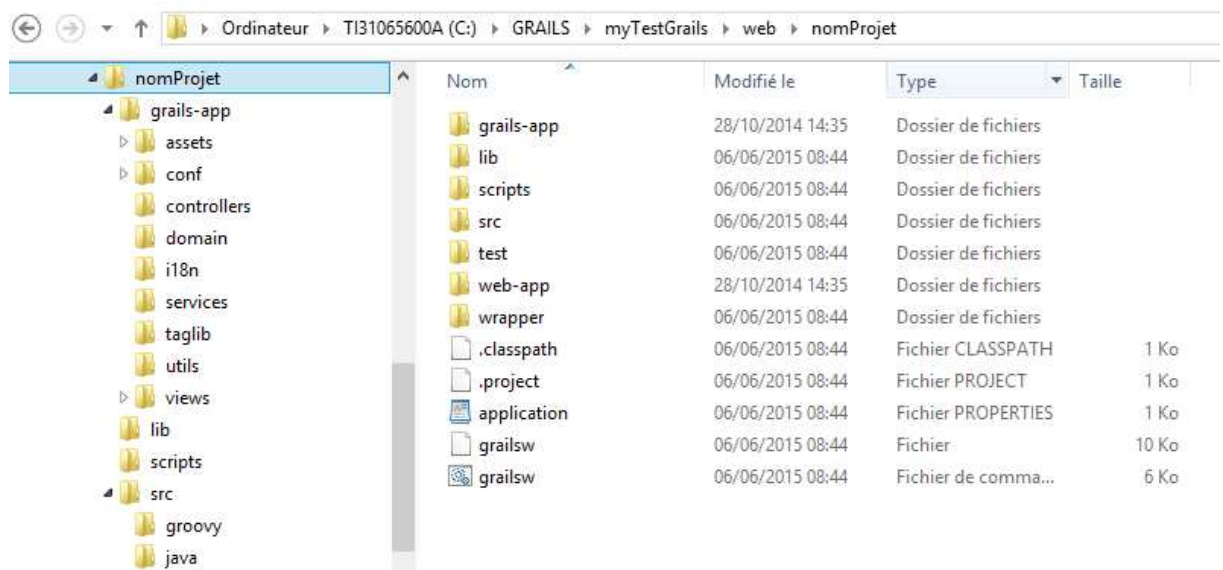
- Création des classes de domaine ;
- Scaffolding ;
- Running.

### II.2.3.1- Création des classes de domaine

Les « **Domain** » sont des définitions du modèle. C'est la première étape dans la création d'un projet *grails*.

Nous avons utilisé dans ce projet la version **2.4.4**.

La commande *grails create-app nomProjet* permet de créer un projet nommé *nomProjet*. *Grails* va construire un répertoire *nomProjet* avec un contenu tel que représenté par la figure ci-dessous :



**Figure 3 : Contenu d'un dossier de projet grails**



Dès que le projet est créé, on se positionne à l'intérieur en invite de commande (*cd nomProjet*).

La commande **grails create-domain-class Etudiant** va créer un fichier **Etudiant.GROOVY** dans le dossier **grails-app/domain**.

Juste en exemple, nous éditerons ce fichier de la façon suivante :

```
class Etudiant {  
    String nom  
    Date dateNaissance  
}
```

On ajoutera dans cette classe, le code ci-dessous :

```
static constraints = {  
    nom blank : false  
    dateNaissance max : new Date()  
}
```

Ce bloc permet de définir les contraintes sur les différents champs du domaine.

C'est également dans ces classes que se définissent les relations entre les différentes entités.

Le langage ainsi utilisé est du **Groovy**, dont nous en avons parlé précédemment. Ce même langage nous permettra de définir le comportement des contrôleurs.

### II.2.3.2- Scaffolding

Le **scaffolding** est une action de mise sur pied d'un échafaudage. C'est là que repose la génération des futures fonctionnalités.

Il se gère au niveau des **controllers** et permet de spécifier les actions du contrôleur.

La commande **grails** qui permet de créer un contrôleur pour le domaine Etudiant est : **grails create-controller Etudiant**.

Un fichier **EtudiantController** sera créé dans le dossier **grails-app/controllers**. En exemple, nous éditerons le code tel que ci-dessous :

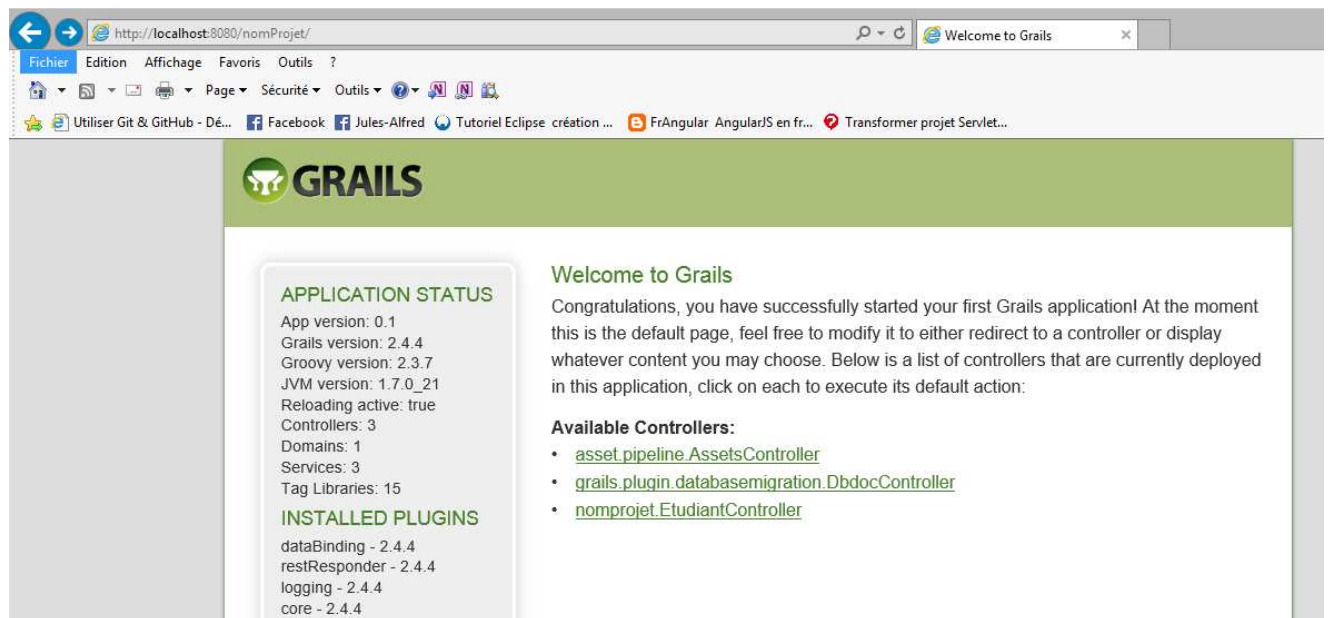
```
class EtudiantController {
    def scaffold = Etudiant
}
```

Le *scaffold* prend un domaine pour lequel le contrôleur définira les fonctionnalités CRUD. Ainsi, on pourra créer un *Etudiant*, modifier ses champs et le supprimer ; également, l’affichage de la liste des instances d’*Etudiant* créées sera disponible.

### II.2.3.3- Running

L’exécution d’un projet *grails* se fait par la commande *grails run-app*. On verra sur la console, le message *Server running. Browse to http://localhost:8080/nomProjet*.

*Grails* nous invite ainsi à exécuter l’application sur un navigateur à l’adresse indiquée. Le navigateur nous présentera l’interface suivante :



**Figure 4 : Première page d’un projet sur grails**

Un clic sur le contrôleur *EtudiantController* ouvre les interfaces permettant la création, la modification, la suppression et l’affichage des instances d’*Etudiant* (cf **Annexe1**).

On peut constater que son serveur embarqué utilise par défaut le port **8080**. Mais, il peut être modifié si un autre serveur (*tomcat* par exemple) utilise déjà ce port. Dans ce cas, la commande d'exécution, pour le port **5050** par exemple, sera *grails -Dserver.port=5050 run-app*.

#### II.2.4- Avantages de Grails

Les avantages que l'on peut avoir dans l'utilisation de Grails comme RAD, peuvent être les suivants :

- ❖ Web MVC : Facile à utiliser ;
- ❖ GSP : Langage de Template simple et complet ;
- ❖ Serveur embarqué ;
- ❖ GORM : Modélisation et accès aux données ;
- ❖ Base de données simulée (développement) ;
- ❖ Internationalisation ;
- ❖ Tests & Tests unitaires ;
- ❖ Documentation très riche.

### II.3- Les outils JUnit et Git

Il n'est pas toujours évident de parvenir rapidement au but que l'on se fixe dans la réalisation programmes. Pour ne pas rendre la tâche encore plus complexe, il peut être nécessaire d'utiliser des méthodes et outils permettant de ne pas s'éloigner du but. Les deux outils, **JUnit** et **Git**, que nous présentons dans cette partie nous permettent de détecter les erreurs de régression et de gérer les versions, respectivement.

#### II.3.1- JUnit

**JUnit** aide le développeur à minimiser les bugs. Un programmeur qui utilise **JUnit** va, avant d'écrire un morceau de programme, se demander ce que ce morceau de programme fait. Ensuite, il va, grâce à ce framework, écrire un bout de code qui sera capable de vérifier si la tâche a été bien réalisée. C'est là la puissance de **JUnit**. Il empêche que le programmeur ne programme « hors sujet », en avançant avec des

erreurs qui pourront être plus difficiles à traiter plus tard. Ainsi donc, avec **JUnit**, lorsque l'on modifie le code, l'on est sûr que l'on n'a pas introduit un bug.

### II.3.2- Git

Toujours dans un souci d'efficacité et de rapidité, le programmeur ne doit pas se permettre une mauvaise gestion des versions. Une version est tout simplement l'état du projet à un moment donné. On peut toujours avoir besoin, dans un projet, de retrouver un état antérieur. **Git** est un outil qui permet de sauvegarder et de restaurer des états.

Lorsque **Git** est bien installé, pour commencer à suivre un projet existant dans **Git**, il suffit de se positionner dans le répertoire du projet et exécuter la commande **git init**. Cela y va créer un sous-dossier nommé **.git**.

La commande **git add fichier** permet d'indexer les fichiers du projet qui seront suivis en version.

Pour cloner un projet existant, on fait **git clone url\_projet**.

La commande **git commit** permet de valider les modifications effectuées sur les fichiers indexés.

Le cycle de vie des états d'un fichier **Git** peut être illustré par la figure ci-dessous :

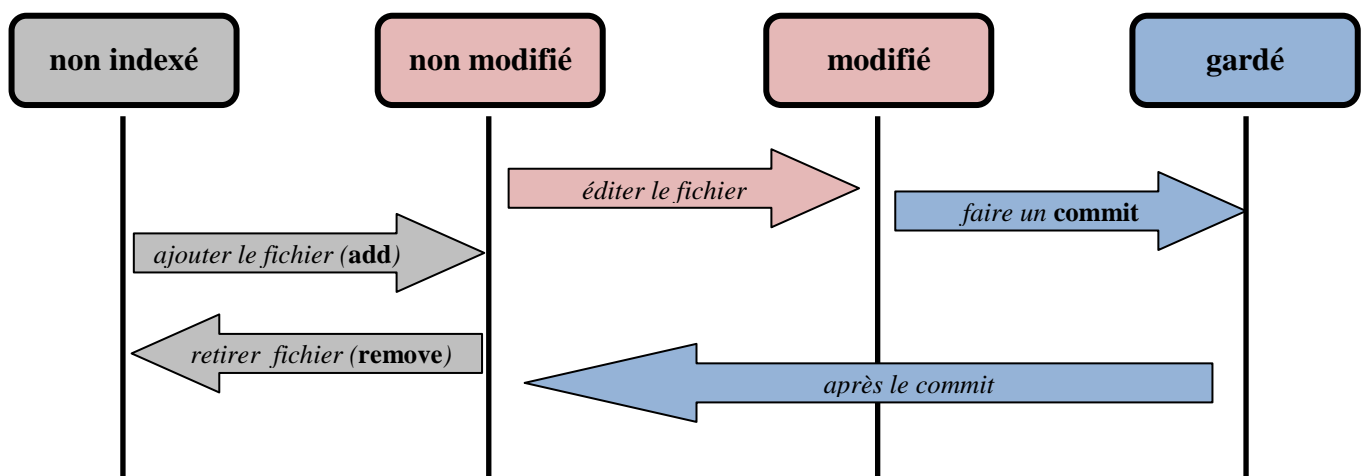


Figure 5 : Cycle de vie des états d'un fichier dans Git.

## II.4- Le pattern MVC

### II.4.1- Définition d'un pattern

Un pattern est « une solution à un problème dans un contexte bien défini ». Ses quatre éléments d'un pattern sont :

- son nom ;
- son but ;
- comment il résout le problème ;
- les contraintes à considérer dans la solution.

Les patterns nous permettent de réutiliser les solutions qui ont marché pour les autres ; « pourquoi réinventer la roue ? ». Ils nous permettent également d'avoir un vocabulaire commun.

### II.4.2- Présentation du MVC

MVC (Modèle – Vue – Contrôleur) est un design de conception d'interface utilisateur permettant de découpler le modèle (logique métier et accès aux données) des vues (interfaces utilisateur).

Des modifications de l'un n'auront ainsi, idéalement, aucune conséquence sur l'autre ce qui facilitera grandement la maintenance.

### II.4.3- Principe du MVC

**Modèle** : gère les données et reprend la logique métier (le modèle lui-même peut être décomposé en plusieurs couches mais cette décomposition n'intervient pas au niveau de MVC). Le modèle ne prend en compte aucun élément de présentation !

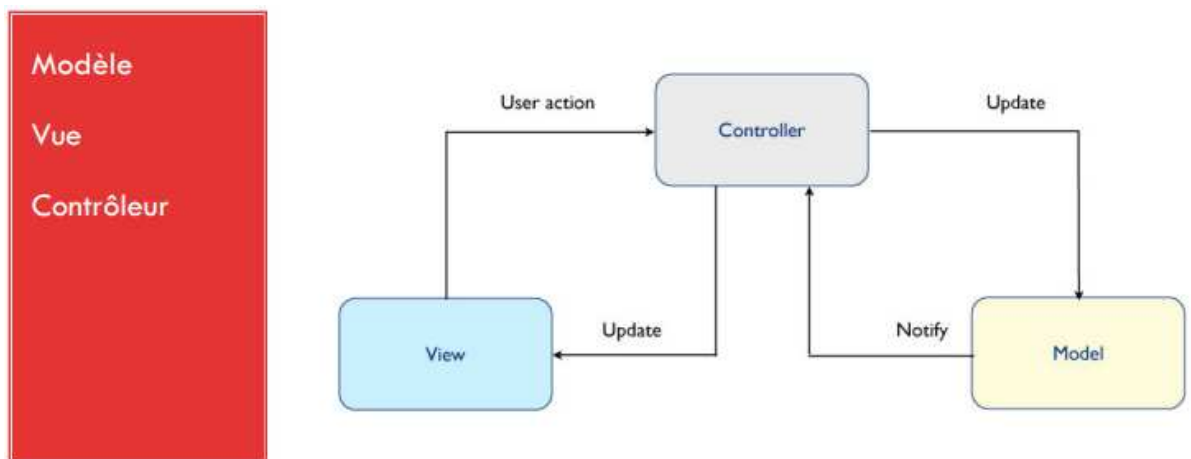
**Vue** : elle affiche les données, provenant exclusivement du modèle, pour l'utilisateur et/ou reçoit ses actions. Aucun traitement, autre que la gestion de présentation, n'y est réalisé.

**Contrôleur:** son rôle est de traiter les événements en provenance de l'interface utilisateur et les transmet au modèle pour le faire évoluer ou à la vue pour modifier son aspect visuel (pas de modification des données affichées mais des modifications de présentation).

Le contrôleur « connaît » la (les) vue(s) qu'il contrôle ainsi que le modèle.

Il pourra appeler des méthodes du modèle pour réagir à des événements, il pourra faire modifier à la vue son aspect visuel. Il pourra aussi instancier de nouvelles vues. Pour faire cela, le contrôleur sera à l'écoute d'événements survenant sur les vues.

La vue observera le modèle qui l'avertira du fait qu'une modification est survenue. Dans ce cas, la vue interrogera le modèle pour obtenir son nouvel « état ».



**Figure 6 : Schématisation du MVC**

## **II.5- Construction d'un projet UML**

UML se définit comme un langage de modélisation graphique et textuel destiné à comprendre et décrire des besoins, spécifier et documenter des systèmes, esquisser des architectures logicielles, concevoir des solutions et communiquer des points de vue.

UML unifie à la fois les notations et les concepts orientés objet. Il ne s'agit pas d'une simple notation graphique, car les concepts transmis par un diagramme ont une sémantique précise et sont porteurs de sens au même titre que les mots d'un langage.

UML unifie également les notations nécessaires aux différentes activités d'un processus de développement et offre, par ce biais, le moyen d'établir le suivi des décisions prises, depuis l'expression de besoin jusqu'au codage.

Le fil tendu entre les différentes étapes de construction permet alors de remonter du code aux besoins et d'en comprendre les tenants et les aboutissants. En d'autres termes, on peut retrouver la nécessité d'un bloc de code en se référant à son origine dans le modèle des besoins.

UML 2 s'articule autour de treize types de diagrammes, chacun d'eux étant dédié à la représentation des concepts particuliers d'un système logiciel. Ces types de diagrammes sont répartis en deux grands groupes :

- Six diagrammes structurels:
  - Diagramme de classes – Il montre les briques de base statiques : classes, associations, interfaces, attributs, opérations, généralisations, etc.
  - Diagramme d'objets - Il montre les instances des éléments structurels et leurs liens à l'exécution.
  - Diagramme de packages - Il montre l'organisation logique du modèle et les relations entre packages.
  - Diagramme de structure composite – Il montre l'organisation interne d'un élément statique complexe.
  - Diagramme de composants – Il montre des structures complexes, avec leurs interfaces fournies et requises.
  - Diagramme de déploiement – Il montre le déploiement physique des « artefacts » sur les ressources matérielles.
- Sept diagrammes comportementaux:

- Diagramme de cas d'utilisation - Il montre les interactions fonctionnelles entre les acteurs et le système à l'étude.
- Diagramme de vue d'ensemble des interactions - Il fusionne les diagrammes d'activité et de séquence pour combiner des fragments d'interaction avec des décisions et des flots.
- Diagramme de séquence - Il montre la séquence verticale des messages passés entre objets au sein d'une interaction.
- Diagramme de communication - Il montre la communication entre objets dans le plan au sein d'une interaction.
- Diagramme de temps – Il fusionne les diagrammes d'états et de séquence pour montrer l'évolution de l'état d'un objet au cours du temps.
- Diagramme d'activité - Il montre l'enchaînement des actions et décisions au sein d'une activité.
- Diagramme d'états – Il montre les différents états et transitions possibles des objets d'une classe.

## **II.6- Le Développement Agile**

### **II.6.1- Les principes du Manifeste Agile**

La notion de méthode agile est née à travers un manifeste signé en 2001 par 17 personnalités du développement logiciel.

Ce manifeste prône quatre valeurs fondamentales :

- « Personnes et interactions plutôt que processus et outils » : dans l'optique agile, l'équipe est bien plus importante que les moyens matériels ou les procédures. Il est préférable d'avoir une équipe soudée et qui communique, composée de développeurs moyens, plutôt qu'une équipe composée d'individualistes, même brillants. La communication est une notion fondamentale.



- « Logiciel fonctionnel plutôt que documentation complète » : il est vital que l'application fonctionne. Le reste, et notamment la documentation technique, est secondaire, même si une documentation succincte et précise est utile comme moyen de communication. La documentation représente une charge de travail importante et peut être néfaste si elle n'est pas à jour. Il est préférable de commenter abondamment le code lui-même, et surtout de transférer les compétences au sein de l'équipe (on en revient à l'importance de la communication).

- « Collaboration avec le client plutôt que négociation de contrat » : le client doit être impliqué dans le développement. On ne peut se contenter de négocier un contrat au début du projet, puis de négliger les demandes du client. Le client doit collaborer avec l'équipe et fournir un feedback continu sur l'adaptation du logiciel à ses attentes.

- « Réagir au changement plutôt que suivre un plan » : la planification initiale et la structure du logiciel doivent être flexibles afin de permettre l'évolution de la demande du client tout au long du projet. Les premiers releases du logiciel vont souvent provoquer des demandes d'évolution.

### **II.6.2- La modélisation agile (AM)**

La « modélisation agile » prônée par Scott Ambler s'appuie sur des principes simples et de bon sens, parmi lesquels :

- Vous devriez avoir une grande palette de techniques à votre disposition et connaître les forces et les faiblesses de chacune de manière à pouvoir appliquer la meilleure au problème courant.

- N'hésitez pas à changer de diagramme quand vous sentez que vous n'avancez plus avec le modèle en cours. Le changement de perspective va vous permettre de voir le problème sous un autre angle et de mieux comprendre ce qui bloquait précédemment.

- Vous trouverez souvent que vous êtes plus productif si vous créez plusieurs modèles simultanément plutôt qu'en vous focalisant sur un seul type de diagramme.

## Conclusion

Dans ce chapitre de généralités, il a été question de présenter certains outils utilisés dans le cadre de notre travail. Nous avons à cet effet présenté la technique du *RAD*, les outils Grails, *JUnit* et *Git*, l'architecture *MVC*, le langage *UML* et la méthode de développement *Agile*. Notons également que tout le long du projet, nous avons adopté une attitude « Agile », telle que décrite ci-dessus.

## CHAPITRE III : ANALYSE ET CONCEPTION

### Introduction

La nécessité de mettre sur pied un logiciel vient des vagues besoins humains. Pour ne pas tomber dans le piège du « *vite fait, mal fait* », il est très important de passer par la phase de modélisation. Cela permettra de mieux appréhender les besoins.

Un modèle est la simplification de la réalité permettant de mieux comprendre le système que l'on développe. La modélisation est la construction des modèles. C'est la première phase dans la mise en œuvre d'un système informatique digne de ce nom. Elle permet de passer de la forme abstraite représentée par des idées à la forme concrète, celle des codes et des configurations. Plusieurs méthodes proposent leurs techniques pour de telles réalisations. Pour notre projet, qui constitue la conception d'un générateur d'application WEB, nous utiliserons le langage UML (Unified Modeling Language), issu des méthodes « Booch » et « OMT ».

Dans une première phase, celle de l'analyse, nous allons d'abord appréhender les besoins du programmeur dans l'utilisation du générateur. Il s'agira ici de recenser les fonctionnalités requises pour une bonne génération d'application. Ensuite, il sera question de présenter l'IHM de l'application générer telle que désirée par le service de génie logiciel de ABLE-SARL.

La deuxième phase constituera essentiellement la conception de notre système de génération. Il s'agira de l'établissement du diagramme des cas d'utilisation, des diagrammes de séquences et des diagrammes de classes.

## III.1- Analyse

### III.1.1- Cahier des charges

La première étape dans le cycle de développement d'un logiciel est l'expression des besoins. Cette phase se termine par un document, généralement contractuel, qui résume ce que l'on attend du logiciel : on parle de « **cahier de charges** ».

Dans notre travail, il est question de mettre sur pied un générateur de squelettes d'applications web. Le cahier des charges est reparti en deux blocs : un bloc pour le **générateur**, un autre pour l'**application générée**.

#### *III.1.1.1- A propos du générateur*

Le générateur sera utilisé par des programmeurs, et cela devra leur permettre de générer des squelettes d'applications web. Un squelette généré sera fonction des valeurs entrées et des configurations faites par le programmeur. Cela lui permettra donc d'aller très vite dans l'implémentation des logiciels.

Le programmeur pourra créer un projet à partir duquel il générera une application. L'application générée utilisera les technologies du J2EE. Elle sera OPEN SOURCE, pour permettre l'ajout de nouvelles fonctionnalités et permettre des éventuelles modifications au gré du programmeur. Aussi, elle devra respecter l'architecture MVC et fonctionner mode en client léger.

Pour décrire les entités du projet, le programmeur devra entrer :

- ❖ soit un code Java des classes métiers ;
- ❖ soit un script de base de données ;
- ❖ ou les deux.

Le générateur devra aussi permettre au programmeur d'entrer les informations sur les entités manuellement. Ces informations concernent le nom de l'entité, ses attributs, et à chaque fois, le nom, le type et la description de l'attribut.

Le programmeur pourra définir manuellement les contraintes sur les attributs des entités et également établir les relations inter-entités.

Il pourra sauvegarder l'état du système à un moment donné et pouvoir le recharger au moment souhaité. On dira qu'il enregistre un projet.

Le générateur devra prendre en compte les versions des applications générées.

Les contrôleurs seront générés automatiquement. Mais dans un mode avancé, le programmeur devra pouvoir modifier au besoin, les codes des contrôleurs, les codes des modèles, les codes des vues, et pourra également effectuer des configurations relatives au SGBD et au mapper utilisés.

Enfin, le programmeur peut décider, à tout moment, de construire le construire et de l'exécuter.

#### Récapitulatif des exigences fonctionnelles et non fonctionnelles du générateur

Le tableau qui suit résume les besoins en précisant à chaque fois catégorie, description, justification et priorité.

Catégorie	Identifiant	Description et Justification	Priorité
<b>Fonctionnelle</b>	<b>E01 :</b> Création du projet	Un projet est tout simplement l'ensemble d'informations décrivant l'application à générer. Il s'agit du nom du projet, de la version de l'application et de ses entités.	HAUTE
	<b>E02 :</b> Gestion des entités	Une entité a un nom et un ensemble d'attributs. Le programmeur devra pouvoir ajouter, supprimer ou modifier une entité. Les relations entre les entités doivent également être spécifiées. Les entités sont entrées manuellement ou peuvent être lues dans un fichier code Java de classes d'entité, dans un script de base de données.	HAUTE

<b>Fonctionnelle</b>	<b>E03 :</b> Sauvegarde d'états du système	Le programmeur peut à tout moment vouloir garder l'état du système. Si les configurations ne sont pas encore toutes faites et que le programmeur n'est pas encore prêt pour lancer la génération, il devra pouvoir enregistrer l'état. Il pourra à tout moment restaurer l'état.	MOYENN E
	<b>E04 :</b> Génération	Le générateur construit une application à partir des informations du projet.	HAUTE
	<b>E05 :</b> Exécution du projet	Le générateur exécute l'application construite.	
	<b>E06 :</b> Mode avancé	Le programmeur dans ce mode modifie les codes et effectue des configurations.	
<b>Non fonctionnelle</b>	<b>E07 :</b> Performance	La génération d'une application à partir d'un projet devra se faire en moins de 10min. C'est la rapidité dans la génération qui donnera du crédit au générateur.	HAUTE
	<b>E08 :</b> IHM optimal	Interface Homme Machine sera améliorée le mieux possible. Le programmeur doit atteindre son but avec le moins d'interfaces possible. Un dialogue de validation sera adressé à l'utilisateur avant toute opération sensible ou non réversible !	MOYENN E

**Tableau 1 : Récapitulatif et description des exigences du générateur**

### *III.1.1.2- A propos de l'application générée*

L'application générée pourra être directement exploitée par un utilisateur. Pour cela, il devra avoir un jeu d'interfaces graphiques permettant de faire des enregistrements, d'afficher des listes de données, de visualiser les valeurs d'une donnée, de faire des modifications et des suppressions. Il s'agit là d'un **CRUD** (Create-Read-Update-Delete) complet.

La **recherche** d'une donnée doit être simplifiée dans l'application. Au lieu de parcourir des listes la recherche d'une donnée, l'utilisateur devra pouvoir, à partir d'un **mot indicatif**, trouver la donnée recherchée.

La **sécurité** dans l'application générée doit également directement intégrée. Elle sera être gérée via l'utilisation des comptes. Un compte sera soit de type « *Administrateur* », soit de type « *Utilisateur* ».

Un utilisateur devra s'identifiera par un **login** et un **mot de passe**. Une fois connecté, l'utilisateur a, sur les différentes données, les droits que lui donne l'Administrateur.

Seul un Administrateur (*compte de type administrateur*) pourra créer des comptes, il aura d'ailleurs des droits pour toute autre opération.

Par contre, un utilisateur simple (*compte de type utilisateur*) ne pourra que consulter les informations sur les entités, tel que afficher une liste d'entités... Mais il ne pourra pas consulter une information relative à un compte.

### **III.1.2- Spécifications du système pour le générateur**

L'expression des besoins est insuffisante et trop imprécise pour servir de base à la réalisation du système. Il est nécessaire pour l'affiner et la compléter de mettre en place une phase d'élaboration des spécifications qui servira de base de travail aux concepteurs du système.

L'objectif des spécifications est de délimiter précisément le système et de décrire les différentes manières de l'utiliser du point de vue utilisateurs.

Pour cela, nous déterminerons quels sont les acteurs et quels sont les cas d'utilisation de notre système.

#### *III.1.2.1- Acteur du système*

Les acteurs sont les utilisateurs extérieurs au système qui ont une bonne connaissance des fonctionnalités du système. Cette connaissance n'est pas une connaissance de la structure du système, mais simplement de son interface d'utilisation.

Le générateur est un outil pour le programmeur. Il devra lui permettre d'accélérer les mises sur pied des logiciels.

Ainsi donc, le seul acteur du système de génération c'est le **programmeur**.

### *III.1.2.2- Cas d'utilisation*

Les use cases (ou cas d'utilisation en français) constituent le concept principal de la méthode OOSE de Ivar Jacobson, l'un des pères de l'UML. Dans le processus de standardisation des méthodes objets ayant abouti au formalisme UML, le concept de use cases a été repris dans le but d'effectuer une bonne délimitation du système et également d'améliorer la compréhension de son fonctionnement. Ils représentent donc le moyen de décrire le caractère fonctionnel des objets, on parle de représentation orienté « fonctionnalité » du système.

L'exploitation du **Tableau 1**, du récapitulatif des exigences du générateur, nous permet d'établir les cas d'utilisateur suivants :

La principale utilisation du système consiste à **générer une application** (*confère exigence E04*). Le programmeur pourra **exécuter l'application générée**.

Le programmeur devra pouvoir, selon l'exigence **E03**

- **sauvegarder** l'état du système à un moment donné (enregistrer un projet) ;
- **charger** un état sauvegardé (ouvrir un projet).

De l'exigence **E02**, on aura les uses case :

- **ajouter, modifier, supprimer** une entité dans un projet;
- **ajouter, modifier, supprimer** un attribut dans une entité;
- **établir une relation, supprimer une relation** entre deux entités.

Appelons **projet** un ensemble d'entités d'un même projet. On pourra alors ajouter qu'un programmeur peut également **créer un projet** (exigence **E01**).



De l'exigence **E06**, on tire que le programmeur peut **ouvrir le mode avancé** où il pourra **modifier les codes** (d'un modèle, d'un contrôleur ou d'une vue) et **effectuer des configurations**.

Le tableau ci-dessous récapitule tous les cas d'utilisation recensés dans le système du générateur :

Identifiant	Nom du use case
cas_01	créer un nouveau projet
cas_02	enregistrer un projet
cas_03	ouvrir un projet
cas_04	ajouter une entité dans un projet
cas_05	modifier une entité dans un projet
cas_06	supprimer une entité dans un projet
cas_07	ajouter un attribut à une entité
cas_08	modifier un attribut d'une entité
cas_09	supprimer un attribut à une entité
cas_10	établir une relation entre 02 entités
cas_11	supprimer une relation
cas_12	générer une application
cas_13	exécuter l'application
cas_14	ouvrir le mode avancé
cas_15	modifier les codes
cas_16	Effectuer des configurations

**Tableau 2 : Récapitulatif des cas d'utilisation du générateur**

### *III.1.2.3- Description des cas d'utilisation*

*La méthode que nous utilisons prévoit plusieurs formes de description des cas d'utilisation. Mais, dans notre travail, nous ferons les descriptions, juste sous **forme textuelle** et sous **forme de scénario**.*

### ➤ **Contraintes de conception**

Mais, avant de passer à cette phase, il est important de mentionner certaines contraintes de conception.

La première contrainte concerne les noms des **entités** ceux des **attributs**. C'est que dans un même **projet**, deux **entités** ne doivent pas avoir le même nom, de même que dans une **entité**, deux **attributs** ne doivent pas porter le même nom.

La deuxième contrainte concerne la répercussion de la suppression d'une **entité**. C'est que la suppression d'une **entité** entraîne la suppression de toutes ses **relations**.

### ➤ *Description des cas d'utilisation*

*cas\_01 : créer un nouveau projet* (forme textuelle)

- ❖ le programmeur entre un *nom* (obligatoire) et une *version* ;
- ❖ le système crée un *projet* avec ces éléments.

*cas\_02 : enregistrer un projet* (forme textuelle)

- ❖ le système crée un *fichier spécial* portant le nom du *projet* ;
- ❖ le système y sauvegarde toutes les informations du *projet*.

*cas\_03 : ouvrir un projet* (forme textuelle)

- ❖ le programmeur sélectionne un *fichier spécial* ;
- ❖ le système ouvre crée un *projet* et le charge avec les informations du fichier.

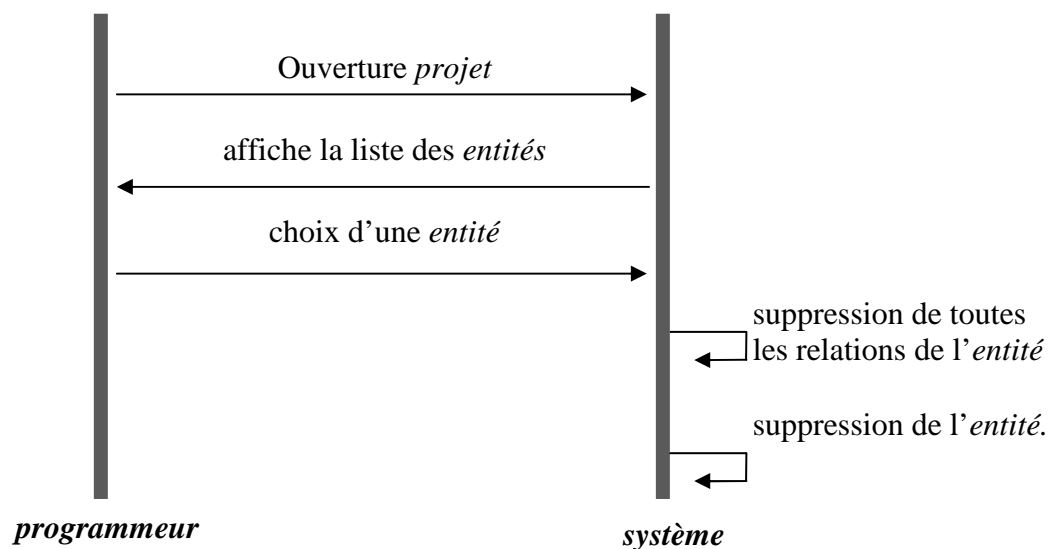
***cas\_04 : ajouter une entité dans un projet*** (forme textuelle)

- ❖ le programmeur ouvre ou crée un *projet* ;
- ❖ entre un *nomEntite* ;
- ❖ le système crée une *entité* ayant ce nom ;
- ❖ puis l'ajoute au *projet*.

***cas\_05 : modifier une entité dans un projet*** (forme textuelle)

- ❖ le programmeur sélectionne, dans un *projet* ouvert, une *entité* ;
- ❖ entre les nouvelles valeurs ;
- ❖ le système modifie les propriétés de *l'entité* avec les nouvelles valeurs.

***cas\_06 : supprimer une entité dans un projet*** (forme de scénario)

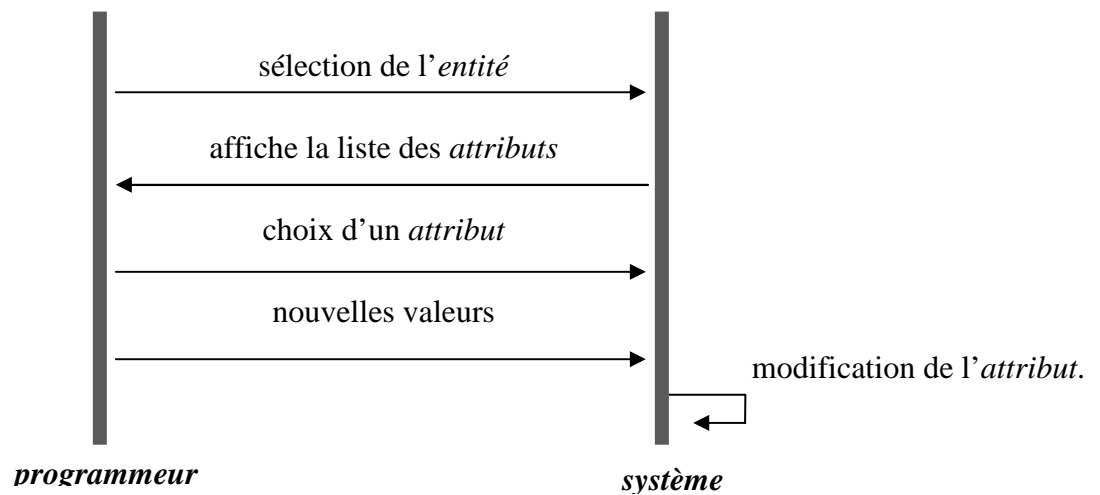


**Figure 7 : Scénario du cas d'utilisation « supprimer une entité dans un projet »**

***cas\_07 : ajouter un attribut à une entité***(forme textuelle)

- ❖ le programmeur sélectionne, dans un *projet* ouvert, une *entité* ;
- ❖ entre un *nomAttribut*, un *typeAttribut*, une *descriptionAttribut* et des *contraintesAttribut* ;
- ❖ le système crée un *attribut* avec ces valeurs ;
- ❖ puis l'ajoute à *l'entité* sélectionnée.

***cas\_08 : modifier un attribut d'une entité*** (forme de scénario)

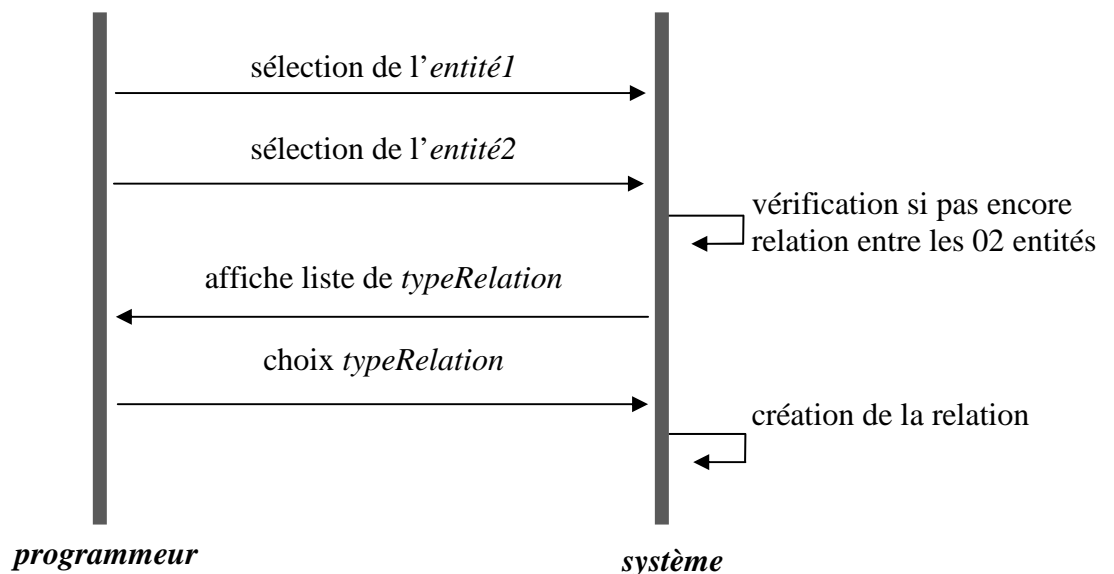


**Figure 8 : Scénario du cas d'utilisation « modifier un attribut d'une entité »**

***cas\_09 : supprimer un attribut à une entité***

- ❖ le programmeur sélectionne, dans un *projet* ouvert, une entité ;
- ❖ le système affiche les *attributs* de l'entité ;
- ❖ le programmeur sélectionne un *attribut* ;
- ❖ le système supprime l'*attribut* sélectionné.

***cas\_10 : établir une relation entre 02 entités*** (forme de scénario)



**Figure 9 : Scénario du cas d'utilisation « établir une relation entre 02 entités »**

***cas\_11 : supprimer une relation*** (forme textuelle)

- ❖ le système affiche les *relations* du projet ouvert ;
- ❖ le programmeur sélectionne une *relation* ;
- ❖ et le système la supprime.

***cas\_12 : générer une application*** (forme textuelle)

- ❖ le programmeur crée ou ouvre un *projet* ;
- ❖ le système construit une application web OPEN SOURCE avec les informations du projet.

***cas\_13 : exécuter l'application*** (forme textuelle)

- ❖ le programmeur génère une *application* ;
- ❖ le système exécute l'*application* générée.

***cas\_14 : ouvrir mode avancé*** (forme textuelle)

- ❖ le système ouvre une interface avec des fonctionnalités permettant de modifier les codes et d'effectuer des configurations.

***cas\_15 : modifier les codes*** (forme textuelle)

- ❖ le programmeur ouvre le mode avancé ;
- ❖ le système affiche les fichiers regroupés par niveau (*fichiers du modèle, fichiers des contrôleurs, fichiers des vues*) ;
- ❖ le programmeur choisit un fichier ;
- ❖ le système ouvre le fichier en édition.

***cas\_16 : effectuer des configurations*** (forme textuelle)

- ❖ le programmeur ouvre le mode avancé ;
- ❖ le système affiche les fichiers de configuration (*de SGBD, de mapping, de démarrage*) ;
- ❖ le programmeur choisit un fichier ;
- ❖ le système ouvre le fichier en édition.

### III.1.3- Spécification du système de l'application web résultante de la génération

#### III.1.3.1- Acteurs et cas d'utilisation

Les acteurs de ce système sont les utilisateurs et les administrateurs.

Les cas d'utilisation seront essentiellement CRUD.

Un utilisateur peut consulter des informations, en dehors de celles des comptes. Il peut également effectuer des recherches.

Un administrateur est aussi un utilisateur. Seulement, il a tous les droits. Il peut créer, modifier, supprimer ou consulter les données.

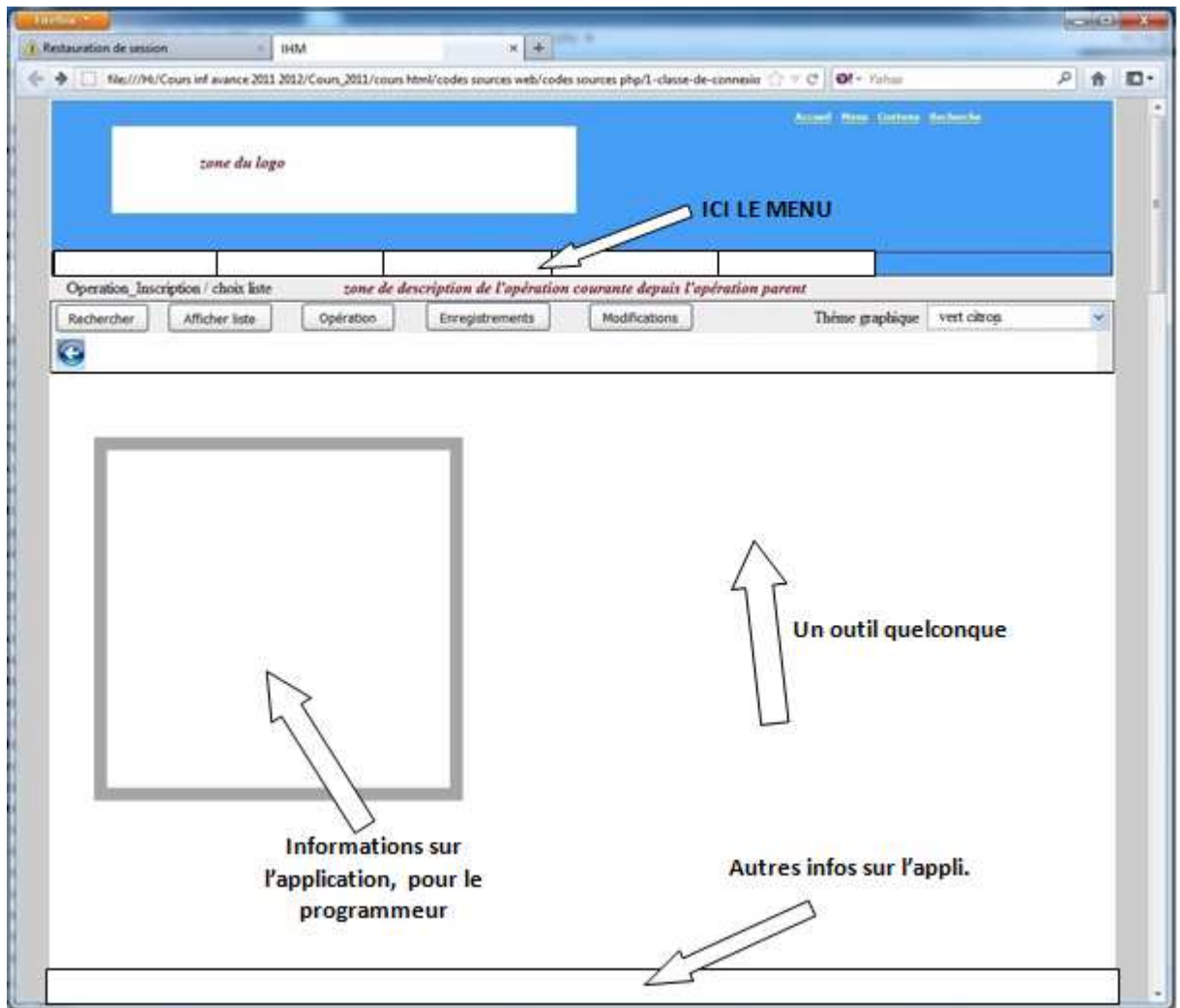
Les cas d'utilisation peuvent être résumés dans le tableau ci-dessous :

N°	Use cases	Description sous forme textuelle	Acteurs
1.	<i>se connecter</i>	i- l'utilisateur entre son <b>login</b> et son <b>password</b> ii- le système vérifie l'authentification iii- le système connecte l'utilisateur si succès.	<i>Administrateur et utilisateur simple</i>
2.	<i>Rechercher</i>	i- l'utilisateur entre un mot clé ii- le système affiche les données ayant cette valeur	
3.	<i>Consulter</i>	i- l'utilisateur sélectionne un ensemble ii- le système affiche la liste des éléments iii- l'utilisateur sélectionne un élément iv- le système affiche les informations de l'élément sélectionné.	
4.	<i>Ajouter</i>	i- l'Administrateur sélectionne un ensemble ii- entre des valeurs iii- le système crée un élément avec les valeurs entrées.	<i>Administrateur uniquement</i>
5.	<i>Modifier</i>	i- l'administrateur sélectionne un ensemble ii- le système affiche les éléments de l'ensemble iii- l'administrateur sélectionne un élément iv- entre de nouvelles valeurs v- le système modifie les informations de l'élément sélectionné avec les nouvelles valeurs.	
6.	<i>Supprimer</i>	i- l'administrateur sélectionne un ensemble ii- le système affiche les éléments de l'ensemble iii- l'administrateur sélectionne un élément iv- le système supprime l'élément sélectionné.	

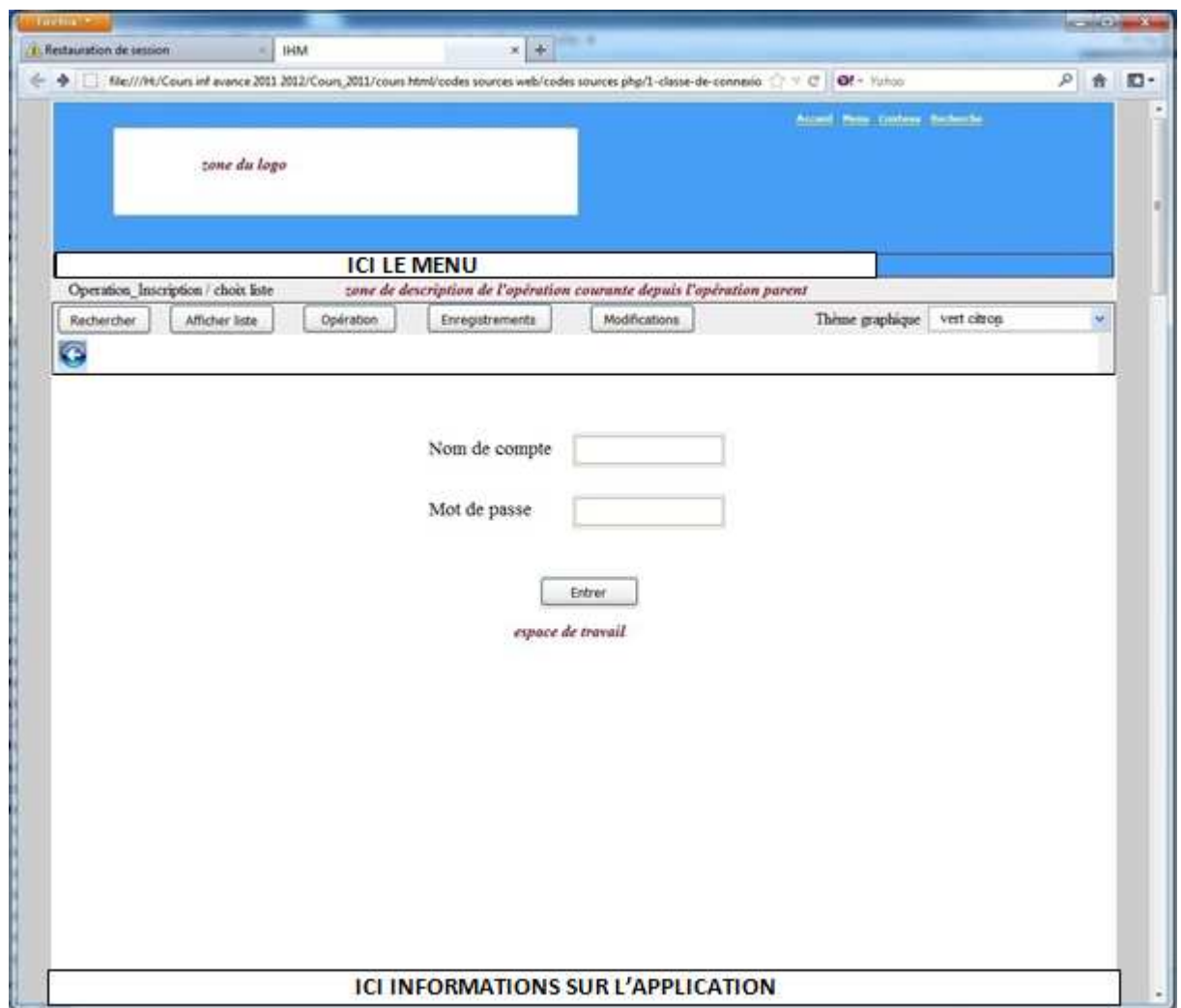
**Tableau 3 : Description des cas d'utilisation de l'application générée, par type d'utilisateur**

### III.1.3.2- IHM de l'application web générée

Une autre spécification que nous pouvons apporter à l'application web générée est son Interface Homme Utilisateur (IHM). L'interface principale et l'interface de connexion auront respectivement les apparences suivantes :



**Figure 10 : Maquette de l'interface principale de l'application générée**



**Figure 11 : Maquette de l'interface de connexion de l'application générée**



## III.2- Conception

Dans cette phase de conception, nous allons décrire, en utilisant le langage de modélisation UML, le fonctionnement des futurs systèmes : le système du générateur d'applications web et le système de l'application web générée. Nous illustrerons ces descriptions par des diagrammes que nous concevrons :

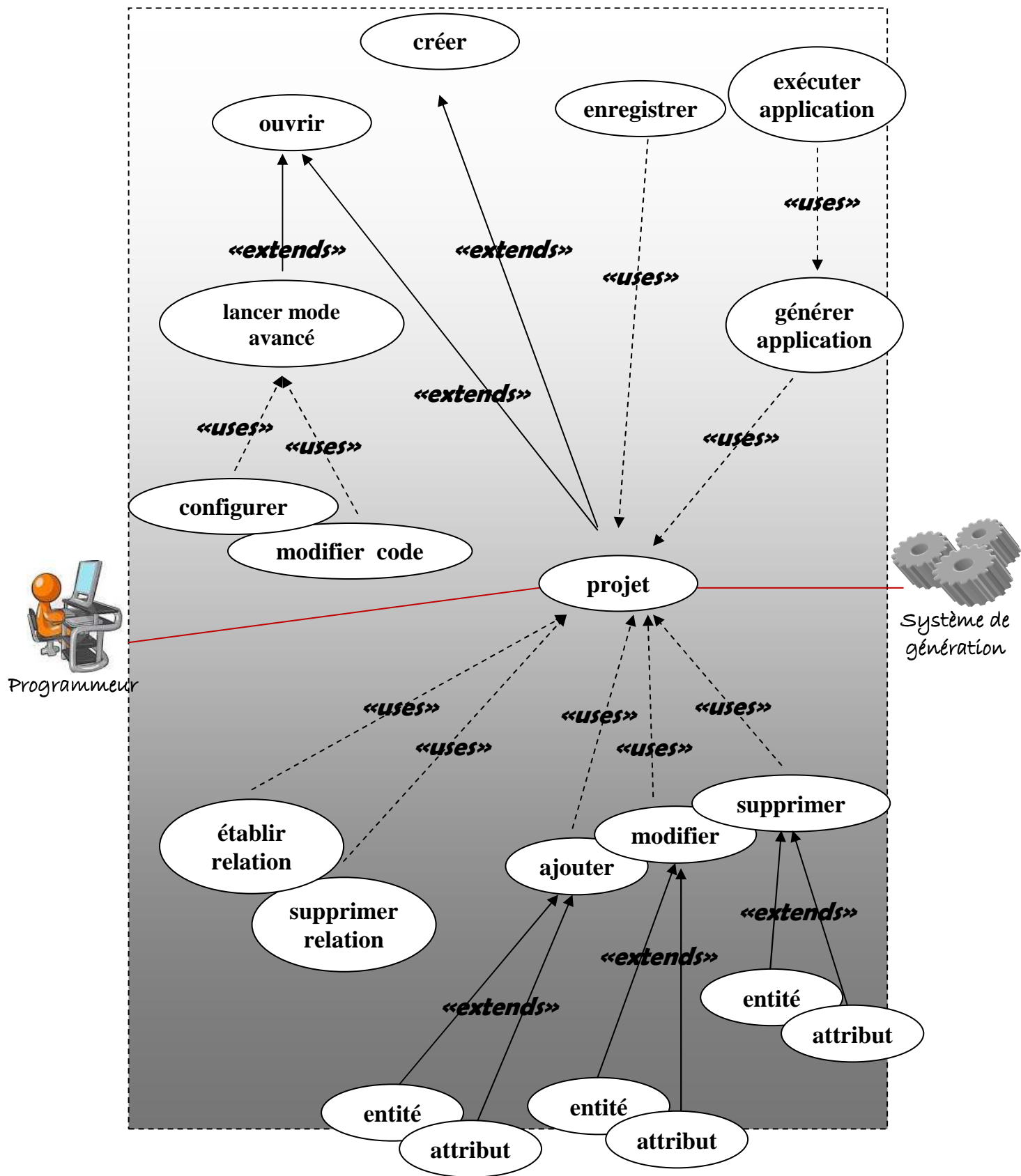
- un diagramme des cas d'utilisation pour du générateur ;
- un diagramme des cas d'utilisation pour l'application qui sera générée ;
- un diagramme des classes pour le générateur.

### III.2.1- Diagrammes des use cases

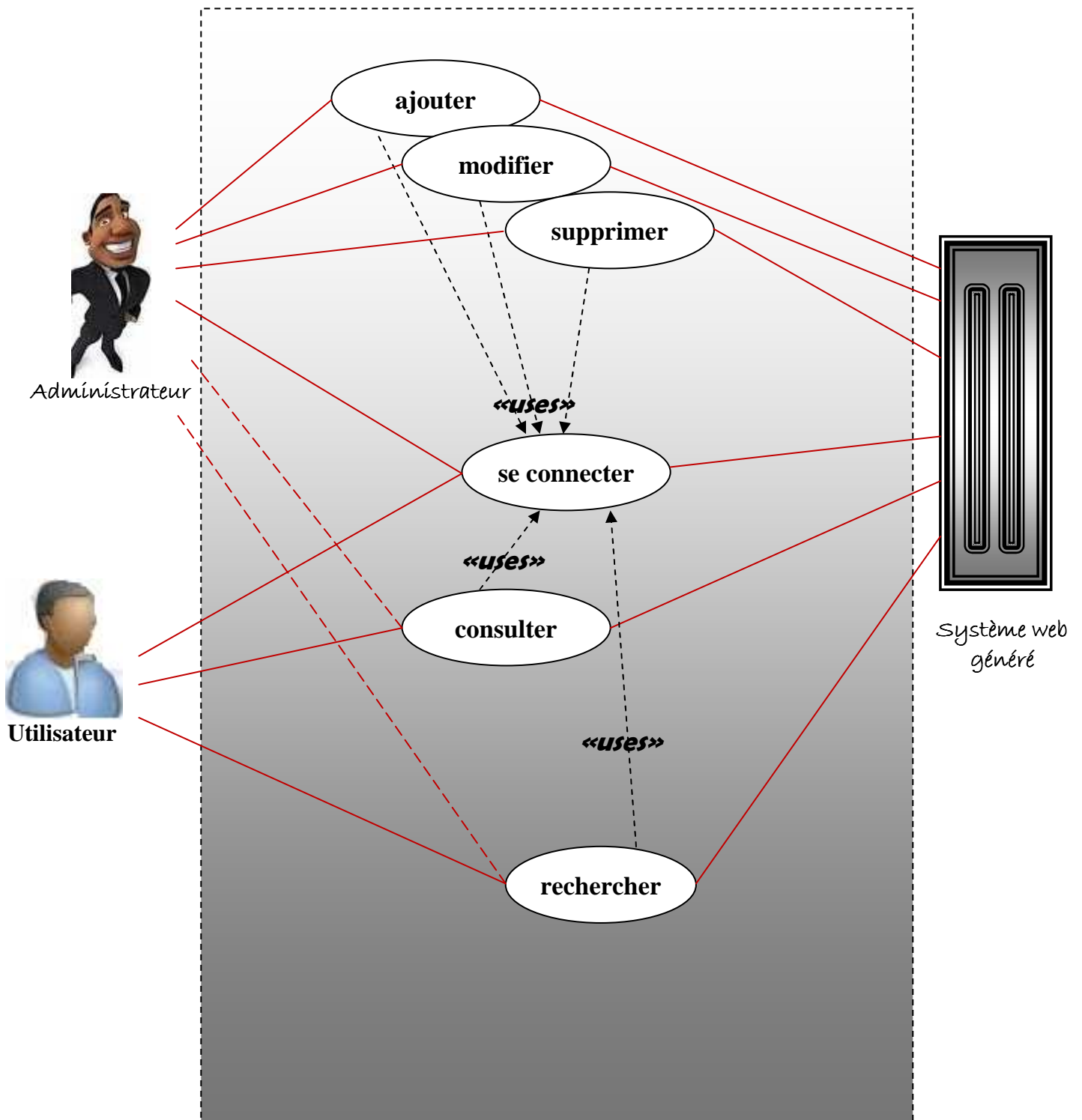
Un diagramme des use cases illustre l'organisation générale de l'utilisation du système par ses acteurs. L'analyse des cas d'utilisation faite dans la partie précédente nous permet de concevoir les deux diagrammes ci-dessous. Le premier pour le générateur et le second pour l'application générée.

Le mot clé *extends* regroupe les use cases sémantiquement égaux. Dans notre cas, *supprimer une entité* et *supprimer un attribut* sont tous des use cases de suppression, par exemple.

Le mot clé *uses* signifie qu'un use case fait appel à un autre, le use case pointé est une sous partie de l'autre. Par exemple, pour *configurer*, il faut au préalable *ouvrir le mode avancé*.



**Figure 12 : Diagramme des cas d'utilisation du générateur**



***Figure 13 : Diagramme des cas d'utilisation de l'application générée***

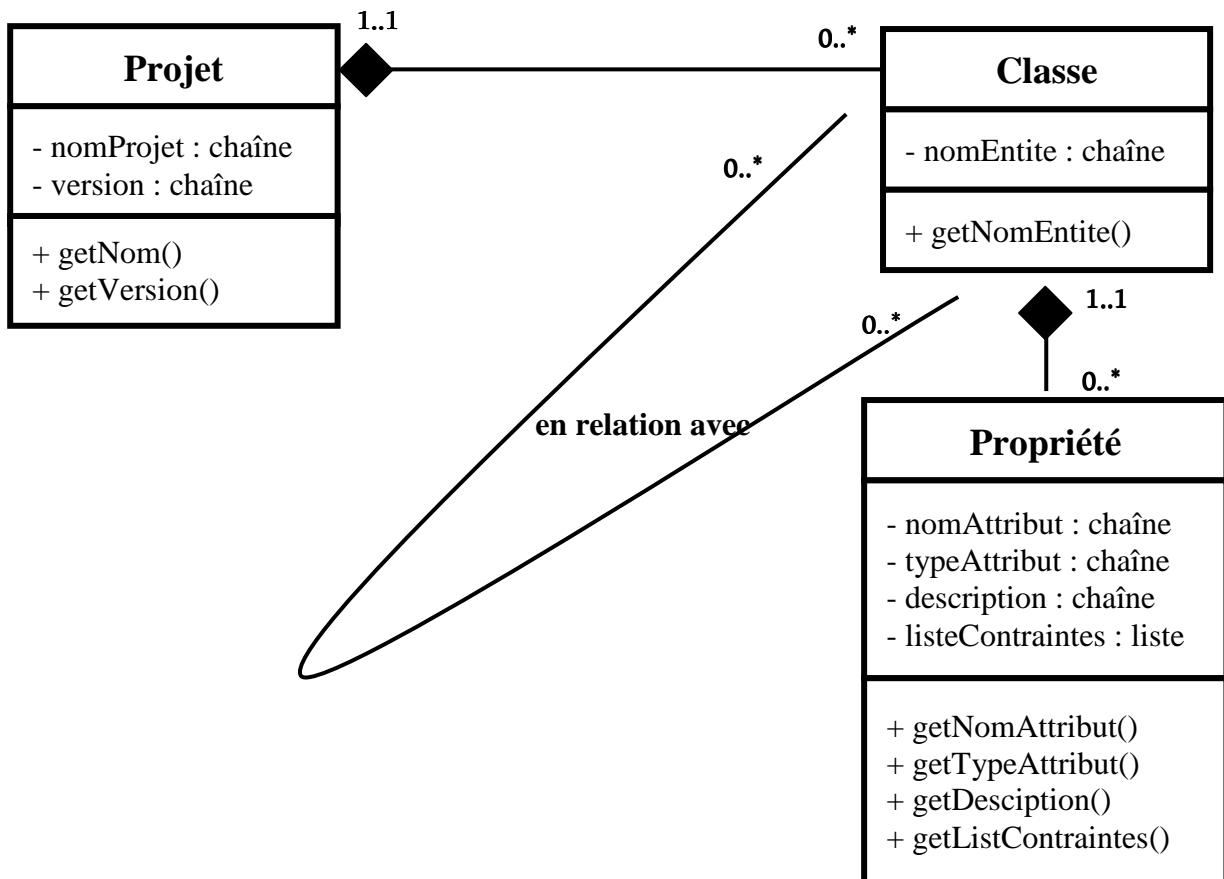
### III.2.2- Diagramme de classes du générateur

#### III.2.2.1- Description des entités

Les entités que nous pouvons recenser sont les suivantes :

- ✚ l'entité **attribut** qui possède :
  - un nom (**nomAttribut**) ;
  - un type (**typeAttribut**) ;
  - une description (**descriptionAttribut**) ;
  - un ensemble de contraintes (**listeContraintes**).
- ✚ L'entité **entité** qui a :
  - un nom (**nomEntite**) ;
  - un ensemble d'attributs (**listeAttributs**).
- ✚ l'entité **projet** qui a :
  - un nom (**nomProjet**) ;
  - une version (**versionProjet**) ;
  - un ensemble d'entités (**listeEntites**).

#### III.2.2.2- Représentation du diagramme



**Figure 14 : Diagramme de classes**

## **Conclusion**

Il était question dans cette partie de représenter, par des modèles, le système à mettre en place. Il s'agit d'un générateur de squelettes d'application web. Il a fallu donc illustrer les attentes du générateur et celles de l'application qui sera générée.

En utilisant le langage UML et les méthodes qui la constituent, nous avons abordé la modélisation en deux étapes. La première étapes, la phase d'analyse, nous a permis de lister les résultats attendus, en termes de fonctionnalités et de performance, ce que nous avons appelé exigences fonctionnelles et non fonctionnelles. Et, de cela, nous avons effectué les descriptions des différents systèmes. Dans la phase de conception, nous avons illustré par des diagrammes les résultats.

Cette modélisation facilitera dans la suite, l'implémentation du projet.

## CHAPITRE IV : IMPLEMENTATION ET IMPLANTATION

### Introduction

Dans le cycle du développement d'un logiciel, l'implémentation est la phase qui consiste à mettre en exécution les résultats de la conception. Il s'agit dans notre cas d'écrire des algorithmes, de configurer des outils... pour mettre sur pied un générateur de squelettes d'applications web pour l'entreprise ABLE-SARL. Nous avons réalisé cet objectif. Le produit résultant a été baptisé **Wable**. Ce chapitre consistera à présenter le travail effectué dans son fond. Nous allons présenter les outils utilisés, parmi lesquels **Grails**, qui est framework permettant de faire des développements avec des RAD. Nous présenterons les configurations qui y ont été apportées. Nous commenterons également les codes que nous avons édités dans le projet. Il s'agira également de décrire le processus d'implantation de ce générateur au sein de l'atelier de génie logiciel de l'entreprise.

### IV.1- Implémentation du générateur

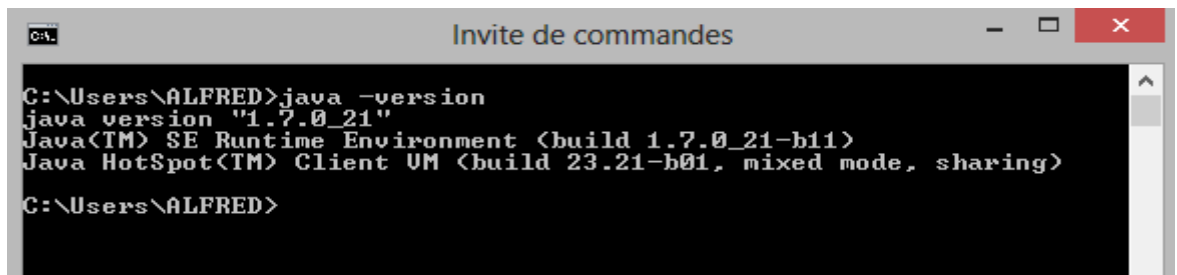
#### IV.1.1- Environnement de développement

##### *IV.1.1.1- L'IDE NetBeans*

Un IDE (Integrated Development Environment) est un environnement de développement intégré réunissant tous les outils nécessaires à la création d'application. Celui que nous avons utilisé pour la mise en œuvre du logiciel de génération est **NetBeans 7.3.1**.

##### *IV.1.1.2- Le langage Java*

Le langage utilisé pour écrire nos algorithmes est le **Java**. La version utilisée est la **1.7**, tel que présenté dans l'invite de commande capturée ci-dessous :



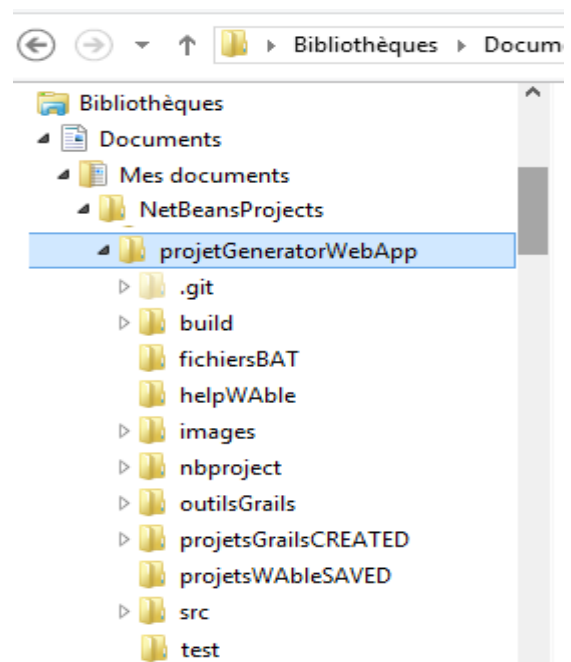
```
C:\Users\ALFRED>java -version
java version "1.7.0_21"
Java(TM) SE Runtime Environment (build 1.7.0_21-b11)
Java HotSpot(TM) Client VM (build 23.21-b01, mixed mode, sharing)
C:\Users\ALFRED>
```

**Figure 15 : Version du java utilisée**

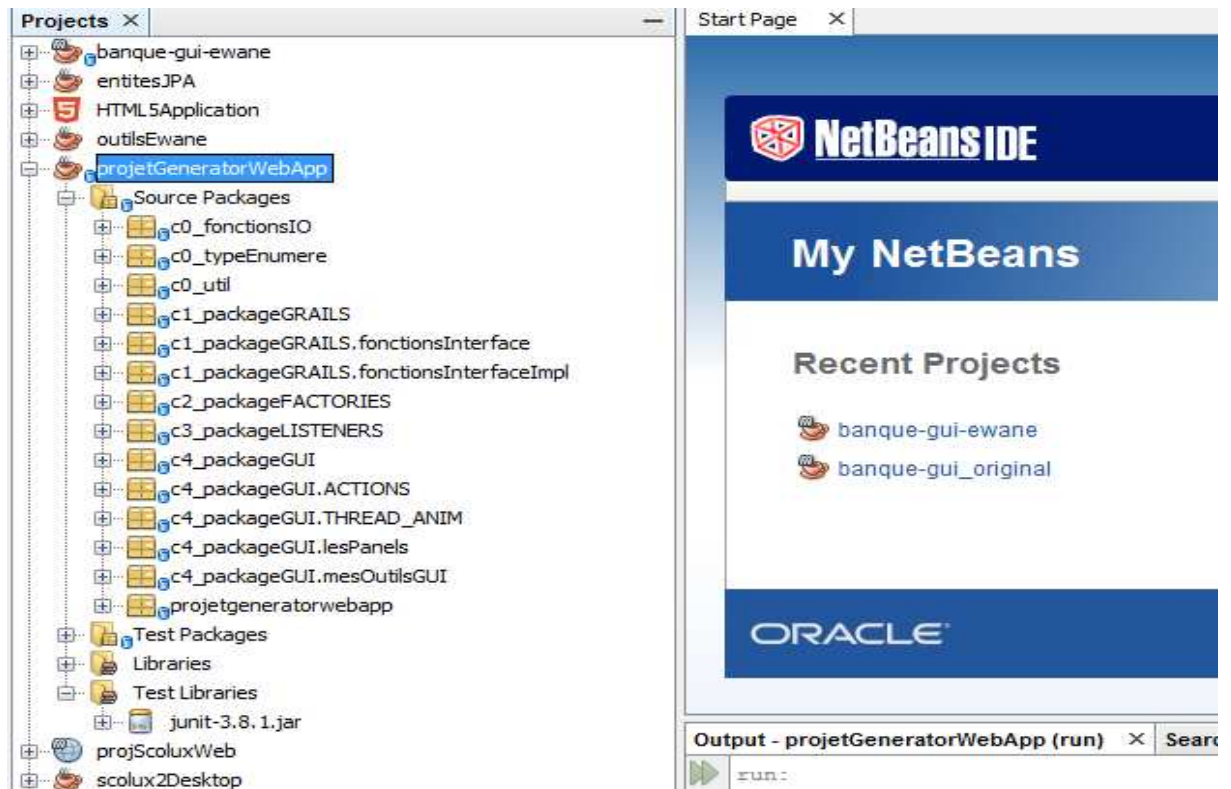
## **IV.1.2- Présentation du projet**

### *IV.1.2.1- L'arborescence*

L'arborescence du projet représente ici l'empaquetage et l'organisation des différents fichiers du projet. Le projet Java que nous avons créé sur NetBeans est constitué en paquets, en fonction de leurs utilités. Les captures ci-dessous présentent cette arborescence, d'abord dans un explorateur Windows, et ensuite dans l'IDE NetBeans.



**Figure 16 : Arborescence du projet  
dans l'explorateur Windows**



**Figure 17 : Arborescence du projet dans l'IDE NetBeans**

#### IV.1.2.2- Les classes « métier »

Le paquetage *c1\_packageGRAILS* contient les fichiers *Attribut.java*, *ClasseEntite.java*, *ProjetGRAILS.java* et *Relation.java*. Ces quatre fichiers JAVA constituent les principales classes « métier » du projet. Voici un aperçu de leurs différents codes :

```
public class Attribut implements Serializable {
    private String nomAttribut;
    private String type;
    private String contraintes;
    private String description;

    public Attribut() {
    }

    //getters et setters
}
```

```
public class ClasseEntite implements Serializable{
    private String nomEntite;
    private List<Attribut> listeAttributs;

    public ClasseEntite() {
    }

    //getters et setters
}
```



<pre> public class <b>ProjetGRAILS</b> implements Serializable{      private String nomDuProjet;     private String version;     private List&lt;ClasseEntite&gt; listeDesEntites;     public String repertoireDuProjet;     public String repertoireDomain;     public String repertoireController;     public String nomPackage;     private List&lt;Relation&gt; listeRelations;      public ProjetGRAILS() {     }      //getters et setters } </pre>	<pre> public class <b>Relation</b> implements Serializable{      private ClasseEntite entite1;     private ClasseEntite entite2;     private TypeRelation typeRelation;     private boolean biDirect;      public Relation(ClasseEntite entite1, ClasseEntite entite2, TypeRelation typeRelation, boolean biDirect) {         this.entite1 = entite1;         this.entite2 = entite2;         this.typeRelation = typeRelation;         this.biDirect = biDirect;     }      //getters et setters } </pre>
---	--

Remarquons que les trois premières classes (*Attribut*, *ClasseEntite* et *ProjetGRAILS*) n'ont que de constructeur sans paramètres. Ceci est fait exprès, puisque les objets de ces classes seront créés plutôt dans les fabriques que nous présentons dans la partie suivante.

#### IV.1.2.3- Les classes *Factoring*

La création d'un **projet**, d'**entité** ou d'un **attribut** demandant l'observation de plusieurs contraintes, nous avons trouvé important de créer une fabrique qui se chargera de vérifier tout cela et renvoyer le produit désiré. Cette fabrique est ici représentée par des classes, dites *Factoring*, du paquetage *c2\_packageFACTORIES*.

Ainsi donc, toutes les opérations liées à la gestion d'un **projet**, d'une **entité** ou d'un **attribut** sont regroupées dans ce paquetage. C'est le cas par exemple des fonctions *getEntity(params)* de la classe *EntitiesFactory* et de *etablirRelation(params)* de la classe *ProjetFactory*, que nous présentons ci-dessous.

```

public static ClasseEntite getEntity(String nomEntite) {

    ClasseEntite ce = new ClasseEntite();
    nomEntite = arrangeNomEntite(nomEntite);

    if (FonctionsDUsine.mauvaisNom(nomEntite)) {
        return null;
    } else {
        ce.setNomEntite(nomEntite);
        List<Attribut> listeAttributs = new ArrayList<>();
        ce.setListeAttributs(listeAttributs);
        return ce;
    }
}

```

Le paramètre de la fonction *getEntity*, représentée ci-contre, est utilisé dans deux autres fonctions : *arrangeNonEntite* et *mauvaisNom*.

La première retire les éventuels caractères spéciaux dans le nom en paramètre et met la première lettre en majuscule (juste conventionnel) ; la deuxième se vérifie si malgré les arrangements, le nom n'est pas un nom valide.

La deuxième fonction est ***etablirRelation***, ci-dessous représentée. Elle établie dans un projet, une relation entre deux entités. Elle prend également en paramètre le type de relation. Le programmeur pourra aussi préciser si cette relation est bidirectionnelle ou non.

```
public static boolean etablirRelation( ProjetGRAILS projet,
    String nomEntite1, String nomEntite2, TypeRelation typeRelation, boolean biDirect ) {

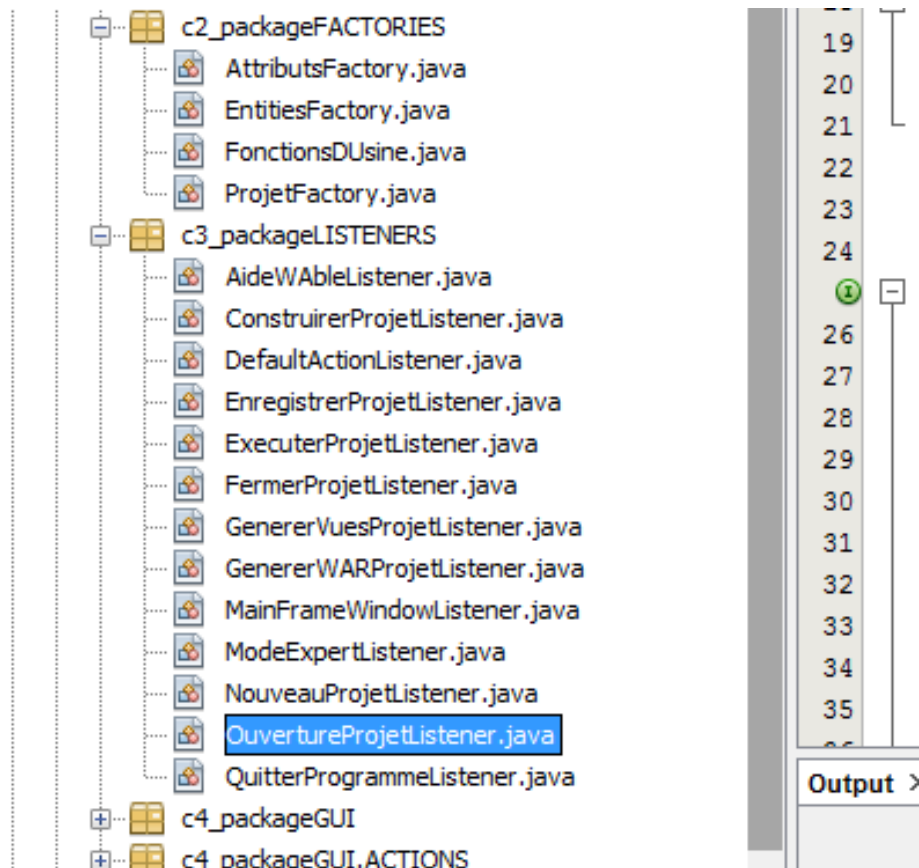
    ClasseEntite entite1 = retourneClasseEntite(projet, nomEntite1);
    ClasseEntite entite2 = retourneClasseEntite(projet, nomEntite2);
    try {
        if (existsRelation(projet, entite1, entite2)) {
            return false;
        }
        projet.getListeRelations().add(new Relation(entite1, entite2, typeRelation, biDirect));
    } catch (NullPointerException exc) {
        return false;
    }
    return true;
}
```

On peut constater qu'avant d'établir une relation, cette fonction vérifie d'abord, via la fonction ***existeRelation***, si une telle relation existe déjà ; auquel cas la relation n'est pas établie. On évite ainsi des redondances dans le projet.

#### *IV.1.2.4- Les classes LISTENER*

Pour programmer une application correctement, il est important de créer ses propres ***Listeners***. Il s'agit là des classes qui écoutent les évènements au niveau de la vue et lance les exécutions adéquates. Ces objets, en Java, se trouvent dans le paquetage ***java.awt.event***.

Le paquetage ***c3\_packageLISTENERS*** de notre projet est celui dans lequel ces « écouteurs » ont été regroupés.



**Figure 18 : Contenus des paquetages FACTORIES et LISTENERS**

Un zoom dans les deux premiers fichiers des *listeners* (*AideWableListener.java* et *ConstruireProjetListener.java*) nous présente les codes suivants :

```
package c3_packageLISTENERS;

import c0_fonctionsIO.MesFonctionsIO;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

/**
 *
 * @author ALFRED
 */
public class AideWableListener implements ActionListener {

    @Override
    public void actionPerformed(ActionEvent ae) {
        MesFonctionsIO.ouvrirFichier("helpWable\\Help_Wable.chm");
    }
}
```

```
package c3_packageLISTENERS;

import
c4_packageGUI.THREAD_ANIM.ConstruireProjetGRails;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

/**
 *
 * @author ALFRED
 */
public class ConstruireProjetListener
    implements ActionListener {

    @Override
    public void actionPerformed(ActionEvent ae) {
        new ConstruireProjetGRails().start();
    }
}
```

Lorsqu'un évènement est effectué sur le bouton d'aide (un clic dans notre cas), l'écouteur *AideWableListener* déclenche l'action en appelant la fonction **MesFonctionsIO.ouvrirFichier("helpWable\\Help\_Wable.chm");** qui consiste à ouvrir le fichier d'aide de Wable.

De même, lance la construction du projet, au niveau graphique, le *listener* récupère l'évènement et déclenche l'action **new ConstruireProjetGRails( ).start( ) ;** qui consiste à instancier et à lancer un thread qui va réellement lancer le processus de construction.

#### IV.1.3- Persistances des objets

La persistance d'un objet est le fait de le faire continuer à exister dans le temps. C'est tout simplement la sauvegarde de cet objet. L'objet principal que nous sauvegardons dans notre cas est le **projet**.

Un *projet Wable*, qui est un ensemble d'**entités**, chaque **entité** ayant un ensemble d'**attributs**, est enregistré dans fichier d'extension **.WABLE**. Ainsi donc, si un programmeur peut à tout moment sauvegarder un projet, et continuer l'entrée des informations quand il le souhaite.

Pour qu'un objet **Java** soit persisté, il faut que sa classe implémente l'interface *Serializable*. On constate bien dans le code ci-dessous que la classe *ProjetGRAILS* implémente cette interface.

```
public class ProjetGRAILS implements Serializable{

    private String nomDuProjet;
    private String version;
    public String repertoireDuProjet;
    public String repertoireDomain;
    public String repertoireController;
    public String nomPackage;
    private List<ClasseEntite> listeDesEntites;
    private List<Relation> listeRelations;

    public ProjetGRAILS() {
        ...
        ...
        ...
    }
}
```

Aussi, tous les objets qui liés à un objet Sérialisable doivent être Sérialisable. C'est pourquoi dans leurs définitions, les classes *ClasseEntite* et *Relation* ont également implémenté l'interface *Sérialisable*.

Le bout de code ci-dessous est celui qui nous permet de sauvegarder un projet. L'objet à enregistrer est *projetG*.

```
String chmFichier = "" + projetG.getNomDuProjet() + ".WABLE";
try {
    FileOutputStream fos = new FileOutputStream("projetsWableSAVED/" + chmFichier);
    ObjectOutputStream objIO = new ObjectOutputStream(fos);
    objIO.writeObject(projetG);
    objIO.close();
} catch (IOException e) {
    System.out.println("erreur de persistance11");
}
```

L'objet est enregistré sous le nom *projetG.getNomDuProjet( )* dans le dossier **projetsWableSAVED**, qui est un répertoire du projet.

Consulter le code permettant de charger le projet en *Annexes*.

#### IV.1.4- Génération et exécution du projet

La génération dans notre logiciel consiste tout simplement à construire un projet *Grails* avec les informations d'un projet *Wable*. Rappelons que *Wable* est le nom donné au logiciel conçu. Cette partie consiste à présenter le processus de génération et l'exécution d'un projet Grails à partir de notre application.

##### IV.1.4.1- La méthode *contruireLeProjet* de l'interface *ServiceGrails*

Le paquetage *c1\_packageGRAILS.fonctionsInterface* regroupe des interfaces qui portent les différents services Grails de notre application. Une de ces interfaces est, par exemple, *ServicesGrails*. Une capture du code du le fichier *ServicesGrails* nous montre ceci :

```

5 package c1_packageGRAILS.fonctionsInterface;
6
7 import c1_packageGRAILS.ClasseEntite;
8 import c1_packageGRAILS.ProjetGRAILS;
9 /**
10  * @author ALFRED
11  */
12 public interface ServicesGrails {
13     boolean contruireLeProjet(ProjetGRAILS prjGrails, String dossierProjet);
14     boolean attribuerProprietesApp(ProjetGRAILS prjGrails, String nom, String version);
15     boolean genererLeWAR(ProjetGRAILS prjGrails);
16     //cette fonction crée un domain-class dans un projet rails
17     boolean genererAllEntitesGrails(ProjetGRAILS prjGrails);
18     boolean genererEntiteGrails(ClasseEntite domain, ProjetGRAILS prjGrails, String cheminRep);
19     //crée un crotoller dans un projet rails
20     boolean genererAllControllersGrails(ProjetGRAILS prjGrails);
21     boolean genererControllerGrails(String nomController, ProjetGRAILS prjGrails, String cheminRep);
22 }
23

```

**Figure 19 : Capture du fichier ServicesGrails.java**

Remarquons à la 13<sup>ème</sup> ligne de ce code, la fonction ***boolean contruireLeProjet (ProjetGRAILS prjGrails, String dossierProjet).*** Cette fonction est implémentée dans la classe ***ServicesGrailsImpl*** du paquetage ***c1\_packageGRAILS.fonctionsInterfaceImpl.***

Elle transforme le projet en paramètre en un projet Grails réel, dans l'emplacement correspondant au chemin indiqué par le paramètre ***dossierProjet.***

Son code est le suivant :

```

public class ServicesGrailsImpl implements ServicesGrails {
    @Override
    public boolean contruireLeProjet(ProjetGRAILS prjGrails, String dossierProjet) {
        String repertoire = dossierProjet + "\\\" + prjGrails.getNomDuProjet();

        File fb = new File(dossierProjet);
        if (!fb.exists()) {
            MesFonctionsIO.creationRepertoire(dossierProjet);
        }
        MesFonctionsIO.copieRepertoire("outilsGrails/squeletteGrails", repertoire);

        String nomPackage = prjGrails.getNomDuProjet().toLowerCase();
        String repDomain = repertoire + "/grails-app/domain/" + nomPackage;
        String repController = repertoire + "/grails-app/controllers/" + nomPackage;

        MesFonctionsIO.creationRepertoire(repDomain);
        MesFonctionsIO.creationRepertoire(repController);

        prjGrails.nomPackage = nomPackage;
        prjGrails.repertoireDuProjet = repertoire;
        prjGrails.repertoireDomain = repDomain;
        prjGrails.repertoireController = repController;

        genererAllEntitesGrails(prjGrails);
        genererAllControllersGrails(prjGrails);
        return true;
    }
}

```

#### IV.1.4.2- Le fichier *runGrails.bat*

Rappelons que dans les généralités, nous avons dit que pour exécuter un projet **Grails**, il faut se tenir dans le répertoire du projet et exécuter la commande ***grails run-app***.

Lorsque le projet **Grails** est construit, c'est exactement ce même principe qui est utilisé pour exécuter le projet.

Le fichier ***runGrails.bat*** est un fichier **.bat** dans lequel nous copions les commandes nécessaires pour cette exécution.

Le bout de code ci-dessous est celui permettant de lancer cette exécution :

```
String rep = CreationProjet.projetEnCreation.projetGRAILS.repertoireDuProjet;
if (rep == null) {
    AlerteDialogue.erreur("Veuillez d'abord construire le projet SVP !");
} else {

    MesFonctionsIO.ecrireDansFichier("fichiersBAT/runGrails.bat", "cd "+rep+"\ngrails run-app");
    try {
        String commando = "fichiersBAT\\runGrails.bat";
        String[] command = {"cmd.exe", "/C", "Start", commando};
        Runtime.getRuntime().exec(command);
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

NB : la fonction ***ecrireDansFichier*** écrit dans un fichier (1<sup>er</sup> paramètre) la chaîne de caractères du 2<sup>e</sup> paramètre.

L'application permet également de générer le **.WAR** du projet. Le même principe ci-dessus est appliqué, avec la commande Grails ***grails war***.

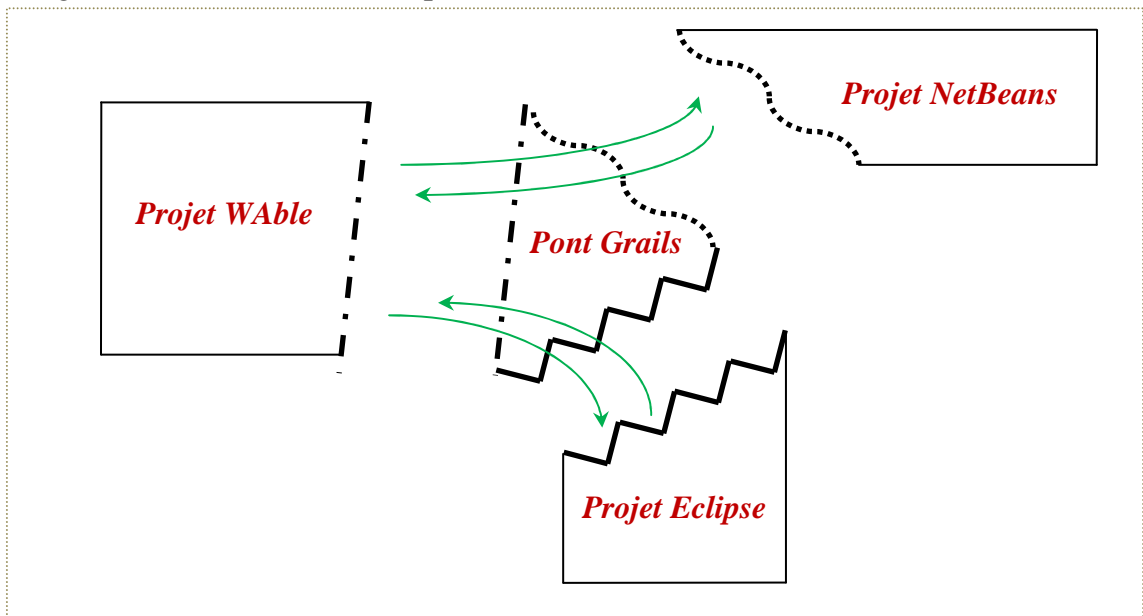
#### IV.1.4.3- Importation et exportation de projet

Tel que nous le verrons dans le chapitre des résultats et commentaires, on peut bien importer un projet *Grails* dans **Wable**. On peut également exporter un projet **Wable** vers *Grails*, vers *NetBeans* ou même vers *Eclipse*.

Ceci se fait grâce à ce que nous appelons le *pont Grails*. **Wable** sait transformer son projet en projet *Grails* (nous l'avons appelé ci-dessus la génération du

projet) et les IDE *NetBeans* et *Eclipse* savent déjà exploiter (importer et exporter) les projets *Grails*.

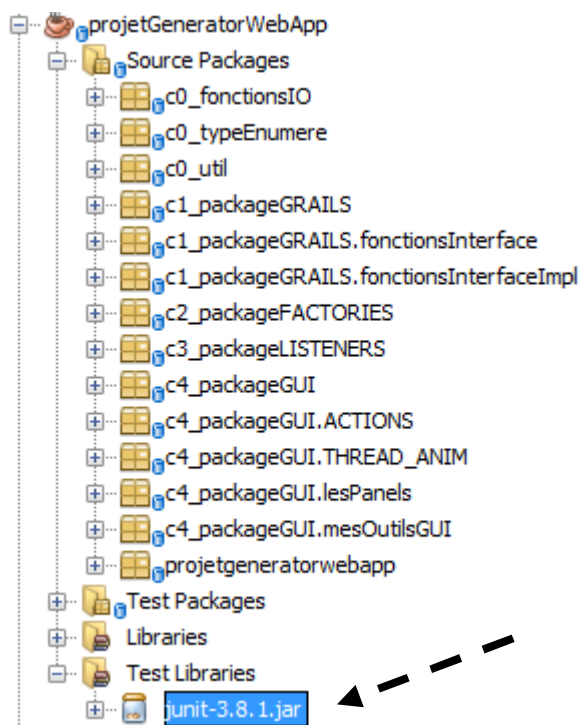
La figure ci-dessous illustre ce pont.



**Figure 20 : Représentation du pont Grails dans l'import-export du projet**

#### IV.1.5- Gestion des tests (JUnit)

Tel qu'on peut le constater dans l'arborescence du projet sur *NetBeans*, la version de *JUnit* utilisée est la **3.8.1**. Il s'agit d'un fichier JAR qui a été placé dans la librairie des tests unitaires.



**Figure 21 : JUnit dans la librairie du projet**



#### IV.1.6- Gestion des versions avec Git en local

Les versions du projet ont été gérées avec *Git*, dans un serveur local de l'entreprise ABLE-SARL. La version de l'outil utilisée est la **1.9.5** comme l'indique l'invite de commandes *Git* ci-dessous :

A screenshot of a Windows command prompt window titled 'MINGW32:/c/Users/ALFRED'. The window has a black background with white text. It displays the following text: 'Welcome to Git (version 1.9.5-preview20141217)', 'Run \'*git help git*\' to display the help index.', 'Run \'*git help* <command>\' to display help for specific commands.', a prompt 'ALFRED@EWANE ~', the command '\$ *git --version*', the output 'git version 1.9.5.msysgit.0', another prompt 'ALFRED@EWANE ~', and a final prompt '\$'.

```
MINGW32:/c/Users/ALFRED
Welcome to Git (version 1.9.5-preview20141217)

Run 'git help git' to display the help index.
Run 'git help <command>' to display help for specific commands.

ALFRED@EWANE ~
$ git --version
git version 1.9.5.msysgit.0

ALFRED@EWANE ~
$
```

**Figure 22 : Invite montrant la version de Git utilisée**

Nous exécutons la commandes *git pull* pour récupérer et fusionner automatiquement la branche du serveur de ABLE avec celle de notre poste de travail. Egalement, la commande *git push* nous permettait de pousser vers le serveur notre branche.

Chaque matin du des jours de travail, nous faisons un *git pull* et en fin de journée, le travail était fusionné à celui du serveur par un *git push*. Ceci nous a permis de faire certains feed-back sans souci.

#### IV.2- Construction du squelette Grails

L'application **Wable** que nous avons développée et présentée jusqu'ici, n'est qu'un utilisateur de **Grails**. Il exploite tout simplement un squelette **Grails** construit au préalable.

Dans cette partie, nous allons présenter les configurations effectuées sur **Grails**. Nous allons également présenter les contrôleurs que nous avons intégrés dans le squelette, présenter la conception de l'interface graphique et enfin les techniques utilisées pour la sécurité dans l'utilisation de l'application web générée.

## IV.2.1- Configurations effectuées sur Grails

### IV.2.1.1- Connexion à une base de données

Dans un projet **Grails**, le dossier *grails-app\conf* contient certains fichiers de configuration. Le fichier ***DataSource.GROOVY*** permet d'effectuer des configurations liées à la BD utilisée.

Dans le bloc **dataSource**, on entre les paramètres tels que le *driver*, le *nom d'utilisateur* et le *mot de passe* pour se connecter à la BD. On spécifie également l'*URL* de la BD.

```
dataSource {
    pooled = true
    driverClassName = "com.mysql.jdbc.Driver"
    username = "able"
    password = "123456789"

    url = "jdbc:mysql://localhost:3306/bdWable?
        autoreconnect=true"
}
```

#### Exemple de configuration du dataSource

Notons que pour que la connexion soit établie avec *mysql*, il a fallu télécharger puis copier le fichier *mysql-connector-java-5.1.14-bin.jar* dans le dossier **lib** du projet **Grails**.

### IV.2.1.2- Le mappeur Hibernate

C'est également dans le fichier ***DataSource.GROOVY*** que l'on précise certains paramètres de *Hibernate*. *Hibernate* est un outil, intégré à **Grails** permettant d'effectuer des mappings. Il fait des correspondances entre les entités et la BD. C'est lui qui se charge de convertir les objets en données réelles dans la BD.

Sa configuration se dans le bloc **hibernate**. Il est important de bien configurer le *cache.region.factory\_class* pour ne pas avoir l'exception *ClassNotFoundException*:  
*org.hibernate.impl.SessionFactoryImpl*.  
Les autres champs peuvent être laissés par défaut tel que **Grails** les a configurés.

```
hibernate {

    cache.use_second_level_cache = true
    cache.use_query_cache = false

    cache.region.factory_class =
    'net.sf.ehcache.hibernate.EhCacheRe
    gionFactory'

}
```

#### Exemple de configuration d'hibernate

Il faut également dans le fichier **BuildConfig.GROOVY** du même dossier, préciser le plugin hibernate utilisé. Cette configuration se fait dans le bloc **plugin**. Dans notre cas, nous avons fait : *runtime ":hibernate:3.6.10.17"*.

#### *IV.2.1.3- Installation du plugin Searchable*

Le plugin **Searchable** est celui qui nous permet d'attribuer des options de recherche aux différents controllers. C'est grâce à lui qu'on peut trouver un objet, seulement en entrant la valeur de l'un de ses champs.

Là aussi, il faut trouver le plugin et l'ajouter dans le fichier **BuildCinfig.GROOVY**. Dans le bloc **plugin**, nous avons ajouté : *compile ":searchable:0.6.9"*.

Il est toujours important, après toute configuration, d'exécuter le projet en ligne. Cela permet généralement de télécharger des fichiers manquants.

### **IV.2.2- Les contrôleurs**

#### *IV.2.2.1- Les contrôleurs générés et le Scaffolding*

Un projet **WABLE** est composé d'**entités**. Dans la construction d'un projet **Grails** à partir d'un projet **WABLE**, chaque **entité** génère une classe « **Domain** » et une classe « **Controller** ».

Pour que les fonctionnalités CRUD (Create, Read, Update, Delete) puissent s'appliquer à un « **Domain** », il faut préciser dans son « **Controller** », l'instruction GROOVY *def scaffold = true*.

On dit qu'on a appliqué le « *scaffolding* » ; et, dans tous les contrôleurs de l'application générée, cela a été appliqué.

Dans ce projet, nous avons utilisé la version **2.1.2** du *scaffolding*.

#### IV.2.2.2- MyOwn Controller

Pour définir certaines pages par défaut, telle que la page d'accueil par exemple, il a fallu créer un contrôleur spécial. Ce contrôleur, nous l'avons appelé **MyOwnController**. Il est directement intégré dans notre squelette web généré.

Son code GROOVY est le suivant :

```
package squelettegrails

class MyOwnController {
    def firstPage = {}
    def secondPage = {}
    def listes = {}
    def enregistre = {}
    def modifie = {}
    def operation = {}
}
```

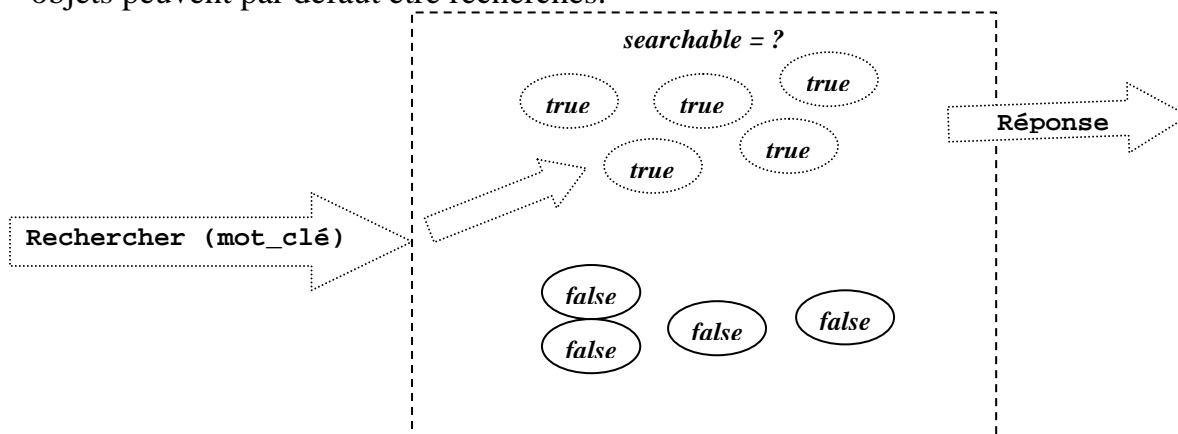
#### IV.2.2.3- UserController

Un autre contrôleur directement intégré au squelette est **UserController**. C'est le contrôleur du domaine **User**. **User** est une classe GROOVY, créée pour gérer la sécurité dans l'application. Le paragraphe IV.2.4, présentant la sécurité de l'application, illustrera davantage cette partie.

#### IV.2.2.4- La recherche

Lorsque le plugin **Searchable** est bien installé, il suffit, pour qu'un domaine soit « *cherchable* », d'ajouter dans son code GROOVY l'instruction :  
**searchable = true.**

Ceci est appliqué dans tous domaines générés par **WABLE**, ce qui fait que tous les objets peuvent par défaut être recherchés.



**Figure 23 : Illustration de la recherche d'un objet dans l'application**

Cette figure illustre bien la recherche d'un objet. La requête est dirigée seulement vers les objets qui ont défini *searchable = true*.

### IV.2.3- Construction de l'interface graphique

Une autre tâche capitale qu'il y eut à faire, fut de fabriquer les interfaces graphiques du squelette généré selon les spécifications de l'IHM du chapitre III. Ces interfaces sont constituées des pages GSP (Groovy Server Pages) que nous présenterons dans la suite. Les pages GSP sont intégrées dans du HTML ; et, pour leur bonne apparence, nous avons utilisés du CSS.

#### IV.2.3.1- Les pages GSP

Les pages GSP sont des pages web qui intègrent du code GROOVY. Ce sont des pages qui peuvent avoir des attitudes statique ou dynamique.

Par exemple, pour fabriquer un lien, on utilise la balise `<g:link>`. Le code ci-dessous est celui du menu des listes.

```
<g:link controller="MyOwn" action="listes">Les listes</g:link>
```

Lorsqu'un utilisateur fait un clic au niveau de la vue, le contrôleur *MyOwn* est interpellé et lance l'action *listes* qui se trouve dans son code.

Un exemple dynamique peut être l'affichage de tous les contrôleurs trouvés dans l'application. Considérons le bout de code suivant :

```
<g:each var="c" in="${grailsApplication.controllerClasses.sort { it.fullName } }">
  <g:if test="${!c.logicalPropertyName.equals('searchable')
    &!c.logicalPropertyName.equals('dbdoc')
    &!c.logicalPropertyName.equals('assets')
    &!c.logicalPropertyName.equals('myOwn')}">

    <g:link controller="${c.logicalPropertyName}" action="create">
      < ${c.logicalPropertyName} >
    </g:link>
  </g:if>
</g:each>
```

Ce code se trouve dans la page de création d'une entité. On parcourt tous les contrôleurs grâce à la balise `<g :each>` et avec `<g :link>`, on crée un lien pour chaque contrôleur vers son action *create*, qui correspond à la création d'un objet du domaine.

Les pages GSP correspondant au CRUD (c'est-à-dire création, lecture, modification et suppression) qui sont automatiquement générés se trouvent dans le répertoire de l'application à l'emplacement `|target|work|plugins|scaffolding-2.1.2|src|templates|scaffolding`.

La page de recherche se trouve dans le fichier `index.GSP` dans le dossier `|target|work|plugins|searchable-0.6.9|grails-app|views|searchable` du projet. Nous y avons apporté des modifications avec du HTML et du CSS pour avoir l'interface de recherche telle que présentée dans le chapitre de résultats et commentaires.

Les contrôleurs à afficher pour le CRUD doivent exclusivement être ceux des entités entrées par le programmeur. Ainsi donc, il faudra empêcher que les autres contrôleurs (ceux par défaut dans l'application) n'apparaissent lors des choix des contrôleurs.

Les contrôleurs qui ne doivent pas s'afficher, dans notre cas, sont quatre :

- ✚ *searchable* : c'est le contrôleur du plugin **searchable-0.6.9** qui permet la recherche d'objets dans l'application ;
- ✚ *dbdoc* : il s'agit là d'un contrôleur GRAILS permettant l'accès à la documentation sur la BD;
- ✚ *assets* : cet autre contrôleur GRAILS se charge de l'accès au contenu du répertoire de travail `grails-app|assets`;
- ✚ *myOwn* : ce contrôleur est celui que nous avons intégré dans le projet et qui a été présenté en IV.2.2.2.

Pour donc empêcher à ces contrôleurs de s'afficher, nous avons utilisé la balise GROOVY `<g :if>`, qui permet de faire des tests. Dans le bout de code ci-dessous, les tests ont été effectués sur le nom des contrôleurs.

```

<g:if test="\${!c.logicalPropertyName.equals("searchable")
    &!c.logicalPropertyName.equals("dbdoc")
    &!c.logicalPropertyName.equals("assets")
    &!c.logicalPropertyName.equals("myOwn")}">

    ...
    whatToDo...
    ...
</g:if>

```

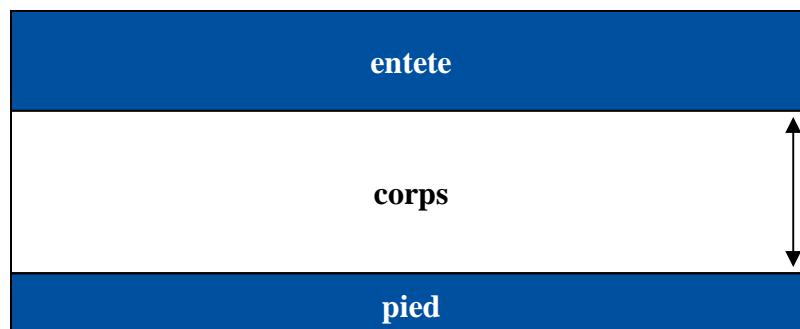
L'instruction *whatToDo* est exécutée si le contrôleur n'est pas l'un des quatre cités plus haut. C'est donc ce que nous faisons pour afficher exclusivement les contrôleurs correspondant aux entités entrées par le programmeur.

#### IV.2.3.2- Le HTML et les codes CSS

La balise *<div>* du HTML permet de créer des blocs. Nous l'avons utilisée dans notre projet pour créer trois blocs dans notre application :

- le bloc du haut, appelé **entete** ;
- celui du milieu appelé **corps** ;
- et enfin celui du bas appelé **pied**.

Ces blocs sont disposés suivant la figure ci-dessous :



**Figure 24 : Disposition des blocs dans l'application web générée**

Leurs codes CSS respectifs sont les suivants :

<pre>#entete {     background: #0050a0;     padding-top: 1em;     position:fixed;     left: 0%;     right:0%; }</pre>	<pre>#corps {     background: #fff;     float: left;     clear: left;     padding-top: 8em;     padding-bottom: 2em; }</pre>	<pre>#pied {     background: #0050a0;     position:fixed;     bottom: 0%;     left: 0%;     right: 0%;     height: 20px; }</pre>
---	--	--

On peut remarquer que l'entête et le pied ont des positions fixes (*position :fixed*). Seul le corps pourra défiler au scroll.

Les balises HTML et les instructions CSS ont de façon générale été les bases de construction du côté statique de toutes nos pages web ; le dynamisme étant assuré par du code GROOVY et JAVA, et légèrement du JavaScript.

#### IV.2.4- Sécurité de l'application

La sécurité des données est une caractéristique qui devra être garantie dans toute application de gestion. Dans notre application web générée, elle se fait via la vérification des comptes d'utilisateurs. Nous présenterons ici le domaine et le contrôleur qui ont été utilisés. Ensuite, nous présenterons comment se fait le filtrage et en fin il s'agira des vues d'authentification.

##### IV.2.4.1- Le domaine User

Un utilisateur est authentifié par un **login** et un **mot de passe**. Nous avons créé une classe domaine nommé *User*, qui regroupe, en plus de ces deux informations, le type de compte, que nous appelons *role*.

Son code GROOVY est le suivant :



```

package squelettegrails
class User {
    String login
    String password
    String role = "user"

    static constraints = {
        login(blank:false, nullable:false, unique:true)
        password(blank:false, password:true)
        role(inList:["admin", "user"])
    }
    boolean isAdmin(){
        return role == "admin"
    }
    String toString(){
        login
    }
    def beforeInsert = {
        password = password.encodeAsABLE()
    }

    def beforeUpdate = {
        password = password.encodeAsABLE()
    }
}

```

Les blocs ***beforeInsert*** et ***beforeUpdate*** définissent les actions à faire juste avant l'insertion et la modification d'entités, respectivement. Dans ce code, ces actions représentent l'encodage du mot de passe, que nous aborderons davantage en *IV.2.4.4*.

#### IV.2.4.2- Le contrôleur *UserController*

Un contrôleur gère les informations entre le domaine précédent et la vue. C'est le contrôleur nommé ***UserController***. C'est ici que se décrivent les algorithmes d'authentification et de connexion.

Il s'agit d'une classe GROOVY dont le code suit :

```

Package squelettegrails

import static org.springframework.http.HttpStatus.*
import grails.transaction.Transactional

@Transactional(readonly = true)
class UserController {
    def beforeInterceptor = [action:this.&auth,
        except:['login', 'logout', 'authenticate']]
    def auth() {
        if(!session.user) {
            redirect(controller:"user", action:"login")
            return false
        }
        if(!session.user.admin){
            flash.message = "Accès limité ! Contactez
l'Administrateur..."
            redirect(controller:"myOwn", action:"firstPage")
            return false
        }
    }

    def login = {}
    def logout = {
        flash.message = "Goodbye ${session.user.login}"
        session.user = null
        redirect(action:"login")
    }
    def authenticate = {
        def user =
        User.findByLoginAndPassword(params.login,
params.password.encodeAsABLE())
        if(user){
            session.user = user
            flash.message = "Hello ${user.login}!"
            redirect(controller:"myOwn", action:"secondPage")
        }else{
            flash.message =
"Désolé, ${params.login}. Veuillez recommencer..."
            redirect(action:"login")
        }
    }

    ...

    def show(User userInstance) {
        respond userInstance
    }

    def create() {
        respond new User(params)
    }

    ...

```

#### IV.2.4.3- Le filtrage

Le filtre est défini dans le répertoire *grails-app\conf\squelettegrails*. Il s'agit d'une classe GROOVY nommée *AdminFilters* qui définit les actions qui peuvent être faites sur des entités, en fonction du type de compte d'utilisation.

Tel que présenté dans le code GROOVY ci-dessous, nous avons dans le bloc *filters* défini un filtre qui stipule que seul un compte de type **Admin** pourra effectuer, sur tout contrôleur, les actions *create/edit/update/delete/save*.

```
package squelettegrails

class AdminFilters {
    def filters = {
        adminOnly(controller:'*',
            action:"(create|edit|update|delete|save)") {
            before = {
                if(!session?.user?.admin){
                    flash.message = "DÃ©solÃ©, vous n'avez pas ce droit !
                                Contactez l'Administrateur..."
                    redirect(controller:"myOwn", action:"firstPage")
                    return false
                }
            }
            ...
        }
    }
}
```

#### IV.2.4.4- Le cryptage

On peut remarquer la fonction *encodeAsABLE()* dans le domaine **User** et également dans son contrôleur **UserController**. Il s'agit d'une fonction qui crypte le mot de passe de l'utilisateur. Ceci permet de masquer les informations d'un utilisateur, même dans la BD.

Le fichier *ABLECodec.GROOVY* du répertoire *\grails-app\utils* définit ce cryptage. Son code est le suivant :

```
class ABLECodec {
    static encode = {
        target-> CodageABLE.coder(target)
    }
}
```

*coder()* est une fonction statique JAVA, de la classe *CodageABLE*.

```

class CodageABLE {
    public static String coder(String pass){

        //instructions de cryptage sur pass

        return pass;
    }
}

```

Ce fichier, *CodageABLE.JAVA*, se trouve également dans le répertoire *\grails-app\utils*.

Seule la connaissance de l'algorithme de ce cryptage peut permettre à un tiers de décrypter une information. C'est donc dans un souci de confidentialité pour l'entreprise ABLE-SARL que les instructions de cryptage, telles que implémentées, ne sont pas présentées ici.

D'ailleurs, chaque programmeur peut l'implémenter à sa guise. Il devra tout simplement respecter la signature *public static String coder (String passwd)*.

#### IV.2.4.5- L'interface de connexion

L'interface de connexion est une construite dans le fichier *login.GSP* du répertoire *\grails-app\views\user*. La maquette de cette page peut être représentée comme dans la figure suivante :

entete

Zone des messages de connexion

Nom de compte :

Mot de passe :

Login

pied

**Figure 25 : Maquette de l'interface de connexion**

La zone de messages de connexion est le lieu où s'affichent les messages du genre « *échec de connexion, veuillez recommencer....* ».

Le contenu du fichier *login.GSP* peut être retrouvé en **Annexes**.

#### **IV.3- Implantation au sein de l'entreprise**

L'implantation de ce générateur n'a pas du tout été une tâche fastidieuse, du moment que cela répondait à un réel problème au sein de l'entreprise. Le besoin de créer des fichiers d'exécution et d'installation ne s'étant pas fait ressentir, l'application a été adoptée comme fonctionnant directement depuis l'IDE NetBeans. Aucun souci à cet effet puisqu'il s'agit là d'un programme pour programmeur !

Seulement, certains peuvent avoir d'handicap dans l'utilisation de l'outil. C'est pourquoi un document appelé **Guide d'exploitation de Wable** a été mis sur pied, pour faciliter l'utilisation du générateur. Ce guide peut être retrouvé en **Annexes**.

### **Conclusion**

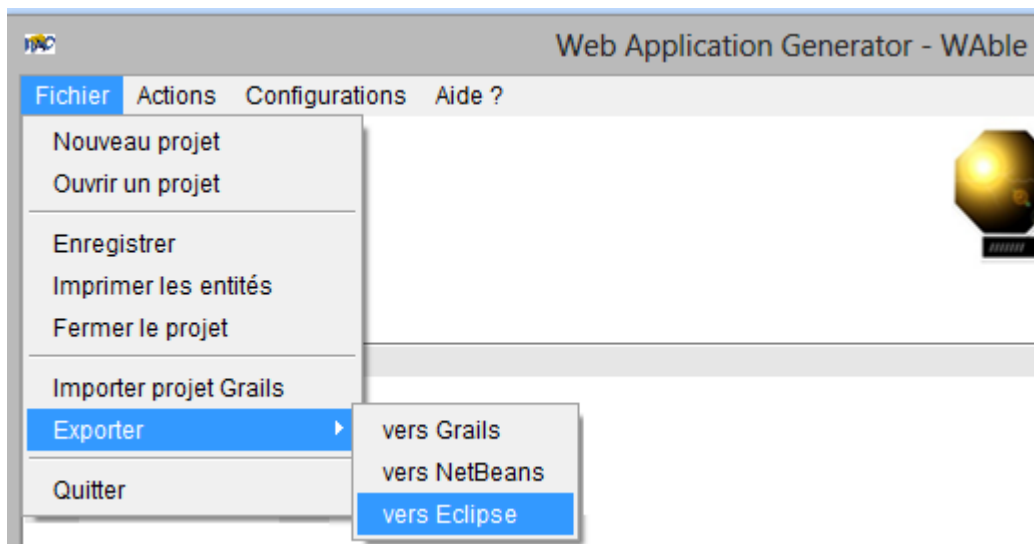
Nous avons présenté dans ce chapitre, l'implémentation de l'application **Wable**. Ce logiciel de génération de squelettes d'application web a été construit principalement suivant l'outil **Grails**. Nous avons présenté les algorithmes écrits et les configurations effectuées sur des outils afin de mettre sur pied ce logiciel. Nous avons également décrit le processus d'exploitation de ce générateur, utile au sein de l'atelier de génie logiciel de l'entreprise.

## CHAPITRE V : RESULTATS ET COMMENTAIRES

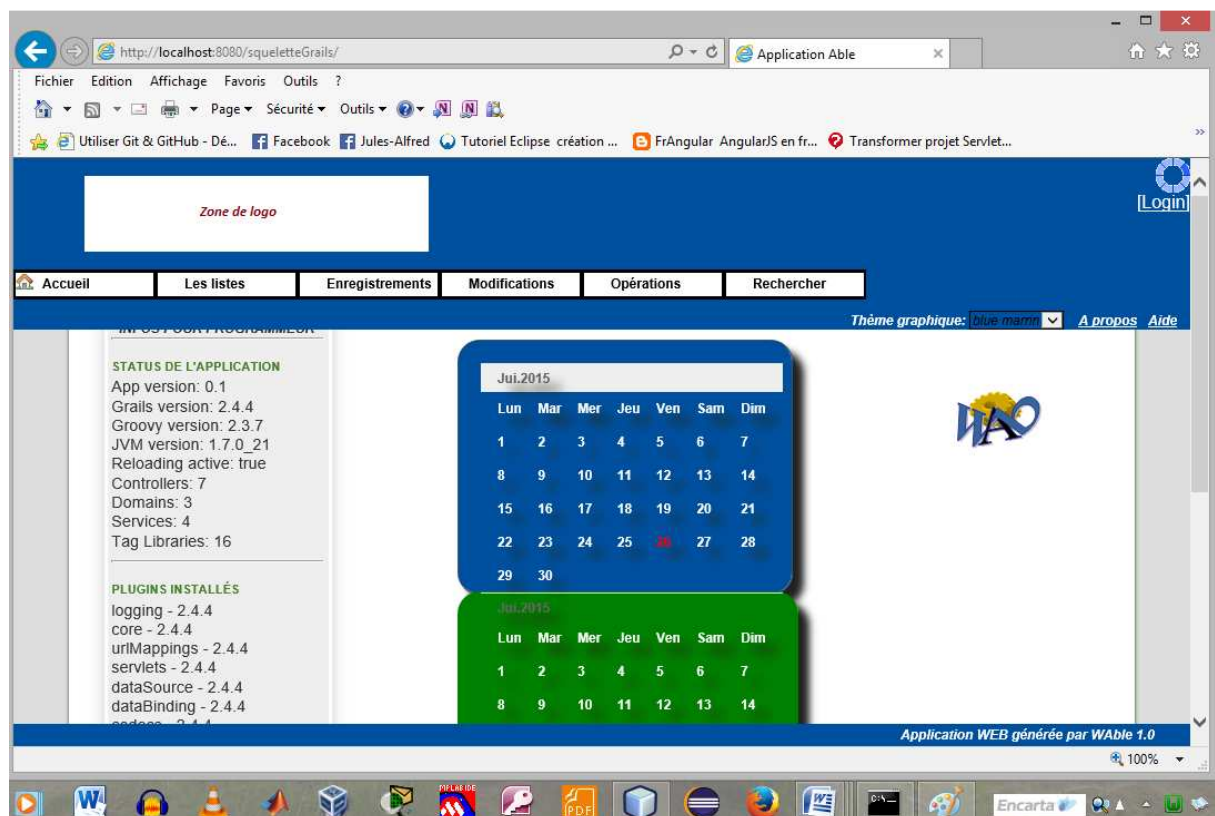
Les résultats du travail effectué seront dans ce chapitre présentés sous forme de capture d'écran des interfaces des applications conçues.



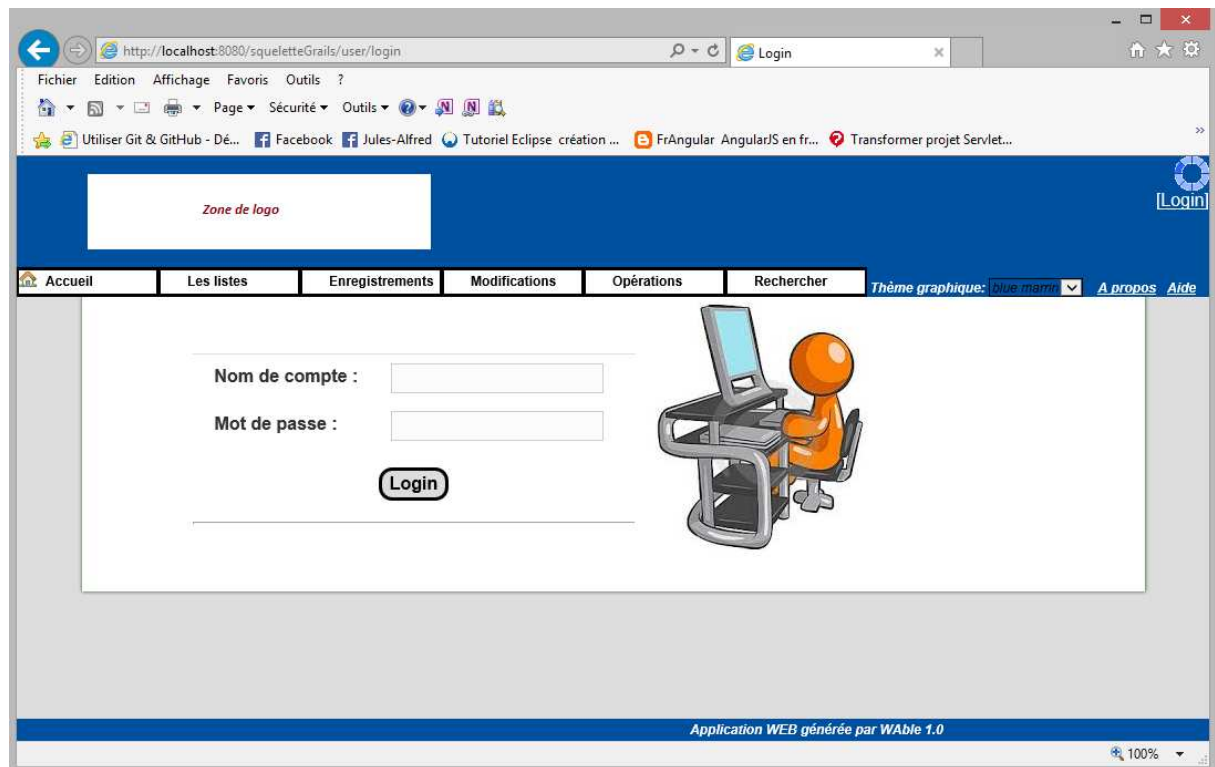
**Figure 26 : Première interface de Wable**



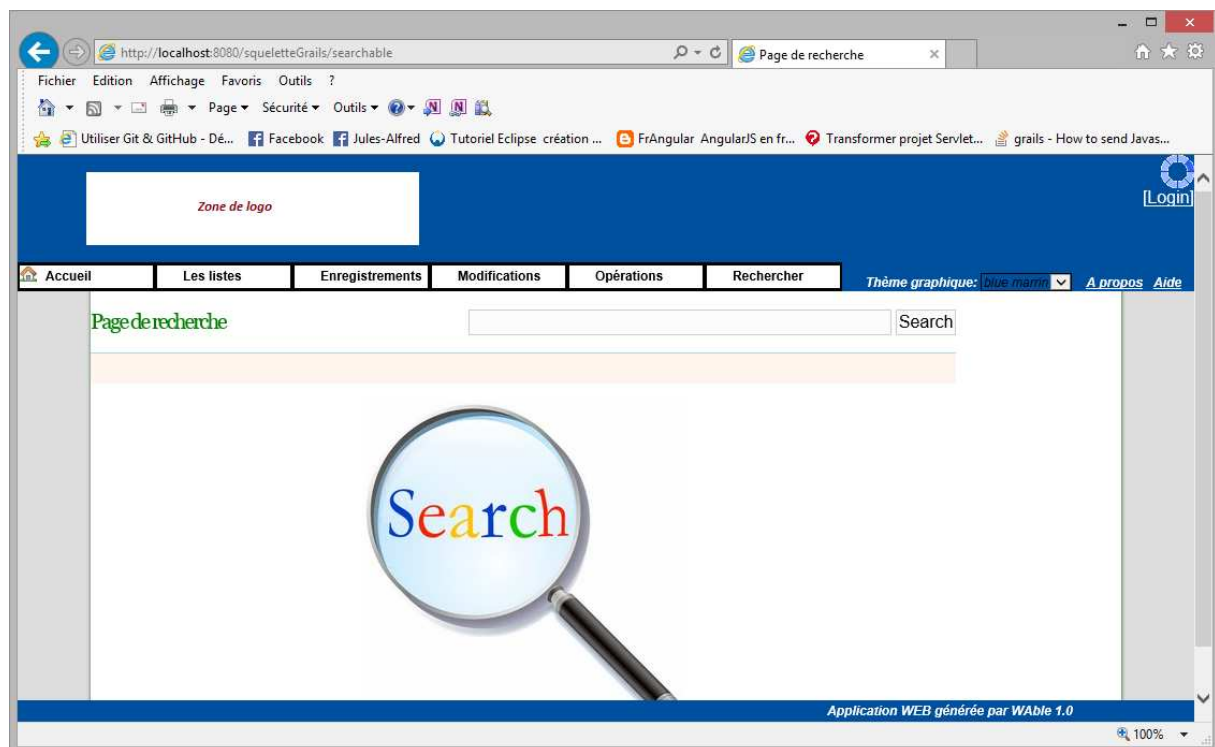
***Figure 27 : Importation ou exportation d'un projet sur Wable***



***Figure 28 : Interface d'accueil de l'application web générée par Wable***



**Figure 29 : Interface de connexion de l'application web générée**



**Figure 30 : Interface de recherche d'un objet dans l'application générée**



Il est tout de même important de préciser que, suivant le cahier des charges, l'exigence **E02** (Gestion des entités) n'a pas été complètement respectée. On devrait entrer les entités de trois façons - manuellement – à partir des classes d'entité JAVA – à partir d'un script de base de données.

Mais l'entrée des entités à partir d'un script de base de données reste encore en cours de conception...

Ce qui nous donne donc, sur 08 exigences, 07 complètement respectées ; soit un résultat de **87,5%**.

## CONCLUSION GENERALE ET PERSPECTIVES

Ce travail a consisté à accélérer la construction des applications au sein de l'entreprise ABLE-SARL. La célérité a été établie au niveau de l'implémentation grâce à l'utilisation de certains outils.

Nous avons décrit le contexte nous ayant inspiré un tel projet comme étant celui selon lequel le client qui passe une commande ne sait généralement pas attendre le temps d'un développement. Alors que dans le processus d'implémentation, plusieurs tâches similaires sont effectuées à chaque construction, nous avons pensé mettre sur pied un générateur, une sorte de « méta application ». Les outils utilisés pour ce travail ont été présenté dans le chapitre des généralités. Après l'analyse et la conception du système de génération, nous nous sommes lancés dans la configuration des outils et l'écriture de certains algorithmes. Les résultats de ce travail sont estimés à **87,5%**, ce qui montre que nous avons encore du travail à faire.

En perspectives donc, nous atteindrons d'abord les 100%, et ensuite, nous pensons multiplier les importations et les exportations des projets du générateur afin qu'il soit compatible dans plusieurs environnements de développement, voire dans d'autres systèmes d'exploitation.

# ANNEXES

## Annexe1 : Guide d'exploitation de Wable

Le guide d'exploitation de Wable est une petite brochure de 14 pages que nous avons conçue au sein de l'entreprise, permettant d'orienter le programmeur dans l'utilisation du générateur. Les pages suivantes présentent ce document, dans une forme réduite...



# Wable soft



# Guide d'exploitation

Version 1.0

*par*  
***L'Atelier de Génie Logiciel***

*Juin 2015*

## ***Introduction***

Wable est un soft conçu pour la génération des squelettes d'applications web, appartenant à l'entreprise ABLE-SARL. Son principal utilisateur est le programmeur. Ce dernier se sert de cet outil pour générer, avec une rapidité remarquable, une application web, de type Java2EE, à partir des résultats de l'analyse et de la conception du système à concevoir.

L'application web générée intègre automatiquement les fonctionnalités CRUD (Create-Read-Update-Delete). Elle intègre également la gestion de la sécurité des données via les comptes d'utilisateurs. Aussi, la recherche d'une entité, à partir de la valeur d'un de ses attributs y est directement possible.

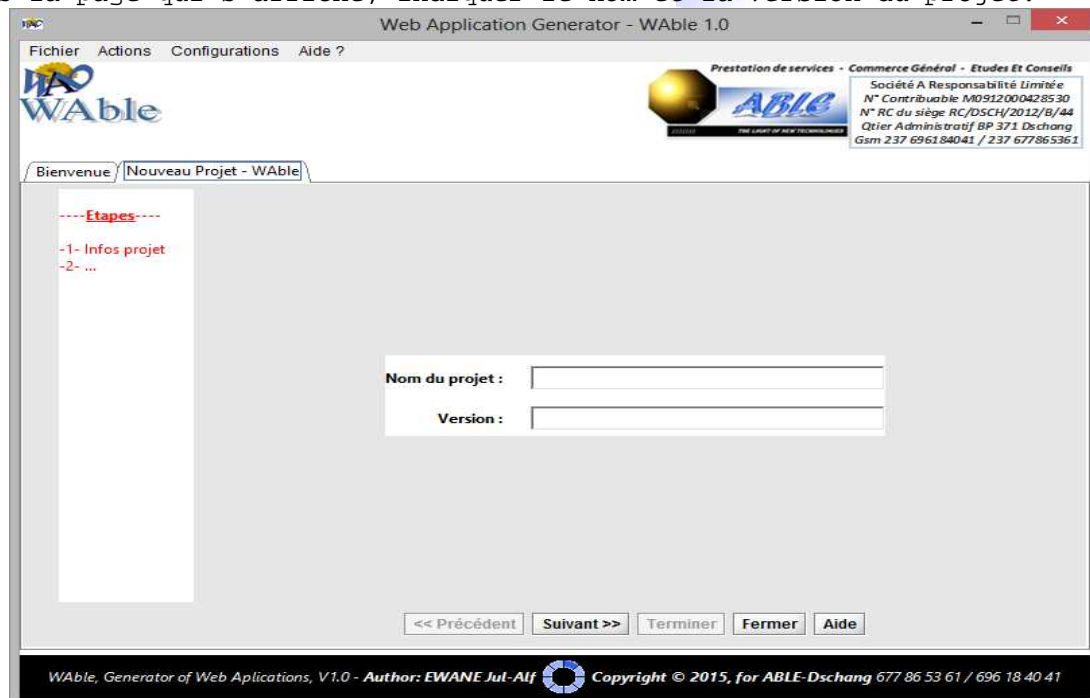
Le programmeur n'a qu'à refaire de légères retouches et/ou à ajouter de nouvelles fonctionnalités, spécifiques au système en construction.

Le présent document va présenter les différentes procédures permettant de partir de l'ouverture de Wable à la génération d'un squelette d'application web.

### ***Création - Ouverture d'un projet***

La création d'un projet Wable se fait dans le menu **fichier** → **Nouveau projet**.

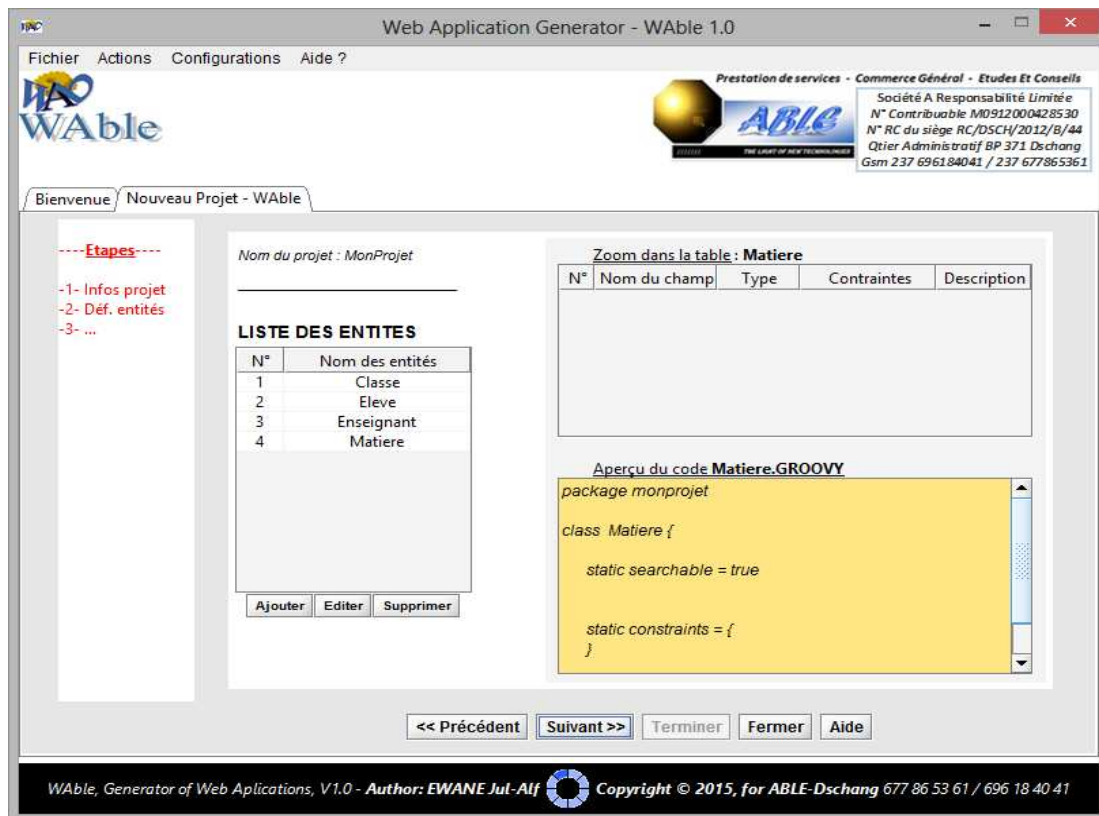
Dans la page qui s'affiche, indiquer le nom et la version du projet.



Cliquer sur le bouton **Suivant>>** pour accéder à l'interface d'entrée des entités.

Notons qu'il faut absolument entrer un *Nom de projet* pour accéder à la page suivante !

La prochaine interface est la suivante :



Cliquer sur **Ajouter** pour ajouter une entité. Le nom entré n'est pas validé s'il commence par un chiffre. Tous les espaces (aux extrémités et dans les intervalles) seront supprimés, de façon à ne former qu'un seul bloc. Aussi, la première lettre du nom de l'entité est automatiquement mise en majuscule.

On pourra sélectionner une entité dans la liste et cliquer sur **Supprimer** si l'on désire l'effacer.

A chaque fois que l'on sélectionne une entité, on peut consulter son code GROOVY dans le tableau nommé Aperçu du code ; on peut également apercevoir ses attributs dans le tableau appelé Zoom dans la table.

Pour ajouter un attribut à une entité, sélectionner l'entité et cliquer sur **Editer** pour voir apparaître à droite, en bas, une interface permettant de définir un attribut.

Dans cette interface, entrer le *Nom de l'attribut* et sa *description*. Sélectionner ensuite son *type*, puis définir les contraintes en cliquant sur la barre devant *Contraintes*.

# ABLE-SARL Atelier de Genie Logiciel

Une fois les entités définies, leurs attributs aussi, il faut indiquer les différentes relations existantes. Dans la prochaine page, choisir deux entités et indiquer le type de relation entre elles.

Web Application Generator - Wable 1.0

Fichier Actions Configurations Aide ?

Bienvenue Nouveau Projet - Wable

---Etapes---

- 1- Infos projet
- 2- Déf. entités
- 3- Déf. relations
- 4- ...

Nom du projet : MonProjet

CHOISIR L'ENTITE 1

Classe

CHOISIR L'ENTITE 2

Eleve

**Définition d'une relation**

Entité 1 : Classe

Entité 2 : Eleve

Type de relation

oneToMany

☒ Bidirectionnelle

Valider la relation

N°	Entité 1	Entité 2	Type Relation
1	Enseignant	Classe	manyToMany (bidirec...

Supprimer la relation

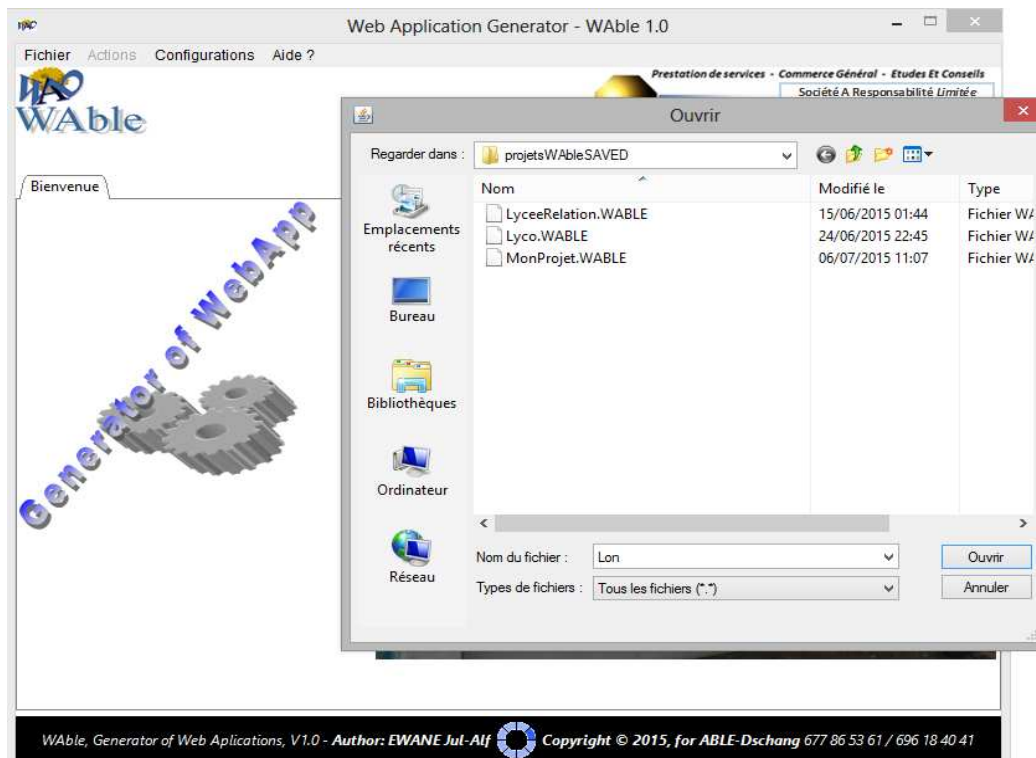
<< Précédent Suivant >> Terminer Fermer Aide

Wable, Generator of Web Applications, V1.0 - Author: EWANE Jul-Alf Copyright © 2015, for ABLE-Dschang 677 86 53 61 / 696 18 40 41

Le projet Wable conçu jusqu'à ce niveau peut à tout moment être sauvegardé. Pour ce faire, cliquer **Fichier** → **Enregistrer**. Et le fichier sera enregistré avec le nom donné au projet, sous l'extension **.WABLE** et dans le dossier **projetsWableSAVED**.

Ce fichier créé renferme toutes les informations sur un projet. Ces informations peuvent à tout moment être restituées lors de l'ouverture d'un projet.

Cliquer sur **Fichier** → **Ouvrir un projet** et sélectionner, dans la fenêtre de dialogue qui s'affiche, le fichier à restaurer dans le dossier de sauvegarde mentionné plus haut.



Une fois le fichier choisi, les données (entités-attributs-relations) sont restaurées dans le projet.

## Construction-Exécution du projet

La quatrième page permet de construire le projet, en cliquant sur le bouton **Construire le projet**. Il s'agit en principe de la génération d'un squelette d'application web à partir des informations indiquées précédemment. Ce squelette est généré sous forme de projet GRAILS, dans le dossier **projetsGrailsCREATED**.



Cette page permet également d'exécuter directement le projet.



## ***Le mode Expert***

Le menu **Configuration** → **Mode expert** ouvre l'interface de configurations avancées. Ici, le programmeur peut effectuer des opérations délicates en accédant à des fichiers sensibles.

### ***- Ajout d'une entité JAVA***

Les classes de domaines peuvent également être extraites d'un fichier JAVA. Ce fichier devra être celui décrivant une entité avec ses attributs et ses accesseurs en lecture et en écriture.

Dans Wable, le programmeur est simplement renvoyé dans un `DialogFile` pour le choix du fichier JAVA.

### ***- Mapping avec Hibernate***

Le mappage permet de faire correspondre deux objets de même nature mais de forme différente. *Hibernate* effectue cette correspondance pour que **Grails** utilise puisse exploiter l'entité JAVA.

Wable ouvre pour cela le fichier **hibernate.cfg.xml** dans lequel le programmeur ajoutera chaque fichier JAVA à mapper. Ce mapping se fait dans le bloc **<session-factory>** et dans la balise **<mapping>**.

Exemple : `<mapping class='com.exemple.MaClasse'/>`

### ***- Configuration du fichier de BD***

Wable ouvre le fichier **DataSource.GROOVY**, dans lequel le programmeur effectue des configurations liées au SGBD. Dans le bloc **dataSource**, on indique la base de données, le driver du SGBD utilisé, l'utilisateur et son mot de passe pour l'accès à la BD.

Exemple de configuration :

```
dataSource {
    pooled = true
    driverClassName = "com.mysql.jdbc.Driver"
    username = "ewane"
    password = "password"
    url="jdbc:mysql://localhost:3306/bdExemple?autoReconnect=true"
}
```

On peut également, dans ce fichier, configurer certains paramètres d'*Hibernate*, ceci dans le bloc **hibernate**.

Exemple de configuration :

```
hibernate {
    cache.use_second_level_cache = true
    cache.use_query_cache = false
    cache.region.factory_class='net.sf.ehcache.hibernate.EhCacheRegionFactory'
}
```

### ***- Configuration des paramètres de démarrage***

Pour la configuration des paramètres de démarrage, le fichier **BootStrap.GROOVY** est ouvert, permettant ainsi, par exemple, la définition d'un utilisateur initial.

### ***- Configuration des paramètres de construction***

Les paramètres de construction se configurent dans le fichier **BuildConfig.GROOVY**. Ce fichier permet, dans le bloc **dependencies**, de définir certaines dépendances. Il permet également d'indiquer les plugins qui seront utilisés lors de la construction (**build**) du projet, lors de sa compilation (**compile**) ou lors de son exécution (**runtime**).

Exemple :

```
dependencies {
```



```
test "org.grails:grails-datastore-test-support:1.0.2-grails-2.4"
}

plugins {
    build ":tomcat:7.0.55"

    // plugins for the compile step
    compile ":scaffolding:2.1.2"
    compile ':cache:1.1.8'
    compile ":asset-pipeline:1.9.9"
    compile ":searchable:0.6.9"

    // plugins needed at runtime but not for compilation
    runtime ":hibernate:3.6.10.17"
    runtime ":database-migration:1.4.0"
    runtime ":jquery:1.11.1"
}
```

## ***- Edition manuelle des codes***

Dans le mode avancé, le programmeur peut également avoir accès aux différents codes des domaines et des contrôleurs (*codes GROOVY*), également aux codes des vues (*GSP* ou *codes HTML*). Le programmeur pourra donc éditer manuellement les codes affichés.

## ***Configuration de l'application générée***

### **==> Modification du logo**

La première modification à effectuer sur le squelette généré peut être le changement du logo par défaut par celui du système que l'on conçoit. Les images de l'application se trouvent dans le dossier **grails-app/assets/images**. L'image du logo est bien **logi.PNG**, qu'il suffit de remplacer.

### **==> Modification du fond**

La modification de la couleur de fond nécessite, pour la présente version du générateur, une connaissance moyenne en CSS. Les codes CSS du projet se trouvent dans l'emplacement **grails-app/assets/stylesheets**.

Les fichiers à ouvrir et à modifier sont ceux dont les noms ont la forme **styles[\*]Able.CSS**. Les blocs du haut, du milieu et du bas sont des **div** dont les **id** sont **entete**, **corps** et **pied**, respectivement.

### **==> Représentation des données dans l'affichage**

Grails a son propre format de représentation des données. Par exemple, **Voiture#2** représente une instance de la classe **Voiture** ; c'est celui dont l'ID est 2.

Mais, le programmeur peut choisir de constituer des formats, propres à chaque entité. Pour cela, il devra surcharger la fonction **toString** dans chaque classe de domaine.

Exemple :

```
String toString(){
    return "${attribut1}, ${attribut2}"
}
```

### **==> Ordre d'affichage des attributs d'une entité**

Le programmeur peut vouloir, lors de l'affichage d'une entité, classer les attributs dans un ordre désiré (par défaut, l'ordre est l'ordre lexicographique). Pour cela, il devra classer les attributs dans le bloc **static constraints** tel qu'il voudrait les voir classer au niveau de la vue.

```
static constraints = {
    name(blank:false, maxSize:50)
    startDate()
    city()
    state(inList:["GA", "NC", "SC", "VA"])
    distance(min:0.0)
```

# ***ABLE-SARL Atelier de Genie Logiciel***

```
cost(min:0.0, max:100.0)
maxRunners(min:0, max:100000)
}
```

## **==> Affichage des listes des données**

Les instances d'une entité peuvent être affichées suivant un tri, mais également suivant une répartition en blocs.

Le tri dans une liste de données est représenté par la syntaxe **sort=attribut**. Le mot clé **sort** indique le tri et **attribut** est l'attribut dont il faut trier les valeurs.

Le programmeur peut également décider d'afficher les données par blocs. Pour cela, il utilise la syntaxe **max=nbre**, où **max** est un mot clé et **nbre**, le nombre maximal de données par bloc.

Par exemple,

**<http://localhost:8080/squeletteGrails/user/index?max=5&sort=login>**

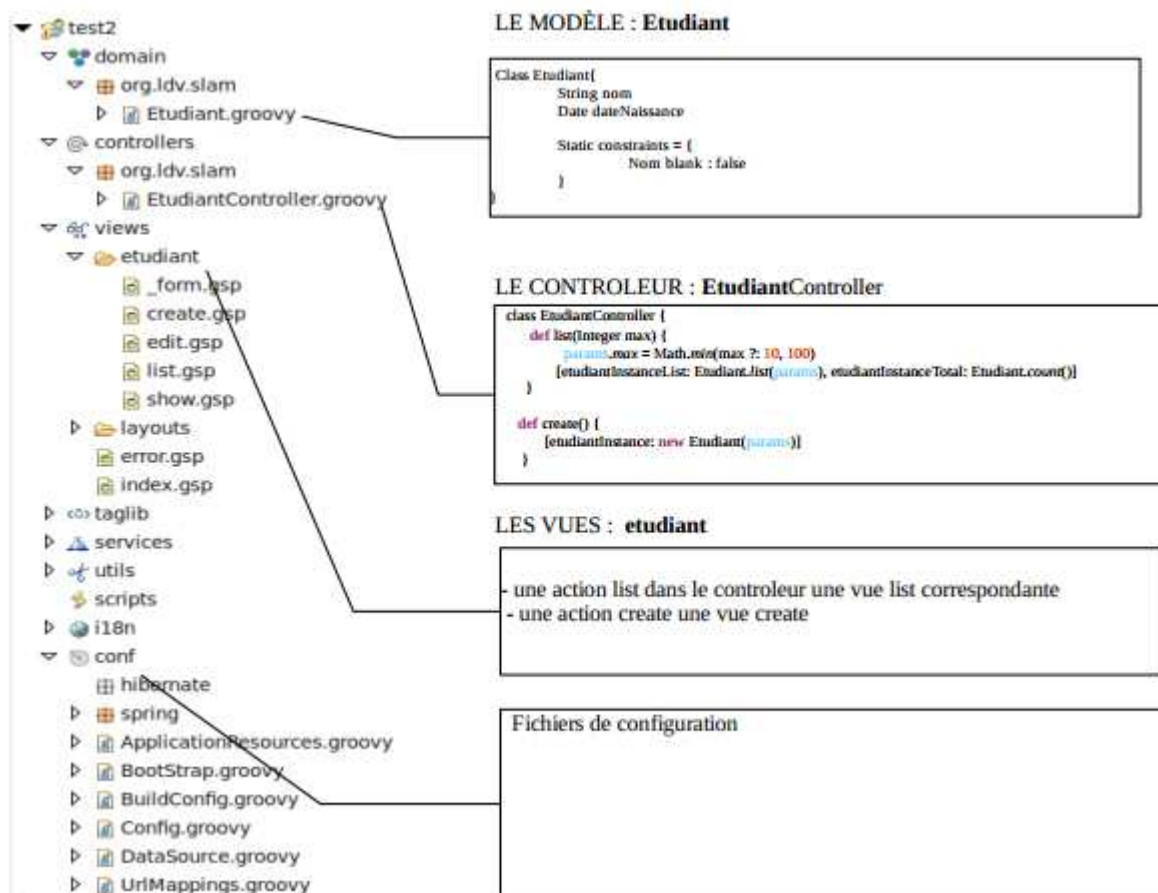
affiche la liste des utilisateurs triée dans l'ordre alphabétique suivant le **login** et par bloc de **cinq**.

## ***Conclusion et perspectives***

Ce support a consisté à présenter l'utilisation du générateur WAbLe. WAbLe est un soft permettant de générer des squelettes d'application web à partir des entités et relations entrées par le programmeur. C'est donc un outil pour le programmeur, lui permettant d'implémenter rapidement des applications web, lorsque la modélisation est déjà faite.

Dans les prochaines versions de ce logiciel, une base de données pourra être directement exploitée à partir de son script. Egalement, l'import-export se fera dans des environnements de travail en plus.

## Annexe 2 : Arborescence d'un projet Grails dans l'IDE *Grails-Tool-Suite*



## Annexe 3 : Gestion simplifiée des collections avec Groovy

### Liste en Java :

```
List<String> lesPays = new ArrayList<String> ();
lesPays.add("France");
lesPays.add("Allemagne");
lesPays.add("Italie");
lesPays.add("Belgique");
```

### Liste en GROOVY :

```
def lesPays = ['France','Allemagne','Italie','Belgique']
```

### Parcours de la liste en JAVA :

```
for(String unPays : lesPays) {
    System.out.println(unPays);
}
```

### Parcours de la liste en GROOVY :

```
lesPays.each {
    println it
}
```

### Map en JAVA :

```
Map<String, String> contacts = new HashMap<String, String>();
contacts.put("durand", "06.12.13.14.16");
contacts.put("dupont", "06.42.32.14.16");
contacts.put("duret", "06.45.83.14.16");
```

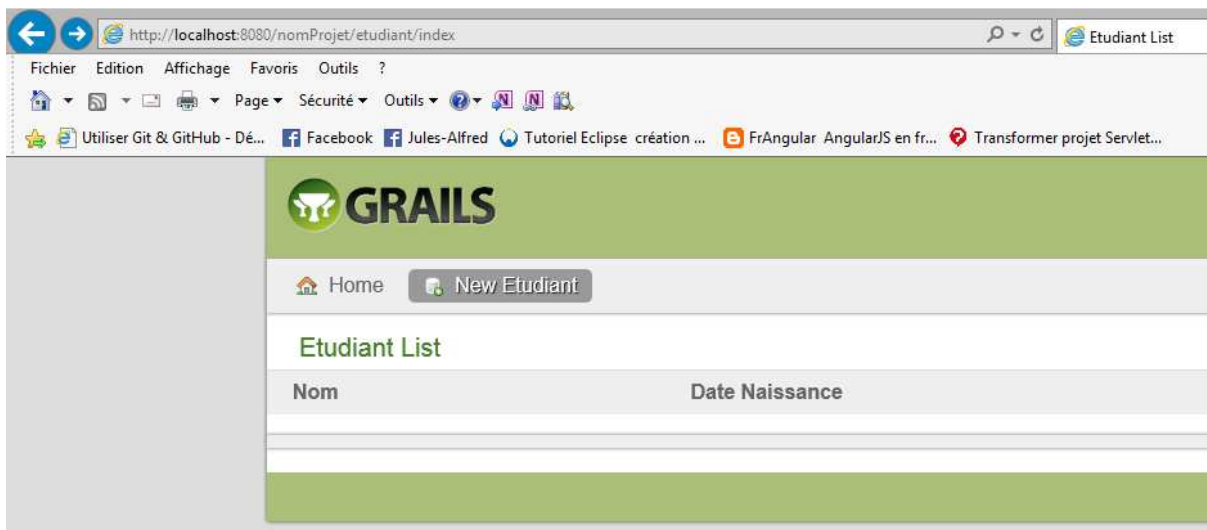
### Afficher le couple clé-valeur

```
for (Map.Entry<String, String> entry : contacts.entrySet()) {
    System.out.println(entry.getKey() + " : " + entry.getValue());
}
```

### Map en GROOVY :

```
contacts = [ "durand":"06.12.13.14.16", "dupont":"06.42.32.14.16", "duret":"06.45.83.14.16" ]
contacts.each() { key, value -> println "${key} == ${value}" }
```

## Annexe 4 : Interfaces par défaut de Grails



*Interface Liste des entités*

The screenshot shows the 'Create Etudiant' form in the Grails application. The header and navigation bar are the same as in the previous screenshot. The page title is 'Create Etudiant'. The form contains two fields: 'Nom \*' with a text input field, and 'Date Naissance \*' with three dropdown menus for day, month, and year. The day is set to '6', the month to 'juin', and the year to '2015'. A 'Create' button is located at the bottom of the form.

*Interface de création d'entité*

The screenshot shows the 'Show Etudiant' page in the Grails application. The header and navigation bar are the same as in the previous screenshots. The page title is 'Show Etudiant'. A message box at the top indicates 'Etudiant 1 created'. Below the message, the details of the student are displayed: 'Nom EWANE Jules-Alfred' and 'Date Naissance 1984-12-21 00:00:00 CET'. At the bottom, there are two buttons: 'Edit' and 'Delete'.

*Interface d'affichage, d'édition et de création*

## Annexe 5 : Code de chargement d'un projet WABLE

```
FileDialog fdg = new FileDialog(MainFrame.fenetre, "Ouvrir");
fdg.setDirectory("projetsWableSAVED");
fdg.setVisible(true);

String rep = fdg.getDirectory();
String fichier = fdg.getFile();
if (fichier == null) {
    return;
}
ProjetGRAILS projetG = null;
try {
    FileInputStream fos = new FileInputStream(rep + "/" + fichier);
    ObjectInputStream objIO = new ObjectInputStream(fos);
    projetG = (ProjetGRAILS) objIO.readObject();
    //objIO.close();
} catch (IOException e) {
    System.out.println("erreur de persistance");
} catch (ClassNotFoundException e) {
    System.out.println("erreur de persistance q");
}
if (projetG == null) {
    System.out.println("je suis user nul projet");
}
```

## Annexe 6 : Code du fichier login.GSP

```
<!DOCTYPE html>
<html>
  <head>
    <meta http-equiv="Content-Type"
      content="text/html; charset=UTF-8" />
    <meta name="layout" content="main" />
    <title>Login</title>
  </head>
  <body>
    <div id="myLogin">
      <g:if test="${flash.message}">
        <div class="message">${flash.message}</div>
      </g:if>
      <g:form action="authenticate" method="post" >

        <center>
          <div>
            <table>
              <tbody>
                <tr class="prop">
                  <td valign="top" class="name">
                    <label id="label">Nom de compte :</label>
                  </td>
                  <td valign="top">
                    <input type="text"
                      name="login" />
                  </td>
                </tr>
                <tr class="prop">
                  <td valign="top" class="name">
                    <label id="label">Mot de passe :</label>
                  </td>
                  <td valign="top">
                    <input type="password"
                      name="password" />
                  </td>
                </tr>
              </tbody>
            </table>
          </div>
          <div>
            <input id="bouton" type="submit" value="Login" />
          </div>
          <br>
          <hr>
        </g:form>
      </div>
      <div id="userImg">
        <asset:image src="utilisateur.png" alt="utilisateur" />
      </div>
    </body>
  </html>
```

## BIBLIOGRAPHIE ET WEBOGRAPHIE

- [ 1 ] A.Budd C. Mall et S. Collison, *Maîtrise des CSS*, PEARSON, 75010 Paris, 2<sup>e</sup> édition, 2010.
- [ 2 ] E. Reboisson, *Introduction au langage de script Groovy*, Developpez.com, 2006.
- [ 3 ] Erich Gamna et al, *Design patterns: elements of reusable object oriented software*, AddisonWesley, 1994.
- [ 4 ] J. Cheynet, *JUnit - Le framework Java open source pour la réalisation de tests unitaires - Dossier réalisé en 3<sup>ème</sup> année*, Université de Marne-la-vallée, Ecole d'Ingénieurs, filière informatique et réseau, 2000.
- [ 5 ] Kolyang, *Introduction au Génie Logiciel*, Editions et Média, 2006
- [ 6 ] P. Roques, *UML2 Modéliser une application web*, EYROLLES, 75240 Paris, 4<sup>e</sup> édition, 2004.
- [ 7 ] P. Roques, *UML 2 par la pratique*, EYROLLES, 6<sup>e</sup> édition, 2008.
- [ 8 ] R. Goetter, *CSS2 Pratique du design web*, EYROLLES, 75240 Paris, 2<sup>e</sup> édition, 2007.
- [ 9 ] S. Davis et J. Rudolph, *Getting started with Grails*, C4Media, 2<sup>e</sup> édition, 2010.
- [ 10 ] V. Douwe, *Cours de génie logiciel 3*, Maroua : Institut Supérieure du Sahel, 2014.
- [ 11 ] P. Ntchamba, *Développement de la base de données et la couche d'accès aux données du module archivage de la plateforme Koosserydesk*, Mémoire de fin de formation d'Ingénieur de Conception, Maroua : Institut Supérieure du Sahel, 2014.
- [ 12 ] A. cherti, *Jargon Informatique*, dictionnaire numérique, version 1.3.6, 2006.
- [ 13 ] S. Chacon\*, *Pro Git*, 2011[En ligne].  
Available: <http://tinyurl.com/amazonprogit>. [Accès le 30 décembre 2014, 15 :32].
- [ 14 ] S. Ambler, *The Best Practices of Agile Modeling*, [En ligne].  
Available: <http://www.agilemodeling.com/>. [Accès le 24 novembre 2014].
- [ 15 ] <http://labs.viaxoft.com/introduction-au-framework-grails/>[Accès le 05 mai 2015, 11 :59].
- [ 16 ] <http://labs.viaxoft.com/grails-partie-2-creation-dune-application-basique/>  
[Accès le 05 mai 2015, 11 :59].



- [ 17 ] <http://labs.viaxoft.com/grails-partie-3-validation/> [Accès le 05 mai 2015, 11 :59].
- [ 18 ] <http://labs.viaxoft.com/grails-partie-4-les-relations/> [Accès le 05 mai 2015, 11 :59].
- [ 19 ] <https://grails.org/plugin/searchable> [Accès le 13 mai 2015, 11 :53].
- [ 20 ] <https://blog.groupe-sii.com/presentation-de-grails/> [Accès le 05 mai 2015, 12 :01].
- [ 21 ] <http://mrhaki.blogspot.com/2009/11/create-grails-project-in-netbeans.html>  
[Accès le 05 mai 2015, 11 :59].
- [ 22 ] <https://spring.io/blog/2010/08/26/reuse-your-hibernate-jpa-domain-model-with-grails>  
[Accès le 22 mai 2015, 00 :03].