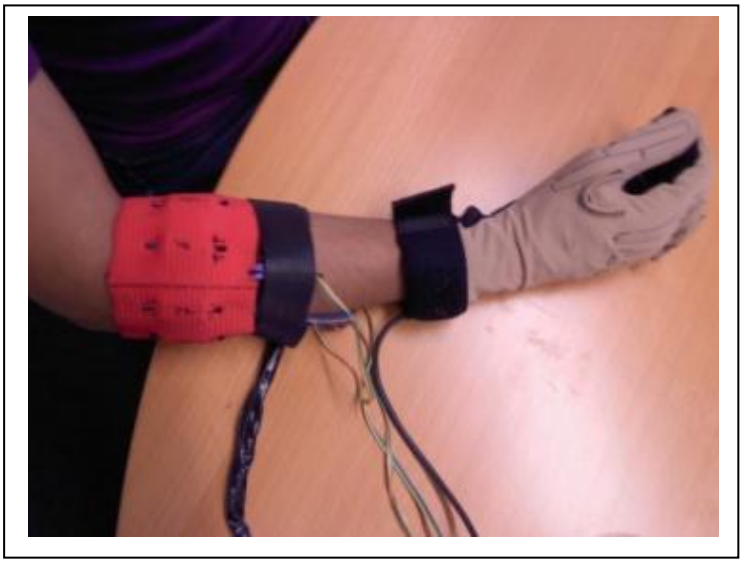
Hes·so VALAIS WALLIS
Haute Ecole Spécialisée de Suisse occidentale
Fachhochschule Westschweiz
University of Applied Sciences Western Switzerland

Travail de Bachelor 2011

Filière Informatique de gestion

« Data acquisitions for electromyograms »

Travail de Bachelor 2011



Etudiant : François Chabbey

Professeur : Henning Müller

www.hevs.ch

Résumé

Ce travail de bachelor est en relation étroite avec le projet de recherche NinaPro (Non-Invasive Adaptive Hand Prosthesis), un projet mené en collaboration avec le centre de recherche IDIAP (Institut d'Intelligence Artificielle Perceptive). Le but de ce projet est d'aider les personnes amputées de la main à améliorer l'apprentissage de l'utilisation de leur prothèse et à acquérir une plus large palette de mouvements. Ce projet se base sur une large récolte de données effectuée sur un ensemble de différents sujets. Les données récoltées consistent d'une part en des signaux musculaires émis par les sujets lors de l'exécution de certains mouvements et d'autre part en des données cliniques concernant les sujets (taille, âge, poids…) et les conditions d'acquisition des données (lieu, date, température de la pièce).

Le but de ce travail de bachelor est de créer une base de données susceptible d'accueillir ces données. Cette base de données devra être accessible à divers intervenants via une interface web, afin de garantir un accès le plus large possible. Naturellement cette application web ne devra être accessible qu'aux personnes autorisées. Un système de droits devra donc être mis en place, ainsi que les outils permettant de l'administrer.

Des outils permettant l'extraction et l'analyse des données devront être fournis aux utilisateurs ; un exemple d'outil d'analyse serait la création de graphiques de distribution pour une variable donnée.

Cette application devra être réalisée en utilisant des techniques de type Ajax, afin de proposer une interactivité semblable à celle que proposent des applications natives. L'interface devra, dans la mesure du possible, rester simple et agréable à utiliser.

Le rapport qui suit retrace les diverses facettes du développement de cette application et propose une synthèse du travail accompli.

Table des matières

1 Introduction

Ce travail de bachelor se situe dans le domaine de l'informatique médicale ; il s'inscrit dans le cadre du projet de recherche NinaPro (Non Invasive Adaptive Hand Prosthesis), mené en étroite collaboration avec le centre de recherche IDIAP (Institut d'Intelligence Artificielle Perceptive). L'objectif de ce travail est la création d'une structure de données permettant de recueillir diverses données collectées[1] tout au long du projet NinaPro. Cette structure de données devra pouvoir être accessible à différents chercheurs, travaillant dans divers lieux. Dans cette optique, une interface web servira à lire et modifier cette structure de données ; ces données étant confidentielles, cette interface devra posséder un système de gestion de droits permettant d'administrer l'accès aux données. Pour mieux cerner le contexte général de ce travail de bachelor, nous commencerons par tracer un résumé général du projet NinaPro

1.1 Contexte du projet

Une personne amputée voit sa qualité de vie grandement amoindrie ; l'usage d'une prothèse permet de retrouver un certain confort de vie, qui reste cependant bien moindre à celui qu'elle possédait avant l'amputation. En effet, « l'état de l'art […] dans le domaine des prothèses non-invasives, n'offre qu'un contrôle grossier de la force, et ne permet que peu de liberté dans les mouvements pouvant être effectués. [8,page 3]»

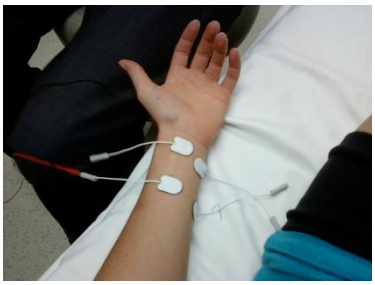


Figure 1 Electromyographie réalisée à l'aide d'électrodes de surface

Le problème vient du fait que les patients interagissent avec une prothèse via des électromyographies, enregistrées grâce à des électrodes de surface, comme le montre la photo ci-contre. Or, un apprentissage long et difficile est nécessaire pour apprendre à contrôler ce genre de prothèses ; de plus, ces prothèses ne permettent qu'un nombre restreint de mouvements basiques (ouverture/fermeture de la main). Pourtant les avancées récentes de la microtechnique ont permis de construire des automates de main possédant un contrôle très précis de la force employée, et ayant une grande variété de mouvements à disposition. Or, comme nous l'avons vu, il y a un réel besoin de prothèses capables de reproduire naturellement un grand nombre de mouvements, tout en ne demandant qu'un effort minimal pour en maitriser l'utilisation. Le projet NinaPro se focalise sur cette partie : « Le but du projet NinaPro est de proposer une famille d'algorithmes capable d'augmenter la dextérité et de réduire le temps d'apprentissage des prothèses... [8, page 4]»

Naturellement, ces algorithmes devront être testés sur de larges ensembles de données, afin d'en vérifier la pertinence. Cette récolte de données constitue le premier volet du projet NinaPro, et c'est dans ce volet que s'inscrit ce travail de bachelor. Le but de ce volet est décrit ainsi : « Le but de ce thème est de développer un protocole reproductible pour acquérir de larges ensemble de données concernant des patients sains ou amputés qui sont

[1] Ces données sont de différents types : il s'agit de données cliniques concernant des sujets, de relevés de signaux musculaires, et de photographies.

amenés à reproduire une série de mouvements […] , et de stocker et analyser ces données […] Le but principal est de constituer une base de données significativement plus large que l'état de l'art actuel […] nous visons un minimum de 30 patients et 100 sujets sains, et 50 postures de main […] [8, page 4]». Le but de ce travail de bachelor est donc de construire cette base de données, et l'application web permettant le stockage, l'extraction et l'analyse des données contenues dans cette base. La nature des données stockées étant étroitement dépendante du protocole utilisé, il convient de décrire ce protocole pour avoir un aperçu des différents types de données qui seront stockées.

1.1.1 Protocole de récolte de données

« La première tâche consiste à développer un protocole d'acquisitions […] [8, page 5]» Pour bien comprendre ce protocole, nous nous sommes proposés comme sujet, ce qui nous a permis de vivre toutes les étapes de ce protocole. Son but est d'effectuer une électromyographie d'un sujet exécutant divers mouvements, et d'acquérir une série d'informations cliniques sur un sujet donné ; les informations cliniques sont recueillies via un formulaire papier, disponible en annexe (cf. annexe H). Ces données comprennent des informations sur le sujet (nom, prénom, âge, taille, température de la peau..), des informations sur l'expérience en cours (lieu, date, température de la salle, etc..). La première étape consiste donc à recueillir ces informations via ce formulaire.

Figure 2 Outils d'acquisition

Le bras du sujet est ensuite recouvert d'électrodes, ces électrodes étant présentes dans un brassard rouge (cf. photo ci-contre). Ces électrodes vont réaliser l'électromyographie. En plus de l'électromyographie, des capteurs de mouvement, présents dans le gant, vont enregistrer les divers gestes du sujet.

Dans un premiers temps, le sujet va s'entrainer à réaliser divers mouvements ; pour ce faire, un film indiquant les gestes à effectuer va être présenté à l'écran, et le sujet devra exécuter les gestes en même temps que le film. Quatre séries de mouvements sont présentées :

1. Des mouvements simples de doigts
2. Des mouvements simples de la main et des doigts
3. Des gestes de préhension
4. Des mouvements fonctionnels

MOUVEMENTS BASIQUES

MOUVEMENTS DE LA MAIN

MOUVEMENTS DE PRÉHENSION

MOUVEMENTS FONCTIONNELS

Source : [8]

Tableau 1 Type de mouvements

Chaque mouvement est répété deux fois. Puis commence l'enregistrement des données proprement dites, les mouvements sont alors répétés dix fois. Un tableau, présent en annexe, résume ce protocole.

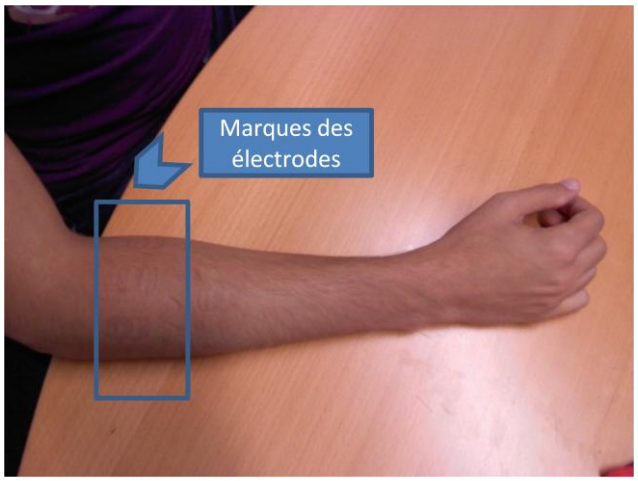


Figure 3 Bras du sujet sans gant

Une fois les exercices effectués, deux photos sont prises : la première photo présente le sujet avec le brassard d'électrodes et le gant. La deuxième photo présente le bras du sujet sans l'appareillage nécessaire à l'acquisition des données (cf. photo ci-contre), le but étant de pouvoir visualiser l'emplacement des électrodes via la marque qu'elles laissent sur le bras du patient.

1.1.2 Données récoltées

Les données récoltées après une acquisition consistent donc en :

- Trois fichiers textes pour chaque série d'exercices, ces fichiers contenant :
  - Les données de l'électromyographie
  - Les mouvements du sujet
  - Un fichier contenant des données de synchronisation
- Deux photos (bras du sujet avec/sans électrode)
- Le formulaire contenant les informations relatives aux patients et à l'expérience

1.2 Motivation du travail de bachelor

Le but de ce travail de bachelor est donc la création d'une structure de données susceptibles d'accueillir toutes ces données, et la création d'une application web permettant d'interagir avec cette structure de données. Cette application web doit contenir les fonctionnalités suivantes :

- Une gestion des utilisateurs et des droits relatifs à ces utilisateurs
- Une page permettant la saisie des données
- Une page permettant la visualisation des données
- Différents outils de recherche doivent être proposés pour visualiser ces données.
- Une page permettant l'extraction des données
- Cette extraction doit pouvoir se faire selon différents critères : âge du patient, état du patient, etc...
- Une page offrant des statistiques sous forme de graphiques, relatives à la fréquentation du site
- Une page offrant des statistiques sous forme de graphiques, relatives aux données saisies

De plus, cette application doit utiliser les technologies mises en œuvre dans le web 2.0 (Ajax…). Comme les coûts inhérents au développement et au déploiement ne seront pas amortis – cette application n'étant pas destinée à produire des rentrées d'argent -, il faudra essayer de les minimiser au maximum ; sans pour autant produire une application de moindre qualité : il faudra donc utiliser au maximum des outils gratuits.

1.3 Remerciements

Nous tenons à remercier toutes les personnes qui ont participées à ce projet et qui nous ont permis de le mener à bien :

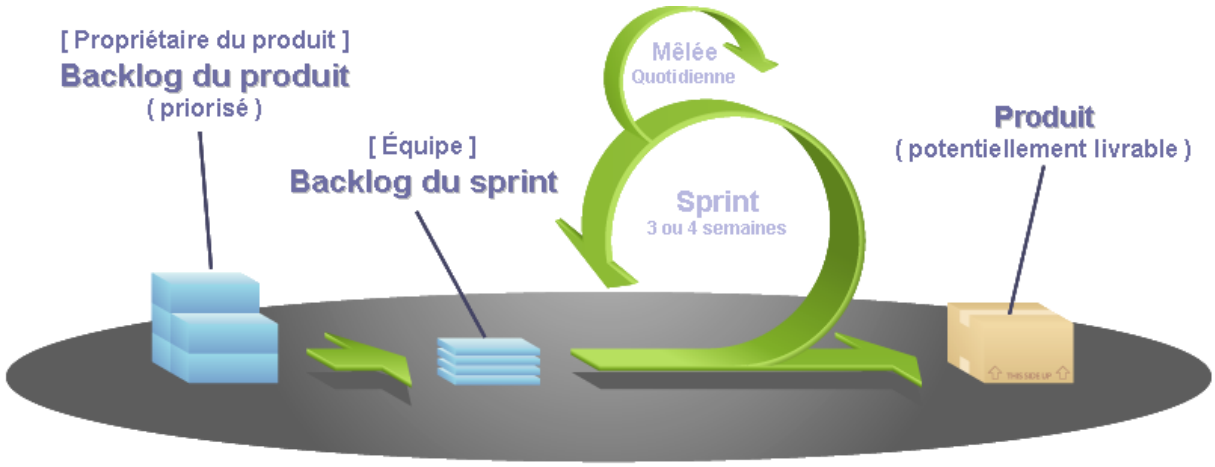
- M. Ivan Eggel pour avoir rapidement installé les différents outils sur le serveur de production
- M. Henning Müller pour sa disponibilité, son encadrement et les conseils qu'il m'a prodigués tout au long de la formation
- M. Manfredo Atzori pour sa disponibilité et ses réponses rapides à nos mails

Je remercie aussi ma grand-mère pour m'avoir hébergé pendant les trois années de la formation, et mon oncle pour les coups de main qu'il m'a donné durant ces dernières années, et toutes les autres personnes qui m'ont aidé le long de ces trois dernières années.

2 Méthodologie de développement

2.1 Scrum

La méthodologie de développement choisie s'inspire de Scrum, adaptée pour les besoins de ce projet. Scrum est avant tout une méthodologie de gestion d'équipe, alors que ce projet est l'œuvre d'une seule personne, mais nous nous sommes inspirés des grandes lignes de Scrum pour sa gestion. La figure ci-dessous représente le schéma classique de la méthode Scrum :



Source : fr.wikipedia.org

Figure 4 Méthodologie Scrum

Sans rentrer dans les détails, cette méthodologie comporte plusieurs étapes :

1. Elaboration d'un « Product Backlog », sorte de cahier de charges constitué d'User Story (Une User Story est un regroupement de fonctionnalités traversant toutes les couches de l'application, décrite du point de vue client)
2. Elaboration d'un « Sprint Backlog » : l'équipe choisit un ensemble d'User Stories à implémenter
3. Ces stories sont implémentées durant un « Sprint », au terme duquel un produit doit pouvoir être présenté au client
4. Celui-ci évalue le produit, et le cycle recommence au point 2

Dans ce projet, le développement s'est élaboré comme suit :

Un cahier des charges a été élaboré avec les différents intervenants du projet lors des premières réunions. Ce cahier des charges correspond dans les grandes lignes à un product Backlog, en ce sens qu'il décrit les grandes fonctionnalités du système. Chacune des ces fonctionnalités peut être vue comme une User Story, car elle possède les mêmes caractéristiques qu'une User Story, à savoir :

- Elle a été couchée sur le papier avec l'utilisateur
- Elle traverse les différentes couches du système
- Il s'agit des fonctionnalités désirées et décrites comme telles par le client final

Le projet est divisé en plusieurs périodes, à l'issue desquelles un livrable est fourni au client, soit sous forme de documents (par exemple le cahier des charges) ou sous formes de prototypes (interface web fonctionnel). Le livrable est alors présenté au client, qui décide des

prochaines fonctionnalités à implémenter (ce qui équivaut au Backlog de Sprint), en accord avec le planning initial. Ceci permet une certaine flexibilité dans le choix des fonctionnalités à implémenter. **Il faut noter qu'une fonctionnalité obligatoire a été introduite en cours de projet (le 25 juillet)**. Il s'agissait de l'upload de cinq photos pour une expérience donnée.

2.1.1 Sprint

La durée d'un sprint est traditionnellement fixée à 3-4 semaines, ce qui offre un temps suffisamment long pour implémenter de nouvelles fonctionnalités. A la fin d'un sprint, une réunion à lieu avec le client. Pendant cette réunion, un débriefing des activités réalisées, difficultés rencontrées et surmontées est réalisé. Le livrable est ensuite présenté au client, qui, comme expliqué plus haut, l'évalue et propose de nouvelles fonctionnalités à réaliser. Le planning est alors mis à jour, et l'itération suivante commence. Dans notre cas, la durée du sprint a été ramenée à une semaine.

2.1.2 Intervenants

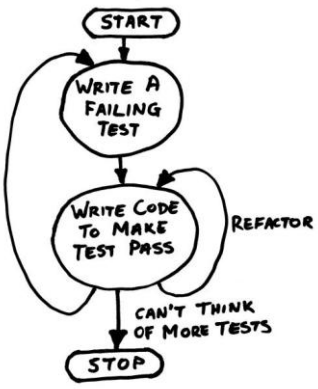
Ce travail de bachelor s'est déroulé en collaboration avec trois intervenants

- M. Muller Henning, qui a tenu le rôle de responsable administratif : Son rôle a été de coordonner les différentes réunions et de superviser la bonne marche de ce travail de bachelor
- M. Manfredo Atzori, qui a tenu le rôle de client : Son rôle a été d'expliciter les besoins clients, d'expliquer le projet NinaPro et de fournir les divers documents/fichiers nécessaires à la réalisation de cette application
- M. Ivan Eggel, qui a tenu le rôle d'administrateur système : Son rôle a été de mettre à disposition un serveur de production ainsi que et d'installer les différents éléments de ce serveur

L'anglais étant la seule langue commune aux intervenants, les réunions se sont déroulées en anglais.

2.2 Développement par les tests

L'une des ambitions de ce projet était de mettre en place un développement guidé par les tests, désigné par l'acronyme TDD (Test Driven Development). Le TDD se caractérise par le fait que les tests sont écrits **avant** de coder. Naturellement, ces tests échouent, car rien n'a été codé. Il faut alors écrire le code nécessaire au bon déroulement des tests, puis, une fois que les tests ont passés, nous pouvons factoriser le code en toute sécurité, car nous disposons de tests pour vérifier le bon fonctionnement du code. On parle de cycle rouge, vert, refactoring. La figure ci-contre résume ce cycle. On distingue généralement deux niveaux de test : les tests unitaires, qui testent le comportement d'une classe donnée, et les tests d'intégration, qui traversent les différentes couches d'une application donnée.



Source : fr.wikipedia.org

Figure 5 TDD

2.2.1 Influence sur le produit

Le TDD, de part le temps nécessaire à l'écriture de tests, ralentit le développement : « The projects also took at least 15% extra upfront time for writing the tests [11, page 1] », mais permet une augmentation du taux de qualité du produit fini, et permet surtout une meilleure maintenance du code. Un code avec une suite de tests peut être modifié sans crainte, car nous pouvons à tout moment tester son bon fonctionnement.

2.2.2 Code hérité

Le code hérité est une notion assez large qui désigne le code qui ne peut être en aucune façon modifié. Le code hérité ne possède pas forcément de connotation négative, en ce sens qu'il peut par exemple s'agir de routines de traitement de factures écrites en Cobol, ce qui rend leurs performances très élevées. De telles routines n'ont aucune raison d'être réécrites en Java ou C#, car leurs performances seraient bien moindre. Par contre, on parle aussi de code hérité dans le cas de codes mal écrits, mal commentés, ou sans tests. L'absence de tests et de documentations, la mauvaise écriture du code empêchent le développeur lambda de comprendre le code et de le modifier. Ce code hérité est donc le cauchemar de tout développeur, et l'écriture de tests constitue un excellent garde-fou pour en empêcher l'apparition.

2.2.3 BDD et tests d'intégration

L'écriture de tests peut même constituer une spécification pour une classe donnée : il s'agit d'une forme "extrême" de TDD que l'on nomme BDD (Behavior Driven Development). Une suite de tests est vue comme une spécification pour une classe donnée. Le développeur, en passant cette suite de tests, aura donc implémenté la spécification pour cette classe donnée.

A un niveau plus élevé, les tests d'intégration peuvent aussi constituer la spécification d'une application donnée. Cette manière de faire est couramment pratiquée chez les utilisateurs du framework Ruby On Rails. Pour s'en convaincre, il suffit de lire un livre traitant de Rails[2], pratiquement chaque exemple s'accompagne d'un test.

Le point 3.6 reprend les points évoqués ci-dessus et montre avec quels outils ils ont été implémentés dans le cadre de notre projet ; nous évaluerons notre utilisation du TDD au point 6.2.6

[2] [7] Ce livre est un tutorial guidant le développeur à travers la création d'un blog; chaque étape est accompagnée d'une série de test

3 Technologies utilisées

Nous allons à présent parcourir les différentes technologies mises en œuvre dans ce projet, ainsi que les outils utilisés pour son développement. Pour chaque technologie et outil, nous envisagerons aussi les principales alternatives présentes sur le marché, et nous expliquerons pourquoi nous ne les avons pas retenues. Nous nous attarderons ensuite sur l'utilisation de l'Ajax, qui constitue une part importante de ce projet. Nous parcourrons ensuite quelques ensembles de technologies alternatives, puis nous terminerons ce chapitre par une description des outils de test utilisés.

3.1 Technologies utilisées côté serveur

3.1.1 Java

Le vocable Java prête à confusion, car il a été intentionnellement utilisé par Sun, puis Oracle, pour désigner sous une même appellation plusieurs éléments distincts:

- Un langage de programmation
- Une machine virtuelle (la JVM)
- Un ensemble de librairies facilitant l'écriture de code
- Un framework facilitant le développement d'application d'entreprises (Java EE)

Reprenons ces points

Java est donc tout d'abord un langage de programmation orienté objet issus d'un projet de Sun datant de 1990. L'un de slogans accompagnant ce langage était "Write once, run everywhere".

La particularité principale de Java est donc que les logiciels écrits dans ce langage sont très facilement portables sur plusieurs systèmes d'exploitation tels qu'UNIX, Windows, Mac OS ou GNU/Linux, avec peu ou pas de modifications. Ce tour de passe-passe est accompli grâce aux compilateurs Java qui "précompilent" le code source afin d'obtenir un bytecode Java, un langage machine propre à Java. Ce bytecode peut ensuite être exécuté sur une machine virtuelle Java(JVM), un programme écrit spécifiquement pour la machine cible qui interprète et exécute le bytecode Java. La JVM offre différents services facilitant le développement, le plus notable étant le Ramasse-Miettes, garbage collector en anglais, un service gérant la création et la destruction des objets et les allocations de mémoire pour ces mêmes objets.

Un ensemble de librairies

Sun a écrit un ensemble de librairies facilitant le développement, ces librairies permettent aux développeurs de ne pas avoir à chaque fois réinventer la roue, et suivent en ce sens le principe DRY (Don't Repeat Yourself). Ces librairies contiennent un ensemble de classes permettant de gérer les problèmes classiques qu'un développeur peut rencontrer : gestion de fichiers, création d'interfaces utilisateurs, structure de données, multithreading, etc.…

3.1.2 Java EE

Le contenu de la présentation qui suit s'inspire du livre d'Antonio Goncalves [6, pages 1-30]

Java EE est un ensemble de spécifications implémentées par différents conteneurs. Un conteneur est un environnement d'exécution qui fournit un certains nombres de services aux composants qu'il héberge. Ces composants utilisent des contrats bien définis pour communiquer avec l'infrastructure Java EE sous-jacente ainsi qu'avec les autres composants. Ces composants doivent être archivés de manière standard (dans des .JAR, ce sont des .ZIP java), avant de pouvoir être déployés.

La figure-ci-dessous représentent une vue d'ensemble de ces conteneurs. Les flèches représentent les protocoles utilisés par un conteneur pour communiquer avec d'autres conteneurs.

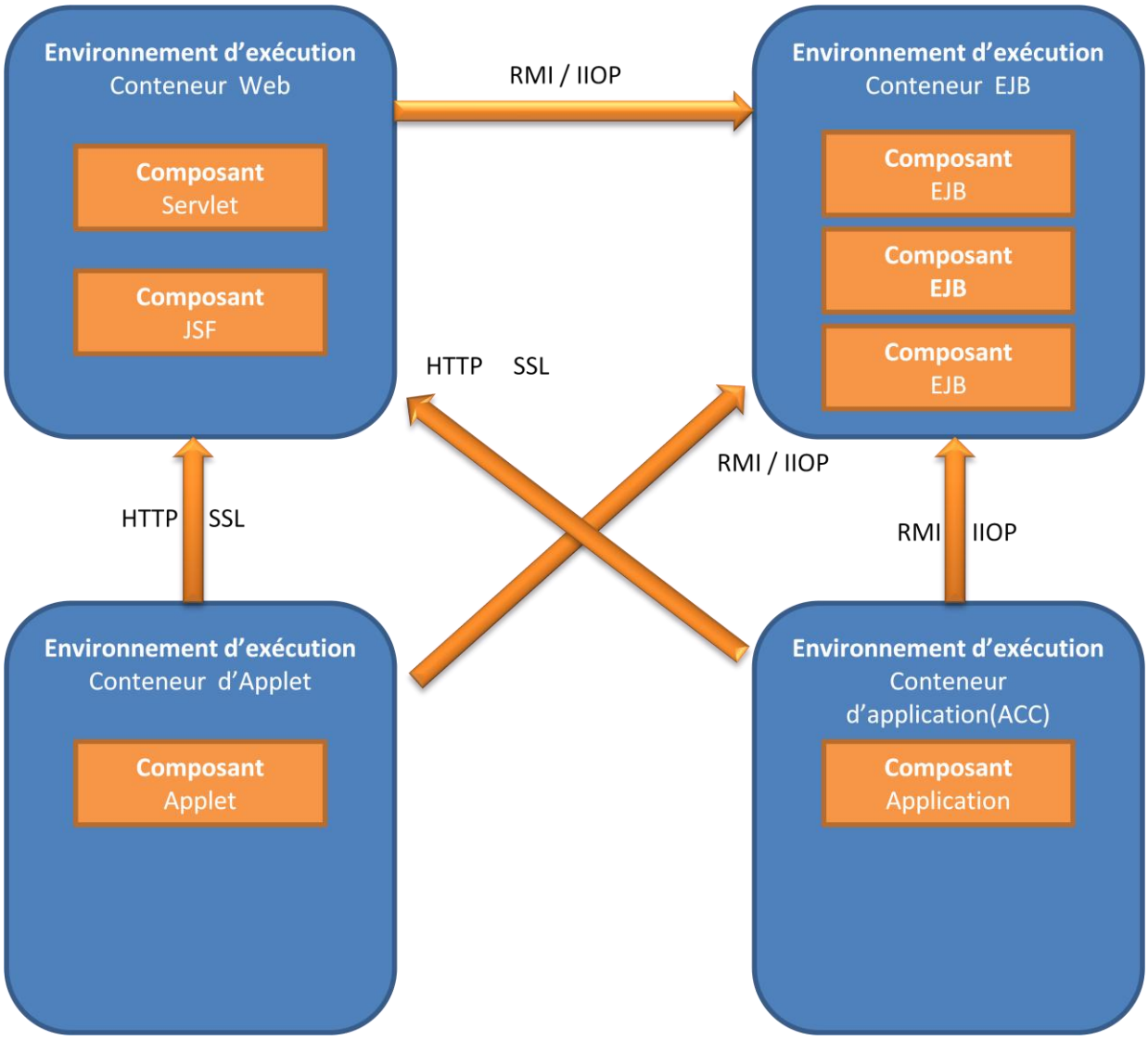


Figure 6 Conteneur JAVA EE

L'infrastructure JAVA est divisée en domaines appelés conteneurs. Chaque conteneur a un rôle donné, possède son API spécifique, et offre divers services à ses composants, et permet d'interagir avec différentes sources de données. Les conteneurs cache les complexités (code de bas niveau) et facilite la portabilité du code. En fonction de l'application qu'un développeur veut construire, il devra comprendre et utiliser les capacités et contraintes de

tel ou tel conteneur. La figure ci-dessous résume les différents services offerts par les conteneurs

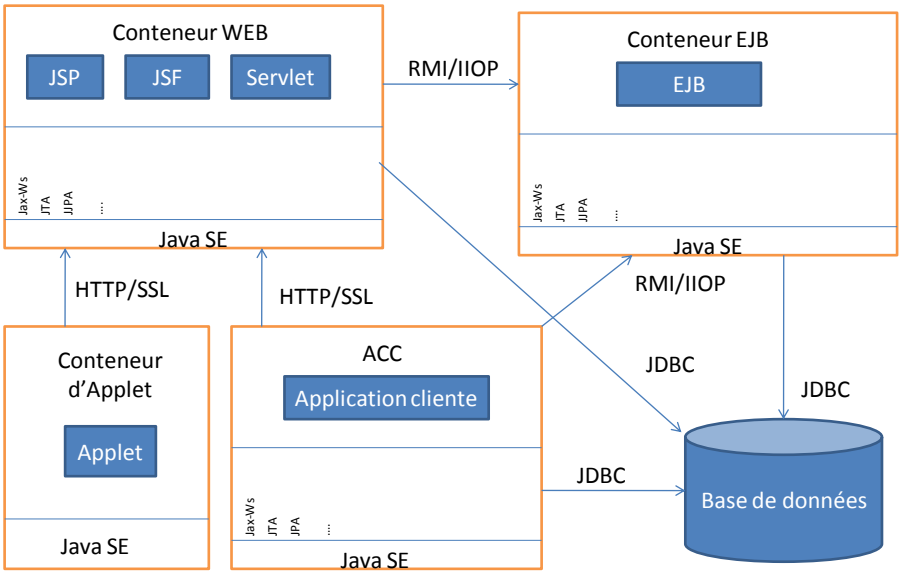


Figure 7 Conteneurs et services

Pour notre application, nous avons choisi de nous limiter à un seul conteneur, le conteneur web. Les services offerts par ce conteneur sont suffisants pour notre application. De plus, nous limiterons ainsi le nombre de JAR à charger par l'environnement d'exécution, et la mémoire utilisée.

3.1.3 GlassFish

Java EE n'étant qu'une spécification, il convient de choisir une implémentation. Nous avons choisi le serveur GlassFish, qui est l'implémentation de référence proposée par Oracle. GlassFish est un serveur simple à installer, et possède une interface d'administration distante simple à utiliser. Comme le montre la figure 8, cette interface permet d'accéder rapidement aux journaux du serveur, d'arrêter et (re)démarrer le serveur, de (re)déployer facilement une application, et de configurer les différentes ressources du serveur. De plus, GlassFish est basé sur une architecture Osgi(Open Services Gateway initiative), cela signifie est qu'il capable de gérer dynamiquement les modules dont il a besoin. Comme nous n'utiliserons que le conteneur web de la spécification EE, nous limiterons ainsi la consommation de mémoire par le serveur.

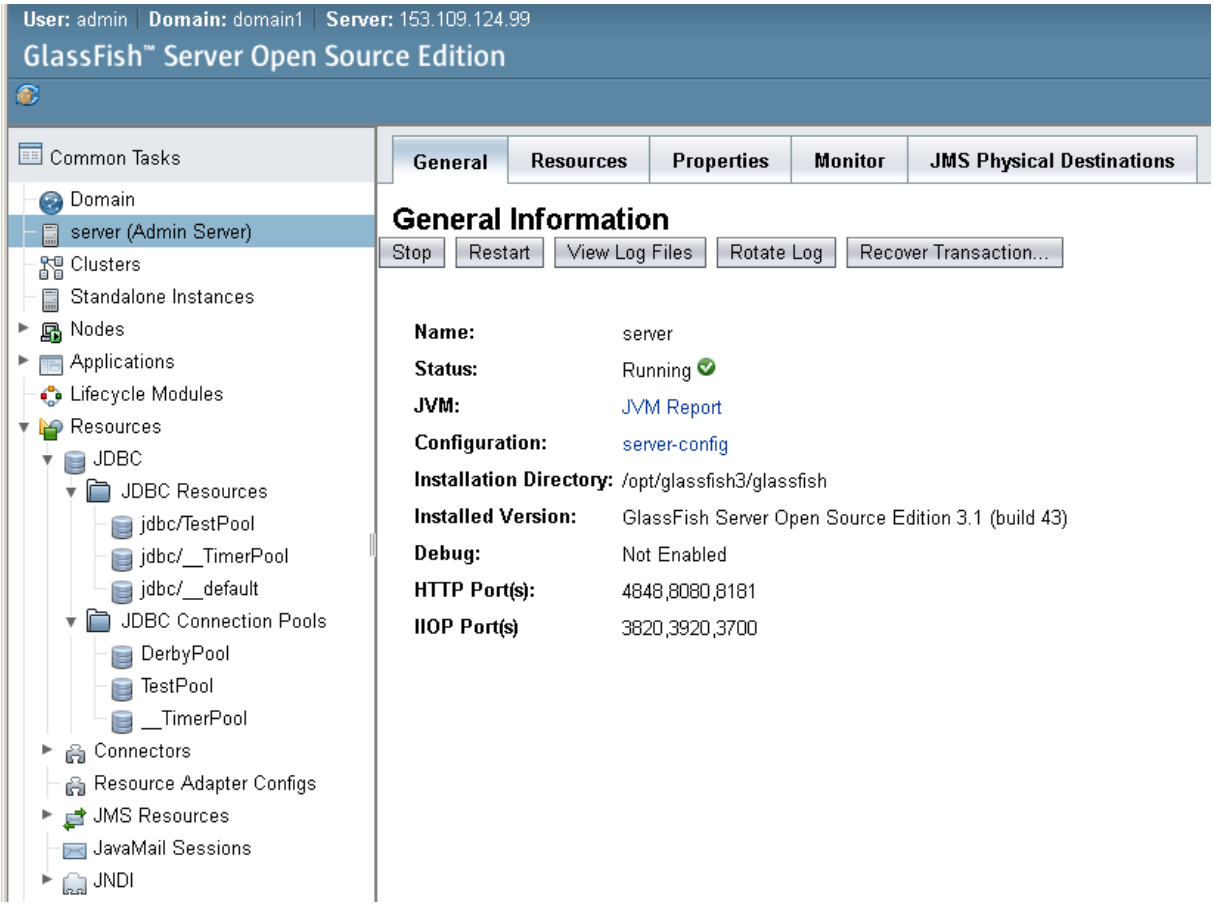
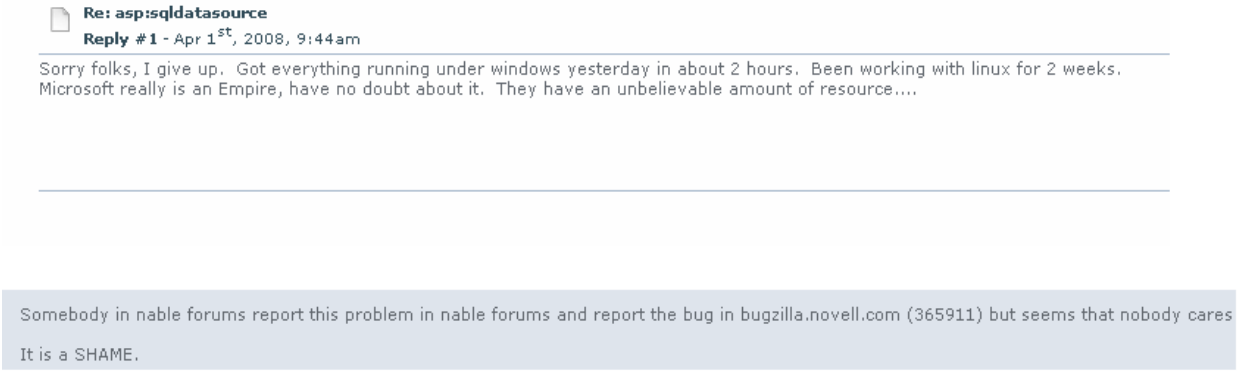


Figure 8 Console d'administration de glassfish

Autres serveurs Java EE

D'autres serveurs implémentent tout ou partie du standard EE, les plus connus étant JBoss, Tomcat (pour la partie web de la spécification). Ces serveurs nécessitent d'être configurés via des fichiers XML, ce qui les rend plus compliqués à gérer. De plus, GlassFish étant l'implémentation de référence, il possède une documentation abondante ainsi que de nombreux forums; ses bugs sont documentés et on peut donc trouver facilement des moyens de les contourner, contrairement à d'autres serveurs, comme le prouve la figure ci-dessous, extraite d'un forum traitant d'un serveur "concurrent":



Source : www.go-mono.com/forums/

Figure 9 Un utilisateur désespéré

Quoiqu'il en soit, notre application a été écrite en n'utilisant que des bibliothèques du standard EE, elle devrait donc être facilement pouvoir être migrée sur un autre serveur respectant ce standard. A titre d'exemple, le serveur de production tourne sur un OS linux, et le serveur de développement sur un OS Windows, et aucune modification n'a dû être effectuée.[3]

3.1.4 MySql

MySql est une base de données disponible gratuitement et abondamment documentée. Elle dispose de connecteurs pour de nombreux langages, ainsi que de nombreux utilitaires permettant de faciliter le développement. MySql est disponible librement sur le web et reste simple à installer.

Moteur utilisé

MySql offre différents moteurs de base de données, chacun étant destiné à différents usages; les deux principaux moteurs étant[4] :

- MyIsam: Ce moteur ne gère pas les clefs étrangères, et ne prend pas en charge les transactions, ce qui lui confère une plus grande rapidité, au détriment de l'intégrité des données. De plus, le verrouillage se fait au niveau des tables entières, et non au niveau des lignes, ce qui le rend peu efficace dans le cas d'écritures concurrentes.
- InnoDB: Ce moteur gère les transactions, les clefs étrangères, et le verrouillage au niveau des lignes. Il est évidemment moins rapide que MyIsam, mais plus sûr. Il 'agit du moteur par défaut de MySql depuis la version 5.5

Notre choix s'est porté sur le moteur InnoDB, afin de préserver l'intégrité des données, ce qui facilite grandement le développement en empêchant des erreurs lors de requêtes SQL et en mettant en lumière tout problème d'intégrité référentielle dans la base de données. Cependant, pour les tables ne possédant pas de clefs étrangères, nous avons utilisé le moteur MyIsam, afin d'augmenter les performances.

Outils utilisés

De nombreux outils accompagnent MySQL et permettent de favoriser le développement et la maintenance de la base de données. Parmi les plus connus, nous avons utilisé :

- PhpMyAdmin est une application web écrite en PHP et tournant sur un serveur Apache qui réside sur la machine hébergeant la base de données. Cette application permet d'effectuer toutes les tâches d'administration courantes depuis un poste distant via une interface graphique simple et conviviale.
- MySqlWorkBench : MySqlWorkBench est une application native se connectant sur une base de données distante. Le principal inconvénient de MySqlWorkBench est qu'il nécessite l'ouverture de ports supplémentaires, typiquement le port 3305, afin de pouvoir se connecter à la DB distante, ce qui limite son utilisation dans un

[3] Naturellement, des problèmes ont été rencontrés dans le code, mais ils étaient dus à une méconnaissance de Java, qui propose une librairie permettant de s'abstraire des différences entre OS

[4] Ces informations sont tirées de la documentation de MySQL, disponible sur le site suivant : http://dev.mysql.com/doc/ (Page consultée le 15 juillet 2011)

réseau d'entreprise qui va ne laisser ouvert que les ports 80 ou 8080 (il 's'agit des ports http). Pour cette raison, nous avons utilisé PhPMyAdmin; nous avons aussi utilisé un plugin conçu pour Eclipse permettant d'accéder à la base de données; nous décrirons ce plugin au point 3.4.1

Autres bases de données

Bien entendu, de nombreuses autres bases de données existent, et nous allons rapidement passer en revue quelques une des bases que nous aurions pu utiliser dans notre projet, et les raisons qui nous ont conduits à ne pas les retenir.

- MSSQL : Il s'agit de la base de données Microsoft ; cette base de données est payante, de plus, l'utiliser prend tout son sens dans une architecture purement Microsoft, ce qui n'est pas le cas de notre projet.
- Derby : Il s'agit d'une base de données écrite en Java, et qui est embarquée sur le serveur GlassFish. Une de ses principales caractéristiques est qu'elle permet l'écriture de procédure stockée en Java, ce qui lui confère une très grande flexibilité. Malheureusement, ses performances sont moins élevées que celle de MySql. Etant donné que MySql est gratuite, notre choix s'est porté sur la base de données la plus performante
- PostgreSQL : Il s'agit d'une base de données open source et performante. Malheureusement, elle ne dispose pas des mêmes outils que ceux accompagnant MySQL. Ces outils permettant d'accélérer le développement, notre choix s'est porté sur MySQL

3.1.5 JPA / EclipseLink

Cette présentation de JPA s'inspire de l'ouvrage [12] :

JPA est une spécification permettant de faciliter la persistance d'objets Java en base de données. En effet, un objet Java et une ligne d'une table d'une DB peuvent être vus comme deux représentations différentes d'un même ensemble de données; malheureusement, il y a de grandes différences entre ces deux représentations : d'un côté, les tables, lignes, colonnes d'une base de données sont relationnels, et utilisent les types de la DB utilisée, alors que les objets Java possèdent les types Java et utilisent des principes objets tels que l'héritage, l'encapsulation, la composition, etc.… De telles différences sont appelées dans la littérature « impendance mismatch », et elles requièrent l'écriture, de la part du développeur, de nombreuses lignes de code[5]. Pour soulager le développeur de ce travail, des ORM (Object-Relational Mapper) ont été développés.

JPA est une spécification standard pour un tel ORM. L'implémentation de JPA que nous utilisons est EclipseLink

Les fonctionnalités offertes par JPA peuvent être décomposées en deux parties principales :

[5] Typiquement, l'écriture de requêtes pour aller chercher des informations dans la base de données, puis l'écriture d'instructions d'affectations de ces informations aux divers champs de l'objet.

Mappage entité – table

L'idée est de faire correspondre à un objet Java certains éléments d'une base de données. Ce travail se fait à l'aide d'annotations qui permettent d'indiquer à quelle ligne d'une table correspond un champ d'un objet. Le principal problème rencontré est le mappage des relations entre les différents objets. Typiquement, dans le cas d'une relation n à n, il faut indiquer la table de relation correspondante et déterminer quel objet est maitre de la relation.

Pour faciliter le mappage, JPA applique le principe de « Convention over Configuration »; il s'agit simplement de minimiser l'écriture de code de la part du développeur en utilisant des conventions simples et évidentes; typiquement, une table correspondant à une classe donnée portera son nom; les tables de relations auront comme nom le nom des classes correspondantes concaténées. Bien sûr, ces conventions peuvent être surchargées, ce qui s'avère indispensable si nous n'avons aucun contrôle sur la base de données.

La figure 10 (page suivante) montre un exemple typique de mappage tiré de notre application. Il s'agit du domaine relatif à la gestion des utilisateurs et des droits. A droite, deux classes Java sont présentes, avec leurs différents champs; ces champs sont annotés pour indiquer à JPA leur contrepartie dans la base de données; les annotations les plus utilisées sont les suivantes :

- @Id : Cette annotation indique à JPA que ce champ correspond à la clef primaire de la table correspondante à l'objet. L'annotation @GeneratedValue permet d'indiquer à JPA la stratégie de génération de la clef primaire; dans le cas de MySql, il s'agit d'une clef primaire auto-incrémentée.
- @Transient : Indique que ce champ ne doit pas être persisté en base de données
- @Column : Définit les caractéristiques de la colonne correspondante dans la base de données
- @OneToOne, @ManyToMany.. : Permet d'indiquer une relation 1 à 1, ou n à n. Il faut indiquer à JPA si une table de relations est utilisée, et les clefs étrangères utilisées.

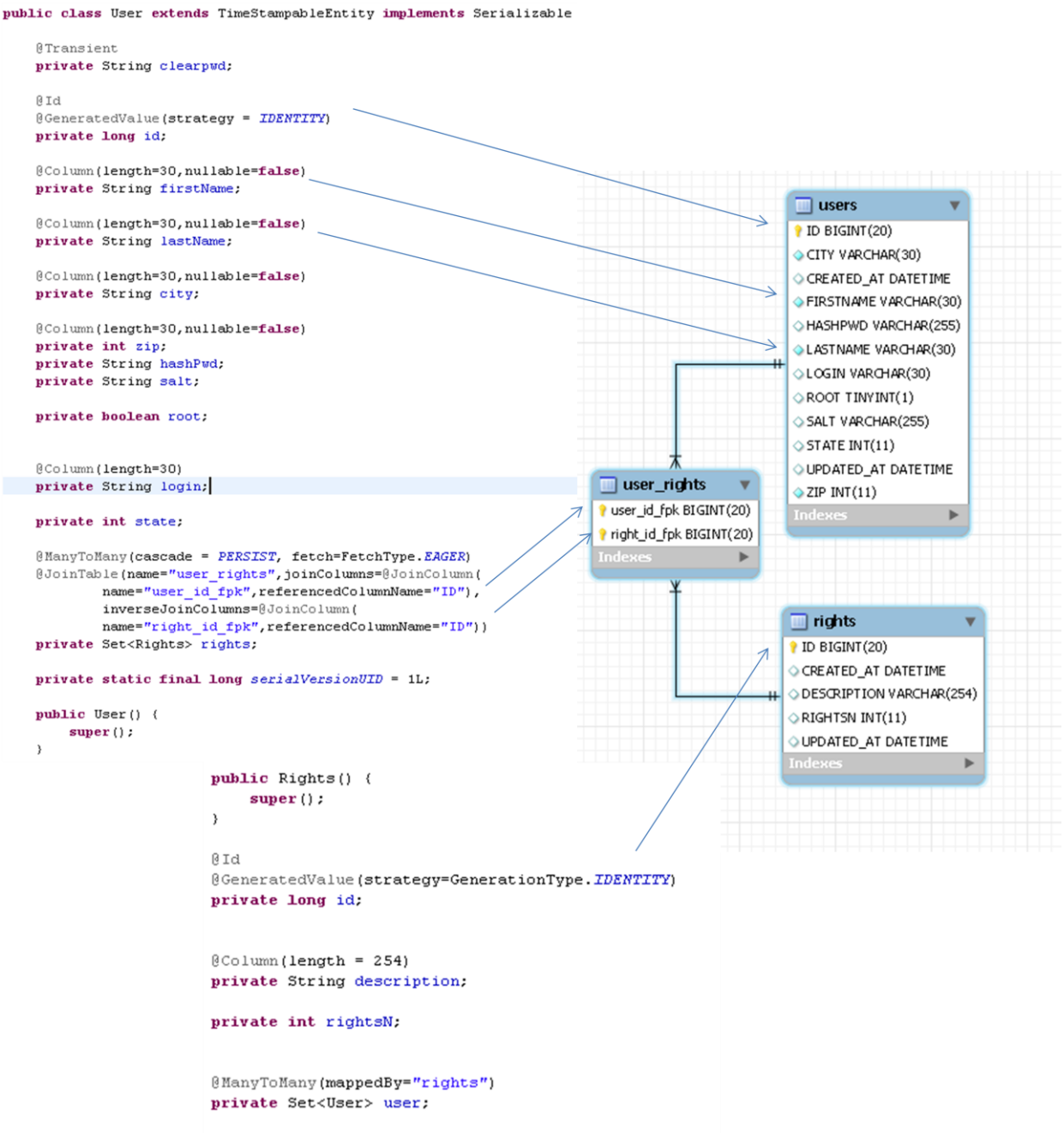


Figure 10 Mappage classe-table

Manipulation des entités

Lorsqu'un objet JAVA est créé ou modifié, il faut tôt ou tard refléter ces changements dans la base de données. L'ORM doit donc, au moment opportun, générer les requêtes SQL adéquates de la manière la plus efficace possible; typiquement l'ORM doit déterminer si l'objet est déjà présent dans la base de données, ce qui va influencer sur le type de requête (INSERT ou UPDATE ou DELETE). De nombreux patterns, que l'on peut retrouver dans la littérature spécialisée[6], sont utilisés par l'ORM, on peut citer à titre d'exemples :

[6] Le livre suivant fait office de référence : Design Patterns: Elements of Reusable Object-Oriented Software

I. **Unit of Work**[7]: Ce pattern maintient une liste des objets pris en charge par l'ORM et coordonne l'écriture des changements faits à ces objets, tout en gérant les transactions.

II. **Identity map**[8] (ou table d'identité): Une table d'identité est un pattern utilisé pour améliorer les performances d'un système en fournissant un cache permettant de prévenir le chargement d'objets déjà présents en mémoire. Les clefs de la table vont être construites en utilisant le type de l'objet, et son id. Si l'objet requis a déjà été retiré de la base de données, la table d'identité pourra fournir directement l'objet; si ce n'est pas le cas, l'objet est chargé de la base de données, puis stocké dans la table.

Mieux vaut connaître ces patterns pour ne pas être étonné par le comportement des ORM, qui peuvent parfois sembler violer le principe de moindre surprise.

JPA offre de nombreuses annotations qui permettent de contrôler finement le cycle de vie des objets prise en charge. Il suffit d'annoter une méthode de l'objet pour qu'elle soit appelée lors de l'étape du cycle de vie correspondant :

Annotation	A quel moment
@PrePersist	Juste avant qu'une entité est persistée pour la première fois dans la BD.
@PostPersist	Après qu'une entité soit persistée dans la DB.
@Pre/PostRemove	Avant/après qu'une entité soit effacée de la base de données
@Pre/PostUpdate	Avant/après qu'un objet soit updaté dans la base de données

Tableau 2 Annotations gérant le cycle de vie

Le code ci-dessus montre un exemple typique d'utilisation de ces annotations, elles permettent ici de gérer la journalisation des modification/créations d'enregistrements dans la base de données[9].

[7] http://en.wikipedia.org/wiki/Identity_map (Page consultée le 22 juillet 2011)
[8] http://msdn.microsoft.com/en-us/magazine/dd882510.aspx (Page consultée le 22 juillet 2011)
[9] Ce mécanisme a été utilisé dans le cadre du projet

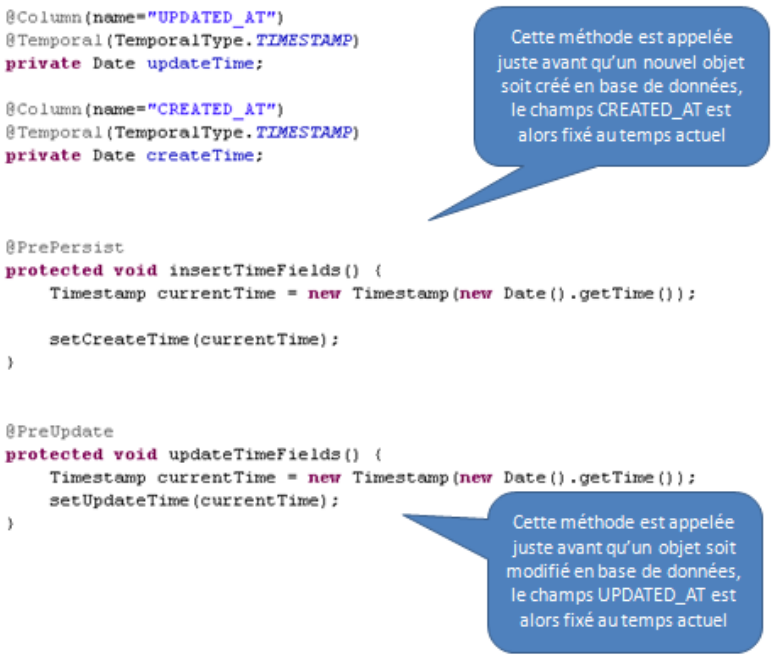


Figure 11 Méthodes intervenant lors de la persistance/mise à jour d'un objet en base

Contraintes et validations

JPA permet aussi de poser des contraintes via des annotations, sur les différents champs d'un objet.

```
//@Past(message="Can't be in the future")
//@Temporal(value = TemporalType.TIMESTAMP)
private String lastMeal;

//@NotNull(message="You must enter a value")
//@Min(value=30,message=" Must be higher than 30")
private Float body_temp;

@Min(value=21,message=" Must be higher than xx")
//@NotNull(message="You must enter a value")
private Integer skin_resistance;

@Min(value=1, message="The weight must be higher than 1")
//@NotNull(message="You must enter a value")
private Float weight;

@Min(value=1, message="The height must be higher than 1")
//@NotNull(message="You must enter a value")
private Float height;

//@NotNull(message="You must enter an age")
//@Min(value=1, message="Must be higher than 1")
//@Max(value = 100, message="Must be lower than 100")
private Integer age;
```

Figure 12 Annotation de contraintes

Ces contraintes sont vérifiées uniquement au niveau applicatif, et ne se retrouvent pas en base de données[10]. JPA déporte donc une partie du travail de validation au niveau applicatif, ce qui peut être vu comme un point négatif, typiquement si l'on souhaite, à contrario, utiliser la base de données pour y implémenter la logique de validation.

Ces contraintes augmentent l'intégrité des données, et peuvent être ensuite réutilisées dans la couche de vue (voire point 3.1.9), typiquement lors de la validation d'un formulaire. Le code à écrire pour la gestion de la vue s'en trouve alors considérablement allégé.

[10] C'est l'annotation @column qui va définir la colonne dans la base de donnée.

En résumé, JPA permet de soulager le développeur de l'écriture du code relatif à la gestion de la base de données : les opérations sur les objets Java sont reflétées en DB sans intervention de la part du développeur, les tables sont créées automatiquement[11]. Dans cette optique, JPA permet d'avoir une indépendance par rapport à la base de données : idéalement, nous pouvons changer de base de données sans problème. Naturellement toute médaille a son revers, et dans le cas de JPA, celui-ci ne pourra pas prendre en compte les fonctionnalités spécifiques à une base de données.

Quelques critiques

JPA donne malheureusement une fausse impression de facilité et de sécurité. Sans prudence, le développeur peut finir par charger toute la base de données en mémoire. Les conséquences peuvent être désastreuses si nous nous trouvons en face d'une DB importante. Le même problème peut survenir lors d'une sérialisation en XML/JSON: toute la DB se trouve d'un coup sérialisée dans un fichier XML

```
01 class Customer {
02
03   @OneToMany(mappedBy="customer")
04   private Set<Order> orders;
05
06 }
07
08 class Order {
09   @ManyToOne
10   private Customer customer;
11 }
```

Source : www.andygibson.net

Figure 13 Deux simples classes

Andy Gibson, dans son blog[12], explique clairement le problème :

Considérons ces deux classes : un client possède un ensemble de commandes. Andy Gibson met en lumière cinq problèmes liés à cette relation :

> Our problem is with the value we get from customer.getOrders() as this set of order entities doesn't really serve any useful purpose and can cause more problems than it solves for the following reasons :
>
> **Dumb Relationship** – It will contain every order for this particular customer when you usually only want a subset of the orders that match a set of criteria. You either have to read them all and filter the ones you don't want manually (which is what SQL is for) or you end up having to make a call to a method to get the specific entities you are interested in.
>
> **Unbounded dataset** – How many orders a customer has could vary and you could end up with a customer with thousands of orders. Combined with accidental eager fetching and loading a simple list of 10 people could mean loading thousands of entities.
>
> **Unsecured Access** – Sometimes we may want to restrict the items visible to the user based on their security rights. By making it available as a property controlled by JPA we lose that ability or have to implement it further down in the application stack.

[11] Les tables sont crées lors du déploiement , JPA offrant différentes stratégies pour la (re)-création des tables.
[12] http://www.andygibson.net/blog/ (Page consultée le 15 juin 2011)

No Pagination – Similar to the unbounded dataset, you end up throwing the whole list into the pagination components and letting them sort out what to display. In most cases, you need to treat each dataset like it will eventually contain more than 30 records so you really need to consider pagination early.

Overgrown object graph – When you request an entity, how much of the object graph do you need? How do you know which pieces to initialize so you can avoid LIEs? This is often the case with JPA, but is also more relevant when you take account of the needs to serialize object graphs to XML or JSON. Sometimes you might need the relationships and sometimes you do not depending on the context you will be using the data in.

Rife with pitfalls – Who saves and cascades what and how do you bind one to the other? You create an order, and assign the customer, do you need to then add it to the customers list of orders or not. What happens if you forget to add it to the customer and you save the customer? Whatever strategy you pick for dealing with this will no doubt end up being implemented inconsistently.

La réponse proposée est de créer une série de méthodes getOrders(), chacune ayant une granularité différente.

3.1.6 JavaServerFaces 2.1

JSF 2.1 est une technologie de vue web. Son but est de faciliter la création de pages web. JSF est une spécification, et son implémentation de référence est le projet Mojarra. Le serveur GlassFish embarque Mojarra; malheureusement, GlassFish utilise une ancienne version de Mojarra comportant des bugs, et il faut donc manuellement remplacer les Jars dans le serveur.

JSF, tout comme les premières versions des EJB (Entreprise Java Beans ; une autre spécification), a rapidement acquis une mauvaise réputation[13] : trop lourd, trop lent, rempli de bogues, trop de configurations à effectuer via du XML, atteint du symptôme de la tour d'ivoire[14]. Antoine Sabot-Durand, dans un billet sur JSF, résume les tares des premières versions de JSF[15] :

- Une technologie de templating ignorant le HTML
- Seul le POST HTTP peut être le point de départ d'un traitement complet côté serveur
- Aucune prise en compte de l'Ajax
- Un écosystème morcelé

Les versions suivantes se sont attelées à corriger ces problèmes et à améliorer JSF et, effectivement, sur le papier, JSF 2.1 paraît intéressant. David Geary résume ainsi les ameliorations apportées: « This book focuses on JSF 2.0, a major improvment over previous

[13] Le site *http://ptrthomas.wordpress.com/2009/05/15/jsf-sucks/* est représentatif des réactions à la première sortie du framework

[14] On reproche souvent aux spécifications du monde JAVA d'être écrite par des gens coupés de la réalité, ou n'ayant pas à développer des applications métier. D'où le nom du symptôme de la tour d'ivoire.

[15] http://blog.ippon.fr/2011/04/27/jsf-je-taime-moi-non-plus/ (Page consultée le 20 juillet 2011)

versions. JSF 2.0 is much simpler to use than JSF 1.x, and it provides new and powerful features, such as easy Ajax integration. [5, pages 1-2] »

Malheureusement, JSF continue à violer ce que l'on nomme « Le principe de la moindre surprise » et force le développeur à employer des détours parfois malheureux. Nous y reviendrons.

« JSF is a component-based framework [5, page 2] ». JSF propose de construire des pages web de la même façon que l'on construit une application native en Swing : en utilisant des composants dont le comportement sera défini dans le code, typiquement à l'aide d'écouteurs d'évènements.

JSF facilite l'écriture de pages web en offrant une librairie de balises; chaque balise représentant un composant graphique (une grille de données, un champ d'entrée texte...).Ces balises peuvent utiliser des informations, ou appeler des méthodes définies dans des objets Java. L'idée étant que derrière une page web, se trouve une série d'objets fournissant les données et les méthodes nécessaires à l'affichage de la page et à la gestion des interactions avec l'utilisateur. Naturellement, dans le cadre d'une application web, le problème principal est la gestion du cycle de vie de ces objets : doivent-ils durer pour une requête donnée, pour une page donnée, ou durant toute la durée de vie de l'application ? Nous verrons que le standard EE et JSF fournissent des outils pour résoudre ces problèmes.

JSF fonctionne comme un contrôleur frontal : un seul contrôleur intercepte les requêtes http du client et en fonction de ces requêtes, va créer ou modifier des vues, qu'il va renvoyer au client. La figure ci-dessous résume le fonctionnement de JSF :

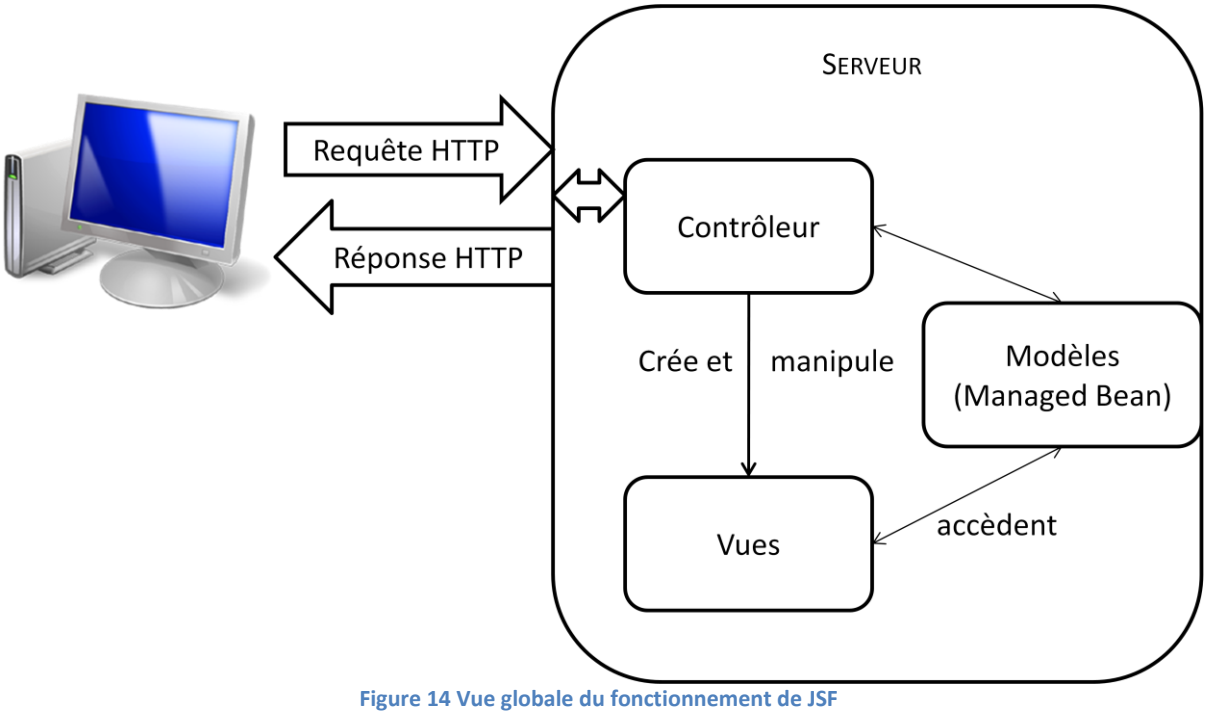


Figure 14 Vue globale du fonctionnement de JSF

Quelles sont les avantages de JSF ?

JSF is not the only component-based web framework, but it is the view layer in the JAVE EE standard. JSF is included in every JAVA EE application server […] JSF is a standard with

> multiple implementations. This gives you a choice of vendors […] as a standard, JSF is continuously improved and updated. [5, page 4]

Pour résumer, JSF est un standard, et dispose donc d'une importante documentation, il est constamment mis à jour, et devrait permettre aux développeurs de s'abstraire des spécificités du développement web pour se concentrer sur la logique applicative. Est-ce que ce but a été atteint ? Nous proposerons un début de réponse à la section 6.3.

3.1.7 RichFaces

RichFaces est une extension de JSF qui offre une librairie de composants supplémentaires. RichFaces facilite et accélère le développement de pages web utilisant l'Ajax. Sa principale force est qu'il s'intègre parfaitement avec JSF 2.0 et ne nécessite aucune configuration supplémentaire. Il suffit de télécharger les Jar correspondants à RichFaces, et ses dépendances, qui sont clairement indiquées. Ces Jars n'ont pas à être installées sur le serveur, elles sont déployées avec le projet.

RichFaces possède un site où une démonstration de tous les composants est proposée, accompagnée du code source, et d'une documentation présentant les différents composants, avec de nombreux exemples[16].

Autres librairies de composants

PrimeFaces est une autre librairie qui possède un très grand nombre de composants (Plus de 100 composants). Malheureusement, PrimeFaces ne possède pas l'équivalent de certaines fonctionnalités de RichFaces, comme la création automatique de fonctions JavaScript. Ces fonctions étant nécessaires à la création des graphiques, nous n'avons pas retenu PrimeFaces.

IceFaces est une autre librairie, aux caractéristiques semblables, mais aux performances moindres. Un comparatif récent, basé sur l'analyse d'un composant semblable entre les trois librairies, sur le site http://www.hightechno.info/2011/05/primefaces-vs-richfaces-vs-icefaces.html, la place bonne dernière au niveau des performances. Le tableau ci-dessous résume les performances (temps et mémoire utilisé par requête) de ces librairies :

Framework	Temps de réponse	Consommation mémoire
IceFaces	36.5 ms	230 Kb
PrimeFaces	13.8 ms	100 Kb
RicheFaces	25.2 ms	100 Kb

Tableau 3 Benchmark .XXXFaces

Ce comparatif met aussi en évidence un bogue dans la gestion des tables[17] de PrimeFaces.

[16] http://www.jboss.org/richfaces (Page consultée le 22 juillet 2011)
[17] La pagination ne fonctionne en que du côté client : le serveur doit donc envoyer toutes les données disponibles au client lord du chargement initial de la page

3.1.8 Weld

Weld est l'implémentation de référence de la spécification JSR-299 (CDI). Le but de Weld est de fournir un ensemble de services améliorant la structure du code applicatif. Ces services fournissent[18]:

- La prise en charge du cycle de vie des objets avec état, ce cycle de vie pouvant être associé à différents contextes.
- une approche de l'injection de dépendance permettant un typage fort.
- un système de gestion d'évènements permettant une gestion fine des interactions entre objets.
- une meilleure approche pour lier des intercepteurs à des objets, avec un nouveau type d'intercepteurs, appelé des décorateurs.

Cycle de vie

Comme nous l'avons dit de plus haut (cf. page 20), la principale difficulté des objets liés à une page web est la gestion de leur cycle de vie.

Le cycle de vie d'un objet peut donc être lié à différents contexte fournis par Weld :

- Le contexte de requête : la durée de vie de l'objet correspond à une requête web : l'objet est donc détruit après que la réponse ait été envoyée au browser, chaque nouvelle requête va donc instancier un nouvel objet. Une classe utilisant ce contexte devra utiliser l'annotation @RequestScoped
- Le contexte de session : La durée de vie de l'objet correspond au temps d'ouverture de la fenêtre du navigateur web (pour IE), ou de l'onglet (pour Firefox). Les objets sont stockés dans une Map. Ce contexte est typiquement utilisé pour gérer les données utilisateur, car ces données devront être conservées sur plusieurs requêtes, jusqu'à ce que l'utilisateur se déconnecte ou ferme le navigateur. Une classe utilisant ce contexte devra utiliser l'annotation @SessionScoped
- Le contexte conversationnel : Ce contexte permet de gérer de manière fine la durée de vie d'un objet. Ce contexte correspond par défaut au contexte de requête, mais une **conversation** peut être initiée dans le code de l'objet. Chaque conversation possède un identifiant unique et gère l'état des objets auxquels elle est liée. Une conversation permet donc de garder l'état d'un objet sur plusieurs requêtes. Une conversation peut être terminée programmatiquement, le contexte de l'objet redevenant un contexte de requête. Le contexte conversationnel trouve naturellement sa place dans le cadre d'interactions de type Ajax: l'objet doit être maintenu sur plusieurs requêtes, mais pas sur la durée d'une session[19] entière.

[18] Ces informations sont tirées de la documentation de Weld, disponible à l'adresse : http://docs.jboss.org/weld/reference/latest/en-US/html/ (Page consultée le 31 juillet 2011)

[19] D'ailleurs, l'implémentation du contexte de conversation et bâtie sur le contexte de session : lorsque la conversation démarre, les objets sont mis dans le contexte de session, lorsqu'elle s'arrête, les objets sont remis dans le contexte de requête

Malheureusement, pour des raisons qui n'ont pu être élucidées que tard dans le projet (mauvaise version du Jar dans le serveur), le contexte conversationnel ne peut pas être utilisé pour tous les classes, ce qui a considérablement compliqué le développement.

Heureusement, JSF fournit aussi un système de gestion de cycle de vie des objets, et propose un contexte de vue, idéal pour la gestion de l'Ajax. Ce contexte reste actif pour toute la durée de la vue; une fois que l'utilisateur change ou recharge la page, l'objet est détruit ou de nouveau instancié.

Passivation

La spécification de JSF exige que toute instance de classe liée à un contexte s'étendant au-delà d'une requête doit pouvoir être passivée. Cela signifie que le serveur peut décider à n'importe quel moment de sérialiser un objet afin de libérer de la mémoire, puis de le désérialiser au moment opportun, pour récupérer des informations. Malheureusement, ce mécanisme de sérialisation/désérialisation n'est pas pleinement fonctionnel pour certains contextes gérés par JSF. Typiquement, lors d'une liste contenant des objets avec des relations profondes, les liens entre certains objets risquent d'être brisés; la parade est alors soit d'utiliser un des contextes de WELD, mais comme dit plus haut, cela ne marche pas à tous les coups, ou alors de reconstruire l'état de l'objet à partir de la base de données entre chaque requête.

A noter que ce comportement n'est pas considéré comme un bug, mais vient directement de ce qui a été écrit dans la spécification. Ce comportement est décrit dans le suivi des problèmes de JSF 2.0, à l'adresse suivante : http://java.net/jira/browse/JAVASERVERFACES-1492

> « If a component binding is used with a View scoped bean, a new instance of the bean will be temporarily instantiated to receive the binding, but will then be replaced by the View scoped instance, causing the set binding value to be lost. »

Ce problème sera repris au point 5

3.1.9 Un exemple de page web réalisé avec JSF 2.0

La figure ci-dessous montre comment ces différentes technologies sont utilisées afin de créer une page web.

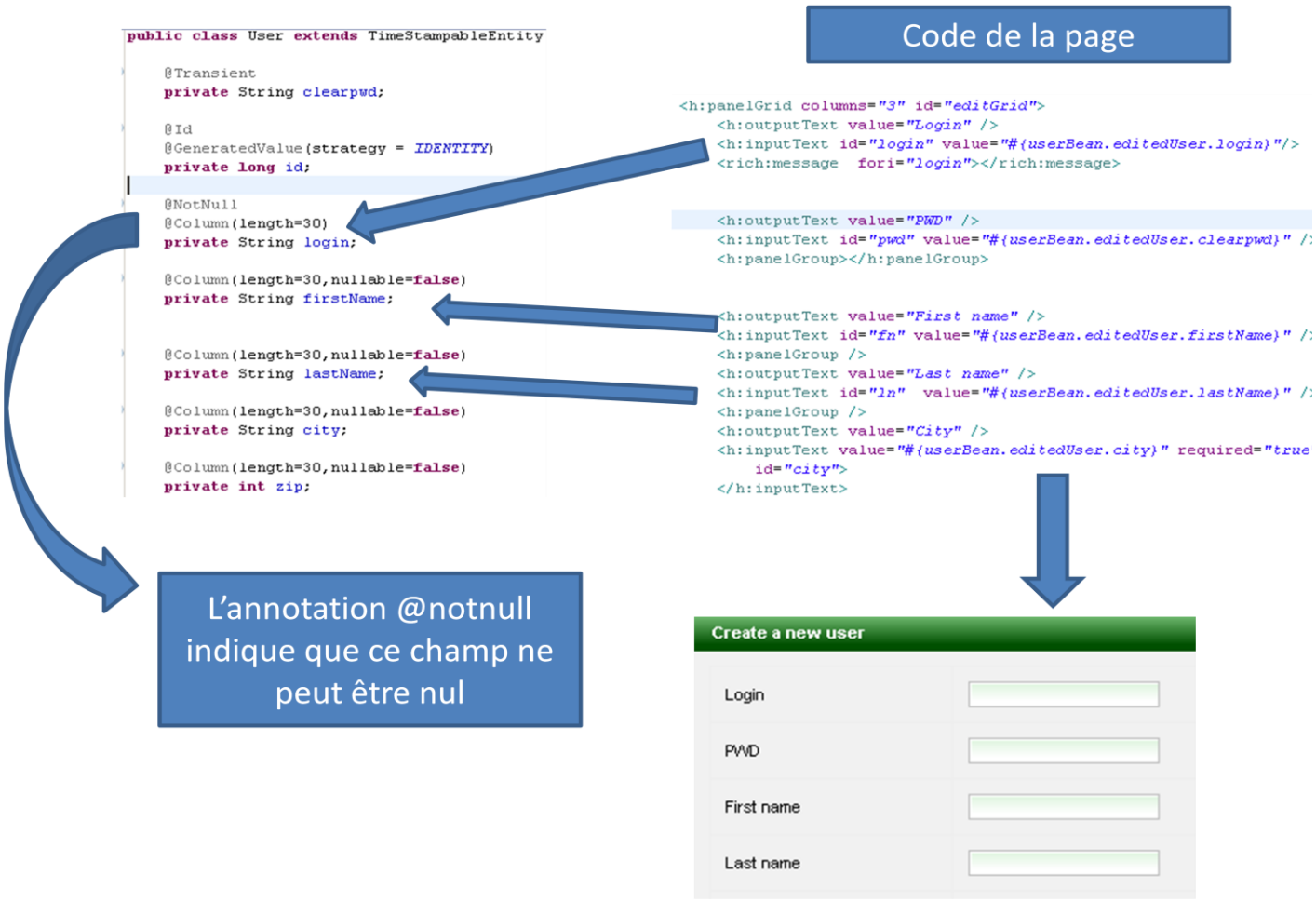


Figure 15 Code Java, code XHTML et formulaire web correspondant

Le code générant le formulaire de la figure 15 est situé en haut à droite ; les balises utilisées possèdent des noms suffisamment évocateurs pour que l'on puisse imaginer les composants qu'elles vont générer à l'écran. Nous pouvons remarquer que chaque balise générant un champ d'entrée texte possède un attribut de type value ; dans le cas du premier champ d'entrée on trouve : « value=''#{userBean.editedUser…..} '' », or userBean est un objet pris en charge par JSF, grâce à l'annotation @ManagedBean, comme le montre la figure ci-dessous :

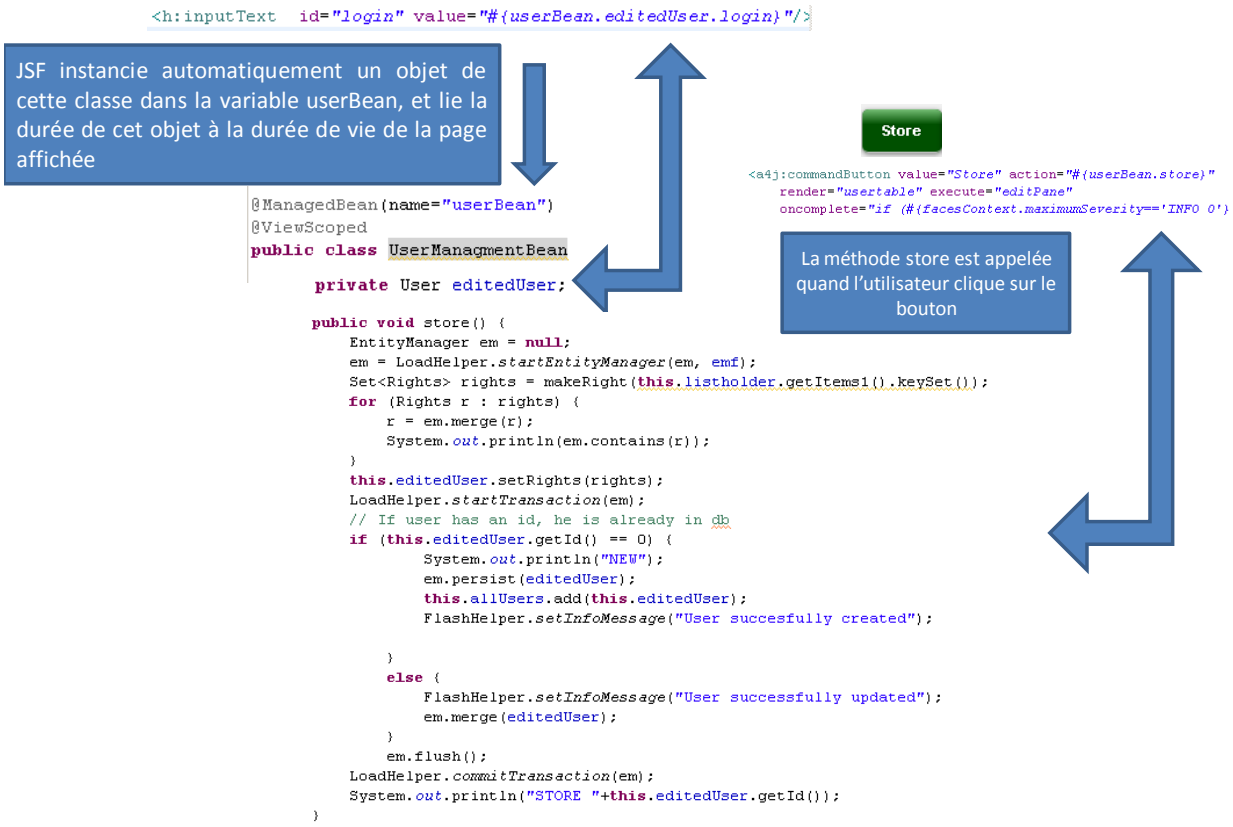


Figure 16 Managed Bean lié à un champ d'entrée text

Lorsque la page a besoin de l'objet userBean pour la première fois, JSF va l'instancier. Or cette classe possède un champ editedUser de type User ; ce champ correspond à l'utilisateur qui est en train d'être créé ou édité. La figure 15 montre les différents champs de la classe User, et nous voyons que cette classe possède un champ login. C'est à ce champ que correspond la valeur du premier champ de saisie de la page.

Lorsque l'utilisateur clique sur le bouton store, JSF va automatiquement mettre à jour les champs de l'objet User, et vérifier que les données soient valides ; si ce n'est pas le cas, il va afficher sur la page des messages d'erreur, correspondant aux problèmes rencontrés. JSF va automatiquement vérifier les contraintes posées sur les champs (cf. point 3.1.5). Si la validation passe, JSF va exécuter la méthode indiquée par l'attribut action de la balise correspondant au bouton. Cette méthode vérifie si le sujet est un nouvel utilisateur (dans ce cas il n'a pas d'id) ou un sujet déjà présent en base de données, et va enregistrer ou modifier cet utilisateur en base.

3.2 Technologies utilisées côté client

Le but final du serveur est de générer des pages web qui sont alors envoyées à l'utilisateur, donc du côté client. Les pages web envoyées se composent de codes HTML, CSS, et Javascript qui seront traités sur la machine cliente, par le navigateur web.

3.2.1 HTML

```
1  <html>
2  <head>
3  <title>Insert title here</title>
4  <style type="text/css">
5  </style>
6  </head>
7  <body>
8      <div id="encadre" style="border:14px solid;black;">
9      <h1>Ceci est un example</h1>
10     <p>
11         Contenu  ----
12     </p>
13     </div>
14 </body>
15 </html>
```

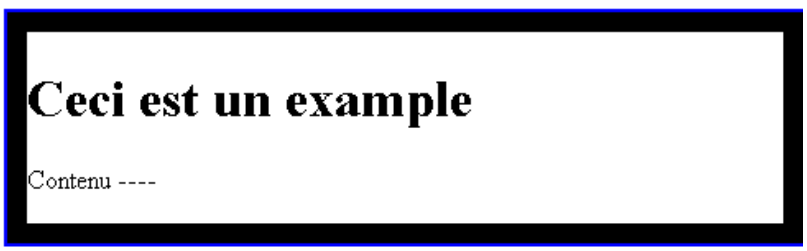


Figure 17 Code HTML

Le HTML, acronyme de HyperText Markup Language, est le format de données conçu pour représenter les pages web. Le HTML permet de décrire et structurer le contenu des pages. Le HTML est utilisé conjointement avec du JavaScript et des feuilles de styles en cascade (CSS). Un navigateur client va lire le code HTML et ses éléments associés pour ensuite le représenter dans sa fenêtre.

La figure ci-contre montre un exemple de document HTML et la page résultante. Les balises <h1> </h1> sont utilisées pour indiquer un titre, et les balises   un paragraphe. Le HTML devrait être utilisé uniquement pour exprimer la structure et la sémantique d'un document. La mise en forme d'un document s'effectue via le CSS

3.2.2 CSS

Dans la figure ci-dessus, la balise se voit attribuer un style CSS qui lui indique d'avoir une bordure noire de 14 pixels. Le style associé à cette balise est directement présent dans le document HTML

Le but du CSS est de permettre la mise en forme du document et de séparer la structure d'un document de ses styles de présentation. Le CSS permet d'associer différents styles aux différentes balises HTML. Les éléments relatifs à la mise en forme d'une page sont alors centralisés dans un même document : la feuille de style CSS.

3.2.3 JavaScript

Le JavaScript permet de manipuler le DOM d'une page HTML. Le DOM, Document Object Model est une recommandation du W3C qui décrit une interface indépendante de tout langage de programmation et de toute plate-forme. Cette interface permet de lire ou de modifier le contenu, la structure, ou le style de documents XML et HTML. Elle permet aussi de gérer les évènements clients sur la page HTML via l'utilisation d'écouteurs. Le JavaScript est donc utilisé pour gérer le comportement d'une page.

3.2.4 Compatibilité entre navigateurs

Bien que le CSS, le DOM, et le HTML soient issus des recommandations du W3C, un organisme indépendant, chaque navigateur gère ces standards de manière différente; ce qui

a donné lieu a de nombreux problèmes de compatibilités pour certains sites. Typiquement IE a pendant longtemps utilisé un modèle de boites différent de celui de FireFox, FireFox permettait la création de coins arrondis sur les boites, etc… La solution à ces problèmes a été, au lieu d'utiliser directement du JavaScript, d'utiliser uniquement des librairies prenant en charge ces problèmes de compatibilité.

JQuery

JQuery est une librairie simplifiant l'écriture de JavaScript. Elle libère le développeur du fardeau de la gestion des incompatibilités entre navigateur. De plus, le framework RichFaces a été prévu pour s'interfacer avec cette librairie.

Flot

Flot est une librairie javascript gérant la création de graphiques de manière simple et intuitive. D'autres librairies existent, parmi les plus connues :

- JFreeChart : Nous l'avions déjà utilisé dans un autre projet; le problème de cette librairie est qu'elle génère des **images.** Du coup, aucune interactivité n'est possible
- Google Chart Tools: Il s'agit d'une librairie très riche offrant de nombreuses options. Cette librairie est encore en évolution, et est légèrement plus compliquée à mettre en place que Flot. Pour cette raison, nous avons retenu Flot

3.2.5 Librairie RichFaces et JSF

RichFaces fournit plusieurs fonctions JavaScript permettant de déclencher et gérer des requêtes Ajax vers le serveur.

3.3 Ajax

Le livrable de ce travail étant une application web utilisant intensivement AJAX, un chapitre entier dédié à Ajax s'imposait[20].

3.3.1 Principes de base

Le terme Ajax a été introduit par Jesse James Garret, un expert web, en Février 2005; Ajax est un acronyme pour « Asynchronous Javascript and XMLHttpRequest » ; de manière plus générale, ce terme est utilisé pour désigner un ensemble de techniques visant à rendre les applications web plus réactives. Dans cette optique, la lettre clef dans Ajax est le A de asynchrone. Dans une application web classique, la seule façon d'envoyer des données au serveur est d'utiliser une soumission de formulaires, qui va alors bloquer le navigateur jusqu'à la réponse du serveur; les requêtes se font de manière synchrone, et il n'y a rien de fondamentalement incorrect à cette manière de faire. Le workflow typique d'une application web se résumait alors à :

1. Visiter une page web
2. Remplir un formulaire
3. Cliquer sur un bouton

[20] Deux ouvrages ont servis à l'élaboration de cette présentation :
[1, pages 339-341]
[5, pages 386-390

4. Attendre
5. Répéter la première étape

En 2005, Mr. Garret remarque qu'assez de gens possédaient assez de navigateurs suffisamment nouveaux, avec des postes suffisamment puissants pour les exécuter correctement, et que ce workflow pouvait être brisé. Les trois capacités importantes que les navigateurs suffisamment nouveaux devaient posséder étaient : le JavaScript, le DOM, et l'objet XMLHttpRequest. Nous avons déjà parlé du DOM et du JavaScript, l'objet XMLHttpRequest correspond à un ensemble de fonctions qu'un script JS peut appeler pour effectuer des requêtes http synchrones ou asynchrones, sans effectuer de soumission de formulaires. En utilisant les capacités de cette objet, nous pouvons effectuer des requêtes asynchrones qui ne bloqueront pas le navigateur, puis updater le DOM, et donc l'affichage du navigateur, avec la réponse du serveur. Le workflow traditionnel se retrouve transformé ainsi:

1. Visiter une page web
2. Remplir des formulaires
3. Le navigateur va envoyer automatiquement des données au serveur qui va effectuer des tâches correspondantes
4. Le navigateur met à jour certaines parties de l'affichage, sans recharger la page
5. Retour au point 2

Nous pouvons constater qu'Ajax n'est pas en soi une technologie, mais un ensemble de techniques s'appuyant sur diverses technologies, qui, quand elles sont utilisées correctement, permettent aux développeurs de fournir une expérience utilisateur plus réactive, avec moins de temps d'attente.

Ajax a tout d'abord été considéré comme un luxe, tant du point de vue du développeur que de l'utilisateur, mais s'est rapidement imposé[21] et a fini par devenir essentiel au développement d'application web compétitive.

3.3.2 Un exemple typique

Pour mieux comprendre Ajax, nous allons utiliser un exemple typique de son utilisation, indépendamment de toute technologie sous-jacente. La validation des entrées de l'utilisateur est un cas typique d'utilisation d'Ajax. La figure ci-dessous décrit les différentes étapes d'une requête Ajax:

1. L'utilisateur frappe la barre d'espace et déclenche un événement clavier sur son navigateur.
2. Une requête Ajax part sur le serveur avec le contenu du champ "Name"
3. Le serveur vérifie le contenu du champ, s'il est vide, il renvoie un message d'erreur en code HTML
4. Le serveur envoie la réponse au navigateur

[21] Les différents éléments permettant d''utiliser Ajax était déjà en place depuis en 1999 ; mais c'est l'apparition d'une « killer app » qui a permis à Ajax de s'imposer ; cette application était Google Maps, offerte par Google au début de l'année 2005

5. Le navigateur reçoit la réponse et modifie l'affichage de manière appropriée. Dans notre cas, il affiche un message d'erreur

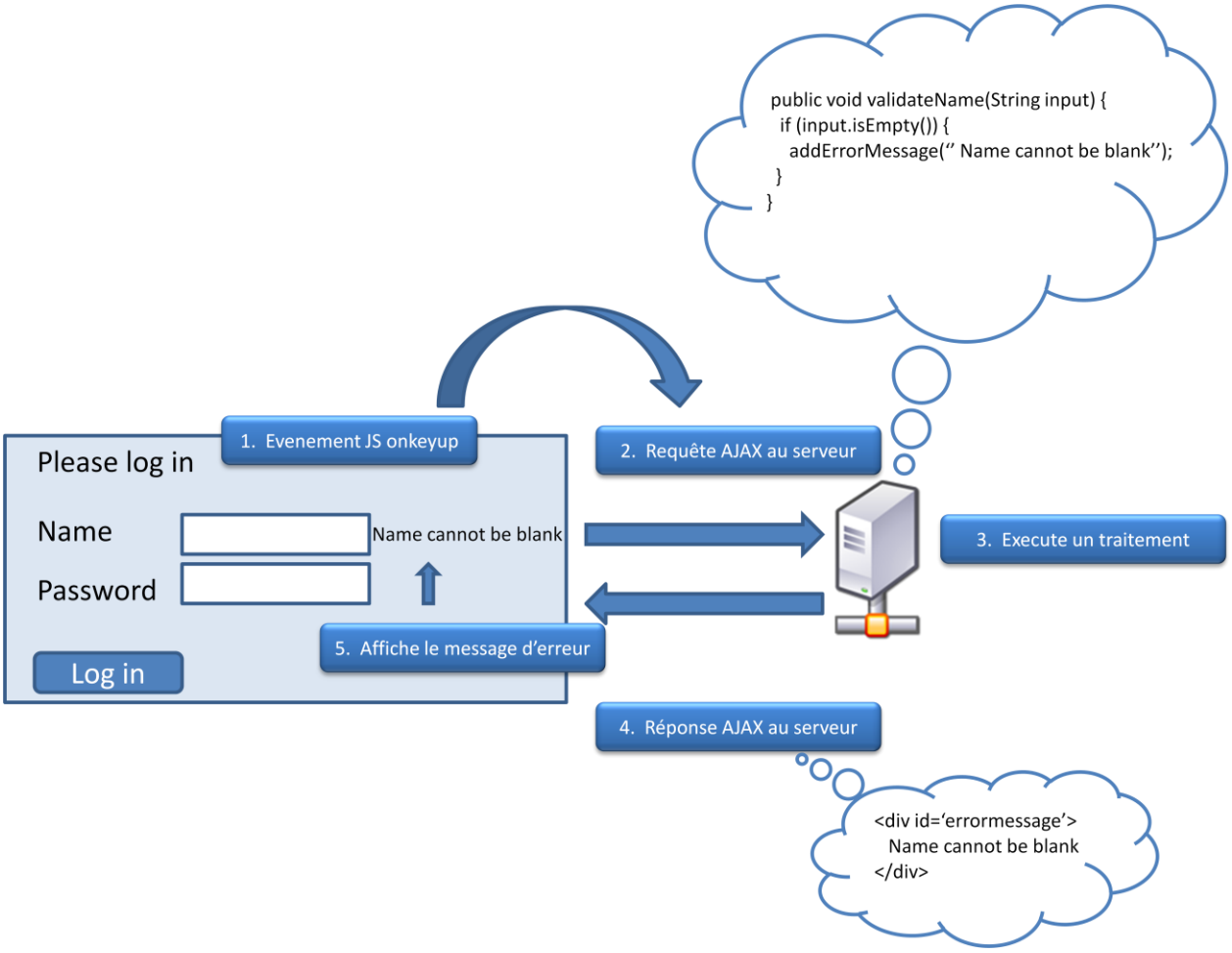


Figure 18 Fonctionnement d’une requête Ajax

3.3.3 Ajax et Richfaces

RichFaces possède une librairie de composants gérant directement l'Ajax. Dans le cas d'usage décrit ci-dessus, il suffit d'attacher la balise <rich:message for= ‘’ ‘’> à l'élément devant être validé grâce à l’attribut for.

La gestion de l'Ajax se fait de manière très simple en indiquant à un composant quelles interactions utilisateurs vont déclencher une requête Ajax, quelle méthode doit être appelée du côté du serveur, et quels éléments doivent être mis à jour sur la page HTML

3.4 Outils de développement

3.4.1 IDE utilisé

Nous avons choisi d'utiliser Eclipse, un environnement de développement gratuit, qui dispose de nombreuses extensions. Eclipse permet le déploiement local et à distance d'applications sur de nombreux types de serveurs.

Data Source Explorer

Une des extensions très utile d'Eclipse est le Data Source Explorer, une extension permettant d'accéder facilement à une base de données, en utilisant le driver java correspondant au type

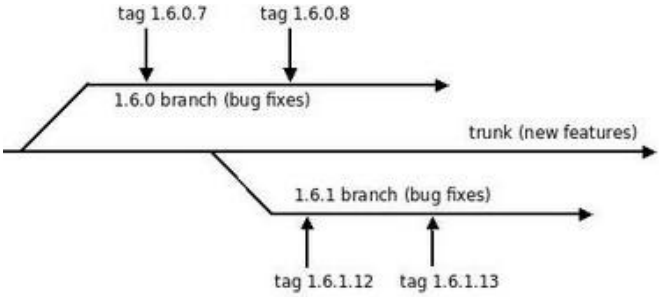
de la base de données. Cette extension permet de lancer des requêtes SQL, d'exporter des données, et d'éditer très simplement les tables de la base de données. Cette extension s'intègre de façon transparente dans Eclipse et facilite grandement le développement de base de données.

3.4.2 Autres IDE

D'autres IDE existent, comme NetBeans, IntelliJ, ou Visual Studio. Tous ces IDE proposant plus ou moins les mêmes fonctionnalités, le critère déterminant devient alors l'habitude que l'utilisateur possède avec tel ou tel IDE.

3.4.3 Git

Git est un système de gestion de versions. Sans rentrer dans les détails, il permet de gérer les différentes versions du code source. Pour ce faire, Git crée une base de données (repository) sur la machine locale qui va stocker toutes les modifications effectuées sur chaque fichier source, ce qui permet d'annuler toute modification du code qui introduirait des bogues.



Source :en-wikiedia.org

Figure 19 Arbre de versions

Les différentes versions peuvent être affichées de manière graphique : on parle alors d'arbre de versions. Un arbre de versions permet d'analyser de manière rapide les changements effectués sur le code source. La figure ci-contre donne un exemple d'arbre de versions.

Le véritable intérêt d'un système de gestions de versions apparaît quand le repository est stocké sur une machine distante : d'une part, on dispose d'une copie de sécurité, et d'autre part, lors du travail en équipe, tous les changements sont stockés sur une même machine. Internet propose différents sites offrant ce service; nous avons choisi GitHub.

3.4.4 GitHub

GitHub est un service web d'hébergement et de gestion de développement de logiciels, utilisant le programme Git. Le service est gratuit pour de petits projets. De plus GitHub permet de créer un wiki, un suivi des bogues, et un suivi global du projet. Il crée automatiquement une série de graphiques, comme par exemple l'arbre de versions. La figure ci-contre représente l'arbre de versions de notre projet, qui se résume à une simple ligne droite, car il n'y a qu'un seul développeur, et nous n'avons pas voulu créer de branches, pour ne pas compliquer inutilement les choses.

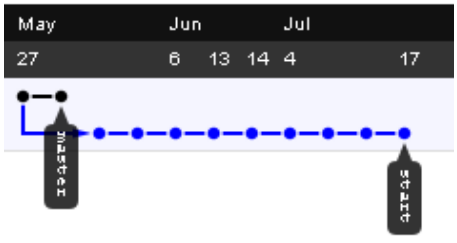


Figure 20 Arbre du projet

3.5 Autres technologies existantes

3.5.1 Google Web Toolkit (GWT)

GWT est un framework orienté composant et client. La spécificité de GWT est que le développeur code sa page en Java, qui est alors compilée en une page web classique avec du HTML, CSS et JavaScript. **Il n'y a donc pas de représentation côté serveur** de l'état de la page. Le serveur est alors entièrement dédié aux traitements des données. GWT reprend le slogan de Java: "Write Once, Run EveryWhere". En effet, comme le montre la figure ci-dessous, le compilateur GWT va produire du code JavaScript pour chaque navigateur web.

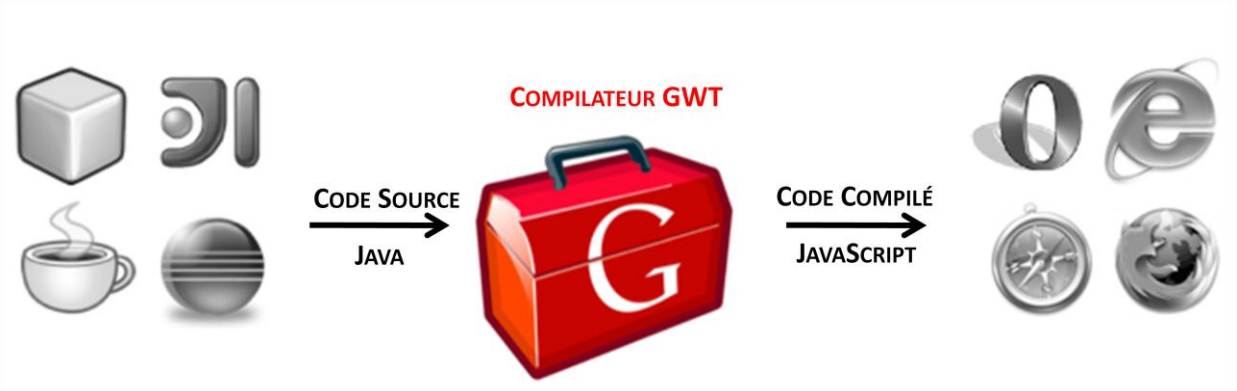


Figure 21 Fonctionnement de GWT

Un autre avantage de GWT est qu'il peut être utilisé conjointement avec Google AppEngine, qui permet alors de déployer l'application sur le Cloud de Google. Malheureusement, Google ne permet pas d'utiliser certaines classes[22] et certains aspects de JPA (relation N-à-N). Notre application utilisant ces fonctionnalités, cet avantage tombe à l'eau. De plus, GWT, à l'époque où nous l'avons utilisé dans un projet, fonctionnait mal avec JPA .

3.5.2 C# / .Net

Microsoft propose une solution complète intégrant un environnement de développement, un langage de programmation, C#, un ensemble de librairies, et une base de données. Tous ces outils possèdent une large base de connaissances et de nombreux forums traitent des problèmes rencontrés. Malheureusement, cette solution Microsoft est payante[23]. Le but de ce projet étant de développer une solution utilisant des outils gratuits, Microsoft se trouve de facto éliminé.

Bien sûr, il existe toujours la solution Mono, un portage Linux de cette suite d'outils. Mais le développement sur Mono reste très difficile (beaucoup de bogues, peu de documentations, outils de développement rudimentaires…)

[22] Google a publié une White List, la liste des classes permise, à l'adresse suivante : http://code.google.com/intl/fr-CH/appengine/docs/java/jrewhitelist.html (Page consultée le 15 juin 2011)

[23] Après quelques recherches sur le net, nous avons constaté que l'environnement de développement à lui seul coute autour de 1000 Euros ; auquel il faut encore rajouter le prix d'un serveur IIS et de la base de données.

3.5.3 Ruby On Rail

Ruby on Rails (Rails) est un framework orienté action destiné au développement web qui s'appuie sur le langage Ruby. Rails, et la communauté qui l'entourent, sont très particuliers à plus d'un titre:

- La plupart des ses utilisateurs codent avec des outils très simples (éditeur texte)
- Rails ne proposent qu'une seule façon "correcte" de faire les choses. S'éloigner de ce chemin rend le développement impossible
- Rails abstrait totalement l'utilisateur de la base de données
- Rails force le développeur à écrire les tests avant de coder
- Le langage Ruby présente de nombreuses particularités comme la génération dynamique de code à la volée.

Tous ces points sont résumés dans la préface de l'ouvrage de référence sur Rails [4, page 3]:

> From the beginning, the Rails turned web development on its head with the insight that the vast majority of time spent on projects amounted to meaningless spin-up. […] **By making decisions for you, Rails frees you to kick off your project with a bang** […]
>
> Rails makes some simple decisions for you […]
>
> Rails is more than a programming framework […] it's also a framework for thinking about web applications. It ships not as a blank slate equally tolerant of every kind of expression. On the contrary […] **it's a designer straightjacket** that sets you free from focusing on the things that just don't matter […] only by understanding the why will you be able to consistently work with the framework instead of against it…

Développer cette application sur Rails aurait été une expérience intéressante, mais développer un projet de cette envergure sur un langage et un framework inconnus nous semblait une prise de risque trop importante.

3.5.4 PHP et MySql

La plateforme LAMP (Linux, Apache, PHP, MySql) et ses nombreuses variantes (WAMP, MAMP, …) est une plateforme couramment utilisée pour la création de sites web. Cette plateforme est constituée d'une suite d'outils open-source; une large communauté utilise cette plate-forme, et de nombreux forums documentent la création de sites web en PHP. Malheureusement, réaliser des sites Ajax en PHP reste relativement difficile, et demande l'écriture de code JavaScript. JSF 2.0 et RichFaces génèrent automatiquement le code JavaScript nécessaire à la bonne exécution des requêtes de type Ajax, ce qui offre un gain de temps important.

De plus, PHP ne force pas le développeur à utiliser un paradigme objet, et permet de mélanger le code PHP directement dans le code HTML, ce qui produit du code peu lisible. Le site http://shootout.alioth.debian.org/u32/benchmark.php?test=all&lang=java&lang2=php propose un benchmark pour mesurer les performances du PHP via celles de Java. Le tableau ci-dessous résume les performances de Java par rapport à PHP :

Benchmark	Temps	Mémoire	Code
Binary-trees	1/71	1/3	=
Mandelbrot	1/65	22X	2X
Fannkuck-redux	1/62	5X	3X
N-Body	1/37	5x	=
Spectral Norm	1/35	=	=
Fasta	1/26	6X	=
K-nucleotide	1/14	2X	2X
Reverse-complement	1/2	1/2X	5X
Regex-dna	1/2	3X	3X
Pidigits		4X	2X

Tableau 4 Benchmark Java PHP

Java est énormément plus rapide que PHP dans tous les cas, au prix d'une consommation mémoire supérieure et de plus de lignes de codes. Cependant, la consommation supplémentaire est négligeable face aux gains de performances.

3.6 Outils de test

3.6.1 Selenium

Selenium est une suite d'outils permettant de réaliser des tests de type boite noire. Il s'agit d'une méthodologie de test qui va uniquement tester les fonctionnalités de l'application, sans tester son fonctionnement interne. Typiquement, dans le cas d'une application web, il s'agira de simuler des interactions utilisateur sur une page donnée, puis de vérifier les changements survenus sur la page en question. Si les changements correspondent à ce qui est attendu, le test est réussi. De la même façon, on peut aller vérifier dans la base de données si les changements attendus sont survenus. Si tel est le cas, le test est réussi. La figure ci-dessous résume cette procédure de test.

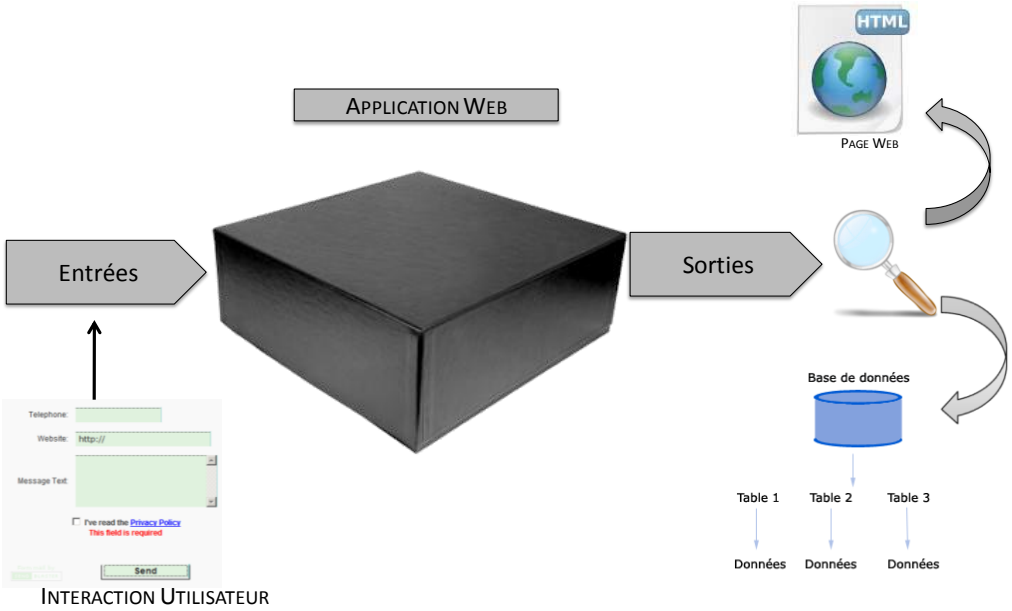


Figure 22 Tests de type boite noire

Le serveur est alors considéré comme une boite noire, car on ne sait rien de ce qui se passe à l'intérieur. Si un test a échoué, aucune indication des problèmes survenus dans le code sera donné; le développeur ne pourra qu'essayer de deviner où les problèmes sont survenus.

Selenium permet de scripter des interactions utilisateur sur un navigateur web dans la majorité des langages du marché. Un tutoriel décrivant le fonctionnement de Selenium, et ses différentes caractéristiques est disponible en annexe. Dans ce projet, Selenium a été utilisé pour vérifier le bon fonctionnement des pages de créations de sujets, d'utilisateurs, et d'acquisitions. Un script simulant la saisie d'informations sur le formulaire est exécuté, puis le test vérifie si les informations saisies sont bien présentes en base de données, et si les informations affichées sur la page sont correctes.

Une des difficultés rencontrées était le test d'upload des images relatives aux acquisitions. Le code pour simuler un upload de fichier semblait impossible à écrire. L'IDE de Selenium est alors venu à la rescousse : cet IDE permet d'enregistrer les interactions souris et clavier sur une page web, puis de produire le code correspondant. En utilisant l'IDE, nous avons pu découvrir le code nécessaire à la simulation d'un upload.

3.6.2 JSFUNIT

Le principal problème des tests « boite noire » est qu'ils ne donnent aucune indication quant au fonctionnement interne de l'application. Il faut alors tester directement dans le conteneur de l'application; on parle alors de tests "boite blanche". Ces tests permettent de tester directement les classes présentes dans le serveur.

JSFUNIT est une librairie permettant d'écrire des tests pour le framework JSF ; ces tests étant réalisés directement dans le conteneur de l'application. Malheureusement, elle nécessite un grand nombre de dépendances, et un grand effort de configuration. Après deux jours passés à tenter de mettre en place des tests avec JSFUNIT, nous avons abandonné l'idée d'utiliser ce framework de test.

JSFUNIT étant encore en version bêta, il faudra encore attendre quelque temps avant d'avoir une version utilisable.

4 Architecture de l'application

4.1 Architecture globale

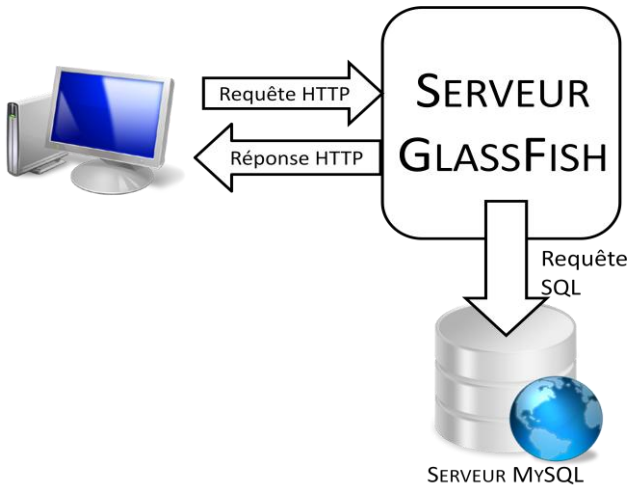


Figure 23 Architecture globale

Au plus haut niveau, notre application possède une architecture classique trois-tiers. Un serveur applicatif GlassFish intercepte les requêtes HTTP, les traite, et exécute les requêtes SQL appropriées vers le serveur de données, en l'occurrence un serveur MySql. Une réponse, sous forme de pages HTML, et alors envoyée au client. A chaque interaction du client, le cycle requête/réponse recommence.

4.2 Structure du code

4.2.1 Modèle

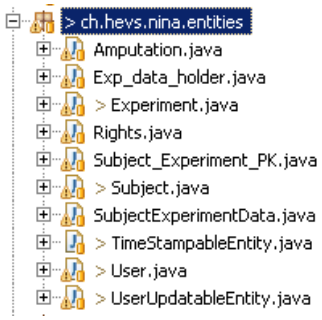


Figure 24 Entités

Les différentes classes métiers ont été regroupées dans un même package. Ces classes gèrent les objets "métiers" de l'application : sujets, utilisateurs, expériences. Ces classes sont prises en charge par JPA qui leurs offre toute une palette de services, qui ont été évoqués plus haut (voire point 3.1.5). La plupart de ces class étendent les classes TimeStampableEntity et UserUpdatableEntity. Ces deux classes prennent en charge la journalisation des évènements en stockant les heures de création et de mise en jour, et le nom de l'utilisateur ayant effectué la mise à jour ou la création de l'objet

A titre de rappel, la figure 11 montre comment s'effectue la journalisation des évènements. Des méthodes sont annotées avec @PrePersist et @PreUpdate, ce qui fait qu'elles sont automatiquement appelées avant que l'objet soit persisté/updaté en base de données.

4.2.2 « Managed Beans »

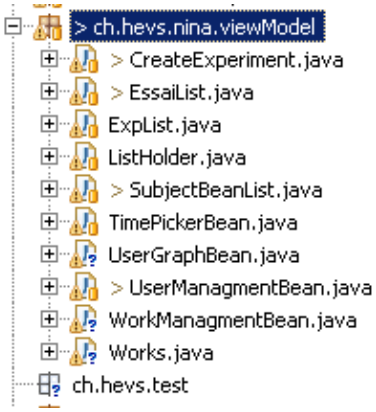


Figure 25 View beans

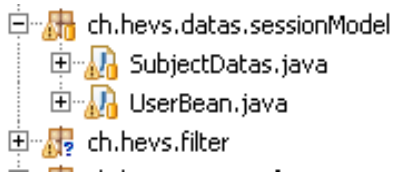


Figure 26 Session beans

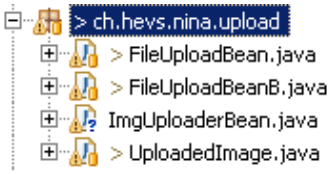


Figure 27 Upload beans

Il s'agit de classes Java dont les instances sont gérées par le conteneur. Leur cycle de vie est défini par l'usage d'annotations (voire point 3.1.8). Ces classes se répartissent dans quatre packages. Dans le package ch.hevs.nina.viewModel sont stockées les classes gérant la génération des pages HTML. Ces classes jouent le rôle de « présenteur »: elles chargent les données de la base, et les rendent accessible à la vue. Ces classes gèrent aussi le comportement des composants présents sur la page web. La plupart de ces classes sont annotées avec l'annotation @ViewScoped, car leur durée de vie correspond à une vue HTML.

Le package ch.hevs.datas.sessionModel contient les classes dont la durée de vie des instances correspond à une session utilisateur. On trouvera donc naturellement une classe gérant le login/logout des utilisateurs. Une session se termine lorsque l'utilisateur clique sur le bougon logout ou ferme le navigateur.

Le package ch.hevs.nina.upload regroupe, pour des raisons de commodité, les classes gérant l'upload de fichiers.

Le package ch.hevs.views.sorting contient les classes gérant le tri. Le fait d'avoir découplé le code gérant le tri du code gérant les vues permet de réutiliser ces classes dans différentes vues.

4.2.3 Classes « Helper »

Plusieurs packages contiennent des classes « Helper » : il s'agit de classes qui sont, pour l'essentiel, constituée de méthodes statiques « utilitaires » : création, effacement de dossier/fichiers, accès à des éléments d'une page, création de fichiers XML, etc...

4.2.4 Vues

JSF propose un langage à balises similaire à l'HTML pour construire des pages web. Ce langage permet de créer très facilement des templates. La figure ci-dessous montre le code correspondant à la page qui affiche la liste des sujets. La balise <ui:composition> indique au serveur d'utiliser le template masterLayout.xhtml. Ce template comprend deux zones qui pourront être définies dans la page appelée. De plus, la balise <ui:include> permet d'inclure d'autres pages dans une page, ce qui permet de scinder une longue page en plusieurs fichiers, afin de rendre la maintenance de l'application plus aisée.

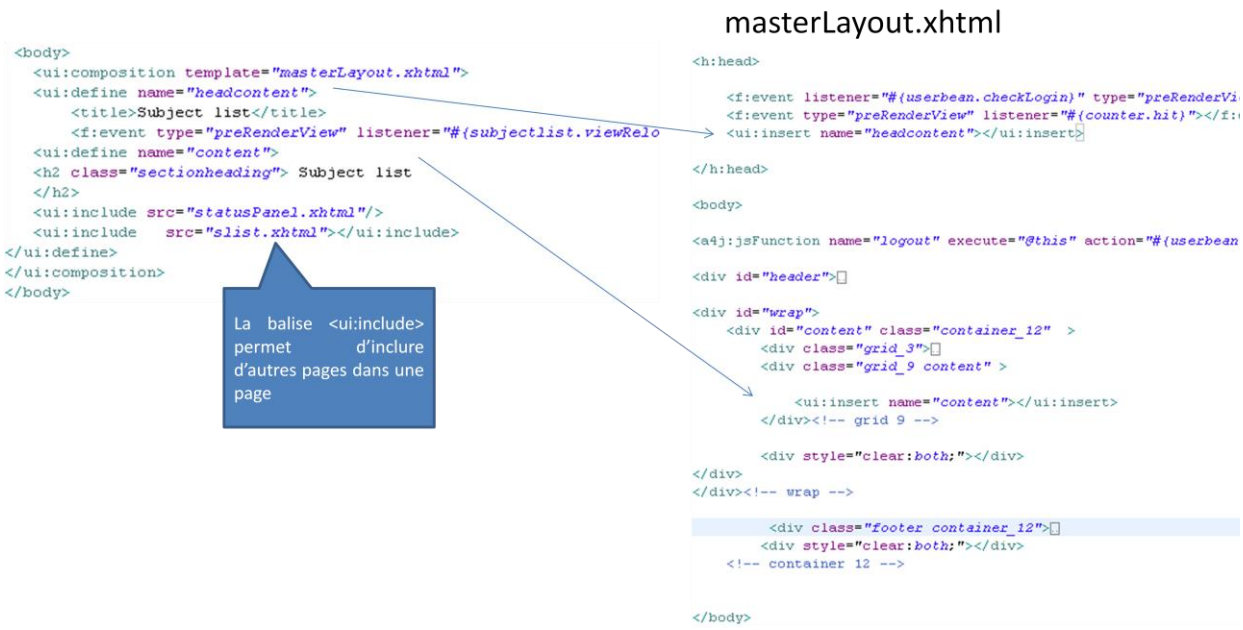


Figure 28 Code de vue inclus dans un layout

4.2.5 Code JavaScript

Différent scripts JS sont stockés dans le dossier JS. Ces scripts sont nécessaires pour la bonne exécution et la réception des requêtes et réponses Ajax. Ces scripts sont aussi utilisés pour la création dynamique de graphiques. Ces graphiques sont créés directement par le navigateur de l'utilisateur final; le serveur se contentant de fournir les données nécessaires à leur création. Nous détaillerons la création de ces graphiques au point 6.1.9

4.3 Modèle physique des données

Le modèle physique de données est disponible en annexe J.

4.4 Stockage des fichiers et images

Les fichiers et les images ne sont pas stockés en base de données, mais dans des dossiers dédiés sur le serveur. Le nom des fichiers est stocké en base, ce qui permet de retrouver les fichiers correspondants.

Stocker ou non des fichiers en base de données est un débat qui agite encore les forums aujourd'hui. Bill Karwin, dans son livre SQL Antipattern [10, page 143-146], avance les raisons suivantes pour stocker des fichiers hors de la base de données :

> There are good reasons to store images or other large objects in files outside the database:
>
> • The database is much leaner without images, because images tend to be large compared to simple datatypes like integers and strings.
>
> • If images are in external files, it's easier to do image previewing or editing.
>
> If these advantages of storing images in files are important and the issues described earlier are not deal-breakers, you may decide that it's the right thing to do in this project.

Bill Karwin rajoute que le stockage de fichiers directement dans la base de données prend son sens dans le cas de fichiers légers, ce qui n'est pas le cas de notre projet : les fichiers zips contenant les données sont lourds (5 MB.) ; tous ces éléments justifient notre choix de ne pas stocker les fichiers dans la base de données.

5 Résultat

L'application NinaWeb, accessible à l'adresse http://153.109.124.99:8080/NinaWeb/ , est le résultat de la mise en œuvre des différentes technologies citées dans les points précédents et de treize semaines de travail.

5.1 Login et menu principal

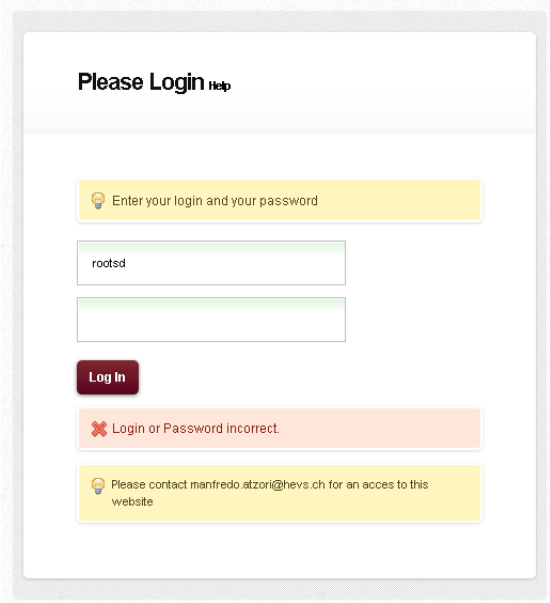


Figure 29 Panneau de login

L'accès au site se fait via un panneau de login. En cas de login incorrect, un message d'erreur s'affiche (cf. figure ci-contre). Une fois le login réussi, l'utilisateur est redirigé sur le menu principal. Les éléments du menu principal varient en fonction des droits de l'utilisateur (cf. figure ci-dessous)

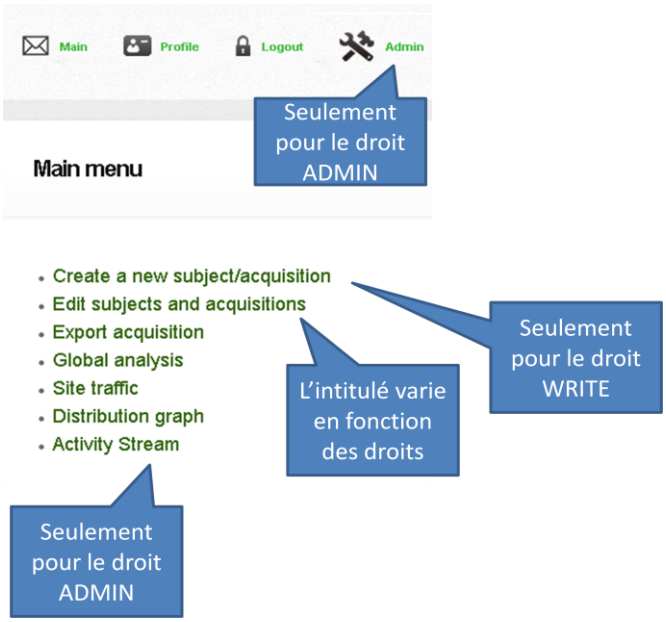


Figure 30 Menu principal

5.2 Gestion des utilisateurs et administration

Un utilisateur peut posséder jusqu'à trois droits différents : Lecture, Ecriture, Administration. Ces droits correspondent aux actions qu'un utilisateur peut entreprendre sur la base de données. L'interface du menu principal va varier en fonction des droits possédés par un utilisateur. Tout utilisateur non-enregistré est automatiquement redirigé sur la page de login.

Un utilisateur tentant d'accéder à une page pour laquelle il ne possède pas les droits nécessaires sera redirigé sur le menu principal.

Au niveau de la base de données, le mot de passe est haché avec l'algorithme SHA2. Pour augmenter la sécurité, un "sel" est généré aléatoirement, haché et stocké en base de données. Cette technique augmente l'inviolabilité du mot de passe.

Une interface est disponible pour les utilisateurs possédant le droit "ADMIN". Cette interface permet de créer, éditer, effacer et bloquer des utilisateurs. Les différents utilisateurs sont accessibles via une grille (cf. figure ci-dessous), disposant de diverses fonctions de tris ascendants ou descendants et de filtres ; les icones permettent de modifier/effacer/bloquer les utilisateurs.

Figure 31 Liste des utilisateurs

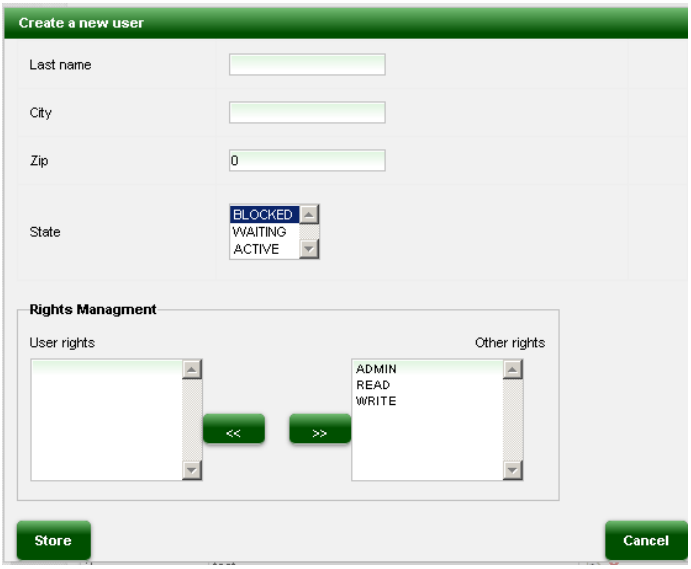


Figure 32 Création d'utilisateur

La création/modification d'un utilisateur se fait via un panneau permettant d'éditer les données de l'utilisateur, et de lui assigner différents droits ; un système de validations empêche de saisir un login qui soit vide, où déjà existant. L'état de l'utilisateur est géré via une liste déroulante. Lorsque l'utilisateur clique sur le bouton Store, un message de succès ou d'erreur s'affiche

Deux autres boutons sont disponibles sur cette interface, le bouton « Reset DB » permet de vider la base de données et de recréer les données sur le serveur ; le bouton « Dump DB » effectue un dump SQL de la base de données.

Cette interface permet aussi de gérer le contenu de la liste déroulante « Work » via une grille.

5.2.1 Gestion du profil utilisateur

Figure 33 Profil utilisateur

A tout moment, l'utilisateur peut modifier son profil, via une interface simple : l'utilisateur n'a qu'à effectuer les modifications dans les champs appropriés, puis cliquer sur le bouton « Update ». Un message indiquant que l'utilisateur a été mis à jour s'affiche alors.

5.3 Saisie des données

La saisie des données se fait sur un seul formulaire que l'utilisateur doit remplir. Ce formulaire vérifie que le sujet en cours de création ne soit pas déjà présent dans la base de données. Le cas échéant, il avertit l'utilisateur et propose de mettre à jour les champs du formulaire via un clic sur un lien (voir figure page suivante)

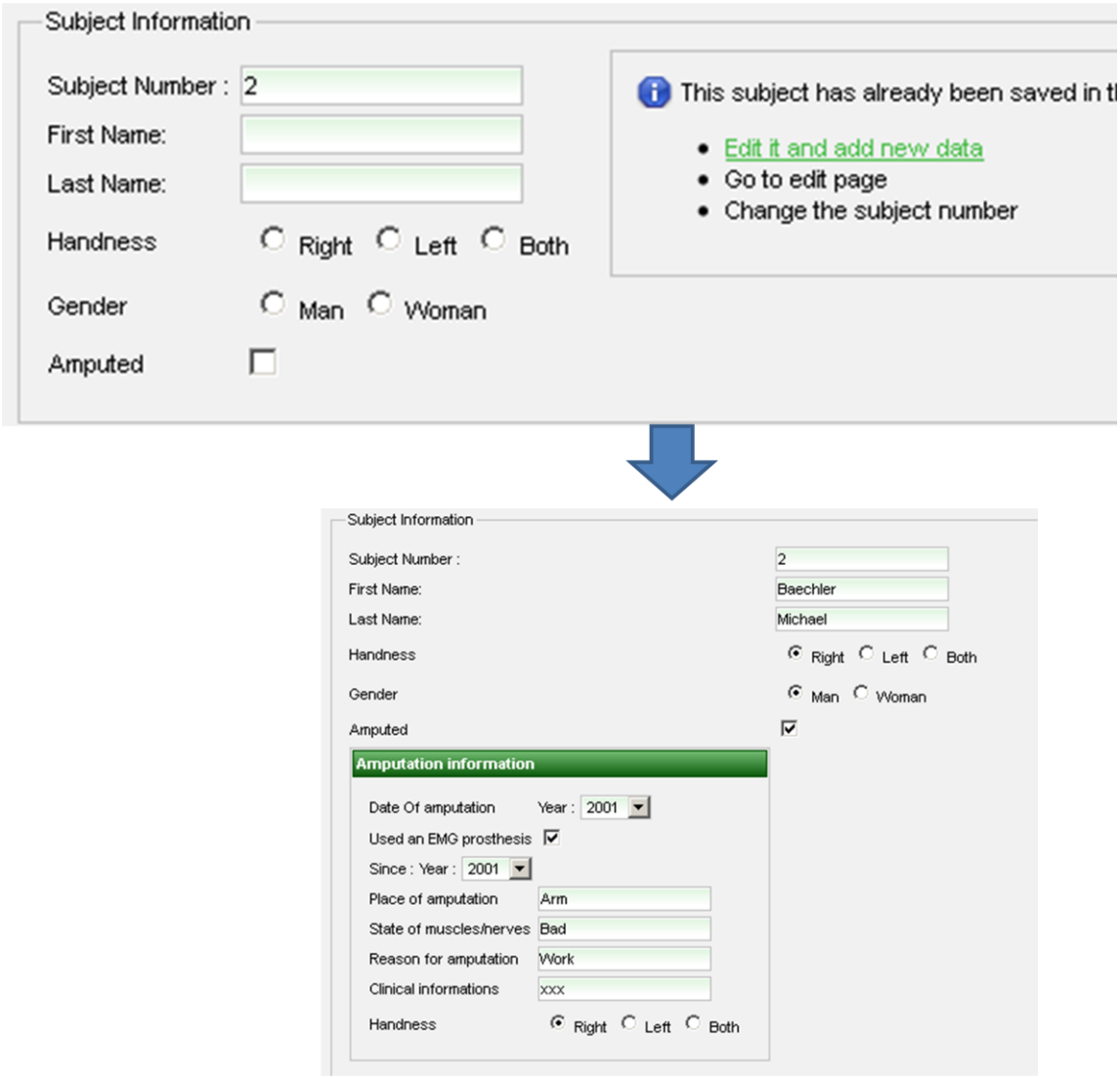


Figure 34 Formulaire de saisie avant et après clic

L'acquisition des données se fait en deux temps : l'utilisateur doit d'abord rentrer les données cliniques relatives au sujet et à l'acquisition, puis cliquer sur un bouton ; une fois ceci fait, un message de succès est affiché et l'utilisateur peut commencer à uploader les fichiers contenant les données des électromyogrammes (voire figure 35). Le serveur va vérifier que le nom des fichiers soit correct (le nom du fichier doit comporter le numéro du sujet et le numéro de l'acquisition et doit être un zip), et que ces fichiers ne soient pas déjà présents sur le serveur. Un message d'erreur sera affiché le cas échéant. Si l'upload s'est déroulé correctement, le serveur va afficher des informations sur le fichier uploadé (taille, contenu).L'utilisateur peut toujours effacer le fichier téléchargé via le bouton « clear files ».

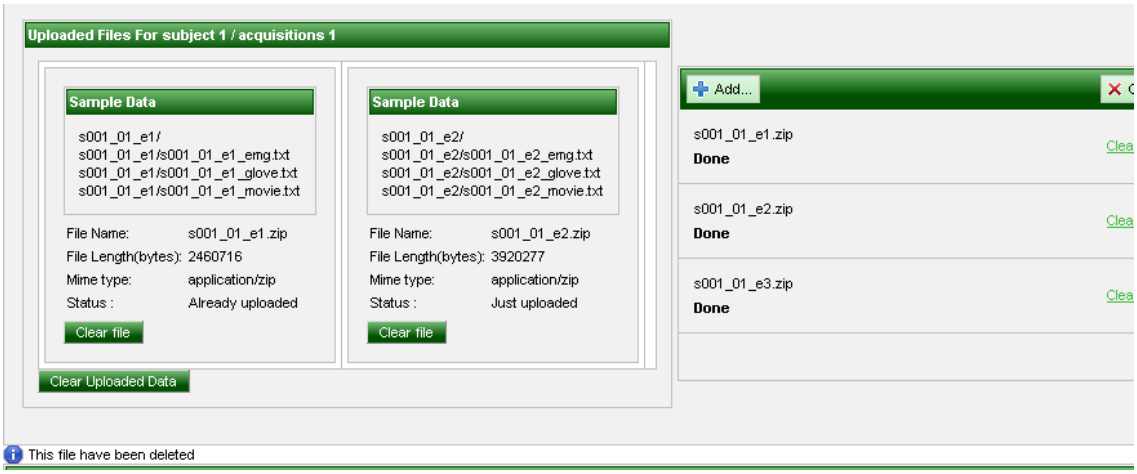


Figure 35 Upload des fichiers

Une fois les données uploadées, l'utilisateur peut revenir sur le menu principal, ou remettre à zéro les champs pour saisir une nouvelle acquisition. L'upload des images se fait de manière similaire.

5.4 Visualisation/Modification des données

Une grille permet d'afficher les différents sujets et leurs acquisitions.

#	Subject Number	First Name	Last Name	Gender	Amputed	All fields	Experiment	Action
		e		Both	Both	Both		
0	100	Georges2	Fake24	M				
1	2	Baechler	Michael	M				
2	1	Manfredo	Atzori	M				

#	Acq. Number	Location	Date	Humidity	Temperature	Action
0	1		Thu Aug 11 00:00:53 GMT+01:00 2011			
1	12	sirre	Tue Aug 02 00:00:00 GMT+01:00 2011			

Figure 36 Liste des sujets/acquisitions

Différentes fonctions de tris sont proposées (ascendants/descendants) ; ces tris sont accessibles en cliquant sur les en-têtes des colonnes ; de plus, plusieurs filtres sont proposés. Ces filtres sont activés en utilisant les listes déroulantes présentes dans les en-têtes des colonnes ou en tapant du texte dans les champs de saisie des en-têtes de colonne. Dans la figure 36, seuls les sujets ayant la lettre e dans leur prénom sont affichés.

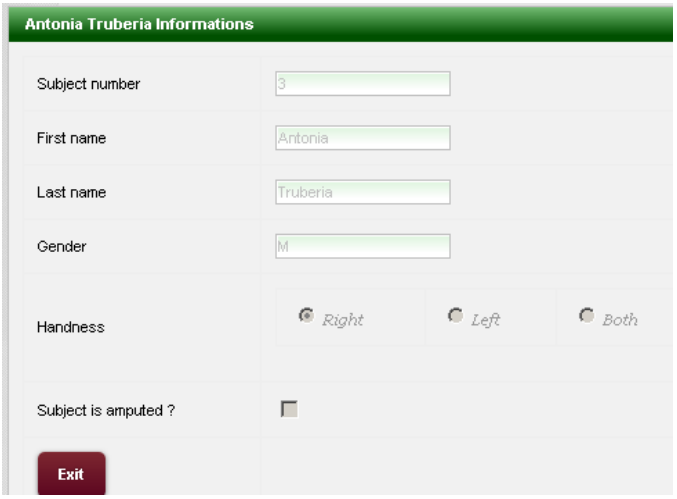


Figure 37 Edition/Visualisation d'un sujet

Les icones présentes dans cette grille permettent d'éditer les données cliniques des sujets et de leurs acquisitions, de supprimer un sujet ou une acquisition, de créer une nouvelle acquisition, d'uploader des images ou des données d'électromyogramme ; si l'utilisateur ne possède pas de droits d'écriture, certains de ces icones n'apparaitront pas, et l'utilisateur ne pourra pas accéder aux champs d'édition, et le bouton d'enregistrement lui sera caché, comme le montre la figure ci-contre. De même, les colonnes « First Name » et « Last Name » n'apparaitront pas si l'utilisateur ne possède que le droit READ. Les différents panneaux d'éditions sont similaires à la figure 37.

La dernière icône permet de générer un rapport PDF pour une acquisition. Un exemple de rapport PDF est disponible à l'annexe F.

5.5 Export des données

L'export des données se fait via une grille affichant les différentes acquisitions.

#	Subject	Acquisition	Amputed	Files	All fields filled	Subject Name	Location	Date less than
			Both					
0	100	12	☑	0	☐	Georges2	sirre	2011-7-2
1	2	1	☑	0	☐	Baechler	sirre	2011-7-2
2	3	1	☐	1	☐	Antonia	sierre	2011-7-2
3	4	1	☐	1	☐	Siriam prasath	idiap2	2011-7-2
4	100	1	☑	0	☐	Georges2		2011-7-11
5	1	123	☐	0	☑	Manfredo	Sierre	2011-7-1

Figure 38 Liste des acquisitions

Différents filtres sont proposés, accessibles en cliquant sur les en-têtes des colonnes : on peut filtrer les acquisitions par leur numéro, par la présence ou/non d'amputation, par le nom de leur sujet, et par leur date. La sélection des acquisitions se fait à la souris, et comme sur Windows, la sélection multiple se fait en utilisant les touches Ctrl (sélection par bloc) ou Shift (sélection par éléments) du clavier.

Figure 39 Contenu du zip

Un clic sur le bouton export prépare un fichier zip contenant un dossier pour chaque sujet, dans lequel se trouve un dossier pour chaque expérience, dans lequel se trouvent les données des électromyogrammes, les photos prises, et un fichier XML (voir annexe G) contenant les données cliniques du sujet et de l'expérience. La figure ci-contre donne un exemple du contenu du zip d'export.

5.6 Graphiques

Une série de graphiques permettant de suivre l'évolution du site sont disponible :

1. Utilisateur créé au fil du temps (cf. figure 47)
   a. Un panneau donnant des informations s'affiche lorsque l'utilisateur passe sur un point du tableau
2. Fréquentation du site (cf. figure 40)
   a. L'utilisateur peut zoomer/dézoomer sur une période de temps donnée : l'axe indiquant le temps change en fonction du degré de zoom.
3. Expériences et sujets au fil du temps
4. Pourcentage des sujets amputés ou non
5. Distribution d'une variable donnée (voire point suivant)

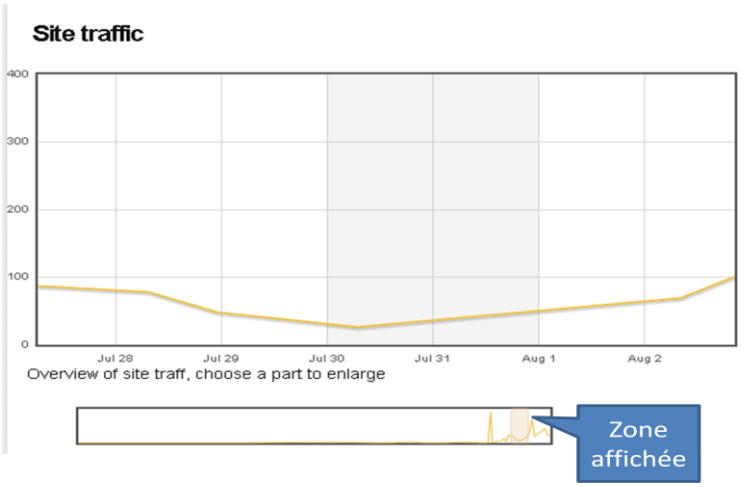


Figure 40 Graphique de fréquentation

5.6.1 Générateur de graphiques

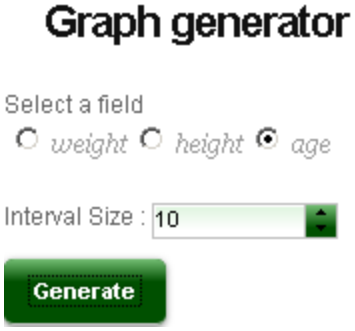


Figure 41 Générateur de graphiques

Le site propose une fonction, qui, lors de l'écriture du rapport, était encore à l'état expérimental : il s'agit d'un générateur de camemberts et d'histogramme fonctionnant de la manière suivante. :

1) L'utilisateur choisit un champ à analyser (cf. figure 41), par exemple l'âge, le poids, ou la taille d'un sujet donné, le but étant de découper ce champ en plusieurs tranches.

   a. L'idée est de créer une série de tranches ; par exemple, pour l'âge, nous aurions la tranche 0-5 ans, puis la tranche 5-10 ans, et ainsi de suite.

2) L'utilisateur choisit la taille de la tranche grâce à un spinner

3) L'utilisateur clique sur le bouton d'envoi, et le graphique s'affiche. En passant sur le graphique, des informations concernant la tranche correspondante s'affichent

4) Le maximum, minimum, et moyenne sont affichés

5) Un histogramme de distribution est généré pour le champ choisi

La capture d'écran ci-dessus montre les graphiques générés :

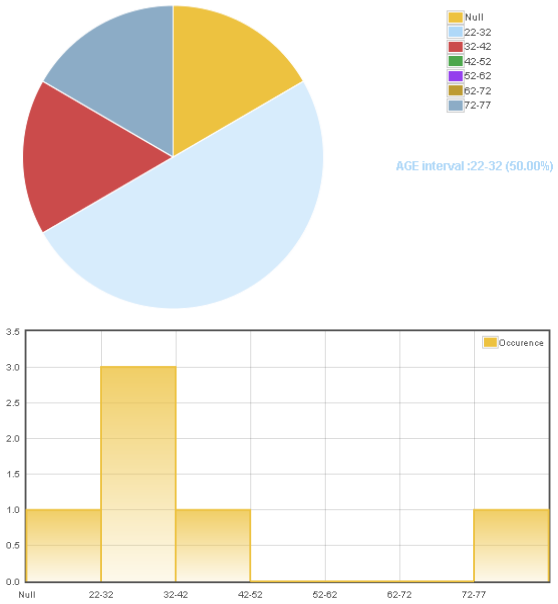


Figure 42 Graphiques de distributions

5.7 Flux d'activités

L'administrateur peut accéder à une interface récapitulant les activités des différents utilisateurs :

Figure 43 Flux d'activités

5.8 Accès sécurisé

Un accès SSL existe à l'adresse https://153.109.124.99:8181/NinaWeb/. Le certificat utilisé est expiré.

6 Evaluation des résultats

Cette partie aborde tout d'abord certains problèmes rencontrés lors du développement, et montre comment ils ont été résolus. Nous ferons une ensuite une évaluation des outils utilisés, des résultats obtenus, et de la méthodologie utilisée. Nous expliquerons ensuite ce que ce projet nous a apportés, puis analyserons entre le planning initial et les heures réalisées. Nous terminerons par la liste des améliorations futures de l'application.

6.1 Problèmes rencontrés

6.1.1 JSF et Weld

Les JARs relatifs à JSF et Weld ne correspondaient pas aux dernières versions de JSF et Weld. Du coup, des erreurs incompréhensibles sont apparues durant le développement. Heureusement, la mise à jour des JARS a pu se faire simplement ; en copiant les bon JARS dans les répertoires du serveur.

6.1.2 Machine de développement et machine de production

Le développement a été effectué sur une machine Windows. Le serveur de production tourne sur un OS Linux, et malheureusement, quelques problèmes sont apparus. La difficulté consistait à trouver l'origine de ces problèmes, à savoir la différence entre les deux machines. Voici la liste de ces problèmes :

- Sur Windows, les chemins s'écrivent avec \, et sur Linux avec /. Heureusement, Java possède une constante permettant de s'abstraire de ces différences
- Sur Windows, MySql est, par défaut, insensible à la casse, alors que sur Linux, il est sensible à la casse
- Sur Windows, le cache de base de données n'a pas le même comportement que sur Linux. Pour éviter tout problème, ce cache a été désactivé

6.1.3 Droits d'accès aux dossiers du serveur

Idéalement, les dossiers contenant les données stockées et générées sur le serveur de production ne devrait être qu'accessible par deux utilisateurs : le serveur, et le root. Lors des premiers déploiements sur la machine de production, les dossiers étaient créés à la main, avec les autorisations 777 (tout le monde pouvait y accéder). Pour remédier à cela, une solution élégante a été trouvée : donner au serveur la possibilité de créer ces dossiers : les dossiers possèdent ainsi directement les bons droits, seul le serveur peut y accéder (et évidemment l'utilisateur root)

6.1.4 Accès concurrent à la base de données

La gestion des connexions à la base de données est laissée aux serveurs, via la création d'un pool de connexions. Le serveur est alors capable de gérer des connexions concurrentes à la base de données, ce que nous serions incapable de faire. Deux (ou plus) personnes peuvent donc accéder au même moment à la base de données : deux connexions seront crées et fermées au moment opportun, et l'application ne bloquera pas. Les différents paramètres du pool de connexion peuvent être réglés sur la console d'administration du serveur.

6.1.5 Journalisation des évènements

La création d'acquisition et de sujets doit contenir des informations de journalisation : date, utilisateur. Comme expliqué au point, JPA permet de résoudre ce problème de manière élégante. Mais il reste encore à journaliser l'utilisateur qui a créé un sujet. La figure ci-dessous montre comment cela est réalisé, de nouveau grâce à l'usage d'annotations.

Figure 44 Journalisation des évènements

Le flux d'activité est géré de la même manière.

6.1.6 Redirection des utilisateurs

```
<f:event listener="#{userbean.checkLogin}" type="preRenderView">
</f:event>
<f:event type="preRenderView" listener="#{counter.hit}"></f:event>
```

Figure 45 Redirection des utilisateurs

Un utilisateur anonyme doit être redirigé sur la page de login. Pour implémenter cette fonctionnalité de manière simple, il suffit d'implémenter un écouteur dans le layout principal du site. Toutes les pages auront cet écouteur. Un écouteur réagit à un évènement particulier, que l'on spécifie grâce à l'attribut type, dans notre cas, il s'agit du chargement de la page. Lorsque la page est chargée, la méthode spécifiée dans l'attribut listener est appelée. Dans notre cas, cette méthode vérifie que l'utilisateur soit enregistré, et si ce n'est pas le cas, le renvoie sur la page de login. Le même principe est utilisé pour les pages d'administration, le serveur vérifie alors que l'utilisateur possède le droit ADMIN.

6.1.7 @ViewScoped et <a4j:mediaOutput>

L'annotation @ViewScoped s'est révélée bien pratique pour la gestion de l'Ajax, mais souffre d'un énorme défaut: certaines données s'évaporent comme par magie, de plus, ce phénomène semble se produire de manière aléatoire. Ce problème a déjà été évoqué plus haut, et il s'agit d'un défaut **fondamental** de JSF. Sans ce défaut, le framework serait

pleinement fonctionnel et vraiment agréable à utiliser. Pour contrer ce phénomène, il n'y a qu'une solution : recharger les données manquantes entre chaque requête, ce qui a compliqué énormément le développement et crée passablement de "mauvais" code. Le composant <a4j:mediaOutput> par exemple, provoque ce comportement : au lieu d'accéder à l'objet référencé par la balise, JSF crée une nouvelle instance de la classe correspondante à cet objet. Il faudrait idéalement utiliser un autre contexte que le contexte de vue.[24]

Quoiqu'il en soit, ce problème a créé une énorme perte de temps et complexifié le développement. Ce problème, selon le tableau de bord du framework JSF, sera résolu lors de la version 2.2

6.1.8 Touches Tab et Enter

La touche Tab permet de passer d'un champ à l'autre sur une page web. Ce comportement était éliminé, sans doute à cause de la gestion de l'Ajax. Il a fallu ré-implémenter ce comportement via du code JavaScript. Un problème similaire survenait avec la touche Return.

L'extrait de code suivant montre comment le problème a été résolu :

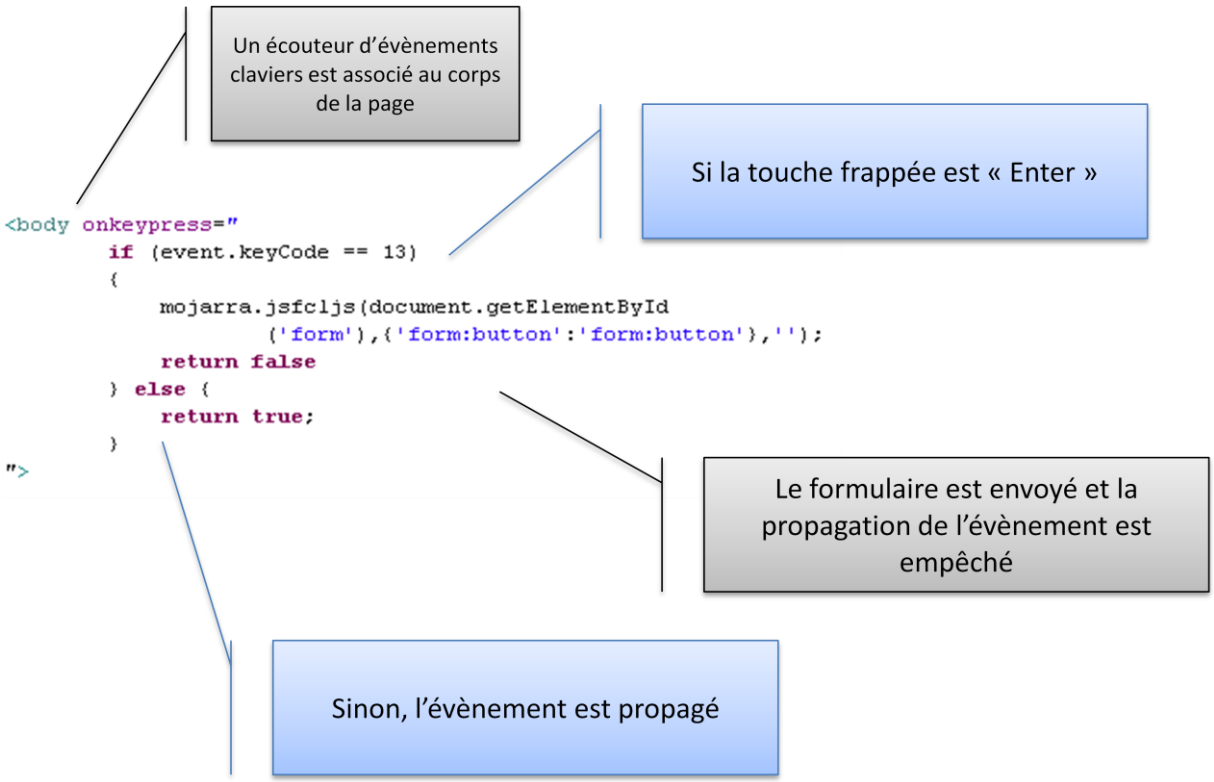


Figure 46 Gestion des évènements claviers

6.1.9 Création de graphiques

La création de graphiques représente une problématique très intéressante, car il s'agit d'un cas qui traverse toutes les couches de l'application. En effet, la création de graphiques se fait côté client: il s'agit de code JavaScript. Mais ce code JS doit récupérer des informations qui

[24] Cette solution est explorée au point 6.3

sont contenues dans le serveur, dans des objets Java. Comment réaliser ce tour de passe-passe ? RichFaces apporte une solution flexible à ce problème, et nous allons montrer comment en suivant pas à pas la création d'un des graphiques utilisés dans le site.

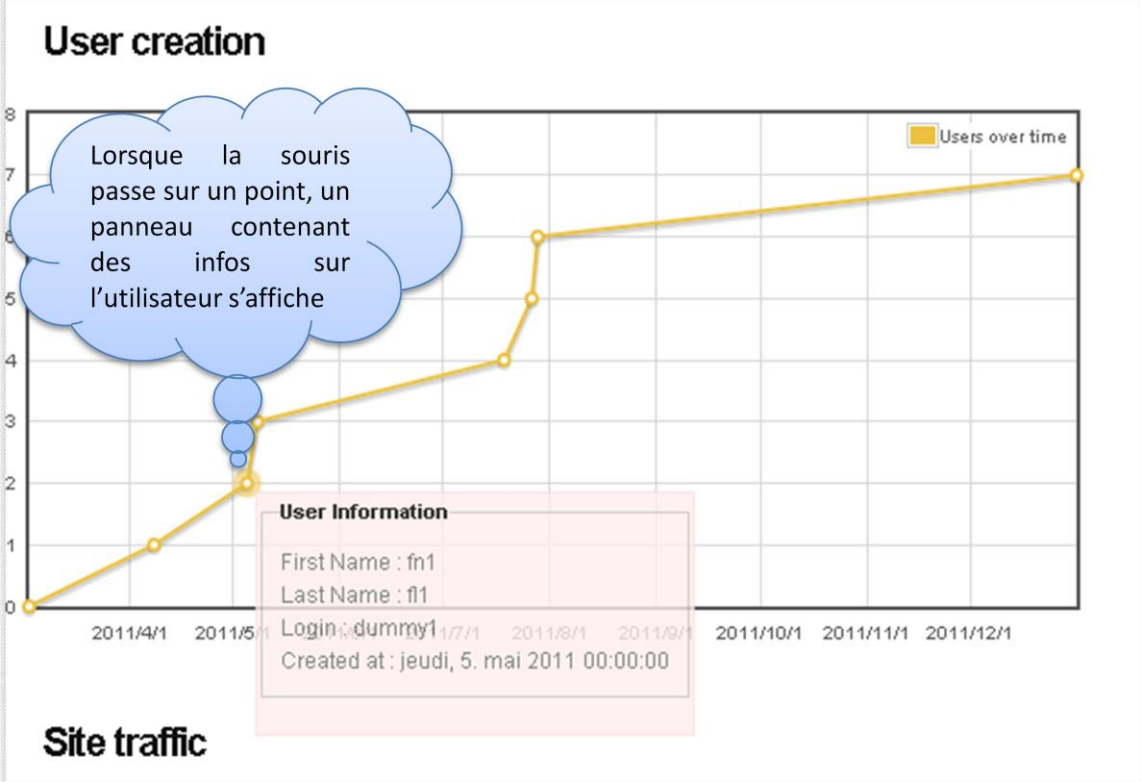


Figure 47 Graphique des utilisateurs

Ce graphique présente l'évolution des utilisateurs à travers le temps. De plus, lorsque l'utilisateur passe sur un point avec la souris, les données correspondantes à l'utilisateur s'affichent. Sa création s'effectue en plusieurs étapes:

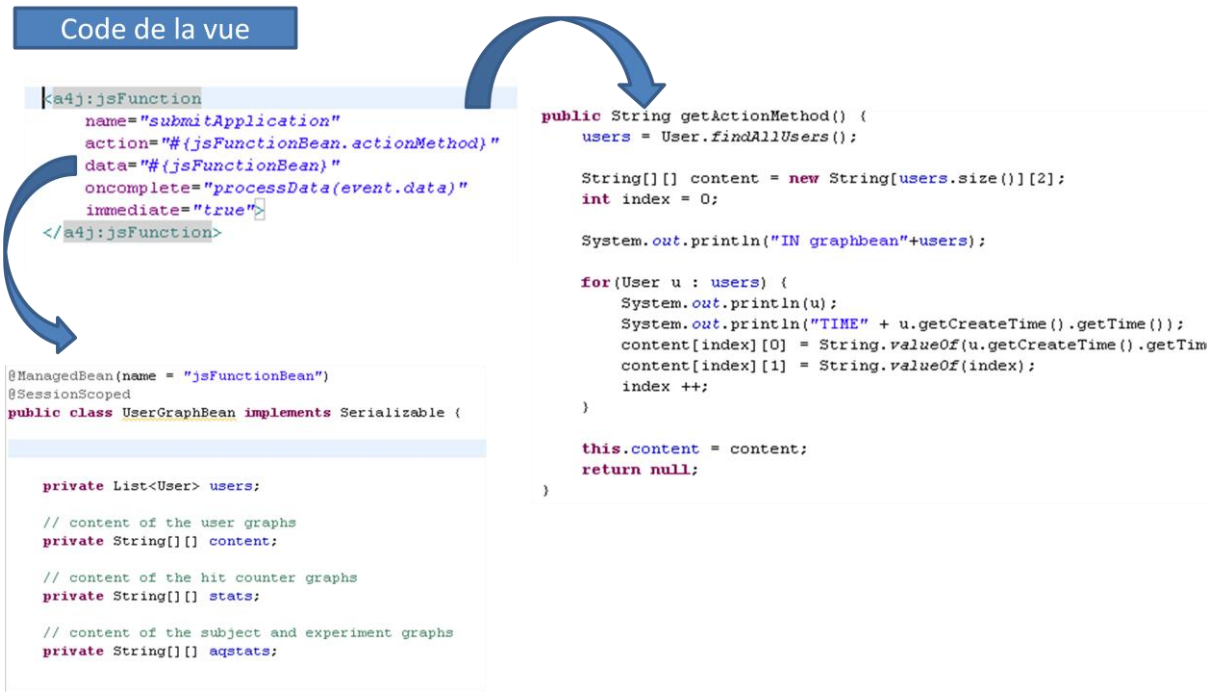
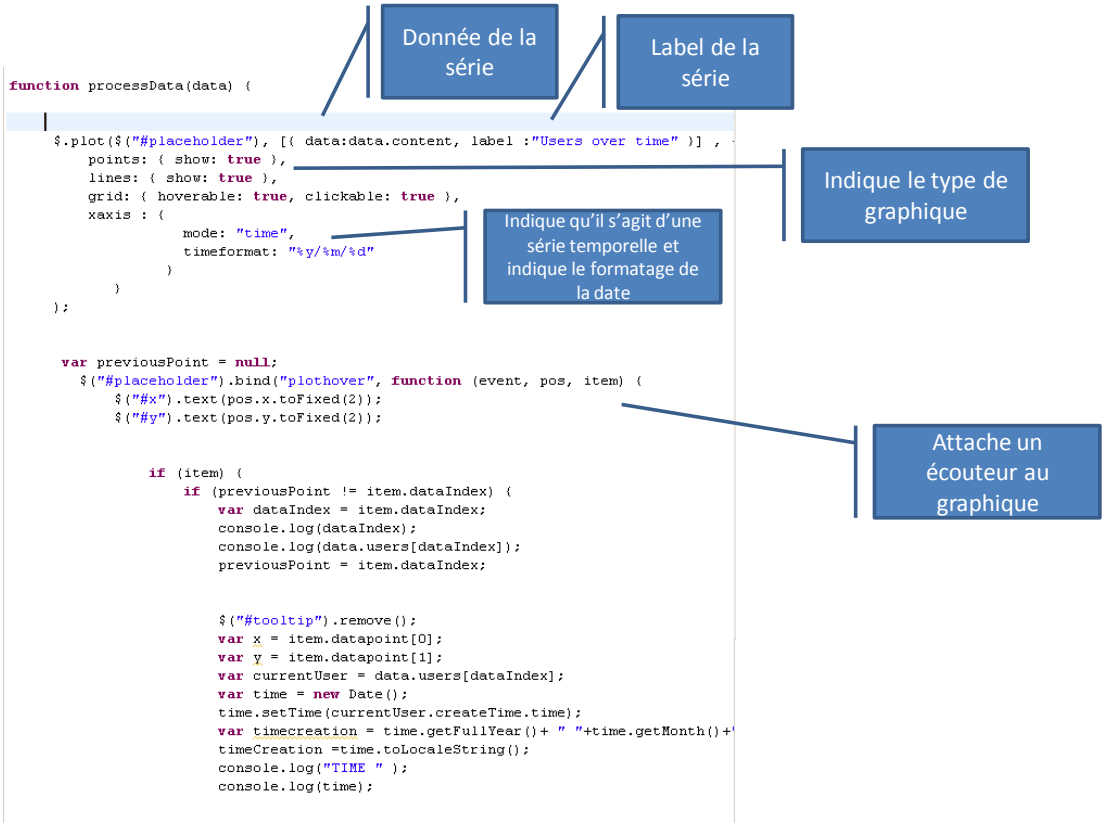


Figure 48 Code Java générant les graphiques

1. La première étape s'effectue dans le code affichant la vue : la balise *<a4j:function>* va créer une fonction JavaScript qui va effectuer un appel au serveur. La méthode appelée côté serveur est indiquée dans l'attribut action. La méthode appelée va chercher tous les utilisateurs dans la base de données, et remplir un tableau à deux dimensions contenant la date de création, et l'index de chaque utilisateur. **A ce moment, les données ne sont encore présentes que dans un objet Java.**
2. L'attribut data de la balise précédente indique à RichFaces de créer un objet JavaScript reflétant l'objet Java *jsFunction*. Cet objet sera disponible via le champ *data* de l'objet JavaScript *event*. Or, à l'étape une, nous avons justement mis à jour le champ *content* de l'objet *jsFunction.* Nous trouverons donc les informations nécessaires à la création du graphique à cet endroit : *event.data.content.*
3. L'attribut *oncomplete* indique à RichFaces d'appeler la fonction *processData* une fois les données prêtes. Nous pouvons voir que les données nécessaires à la création du graphique sont passées à cette fonction. Cette fonction va appeler la fonction plot de la librairie flot, et passer en paramètre les données nécessaires à la création du graphique. La figure ci-dessous détaille la fonction *processData*




```
function processData(data) {

    $.plot($("#placeholder"), [{ data:data.content, label :"Users over time" }] ,
        points: { show: true },
        lines: { show: true },
        grid: { hoverable: true, clickable: true },
        xaxis : {
                    mode: "time",
                    timeformat: "%y/%m/%d"
                }
            }
    );


     var previousPoint = null;
       $("#placeholder").bind("plothover", function (event, pos, item) {
           $("#x").text(pos.x.toFixed(2));
           $("#y").text(pos.y.toFixed(2));


               if (item) {
                   if (previousPoint != item.dataIndex) {
                       var dataIndex = item.dataIndex;
                       console.log(dataIndex);
                       console.log(data.users[dataIndex]);
                       previousPoint = item.dataIndex;


                       $("#tooltip").remove();
                       var x = item.datapoint[0];
                       var y = item.datapoint[1];
                       var currentUser = data.users[dataIndex];
                       var time = new Date();
                       time.setTime(currentUser.createTime.time);
                       var timecreation = time.getFullYear()+ " "+time.getMonth()+"
                       timeCreation =time.toLocaleString();
                       console.log("TIME " );
                       console.log(time);
```

Figure 49 Code Javascript générant les graphiques

La fonction bind permet d'attacher un écouteur au graphique, et de définir la fonction qui sera appelée par cet écouteur. Dans notre cas, cette fonction va créer le panneau détaillant l'utilisateur sélectionné.

6.1.10 Gestion des erreurs

Dans le cycle traditionnel requête-réponse, la gestion des erreurs, avec JSF reste simple : si une erreur se produit côté serveur, l'utilisateur est redirigé sur une page d'erreur. Evidemment, la gestion des erreurs peut être affinée ; typiquement si une requête d'écriture à la base de données ne peut pas être effectuée, la même page pourra être renvoyée à l'utilisateur, avec un message lui proposant de renvoyer sa requête

Les techniques Ajax introduisent de nouveaux problèmes :

- Par la prolifération du code JS qu'elles engendrent, elles augmentent les possibilités d'erreur côté client. Or, une erreur JS bloque le script et le rendu de la page, ce qui doit être évité à tout prix
- Si une erreur survient côté serveur, le mécanisme standard de redirection sur une page d'erreur ne fonctionne pas. Le texte de l'exception est simplement renvoyé en guise de réponse au navigateur, qui ne sait évidemment pas comment interpréter cette réponse. Il faut alors donc coder à la main la gestion des erreurs,

et un autre problème survient : seul les blocs try catch permettent de gérer les erreurs, or ces blocs ralentissent l'exécution du code. Il s'agit donc de bien cibler les endroits potentiels où une erreur pourrait survenir, tout en sachant qu'on ne peut pas « tout » couvrir

6.2 Discussion

Ce projet nous a permis de pousser le framework jusqu'à ses dernières limites, il constitue de fait une sorte de « benchmark » pour son évaluation. Les premiers points de cette partie vont s'attacher à comparer JSF avec d'autres frameworks. Comme l'utilisation d'Ajax constituait une partie importante de ce travail, nous évaluerons son utilisation dans l'application. Puis nous discuterons quelques points particuliers de ce projet. Nous terminerons par une analyse des décalages entre le planning initial et le travail effectué.

6.2.1 JSF vs SEAM

La majorité des problèmes rencontrés auraient pu être résolus en utilisant Seam. Malheureusement, cette solution n'a été vue que vers la fin du projet, et la mise en œuvre de ce framework, couplée à la migration de l'application, nous a paru un pari trop risqué.

6.2.2 Framework orienté actions vs framework orienté composants

Un problème plus fondamental affecte tous les frameworks orientés composants: ces frameworks ne tiennent pas compte des principes sous-jacents d'Internet.

REST est l'acronyme de "Representational State Transfer" inventé par Roy T. Fielding dans sa dissertation "an architecture style of networked systems". Roy T. Fielding participe de puis 1994 aux travaux du W3C sur les sujets URI, HTTP, HTML et WebDAV et a été le co-fondateur du projet Apache, le serveur web qui équipe la majorité des sites web de tout l'Internet. REST décrit certaines caractéristiques du web qui en ont fait son succès. Roy Fielding résume ainsi le terme REST « REST provides a set of architectural constraints that, when applied as a whole, emphasizes scalability of component interactions, generality of interface, independent deployment of components, and intermediary components to reduce interaction latency, enforce security, and encapsulate legacy systems »[25]. REST est un ensemble de contraintes architecturales à utiliser pour la construction de systèmes. Ces contraintes ont pour une large part présidé à l'élaboration du protocole http.

Dans cette optique, chaque URL représente une ressource donnée, et chaque méthode http correspond à une action sur cette ressource. Les frameworks orientés **action** se basent sur ce style d'architecture : chaque requête doit avoir une URL exprimant la ressource donnée, et la méthode http doit correspondre à l'action effectuée sur cette ressource; le tableau suivant démontre que les quatre verbes http correspondent aux quatre actions classiquement effectuées sur une base de données :

[25]FIELDING, Roy Thomas, Architecture Style and the Design of Network-Based Software Architecture. Université de Californie, 2000.

Action	HTTP	SQL
Lire	GET	SELECT
Modifier	POST	UPDATE
Supprimer	DELETE	DELETE
Créer	PUT	INSERT

Tableau 5 HTTP vs SQL

Dans l'optique REST, les données stockées en session doivent être réduites au minimum, et donc simplement se limiter aux données concernant l'utilisateur; on ne peut tout de même pas demander à un utilisateur de se re-logger à chaque page. Pour cette raison, les frameworks orientés action ne maintiennent généralement pas l'état des composants affichés sur la page. En effet, la notion de composants, d'écouteurs, n'existent pas dans ces frameworks. La seule chose qui est importante, c'est quelle action (c'est à dire quelle URL et quel verbe http) entreprendre suite à une interaction de l'utilisateur.

En conséquence, les frameworks actions paraissent plus compliqués à appréhender, car ils n'ont pas une palette de composants prêts à l'emploi, et ne ressemblent pas du tout au développement traditionnel d'applications natives (cf. la librairie Swing dans le monde Java), et laissent au développeur le soin de coder le code JavaScript nécessaire à la gestion de la page affichée. De plus, ces frameworks forcent le développeur à adopter une certaine structure dans l'hiérarchie des fichiers sources. En conséquence, les sites construits avec ces frameworks proposent donc des URL toujours lisibles et une structure du code source à chaque fois similaire. A titre d'exemple, voilà l'URL d'un site construit avec Symfony, un framework PHP : http://www.jobeet.org/en/job/company-105/paris-france/9/web-developer.
L'avantage d'une telle URL est qu'elle donne toutes les informations sur l'action effectuée par l'utilisateur. A l'inverse, les URL générées par les frameworks orientés composants ne sont guère lisible, où se cantonne simplement à un nom de page.

A notre sens, c'est l'une des faiblesses fondamentale de tous les framework orientés composants, c'est-à-dire que d'une certaine façon, ils promeuvent un modèle qui ne correspond pas au style architectural de l'univers où ils sont déployés, c'est-à-dire Internet. Ils trompent littéralement le développeur sur la marchandise, en lui faisant croire qu'il développe une application native, alors que cela n'est pas le cas. Comme le prix à payer n'est pas élevé, le développeur aura tendance à fermer les yeux; mais plus la complexité du site augmente, plus ce prix devient élevé : la gestion de l'état de la page va augmenter et consommer beaucoup de mémoire, de même que le code JavaScript nécessaire à la gestion des divers composants; la notion d'URL finit par complètement disparaître, et n'être plus qu'un simple point d'entrée au site.

Cependant, un développeur n'a pas forcément envie de se plonger dans du JavaScript, et les frameworks de type composant permettent de s'abstraire de cela. Et donc, la question qui se pose est la suivante : est-ce que nous aurions réussi à développer une application semblable en utilisant un framework orienté action, comme Ruby On Rails, où nous aurions du coder le JavaScript à la main ?

Rétrospectivement, cela aurait paru un défi intéressant, mais rien ne garanti alors que le projet aurait abouti.

6.2.3 JSF vs Rails

Une grande différence, et un désavantage majeur pour les applications JSF par rapport aux applications Rails est la facilité avec laquelle on peut déployer dans le cloud un site Rails. Il suffit de s'inscrire sur le site Heroku, qui propose un cloud pour héberger des applications Rails. Heroku est capable d'aller chercher l'application sur GitHub, et de la déployer. Nous avons réalisé un petit blog en Rails, et le déploiement sur le Cloud s'est réalisé en 15 minutes sans problème. De plus Heroku propose un tableau de bord permettant de choisir les capacités de la base de données et de la machine hébergeant l'application ; nous n'avons rien trouvé de semblable pour des applications JSF

6.2.4 JSF vs GWT

Le chapitre précédent comparait deux types de framework différents, nous allons à présenter tenter de comparer JSF avec un autre framework orienté composant –GWT-, avec lequel nous avons déjà développé un projet, il y a six mois. Les principales caractéristiques de GWT sont décrites au point 3.5 ; ce qui est à retenir, c'est que la gestion de la page web est entièrement déportée côté client, le serveur étant dédie au traitement des données. L'approche de GWT est celle du client lourd, avec un serveur ne jouant que son « vrai » rôle : le traitement des données

Ce qui nous avait frappés avec GWT, c'était la facilité avec laquelle on pouvait créer une petite application web en ne codant qu'une seule classe avec une méthode main, comme un programme console Java. Le problème est que le développeur aura tendance à toujours repartir de cette classe main, et ne séparera pas les différentes couches de l'application : vue, contrôleur et modèle se retrouve mêlés dans un seul fichier, un véritable cauchemar pour la maintenance d'une application.

De plus, GWT ne laisse quasiment aucun contrôle sur le code produit : le développeur code en JAVA, mais les artefacts produits après compilation sont des pages HTML, CSS et des scripts JS. Et lors des premières versions de GWT il fallait bien connaître le framework pour oser modifier le code JavaScript généré, et de toute façon, le faire revient à « briser » le projet, car à la prochaine compilation du code Java, les scripts JS sont régénérés.

Dans la préface du livre *Programmation GWT 2*, l'auteur affirmait que les frameworks côté serveur allaient rapidement disparaître [9, pages IV-V]; or, il n'en est rien. JSF 2.1 a d'ailleurs gagné plusieurs adhérents par rapport à la catastrophe que représentait la version précédente, en grande partie grâce à sa flexibilité et ses librairies de composants. De plus, certains frameworks basés sur JSF proposent des modules pour intégrer des applications GWT. **Comme dans le monde économique, ce n'est pas la compétition qui assure la survie, mais la coopération intelligente.**

Pour conclure cette discussion sur les frameworks, nous reprenons la conclusion du billet d'Antoine Sabot-Durand[26] :

> Pour conclusion, j'évoquerai les approches antagonistes actuelles entre les frameworks qui prônent le do it yourself concernant la partie IHM (Ruby on Rails, Play!, Spring MVC) et ceux qui livrent des approches de composants riches laissant peu de contrôle sur le code produit (GWT ou Vaadin par exemple). Ces deux perceptions ont leurs avantages et inconvénients et je me garderai bien de trancher sur le sujet.
>
> Simplement, je remarque que JSF 2.0 est une solution qui permet de jouer sur les deux tableaux, en permettant de développer soit même des composants réutilisables de manière très simple et/ou d'avoir recours à des bibliothèques très riches de composants haut niveau qui vont produire des interfaces riches sans que l'on ai à maitriser HTML 5, CSS3 ou Javascript.
>
> Pour ceux qui se demandent encore pourquoi JSF est la technologie standard d'IHM web de Java EE 6, je pense que la réponse se trouve beaucoup dans cette souplesse et que, même si JSF a encore des progrès à faire, il a le mérite de proposer la bonne péréquation entre la conception d'interfaces simples et la création de client web riches. Une synthèse presque parfaite des frameworks web Java en quelque sorte.

Au final, le choix d'un framework et du langage doit être dicté par les besoins du moment. Cela peut paraître une évidence, mais de nombreux facteurs, qui vont au-delà des préférences personnelles pour tel ou tel langage, rentre en ligne de compte, et un développeur, fan de telle ou telle technologie, risque de ne pas s'intéresser à ces facteurs, sous prétexte qu' « avec mon framework, je peux tout de façon tout faire ». Et comme le dit Abraham Maslow: « It is tempting, if the only tool you have is a hammer, to treat everything as if it were a nail. » Ce qui produit de mauvais résultats dans la plupart des cas.

6.2.5 Trop d'Ajax

Le fait d'avoir voulu construire une application utilisant intensivement les techniques Ajax a introduit passablement de difficulté (Gestion des erreurs, code JS à débugger, annotation @ViewScoped défectueuse, non-respect du style d'architecture Rest, non-respect de l'historique.) Peut-être aurait-il été judicieux d'utiliser les techniques Ajax de manière limitée, pour des actions de tri, de filtrage, d'auto-complétion faisant des appels au serveur, mais de ne pas l'utiliser lors d' « actions lourdes », c'est-à-dire lors d'écriture dans la base de données.

6.2.6 TDD

Le temps nécessaire a l'écriture des tests a été sous-estimé ; l'écriture de tests prend beaucoup de temps, et il est difficile de savoir où s'arrêter : faut-il tester toutes les méthodes, même une simple méthode d'une ligne ? Dans le cas de ce projet, nous avons dû abandonner l'écriture de tests pour pouvoir avancer.

[26] http://blog.ippon.fr/2011/04/27/jsf-je-taime-moi-non-plus/ (Page consultée le 10 juillet 2011)

De plus, beaucoup de temps a été perdu dans la recherche d'un framework de test d'intégration : nous avons tenté de faire fonctionner JSFUnit, mais sans succès, et pour utiliser Selenium, nous avons du passer par une courte phase d'apprentissage ; le temps utilisé correspondait à 3 jours / homme **pour apprendre à utiliser et mettre en place cet outil. Les tests d'intégration devaient alors encore être écrits** (ainsi que le rapport.) Comme la timeline de ce projet était fermée, nous avons préféré mettre de côté l'écriture de tests au profit de l'écriture du rapport.

6.2.7 Retour d'expérience

Ce projet est notre premier projet d'une certaine envergure, et il nous a permis de découvrir toutes les difficultés et particularités du développement d'une « vraie » application :

Design

Construire une interface est toujours une expérience intéressante, voire amusante : la construction de bonnes interfaces renvoit à de nombreux domaines. Malheureusement, nous ne sommes pas très « bon » dans ce domaine : nos compétences avec des logiciels de retouche, comme photoshop, se résume à … zéro ! Nous sommes donc contraints de faire avec ce qui est proposé par RichFaces et JSF pour tenter de construire quelque chose qui soit relativement agréable à utiliser. Cela ne nous a pas empêché d'aller modifier du code traditionnellement réservé aux « web designers », typiquement les feuilles CSS, le code HTML et JS.

Cycle de vie et intégration

Ce projet est le premier projet où nous avons dû « tout » prendre en compte, que ce soit au niveau du **cycle de vie : développement, test, déploiement, documentation,** qu'au niveau des **différentes couches :** configuration des différents serveurs, code Java côté serveur, code JavaScript côté client, gestion de la base de données. Il s'agit véritablement du premier projet où nous avons pu intégrer différentes librairies à différents niveaux, et faire communiquer véritablement les différentes couches de l'application (cf. point Création de graphiques). Ce projet nous a permis aussi de nous familiariser avec le langage JavaScript

Tâches d'administration système

Un aspect souvent négligé lors du développement d'applications web est la création de taches « systèmes » : recréation/export de la base de données, nettoyage des dossiers de stockage de données, tâches de maintenance de la base de données (vérification des doublons…). Lorsqu'un projet brasse un nombre conséquent de données ; créer ces tâches dès le début et disposer d'une interface pour les lancer permet un gain de temps important pour la suite du développement ; de plus, ces tâches seront alors à disposition pour la maintenance future du site. Ce projet a mis en lumière la nécessité de construire ces tâches le plus vite possible

Domaine métier

Ce projet nous a permis d'être confronté à un vrai domaine métier et de sortir des sempiternelles médiathèques ou des mini-applications bancaires que nous avions dû réaliser lors de nos études.

6.2.8 Gestion de projet

Le planning du projet et les heures réalisées sont disponibles en annexe. Globalement le planning a été tenu mais une grande perte de temps s'est produite lors de la recherche d'un outil permettant de réaliser des tests d'intégration. De plus, le déploiement sur un serveur de production a commencé bien plus tôt, ce qui a été une bonne chose, au vu du nombre de problèmes qu'a posé cette mise en production.

Nous pensons que pour réaliser l'ensemble des fonctionnalités notées dans le cahier des charges, qui était assez large, et écrire une véritable suite de test, il aurait fallu d'une part deux semaines de plus et d'autre part ne pas avoir à écrire le rapport.

6.3 Améliorations futures

6.3.1 Refactoring 1 : Suppression des contextes JSF au profit des contextes CDI

Notre application mélange les contextes WELD et les contextes JSF. Cela était du au fait que nous avions choisi d'utiliser le contexte @ViewScoped pour gérer les objets liés à la vue, ce qui a occasionné certains problèmes (cf. supra). De plus, l'utilisation des contextes JSF empêche d'utiliser les services proposés par CDI (intercepteurs, listeners intégrés). Un premier refactoring consisterait à supprimer toutes les annotations @ViewScoped, pour les remplacer soit par :

- @ConversationScoped : Il faut donc gérer le début de la conversation (lorsque l'utilisateur arrive sur la page), et la fin de la conversation (lorsque l'utilisateur quitte la page)
- @SessionScoped : Dans cette optique, la mémoire consommée serait augmentée, la durée de vie du bean durant toute la session ; et il faudrait réinitialiser les champs de l'objet lorsque l'utilisateur quitte la page gérée par l'objet.

6.3.2 Refactoring 2 : Amaigrissement du contrôleur

Dans le monde Ruby, on essaie de minimiser la taille des objets gérant les pages. Dans notre application, ces objets contiennent de nombreuses lignes. Typiquement, ce sont eux qui sauvent les objets métiers en base de données et qui vont chercher les objets dans la base de données. Ce code devrait être déplace directement dans le code de l'objet métier ; une partie de ce refactoring a déjà été commencé.

De plus, certains objets gèrent plusieurs éléments d'une même page. Ces objets pourraient être découpés en plusieurs parties, chaque partie s'occupant d'un seul élément de la page. Ce refactoring a été mis en œuvre pour la gestion de la partie administrative : nous avons une classe qui instancie l'objet s'occupant de l'administration des utilisateurs, et une classe instanciant l'objet gérant le contenu de la liste déroulante « Work »

6.3.3 Refactoring 3 : Classes « Helpers » mélangées

Certaines classes helpers se sont malencontreusement retrouvées dans des packages qui n'étaient pas les bons. Dans le même ordre d'idées, certaines méthodes appartenant à des classes métiers, ou à des classes gérant les pages, seraient plus à leur place comme méthode statique d'une classe helper.

6.3.4 Refactoring 4 : Laisser le conteneur gérer les utilisateurs

Les aspects relatifs à la gestion des utilisateurs sont gérés par notre application ; mais le serveur peut être configuré pour gérer lui-même l'authentification et les autorisations des utilisateurs ; ce qui permettrait de découpler ces aspects de l'application. Un framework (JAAS) destiné à la gestion de la sécurité est embarqué dans le JDK.

6.3.5 Vers une architecture orientée services

Nous terminerons ce rapport en proposant une autre architecture, plus ouverte, pour notre application.

En effet, l'architecture choisie couple les accès à la DB aux pages web : ce sont les contrôleurs respectifs de ces pages qui ouvrent les connexions à la DB, effectuent les requêtes, ferme les connexions. Supposons que nous voulions développer une application mobile proposant les mêmes fonctionnalités : il faudra tout recoder.

Il serait donc intéressant d'implémenter un service web de type REST gérant les différentes opérations liées à la base de données (CRUD des sujets, acquisitions, etc..), l'authentification et les autorisations des utilisateurs. Toutes ces opérations seraient découplées de notre application ; une application native de type mobile ou bureau n'aurait pas à recoder ces opérations, mais simplement les appels aux services web. La figure ci-dessous résume une telle architecture.

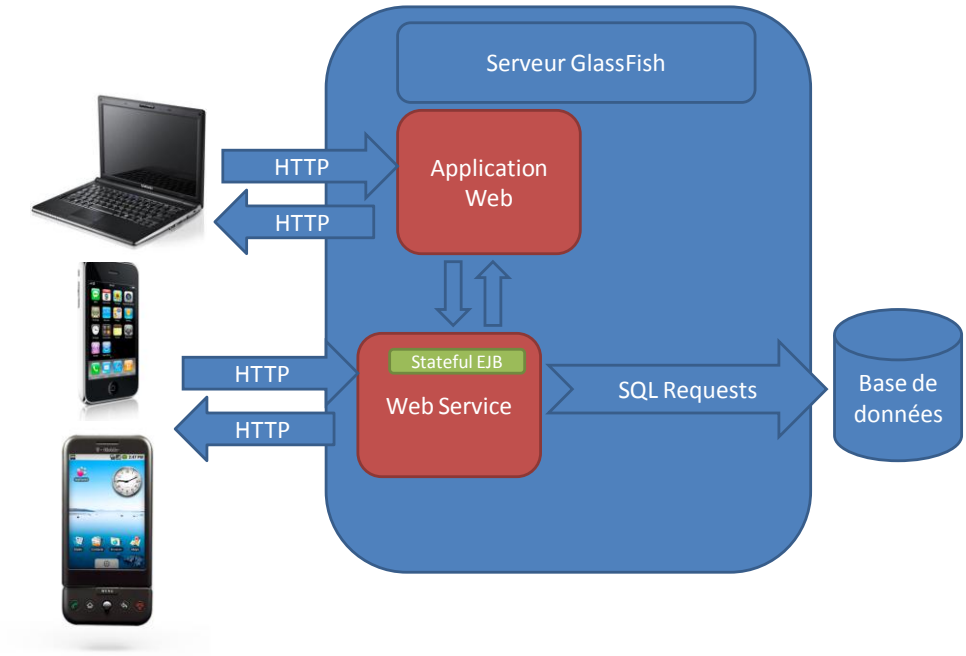


Figure 50 Architecture SOA

7 Bibliographie

[1] BURNS Ed, JSF 2.0-The Complete Reference, McGraw-Hill Books USA, 2010

[2] CHAFFER Jonathan, SWEDBERG Karl : Learning jQuery 1.3, Olton, Packt Publishing UK, 2007

[3] DIX Paul, Service-Oriented Design With Ruby And Rails, Pearson Education, 2010

[4] FERNANDEZ Obie, The Rails 3 Way, Boston, Pearson Education USA; 2010

[5] GEARY David, HORSTMANN Cay: Core JavaServer Face, Third Edition, Boston, Pearson Education USA, 2010

[6] GONCALVES Antonio, Beginning Java EE6 Platform with GlassFish 3, From Novice To Professional, Apress, 2009

[7] HART Michael, Ruby On Rails 3 Tutorial, Boston, Pearson Education USA, 2010

[8] HENNING Mueller, CAPUTO Barbara, VAN DER SANGT Patrick, NINAPRO – Non-Invasive Adaptive Hand Prosthetics, 2010

[9] JABER Sami, Programmation GWT, Eyrolles, 2010

[10]KARWIN Bill, SQL Antipattern, The Pragmatic Programmers, 2010

[11]THIRUMALESH Bath, Evaluating the Efficacy of Test-Driven Development : Industrial Case Studies, Redmond, Microsoft, 2008

[12]YANG Daoqi, Java Persistence with JPA, Outskirsts Press, 2010

8 Glossaire

Ajax :	Acronyme de Asynchronus JavaScript and XMLHttpRequest. Il s'agit d'un ensemble de techniques augmentant l'interactivité des pages web.
Bean :	Objet Java dont le code de sa classe suit certaines conventions.
CRUD :	Acronyme de Create, Read, Update, Delete ; il s'agit des quatre actions typiquement réalisées sur une base de données
DB :	Abréviation de base de données
Électromyographies :	Il s'agit d'un examen qui permet enregistrer l'activité électrique d'un muscle ou d'un nerf.
Framework :	Ensemble de composants (classes, GUI, outils de ligne de commande..) destiné à faciliter le développement d'applications, typiquement en soulageant le développeur du travail de bas niveau.
Jar :	Archive contenant des classes Java
Java :	Langage de programmation orienté objet pouvant s'exécuter sur n'importe quelle plateforme grâce à la machine virtuelle du même nom, la JVM (Java Virtual Machine)
Java EE :	Il s'agit d'un ensemble de spécifications destinées à faciliter le développement d'application en entreprise.
JPA	Acronyme de Java Persistence Api. Il s'agit d'un framework gérant les données relationnelles
SF :	Acronyme de JavaServer Faces. Il s'agit de la spécification du standard JAVA EE traitant de l'affichage et de la gestion des pages Web.
Machine Virtuelle :	Une machine virtuelle transcrit du code précompilé en code machine exécutable sur une plate-forme donnée
SQL :	Acronyme de Structured Query Langage.) Il s'agit d'un langage informatique normalisé qui sert à effectuer des opérations sur des bases de donnée
XML :	Acronyme d'eXtensible Markup Language. Il s'agit d'un langage informatique de balise générique. Ce langage est typiquement utilisé pour transporter des informations entre différentes applications

9 Table des illustrations

9.1 Figures

Table des illustrations

9.2 Tableaux

10 Déclaration sur l’honneur

Je déclare, par ce document, que j'ai effectué le travail de bachelor ci-annexé seul, sans autre aide que celles dûment signalées dans les références, et que je n'ai utilisé que les sources expressément mentionnées. Je ne donnerai aucune copie de ce rapport à un tiers sans l'autorisation conjointe du RF et du professeur chargé du suivi du travail de bachelor, y compris au partenaire de recherche appliquée avec lequel j'ai collaboré, à l'exception des personnes qui m'ont fourni les principales informations nécessaires à la rédaction de ce travail et que je cite ci-après :

- M. Henning Müller
- M. Manfredo Atzori

François Chabbey

11 Annexe

11.1 Annexe A : Accès

Serveur MySQL

Adresse: 153.109.124.99/phpmyadmin

Utilisateur : root

Mot de passe : ma20ny11

Serveur GlassFish

Adresse : 153.109.124.99 :4848/

Utilisateur : admin

Mot de passe : gf11

Accès au site

Adresse : 153.109.124.99 :8080/NinaWeb

Utilisateur root : admin

Mot de passe : root

Utilisateur en lecture seule : read

Mot de passe : read

Repository

Adresse : https://github.com/chabbeyf/NinaPro/tree/start

Le projet peut être téléchargé depuis le site

11.2 Annexe B : Protocole d'acquisition

Training Sequence	
Number of Movements	50
Repetitions of Movements	2
Movement Duration (Seconds)	5
Rest Duration (Seconds)	3
Exercise Duration (Minutes)	13
Pause (Minutes)	5
aining Sequence Duration (Minutes)	18

Experiment	
Exercise 1	Basic Fingers
Number of Movements	12
Repetitions of Movements	10
Movement Duration (Seconds)	5
Rest Duration (Seconds)	3
Exercise Duration (Minutes)	16
Pause (Minutes)	5
Exercise 2	Hand Static Postures & Basic Wrist Movements
Number of Movements	17
Repetitions of Movements	10
Movement Duration (Seconds)	5
Rest Duration (Seconds)	3
Exercise Duration (Minutes)	23
Pause (Minutes)	5
Exercise 3	Grasps
Number of Movements	21
Repetitions of Movements	10
Movement Duration (Seconds)	5
Rest Duration (Seconds)	3
Exercise Duration (Minutes)	28
Pause (Minutes)	5
Exercise 4	Functional Movements
Number of Movements	14
Repetitions of Movements	5
Movement Duration (Seconds)	10
Rest Duration (Seconds)	3
Exercise Duration (Minutes)	15
Experiment Duration (Minutes)	97

TOTAL DURATION (Minutes)	115

11.3 Annexe C : Cahier des charges

Cahier des charges

Introduction

Ce document décrit les différentes fonctionnalités de l'application qui devront être réalisées dans le cadre de ce travail de bachelor. Ces fonctionnalités sont réparties en deux catégories : les fonctionnalités obligatoires et les fonctionnalités facultatives :

- **Fonctionnalités obligatoires** : Il s'agit des fonctionnalités devant être absolument réalisées
- **Fonctionnalités facultatives** : Ces fonctionnalités seront développées en fonction du temps restant à disposition

Structure du projet

Le projet consiste en une application Web. Cette application comporte trois parties distinctes :

1. Création d'une structure de données susceptible d'accueillir les données récoltées
    a. Création d'une base de données
    b. Création d'utilisateurs et de droits pour cette base
    c. Les résultats des expériences doivent être stockés sous forme de fichiers
2. Une interface de saisie et de visualisation de données
    a. Saisie de données concernant des patients
    b. Saisie de données concernant des expériences cliniques
    c. Recherche de données
    d. Mise à jour des données
    e. Mise en place d'un système de gestion de droits
    f. Affichage d'informations concernant les expériences
3. Une interface d'analyse et d'extraction de données
    a. Extraction de données
    b. Analyse statistique de données

Détail des fonctionnalités

Partie 1 : Création d'une structure de données

Ce projet pose comme contrainte de stocker les résultats des expériences sous forme de fichiers textes. La structure de données adoptée se décomposera donc en une base de données stockant les informations relatives aux utilisateurs, sujets, et conditions d'expérience, et de l'autre, un système de fichiers stockant les résultats des expériences menées. La figure 01 représente la structure de données adoptée.

Création d'une structure de données

Fonctionnalités obligatoires :

- Création d'une base de données susceptible d'accueillir les résultats des expériences menées dans le cadre du projet NinaPro.
- Cette base doit être capable de suivre les évolutions des données médicales des sujets au fil des expériences menées.

- Les résultats des expériences devront être stockés sous forme de fichiers textes. Il faudra donc s'assurer que les données relatives à une expérience donnée dans la base de données correspondent aux fichiers correspondants, comme décrit dans la figure ci-dessous.

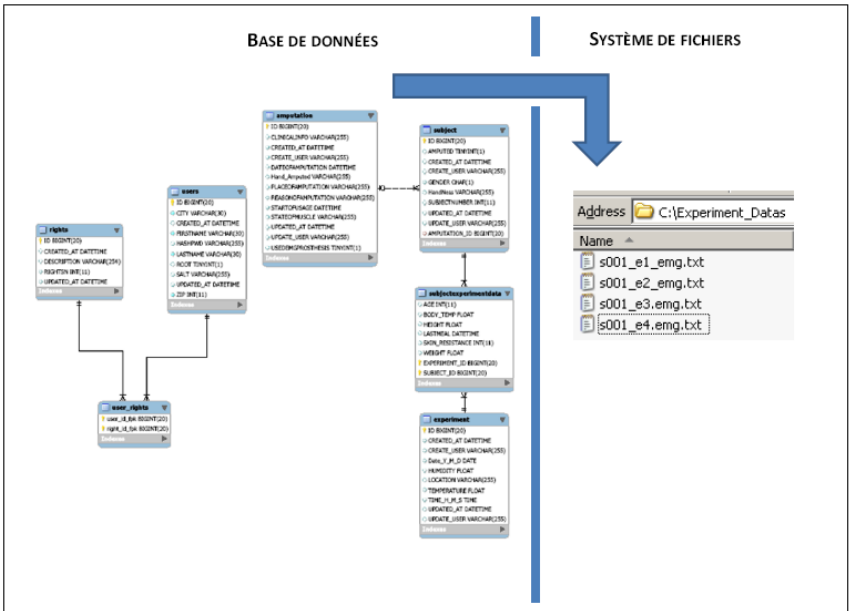


Figure 1 Structure des données

Fonctionnalités facultatives

- Journalisation des évènements de la base de données, les données journalisées étant les suivantes :
  - Date de création et de modification d'un enregistrement
  - Identifiant de l'utilisateur ayant créé ou mis à jour un enregistrement
- Enregistrement des autres fichiers : Chaque fichier concernant une expérience possède deux autres fichiers qui lui sont corrélés.
  - Le premier fichier contient des données de synchronisation par rapport au film de référence
  - L'autre fichier contient des données de référence enregistrée à l'aide d'un gant

Création d'une structure de données

Fonctionnalités obligatoires

- Création de trois rôle utilisateurs, ayant respectivement pour droits
  - Superadministrateur
  - Lecture sur toutes les tables
  - Ecriture sur toutes les tables

Mise à jour des données

Fonctionnalité obligatoire

- Création d'une interface permettant la mise à jour des données

Fonctionnalité facultative

- La première version de cette interface de mise à jour permettrait uniquement de modifier les données d'un seul tenant; c'est-à-dire que l'utilisateur doit choisir une expérience donnée, puis peut après modifier toutes les données correspondantes à cette expérience (sujet de l'expérience, condition de l'expérience, etc..). Il serait intéressant de donner la possibilité à l'utilisateur de modifier uniquement les données concernant un sujet donné.
- Ajout d'une fonctionnalité "Recherche rapide" basée sur l'id du sujet, et la date de l'expérience

Gestion des droits

La sécurisation du site est englobée dans la gestion des droits.

Fonctionnalités obligatoires

- Création d'un système de droits simples
    - Un utilisateur peut se voir assigner un/plusieurs droits; ces droits étant :
        - Administrateur du système
        - Saisie de données
        - Mise à jour de données
        - Consultation de données
        - Suppression de données
- L'accès au site doit être sécurisé; seule la page d'accueil devra être accessible en clair et au public.
    - L'accès se fait via un login / mot de passe
- Création d'une interface d'administration des utilisateurs
    - Ajout/Suppression/Modification des utilisateurs

Fonctionnalités facultatives

- Affinage des droits
- Création de groupes
- Implémentation d'une fonctionnalité permettant à l'utilisateur de récupérer un nouveau mot de passe par e-mail au cas où il aurait perdu son mot de passe.
- Dans l'interface d'administration des utilisateurs, donner la possibilité à l'utilisateur de consulter les activités d'un utilisateur donné
- Les utilisateurs peuvent s'enregistrer sur le site via une interface, et le superadministrateur doit approuver leur enregistrement sur son interface

Affichage d'informations concernant les expériences

Fonctionnalités facultatives

- Chaque expérience suit un protocole précis qui se décompose en quatre exercices distincts. Une partie du site, qui serait ouverte au public, pourrait présenter le protocole des expériences. Une partie administrative serait créée pour permettre de gérer les données affichées.
- Chaque expérience possède un film de démonstration. Ce film est présenté au sujet passant une expérience. Il serait intéressant, pour un sujet et une expérience donnée, de visualiser les mesures de l'expérience donnée avec les images correspondantes du film.

Partie 2 : –Extraction de données

Extraction de données

Fonctionnalités obligatoires :

- L'utilisateur doit avoir la possibilité de télécharger les fichiers concernant une ou plusieurs expériences données (par exemple, toutes les expériences trouvées lors d'une recherche)
- L'extraction doit se faire au format CSV et ZIP (au cas où plusieurs fichiers seraient extraits)

Fonctionnalités facultatives

- L'extraction doit se faire sous d'autres formats:
  - PDF, XLS, TXT…
- L'extraction sous format CVS peut être paramétrable (choix du séparateur)

Analyse de données

Fonctionnalités obligatoires

- Analyse de données via des critères simples :
  - Affichage de moyennes
  - Analyse statistique simple :
    - Diagramme de distribution des sujets en fonction de l'âge, ou d'autres paramètres.
  - Evolution des données des sujets au fils des expériences menées, présentée sous forme de graphiques

Fonctionnalités facultatives

- Affichage de statistiques d'utilisateurs
  - Nombre d'upload par mois.
- Affichage des activités d'un utilisateur donné
  - Pour chaque utilisateur, on peut afficher un fil de ses activités récentes
  - Le superadministrateur peut avoir accès à un fil de toutes les activités récentes, le tableau ci-dessous donne un exemple de fil (Il s'agit du fil d'activité sur un repository)

Affichage d'informations concernant les expériences

Fonctionnalités facultatives

- Chaque expérience suit un protocole précis qui se décompose en quatre exercices distincts. Une partie du site, qui serait ouverte au public, pourrait présenter le protocole des expériences. Une partie administrative serait créée pour permettre de gérer les données affichées.
- Chaque expérience possède un film de démonstration. Ce film est présenté au sujet passant une expérience. Il serait intéressant, pour un sujet et une expérience donnée, de visualiser les mesures de l'expérience donnée avec les images correspondantes du film.

Partie 2 : –Extraction de données

Extraction de données

Fonctionnalités obligatoires :

- L'utilisateur doit avoir la possibilité de télécharger les fichiers concernant une ou plusieurs expériences données (par exemple, toutes les expériences trouvées lors d'une recherche)
- L'extraction doit se faire au format CSV et ZIP (au cas où plusieurs fichiers seraient extraits)

Fonctionnalités facultatives

- L'extraction doit se faire sous d'autres formats:
  - PDF, XLS, TXT…
- L'extraction sous format CVS peut être paramétrable (choix du séparateur)

Analyse de données

Fonctionnalités obligatoires

- Analyse de données via des critères simples :
  - Affichage de moyennes
  - Analyse statistique simple :
    - Diagramme de distribution des sujets en fonction de l'âge, ou d'autres paramètres.
  - Evolution des données des sujets au fils des expériences menées, présentée sous forme de graphiques

Fonctionnalités facultatives

- Affichage de statistiques d'utilisateurs
  - Nombre d'upload par mois.
- Affichage des activités d'un utilisateur donné
  - Pour chaque utilisateur, on peut afficher un fil de ses activités récentes
  - Le superadministrateur peut avoir accès à un fil de toutes les activités récentes, le tableau ci-dessous donne un exemple de fil (Il s'agit du fil d'activité sur un repository)

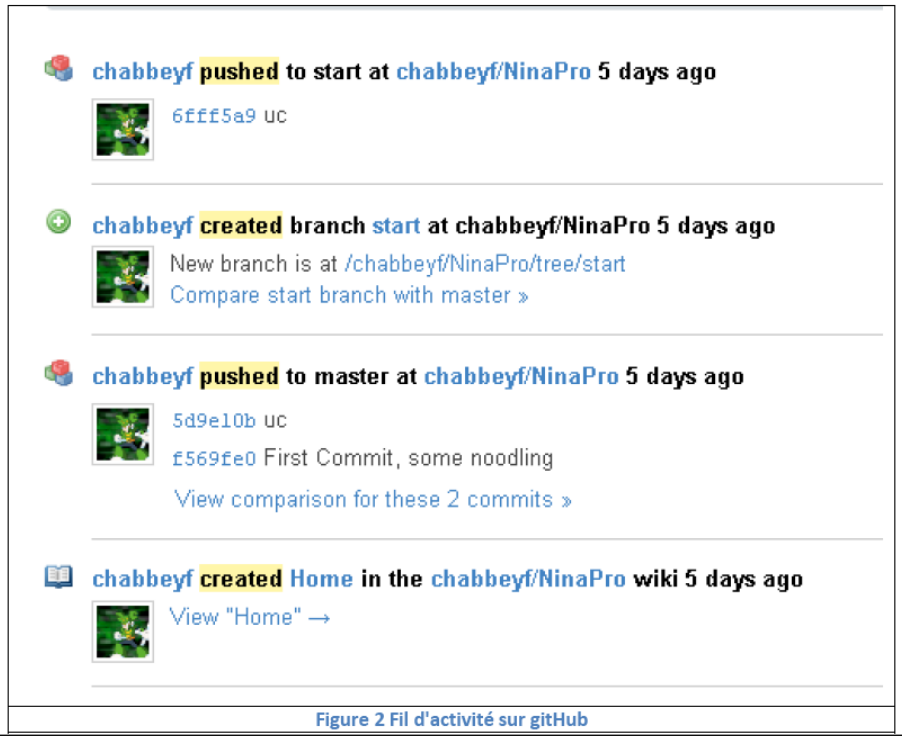


Figure 2 Fil d'activité sur gitHub

11.4 Annexe D : Planning

WP 1 (16.05.11- 13.06.11)

Durée : 4 semaines (20 période par semaine)

- Analyse du projet
- Constitution du cahier des charges
- Mise en place de l'environnement de travail
- Recherche d'outils
- Elaboration du premier modèle physique de données
- Base de données

Livrable

- Cahier des charges
- Planning
- Modèle physique

WP 2 (13.06.11 – 04.07.11)

Durée : 3 semaines (45 périodes par semaine)

- Création d'un système de gestion de droits
- Création d'une interface de saisie
- Création d'une interface de recherche

Livrable

- Application web reprenant les points ci-dessus
- Deuxième version de base de données
- Rapports de travail

WP 3 (04.07.11 – 22.07.11)

Durée : 3 semaines (45 périodes par semaine)

- Création d'un système d'extraction de données
- Création d'une interface permettant des analyses statistiques simples
- Correction/Amélioration des points du WP2

Livrable

- Application web mise à jour
- Rapports de travail

WP 4 (25.07.11 – 2.08.11)

Durée : 2 semaines (45 périodes par semaine)

- Correctif / Amélioration de l'application

- Mise en place des éléments facultatifs

Livrable

- Application web
- Rapports de travail

WP 5

Durée 1 semaine (45 périodes)

- Rédaction de la documentation
- Préparation de la présentation
- Préparation de la JAVADOC
- Déploiement sur serveur de production

Livrable

- Rapport
- Présentation orale (2 septembre)
- JavaDoc
- Application web

11.5 Annexe E : Tutoriel Selenium

Selenium : un tour d'horizon

Selenium est un framework de test pour les applications web disponibles pour la plupart des langages de programmation sur le marché. Il permet typiquement de vérifier la présence d'un élément sur une page, de simuler des interactions sur une page donnée, et bien d'autres choses encore. Selenium possède deux caractéristiques intéressantes :

1. Dans le cas de Java, il s'intègre avec les librairies de tests déjà existantes, ce qui permet alors de créer de véritables tests d'intégrations.
2. Il dispose d'un ID, sous la forme d'une extension Firefox, qui permet d'enregistrer une suite d'interactions avec le navigateur, puis de les restituer sous forme de classes de test. L'intérêt de cet IDE est immédiat; dans le cas où le développeur ne parvient pas à écrire un test, il peut lancer l'IDE, réaliser les interactions désirées dans le navigateur, et regarder le code produit.

Que tester ?

Deux types de tests sont typiquement réalisés avec Selenium :

- D'une part, nous pouvons tester la présence ou la non-présence d'un élément sur une page suite à une requête. Ce genre de test s'apparent à des tests unitaires, dans le sens où l'on ne teste que la partie "vue" de l'application
- D'autre part, comme Selenium s'intègre avec les tests classiques de JAVA, nous pouvons tester d'autres éléments de l'application; typiquement, il s'agira alors de tester les modifications survenues dans la base de données, suite à une série d'interactions utilisateurs, ou bien de vérifier qu'un utilisateur a été redirigé vers la bonne page suite à un login réussi. On parle ici de tests d'intégrations, car ces tests traversent toutes les couches de l'application.

Test de type "black box"

Selenium est un outil de type "black box" car il ne va pas réellement voir ce qui se passé à l'intérieur du serveur, il se limite à analyser les vues présentées par l'application. Les traitements effectués côté serveur restent hors de portée de Selenium, le serveur étant vu comme une "boite noire" fournissant par magie des pages WEB. Dans cette optique, Selenium ne permet que de repérer des erreurs côté serveur, mais pas de comprendre les raisons ayant amenés ces erreurs. D'autres outils doivent être utilisés pour pouvoir tester à l'intérieur du conteneur de l'application.

Différents Navigateurs

Un des intérêts de Selenium est qu'il permet de réaliser les tests sous différents navigateurs, ce qui permet de s'assurer de la portabilité d'une application donnée

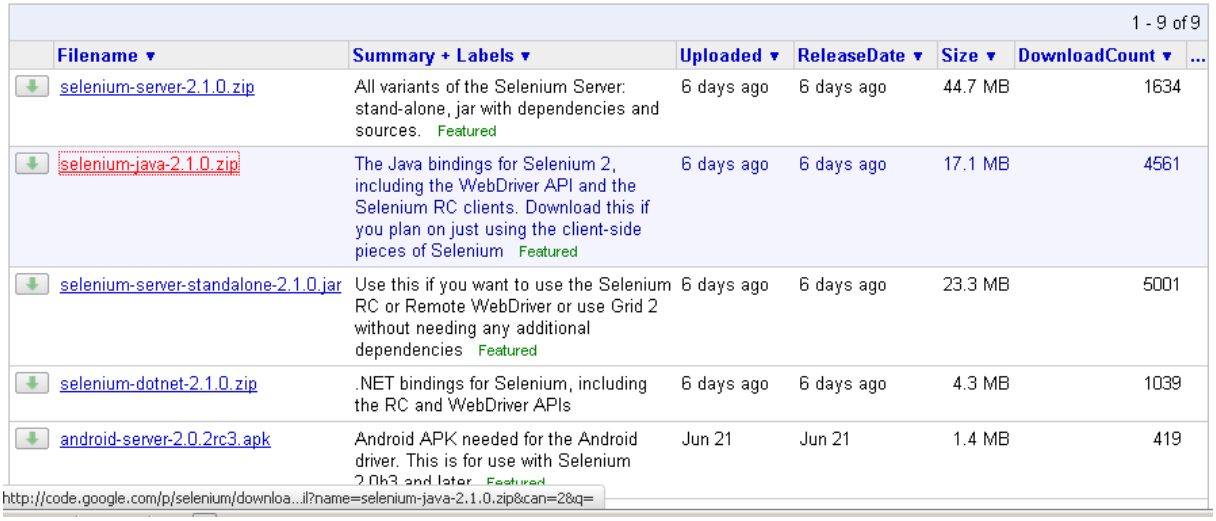
Premier pas

Le but de ce tutoriel est de montrer comment utiliser Selenium dans un environnement de développement de type Eclipse. Pour ce faire, nous monterons deux tests "classiques"; un test de

vue, où nous vérifierons la présence de tel ou tel élément sur une page Web, et un test d'intégration, où nous simulerons des interactions utilisateurs, puis vérifierons que la base de données aie bien enregistré les données saisies.

Installer Selenium

La première étape consiste à aller chercher les JARs nécessaire à l'exécution de tests utilisant Selenium. Le projet Selenium est hébergé sur Google Code, à l'adresse suivante: http://code.google.com/p/selenium/downloads/list

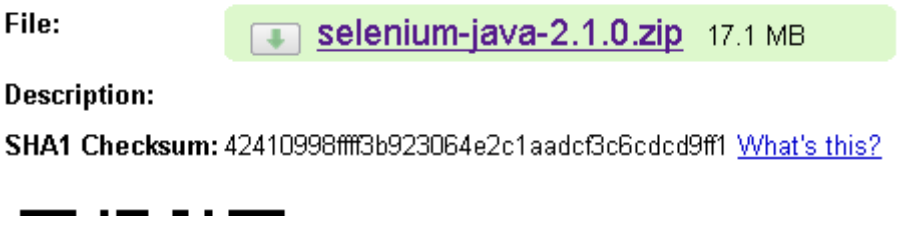


1 - 9 of 9

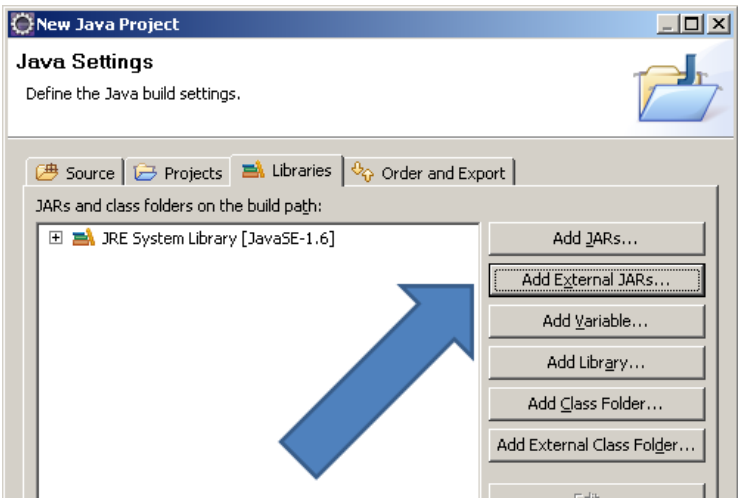
Filename ▾	Summary + Labels ▾	Uploaded ▾	ReleaseDate ▾	Size ▾	DownloadCount ▾	...
selenium-server-2.1.0.zip	All variants of the Selenium Server: stand-alone, jar with dependencies and sources. Featured	6 days ago	6 days ago	44.7 MB	1634	
selenium-java-2.1.0.zip	The Java bindings for Selenium 2, including the WebDriver API and the Selenium RC clients. Download this if you plan on just using the client-side pieces of Selenium Featured	6 days ago	6 days ago	17.1 MB	4561	
selenium-server-standalone-2.1.0.jar	Use this if you want to use the Selenium RC or Remote WebDriver or use Grid 2 without needing any additional dependencies Featured	6 days ago	6 days ago	23.3 MB	5001	
selenium-dotnet-2.1.0.zip	.NET bindings for Selenium, including the RC and WebDriver APIs	6 days ago	6 days ago	4.3 MB	1039	
android-server-2.0.2rc3.apk	Android APK needed for the Android driver. This is for use with Selenium 2.0b3 and later Featured	Jun 21	Jun 21	1.4 MB	419	

http://code.google.com/p/selenium/downloa...il?name=selenium-java-2.1.0.zip&can=2&q=

Il faut alors cliquer, sur le lien selenium-java-2.1.0.zip, une nouvelle fenêtre s'affiche, il faut de nouveau cliquer sur le le lien selenium-java-2.1.0.zip, qui démarrera le téléchargement de fichier.



Une fois le fichier téléchargé, il faut le décompacter sur le disque dur local. Il est temps de lancer eclipse, et de créer un nouveau projet java standard: Il faut alors indiquer à Eclipse que nous allons utiliser les librairies de Selenium. Dans la boite de dialogue de création de projets, il faut se déplacer sur l'onglet "Libraries", puis cliquer sur la button "Add External Jars…", comme indiqué dans la figure de la page suivante. Une boite de dialogue permettant de choisir des fichiers apparaît; il faut sélectionner tous les JARs présents dans l'archive téléchargée.



Première classe

```
import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.firefox.FirefoxDriver;
public class Essai {

    public static void main(String args[] ) {


        // Create a new instance of the firefox driver
        // Notice that the remainder of the code relies on the interface,
        // not the implementation.
        WebDriver driver = new FirefoxDriver();

        // And now use this to visit Google
        driver.get("http://www.google.com");


        // Find the text input element by its name
        WebElement element = driver.findElement(By.name("q"));


        // Enter something to search for
        element.sendKeys("Cheese!");


        // Now submit the form. WebDriver will find the form for us
        // from the element
        element.submit();


        // Check the title of the page
        System.out.println("Page title is: " + driver.getTitle());

    }
}
```

L'interface WebDriver est implémentée par différentes classe, chaque classe correspond à un navigateur du marché :
- FireFoxDriver pour FireFox
- OperaDriver pour Opera
- InternetExplorerDriver pour IE..

La méthode get permet d'accéder à une URL

La méthode sendKeys permet de simuler des frappes de clavier sur un élément

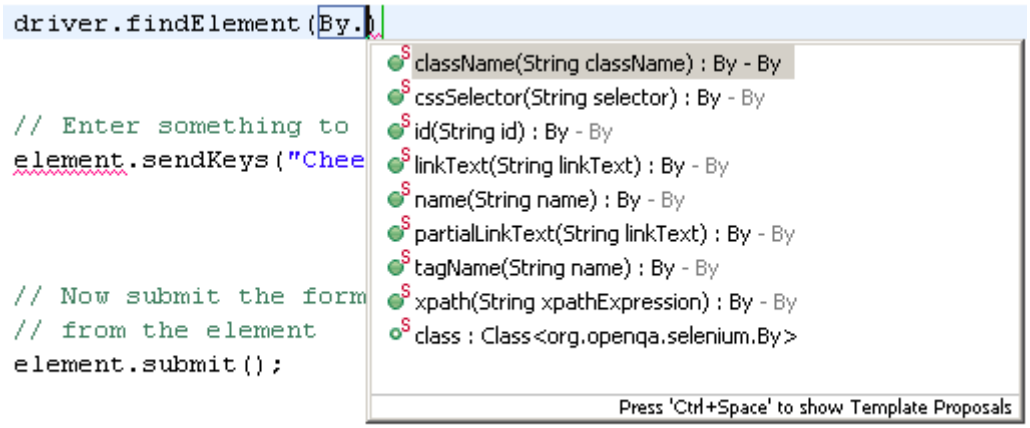
La méthode submit envoye le formulaire

Cet exemple de code se contente de charger la page Web du site google, de remplir le champ avec les lettres "Cheese !", puis d'envoyer le formulaire. Le code affiche alors le titre de la page. Cet exemple s'exécute comme une classe java standard; lors de l'exécution, vous pouvez remarquer

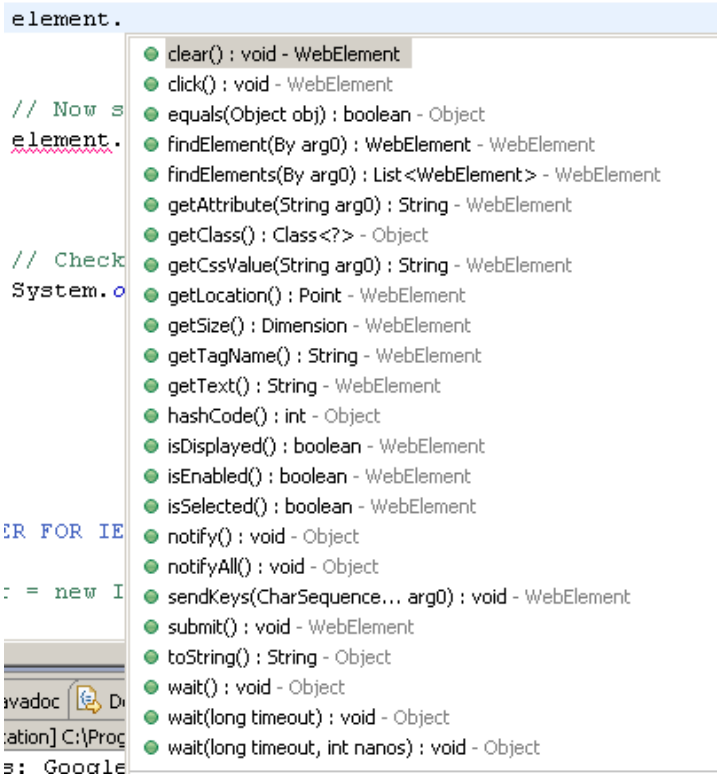
qu'une fenêtre FireFox s'ouvre, et que toutes les actions spécifiées dans le code s'exécutent à l'intérieur de cette fenêtre

La méthode findElement

La méthode la plus utilisée dans le cadre d'écriture de tests est la méthode findElement; cette méthode permet de trouver un élément dans la page. Elle permet d'utiliser différents critères de recherches grâce à la classe statique By.



La figure ci-dessous permet de voir toutes les méthodes recherches qu'offrent la classe statique By : on peut ainsi retrouver un élément par le nom de ses balises, son id, sa classe, ou en utilisant un sélecteur CSS, ce qui permet une très grande flexibilité. Une fois un élément retrouvé, diverses méthodes sont proposées, soit pour interagir avec cet élément, soit pour accéder à ses diverses propriétés.



Cette figure nous montre les différentes méthodes offertes par Selenium. Pour interagir avec cet élément, nous utiliserons les méthodes click() et sendKeys() qui permettent de simuler un clic souris, respectivement une/des frappes claviers. La méthode submit() permettra d'envoyer le formulaire contenant l'élément en question.

Les autres méthodes permettent de récupérer des informations qui seront typiquement utilisées dans des assertions, lors de l'écriture de test.

Premier test

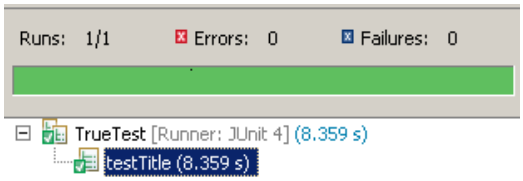
Pour écrire un test, il suffit simplement d'indiquer à Eclipse que nous souhaitons créer un nouveau cas de test, à la place d'une classe standard. L'écriture de tests, dans JUnit 4, se fait à l'aide d'utilisations :

- L'annotation @Before spécifie qu'une méthode sera appelée avant chaque test
- L'annotation @Test spécifie qu'une méthode correspond à un cas de test

La figure ci-dessous montre un cas de test très simple.

```
import org.junit.Before;
import org.junit.Test;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.firefox.FirefoxDriver;
import org.testng.Assert;
public class TrueTest {

    WebDriver driver;

    @Before
    public void setUp() {
            driver = new FirefoxDriver();

            // And now use this to visit Google
            driver.get("http://www.google.com");
    }

    @Test
    public void testTitle() {
        Assert.assertEquals(driver.getTitle(),"Google");
    }

}
```

Avant d'exécuter la méthode de test, la méthode setUp() est appelée, et va récupérer la page du site de Google. Une fois ceci fait, la méthode testTitle vérifie que le titre de la page corresponde bien à "Google". Pour lancer ce test, il faut l'éxecuter comme un test JUnit, et non comme une classe :



Une fois le test exécuté, la fenêtre de résultat s'affiche, et nous pouvons constater que le test s'est effectivement bien déroulé.

Tester le comportement d'une page

A titre d'exemple plus évolué, voilà un extrait des tests utilisés pour vérifier le bon comportement de la page de login.

```
@Test
public void loginFailedBadPwd() {
    login.sendKeys("\b\b\b");
    login.sendKeys("root");
    pwd.sendKeys("prout");
    button.click();

    System.out.println(driver.getPageSource());

    WebElement div = driver.findElement(By.id("errormessage"));
    System.out.println(div.getText());

    Assert.assertEquals(div.getText(), "Login or Password incorrect.");
    //System.out.println(driver.getCurrentUrl());
}

@Test
public void loginSuccess() {

    login.sendKeys("\b\b\b\b\b");
    login.sendKeys("root");
    pwd.sendKeys("admin");
    button.click();

    System.out.println(driver.getPageSource());

    Assert.assertEquals(driver.getTitle(),"Clinical");
}
```

Les variables *login* et *pwd* représentent les champs de la page de login, la variable *button* représente le bouton de login. La première méthode test un échec de login : si l'utilisateur rentre des mauvaises informations, un message d'erreur doit s'afficher sur la page.

La deuxième méthode vérifie que, en cas de login réussi, l'utilisateur soit bien redirigé sur une page dont le titre est "Clinical".

Un test d'intégration

Nous allons tester ici la base de données et la vue Web. Il faut donc configurer la connexion à la base de données avant de réaliser ce test. Comme nous utilisons JPA et MySQL, il faut rajouter les JAR correspondants au projet.

Mise en place

Le code d'initialisation charge la page de login, log un utilisateur root, puis crée une unité de persistance.

```
@BeforeClass
public static void setup() throws InterruptedException {
        System.out.println("STATIC INIT");
        driver = new FirefoxDriver();
        WebElement login;
        WebElement pwd;
        WebElement button;
        driver.get("http://localhost:8080/NinaWeb");
        login = driver.findElement(By.id("form:login"));
        pwd = driver.findElement(By.id("form:pwd"));
        button = driver.findElement(By.id("form:button"));

        login.sendKeys("root");
        pwd.sendKeys("admin");
        button.click();

        Thread.sleep(100);

        driver.getPageSource();

        setUp();

}

public  static void setUp() {
    driver.get("http://localhost:8080/NinaWeb/faces/form2.xhtml");
    emf = Persistence.createEntityManagerFactory("NinaPU");



    em = emf.createEntityManager();
}
```

La méthode prepare() remplit les champs nécessaires à la création d'un sujet, puis clique sur le bouton d'envoi du formulaire. La méthode essaie alors de récupérer le sujet correspondant dans la base de données. C'est alors que les tests peuvent réellement commencer; le but étant de vérifier que ce qui a été rentré dans le formulaire correspond à ce qui a été sauvé en base de données. Pour chaque champ de la base de données, une méthode de test est créée, afin de permettre une meilleure lecture des résultats

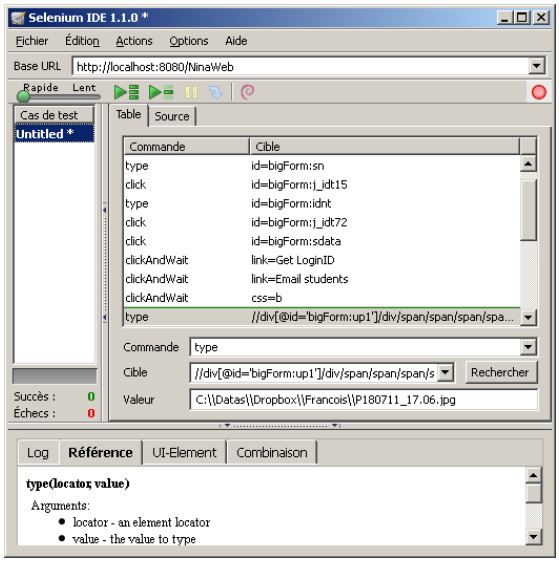
```
public void prepare() {
    driver.navigate().to("http://localhost:8080/NinaWeb/faces/form2.xhtml");
    WebElement sn = driver.findElement(By.id("bigForm:sn"));
    sn.sendKeys("\b1999");
    sn = driver.findElement(By.id("bigForm:firstname"));
    sn.sendKeys("FN");
    sn = driver.findElement(By.id("bigForm:lastname"));
    sn.sendKeys("LN");

    driver.findElement(By.id("bigForm:handradio:0")).click();
    driver.findElement(By.id("bigForm:genderradio:0")).click();
                . . .
    driver.findElement(By.id("bigForm:sdatas")).click();

    emf.getCache().evictAll();

    Query q = em.createQuery("Select s FROM Subject s WHERE s.subjectNumber=?
    q.setParameter(1, 1999);
     su = (Subject) q.getSingleResult();
    sd = su.getSubd().get(0);
```

Envoi du formulaire

Récupération des infos dans la DB

```
@Test
public void checkInDbSn() {
    prepare();
    Assert.assertEquals(su.getSn(),"1999");
}
@Test
public void checkInDbFirstName() {
    Assert.assertEquals(su.getFirstName(), "FN");
}
@Test
public void checkInDbLastName() {
    Assert.assertEquals(su.getLastName(), "LN");
}

        . . .
```

Selenium IDE



Selenium possède un environnement de développement intégré, sous la forme d'extensions pour FireFox. Cette extension est particulièrement utile lors de la création de tests plus évoluées; par exemple pour tester l'upload ou download. Cette extension permet d'enregistrer les différentes interactions de l'utilisateur avec la page Web, puis d'exporter le tout sous forme de classe Java. Dans le cadre du projet, l'IDE a été utilisé pour tester l'upload des fichiers vers le serveur

11.6 Annexe F : Rapport PDF généré

Acquisition report

1. Subject Information

Subject number:1
Handness :Right
Created By :rootsd
Is amputed :false

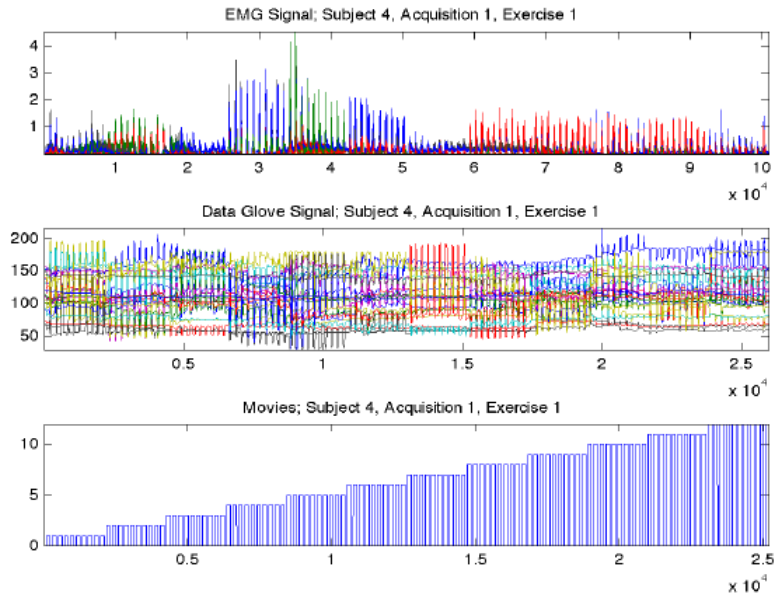
2. Subject Information Related to Acquisition

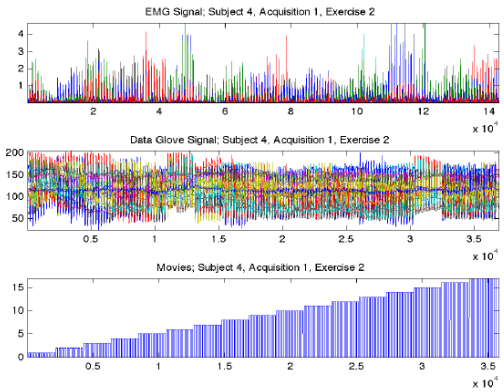
Subject temperature :123.0
Weight :75.0
Height :170.0
Work :Researcher
Skin resistance :123
Time of last meal :12:30

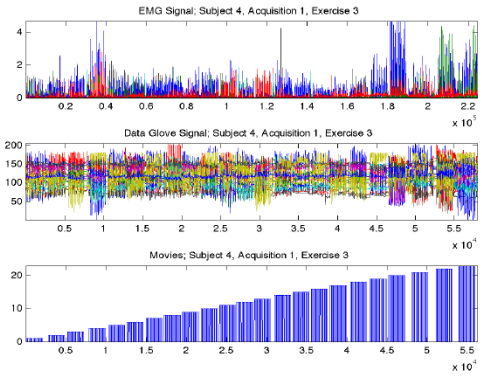
3. Acquisition information

Room temperature :123.0
Room humidity :123.0

4. Pictures



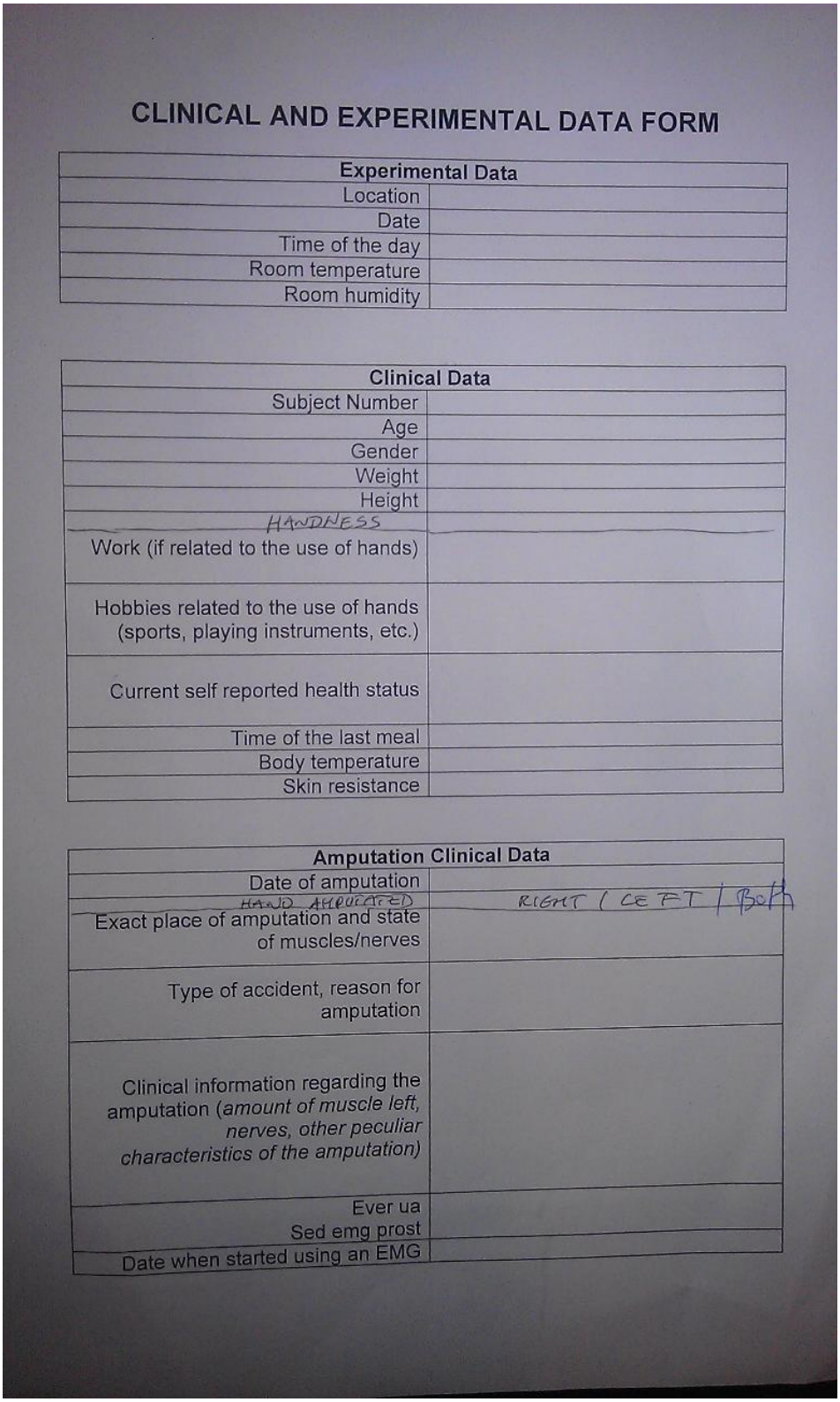




11.7 Annexe G : Fichier XML généré

```
- <experiment>
    <subject hand="Right" subject_number="1"/>
  - <experiment_datas id="123">
      <temperature>123.0</temperature>
      <humidity>123.0</humidity>
      <location>Sierre</location>
      <date_of_exp>Mon Aug 01 09:11:00 GMT+01:00 2011</date_of_exp>
    </experiment_datas>
  - <subject_datas>
      <age>31</age>
      <hobbies>Guitar</hobbies>
      <weight>75.0</weight>
      <height>170.0</height>
      <work>Researcher</work>
      <status>Good</status>
      <body_temperature>123.0</body_temperature>
      <skin_resistance>123</skin_resistance>
      <last_meal>12:30</last_meal>
    </subject_datas>
  </experiment>
```

11.8 Annexe H : Formulaire d'acquisition



CLINICAL AND EXPERIMENTAL DATA FORM

Experimental Data	
Location	
Date	
Time of the day	
Room temperature	
Room humidity	

Clinical Data	
Subject Number	
Age	
Gender	
Weight	
Height	
HANDNESS	
Work (if related to the use of hands)	
Hobbies related to the use of hands (sports, playing instruments, etc.)	
Current self reported health status	
Time of the last meal	
Body temperature	
Skin resistance	

Amputation Clinical Data	
Date of amputation	
HAND AMPUTATED	RIGHT / LEFT / Both
Exact place of amputation and state of muscles/nerves	
Type of accident, reason for amputation	
Clinical information regarding the amputation (*amount of muscle left, nerves, other peculiar characteristics of the amputation)*	
Ever ua Sed emg prost	
Date when started using an EMG	

11.9 Annexe I : Rapport des heures

Semaine 1	
Analyse	16
Semaine 2	
Analyse	5
Recherche	5
Essais	5
Semaine 3	
Design	16
Semaine 4	
Développement DB et JPA	5
Tests	5
Semaine 5	
Recherche Framework test	10
Développement interface graphique	10
Développement Gestion users	10
Semaine 6	
Développement interface admin	22
Développement formulaire saisie	23
Semaine 7	
Tests	15
Développement grille sujets	35
Semaine 8	
Tests	6
Développement grille export	40
Semaine 9	
Tests	5
Déboggage	30
Développement des graphiques	10
Rédaction	10
Semaine 10	
Rédaction	25
Développement	25
Semaine 11	
Rédaction	35
Développement	12
Total	**380**

11.10 Annexe J : Modèle physique des données