Tuteur : John TRANIER
Encadrant : Jan-Georg SMAUS

Le 23 août 2014

Participation au développement d'une application Java EE

dédiée à la conception et à la diffusion de supports pédagogiques

Mathieu SOUM

Stage du 5 mai au 31 août 2014
Chez TiceTime

Master 1 Informatique – Spécialité Développement logiciel / 2013 - 2014

Tuteur : John TRANIER
Encadrant : Jan-Georg SMAUS

Le 23 août 2014

Participation au développement d'une application Java EE

dédiée à la conception et à la diffusion de supports pédagogiques

Mathieu SOUM

Stage du 5 mai au 31 août 2014
Chez TiceTime

Master 1 Informatique – Spécialité Développement logiciel / 2013 - 2014

Je souhaite remercier ici l'équipe pédagogique de la formation pour nous permettre de réaliser ce stage. C'est toujours bénéfique de participer à une expérience professionnelle pour nos compétences mais aussi pour notre savoir-être en entreprise.

Je tiens aussi à remercier Franck Silvestre et John Tranier pour m'avoir permis de réaliser ce stage. Ils m'ont accueilli dans leur équipe et m'ont épaulé tout au long de cette expérience et je leur en suis reconnaissant pour ça.
J'aimerais remercier Vincent Tertre pour son expertise et l'aide qu'il m'a apporté durant ces trois mois de stage.

Enfin, je souhaite remercier aussi ceux qui m'ont accompagné durant ce stage, certes plus moralement que techniquement, je veux parler de Clément, Ophélie, (petit) Clément, Antoine et d'autres que j'oublie sûrement.

Sommaire

Introduction

Contexte du stage

Ce stage s'inscrit dans le cadre de ma formation de Master 1 Informatique parcours Développement Logiciel. Il s'étend sur 3 mois du 5 mai au 1er août 2014.

Durant cette période, j'ai rejoint l'équipe de TiceTime pour travailler sur un outil d'édition de contenu pédagogique. Cet outil est à destination des enseignants et a pour but de leur permettre de créer des cours dans un format de type diaporama et ensuite de l'exporter ou le diffuser dans plusieurs autres formats (PDF, RTF, Diaporama HTML, *Paquetage de Contenu IMS*).

L'intérêt pour l'entreprise de proposer ce stage était principalement d'avancer dans le développement de cet outil. Il n'y avait donc pas d'objectif à proprement dit qu'il faudrait avoir atteint au bout des 3 mois de stage. Les fonctionnalités à implémenter se sont plutôt ajoutées au fil de l'avancement du stage et de la réalisation des fonctionnalités précédentes.

Structure du rapport

Ce rapport de stage sera découpé en 5 chapitres.

Premièrement, un rappel du contexte dans lequel s'est déroulé ce stage. Une description de l'entreprise, de l'équipe avec laquelle j'ai travaillé et des différents outils utilisés.

Ensuite, une présentation détaillée du sujet en précisant le détail des fonctionnalités que j'ai implémentées et de la démarche méthodologique employée.

Puis, le travail effectivement réalisé. On s'attardera essentiellement l'aspect technique, la description fonctionnellement étant décrite dans le chapitre précédent.

Enfin, un bilan personnel pour prendre du recul sur cette expérience et la contextualiser par rapport à ma formation, mon parcours professionnel et mon futur projet professionnel.

Conseils de lecture

Ce document est à destination de mon maître de stage, mon encadrant universitaire ainsi qu'au jury de ma soutenance qui l'aura à disposition. Pour faciliter sa lecture, voici quelques précisions quant aux détails typographiques utilisés dans ce document.

Les citations relatives à du code source telles que les noms de classes par exemple seront écrit avec une `police à chasse fixe`.

Un glossaire à la fin du document répertorie certains termes techniques qui nécessiteraient une définition plus précise. Les mots apparaissant dans ce glossaire seront *en écriture penchée* dans le reste du document.

Chapitre 1

Contexte

1.1 Ticetime

Ticetime est une jeune entreprise créée en 2011 et composée aujourd'hui de deux salariés :

Franck Silvestre Directeur général. Il occupe également un poste de maître de conférence associé à l'Université Paul Sabatier de Toulouse

John Tranier Directeur technique. Il donne également des cours à l'Université Montpellier II.

Historique

juin 2009 Franck occupe la fonction de directeur technique au sein d'OMT-Fylab et recrute John Tranier comme architecte sur le projet Open ENT - Lilie. Cette rencontre marque le début d'une expérience professionnelle unique et d'une amitié.

janvier 2011 Création de la société Ticetime par Franck Silvestre après son départ d'OMT/-Fylab. Franck travaille comme consultant et développeur indépendant principalement sur le projet TD Base, brique pédagogique de l'Open ENT.

janvier 2012 Franck intègre à mi-temps l'Université Toulouse III en qualité de maître de conférences associé. L'idée du projet elaastic et son premier prototype germe devant la frustration rencontrée lors de la conception des supports de cours.

juin 2013 John Tranier intègre comme associé et directeur technique Ticetime pour participer au développement de la société et pour prendre en charge la mise en œuvre industrielle d'elaastic. Ticetime, partenaire de Setec IS, accompagne la région PACA dans la mise en œuvre de son Environnement Numérique Éducatif.

1.2 L'équipe

Durant ce stage, j'ai eu l'occasion de travailler avec :

Franck Silvestre directeur général.
John Tranier directeur technique.
Vincent Tertre alternant en Master 2 DL à Paul Sabatier.

1.3 Les outils utilisés

J'ai développé dans un environnement Fedora 20 (Heinsenbug). Nous travaillions avec divers outils dont certains distants (précisés dans la liste ci-après ; cette liste comprend aussi les frameworks logiciels utilisés).

CloudBees *PaaS* qui embarque le système d'intégration continu Jenkins et qui permet le déploiement automatique d'application web et mobile.

CodeNarc Analyseur de code spécialisé pour Groovy.

Git Gestionnaire de version.

Grails Framework d'application web utilisant le langage Groovy. Il suit le paradigme « codage par convention » masquant certaines configurations au développeur.

GVM (the **G**roovy en**V**ironnement **M**anager) Outil qui permet l'installation d'autres outils relatifs à un environnement Groovy/Grails à des versions particulières et de les mettre à jour simplement.

IntelliJ IDEA *IDE* avec lequel j'ai rédigé tout le code source que j'ai produite. La gestion des projets et des environnements Grails apporte beaucoup de confort de travail. Nous utilisions l'outil de vérification de couverture de code intégré à l'IDE.

Jira *SaaS* de gestion de projet et de gestion des bogues/incidents. Il fournit une interface « tableau Kanban » que nous avons beaucoup utilisé.

Spock Un framework de test et de spécification pour les applications Java et Groovy.

UMongo Application permettant d'explorer et de modifier une base de donnée Mongo. Mongo est une base de données NoSQL regroupant des collections de documents au format Json.

FIGURE 1.1 – Quelques logos des outils utilisés

Chapitre 2

Présentation du sujet

Ayant intégré une équipe travaillant sur un projet existant, je n'ai pas réalisé un travail de mon côté mais plutôt ajouté de nouvelles fonctionnalités à ce projet. C'est pourquoi ce chapitre reprend les différentes fonctionnalités que j'ai développées après une petite introduction du projet global.

2.1 Elaastic

Elaastic est un outil pour concevoir et diffuser des contenus pédagogiques. Il permet de concevoir des supports pédagogiques structurés, interactifs et indépendants du format de destination. Il offre la possibilité de diffuser ces supports sous forme de diaporama ou sous forme de publication au format livre ou au format site interactif dans un *ENT* ou un *LMS*.

Les cours créés avec Elaastic sont organisés en arbre sur trois niveaux :

- Section
  - Sous-section
    - Unité

Des ressources comme des images ou des documents externes peuvent être ajoutés à ces cours pour les enrichir. Nous verrons également qu'il est maintenant possible d'ajouter des questions interactives en tant qu'unité de cour.

D'un point de vue technique, Elaastic se découpe en deux parties. Une partie Grails pour le corps et la partie *back end*. Le système de persistance des données est assurée par Mongo pour la version de développement. La version déployée en production s'appuie sur une base de données Postgre SQL. Côté *front end*, l'interface utilisateur est conçu grâce à AngularJS, un framework Javascript.

FIGURE 2.1 – Logo d'Elaastic

Pour ma part, j'ai essentiellement travaillé sur le corps et la partie back end. Les sections qui suivent détaillent les fonctionnalités sur lesquelles j'ai travaillé.

2.1.1 IMS Content Packaging

IMS Content Packaging est un format standard d'échange d'information structurées en arbre – chapitre, section, sous-section par exemple – défini par l'IMS GLC [1]. Cette structure correspond aux cours créé avec Elaastic et s'adapte bien en format d'exportation. Une des motivations à fournir ce format d'exportation était que ce format est compris par Moodle, plateforme utilisée par Franck et John dans leur activité d'enseignement. Une fois le package généré, il peut être directement importé dans Moodle.

Mon travail sur cette fonctionnalité a été d'intégré au système d'export déjà existant la possibilité d'exporté un cours créé avec Elaastic au format IMS Content Packaging.

2.1.2 WebDAV

WebDAV est un protocole d'échange et d'édition collaborative de fichier. Là encore, la motivation était l'intégration à d'autres LMS comme Moodle. En effet, Moodle peut être configuré pour récupérer se connecter à des serveurs WebDAV pour récupérer ou déposer des fichiers.

L'objectif de cette fonctionnalité était de permettre à l'utilisateur de configurer un serveur WebDAV afin de pouvoir déposer des cours dans les différents formats d'exportation disponibles.

2.1.3 Artifact

Un problème qui est apparu avec la complexification des exportations est le temps que met l'application à générer ces documents. Un système de cache a été imaginé pour ne pas avoir à réexporter des cours qui n'aurais pas été modifiés entre deux exportations. Ces cours généré une première fois dans un format donné correspondent à un artéfact (*artifact* en anglais).

Mon travail sur les artéfacts a été de définir la structure d'un artéfact pour la persistance des données et de mettre en en place le système de cache afin de ne pas utiliser du temps processeur inutilement et fournir un service plus rapide quand cela est possible.

2.1.4 Diffusion

Un des objectifs d'Elaastic est de pouvoir diffuser des cours dans plusieurs formats afin que d'autres personnes – notamment les étudiants – puisse y accéder en dehors des cours. C'est pourquoi il a fallu définir ce qu'était une diffusion d'un point de vu technique et comment l'intégrer à l'application. Une première version simple d'une diffusion permet d'accéder à un cours dans une version donnée dans les formats d'exportation disponibles.

Mon travail sur ce point a été de créer une page contenant une diffusion pour un cours particulier regroupant les différents formats disponibles dans lesquels le cours est disponible.

2.1.5 Question interactive

Franck a travaillé sur Tsaap-Notes, un outil de micro-blogging pour la prise de note collaborative durant les cours magistraux. Cet outil permet de créer des notes dans un format de questions interactives auxquelles les étudiants répondent en temps réel durant les cours.

Dans une optique de couplage de ce système de prise de note collaborative avec les cours créés depuis Elaastic, il faudrait que l'utilisateur puissent créer des unités de cour dans ce format de question interactive.

Mon travail sur les questions interactives a été de rendre possible la saisie par l'utilisateur d'une unité de cours spéciale qui sera une question interactive. Les questions seront rédigées au format Moodle GIFT puis analysées et converties pour l'intégration dans les formats d'exportations.

1. IMS Global Learning Consortium

2.2 Démarche méthodologique

Pour chaque activité réalisée, plusieurs marqueurs de suivi de projet étaient à faire suivre et à mettre à jour au fur et à mesure de l'avancement des tâches.

Pour la gestion de projet et du code source nous utilisions Jira et Git de manières très couplées. Jira nous permet de découper et d'organiser le travail à faire et les anomalies détectées. Ainsi nous avions des *stories*, des tâche et sous-tâches, ou des bogues. Il attribue à chaque élément un identifiant unique de la forme EL-XXX [2].

Là où le couplage avec Git intervient, c'est au niveau du nommage des branches. Nous organisions chaque branche de développement en fonction de l'élément Jira auquel elle faisait référence. Ainsi, chaque branche correspondait à une et une seule story. De plus, nous gardions dans cet identifiant dans les messages de commits pour permettre de descendre au niveau de la tâche ou de la sous-tâche quand des précisions sur celle ci était nécessaire.

Une fois le développement de la fonctionnalité terminée, les tests correspondants écrits et validés et le code nettoyé, la branche passait en *« to review »* sur notre tableau Kanban. Une nouvelle branche Git était alors créée avec l'historique des commits nettoyé. Cette branche reprenait le nom de la précédente et contenait en plus le marqueur « ready ».

Une fois le code revu par John, il me faisait des retours sur ce qu'il avait pu noter. J'apportais ensuite les modifications retenues et je mettais à jour la branche Git correspondante pour l'intégration à la branche principale de développement.

De manière plus globale mais toujours dans la méthodologie de travail, nous faisions des « daily meeting » avec John et Vincent afin de tenir compte du travail fait la journée passée et de ce sur quoi nous allions travailler pour la journée à venir.

Un dernier point sur la démarche de travail que je vais aborder ici est la phase de test. Pour chaque fonctionnalité développée, une contrainte de validation était l'écriture de tests pour une couverture de code optimale. Pour vérifier cette couverture de code, nous utilisions le plugin de couverture de code fournit avec IntelliJ IDEA.

Pour l'écriture des tests – ou plus précisément des spécifications – nous avons utilisé le framework Spock. C'est un framework pour écrire des spécifications. Il permet de spécifier le comportement de chaque méthode d'une classe dans une classe de test – spécification. Ce framework permet de faire des tests sur le comportement de la classe testée en simulant ses interactions avec de faux objets, des *mock objects*. Pour faire cela, il nous permet de suivre le principe du « given, when, then ». Ce principe consiste à spécifier un état de base – given – à lui appliquer un stimulus, un appel de fonction sur un objet réel par exemple – when – et à vérifier que les différentes interactions fonctionnent et que l'objet réel a bien changé d'état – then.

2. EL étant l'abréviation du projet et XXX un numéro unique

Chapitre 3

Travail réalisé

Ce chapitre sera consacré aux solutions retenues et à leur mise en œuvre d'un point de vue essentiellement technique. Chaque fonctionnalité sera séparée dans une section propre et sera développée en suivant le schéma suivant :

— Recherche et analyse
— Solution mise en œuvre
—

Je n'aborderai pas ici la phase de test car je l'ai développée dans la partie démarche méthodologique. En effet, cette phase est la même pour chaque fonctionnalité développée. Je préfère donc me concentrer ici sur l'analyse et les solutions apportées pour chaque fonctionnalité.

3.1 IMS

3.1.1 Recherche et analyse

Dans un premier temps, il m'a fallu me renseigner sur cette norme que je ne connaissais pas du tout. Il est sorti de cette étude que les paquetages de contenu IMS sont des archives au format standard *zip*. Or, un avantage pour moi est que la logique de création d'une archive zip existe déjà dans Elaastic et est utilisée pour l'export de cours au format HTML empaqueté dans une archive zip.

Cependant, un paquetage IMS n'est pas qu'une archive zip quelconque. Une organisation spécifique est nécessaire à l'intérieur de cette archive. Cette organisation est personnalisable et renseignée dans un fichier `manifest.ims` placé à la racine de l'archive. Ce fichier contient des données au format XML décrivant l'organisation du cours empaqueté, les différentes ressources contenues dans l'archive et des méta-données relative à ce cours (auteur, date, licence, etc.). La syntaxe de ce fichier nous permet de lier des ressources à chaque élément du cours. Ainsi, en reprenant la structure de nos sections, sous-sections, unités dans ce fichier manifest et en liant le contenu des unités au format HTML de notre cours en tant que ressources de cette unité, nous aurons un paquetage de contenu IMS complet qui pourra être lu et affiché par le LMS de l'utilisateur reprenant le contenu et les ressources tel qu'il l'a rédigé dans Elaastic.

IMS GLC nous fournit également un schéma XSD pour valider la construction de ce fichier manifest, ce qui nous permettra de vérifier que ce dernier est conforme à la norme et que notre archive sera reconnue par les applications qui l'utiliseront.

3.1.2 Solution mise en œuvre

Pour la création de l'archive zip, j'ai donc repris la logique utilisée pour créer l'archive pour l'export au format HTML « zippé ». Ce système utilise des flux et des entrées pour ajouter

les fichiers à l'archives. Les flux lisent des données binaires tandis que les entrées séparent les différents fichiers de l'archive. La logique est donc la suivante :

On crée un `ZipOutputStream` qui sera notre flux dans lequel nous enverrons le contenu des différents fichiers. Ensuite, on crée une `ZipEntry` qui correspondra à un fichier dans l'archive en lui passant le chemin de ce fichier à l'intérieur de l'archive. Si des répertoires doivent être créés pour satisfaire le chemin du fichier, ils le seront automatiquement. On ajoute cette entrée au flux grâce à une méthode `newEntry()`. Puis on envoie les données correspondant au contenu du fichier de la `ZipEntry` directement dans le flux grâce aux opérateurs flux du langage.

Pour la création du fichier `manifest.ims`, j'ai utilisé la classe `MarkupBuilder` qui permet de construire une structure de données au format XML en spécifiant le nom, les attributs, les enfants et le contenu de chaque élément XML. Cette classe prend un `Writer` à la construction que j'ai instanciée en tant que `StringWriter`.

La première section du fichier manifest est la partie sur les méta-données. Cela se traduit par la section `<metadata>` du fichier manifest.

J'ai ensuite implémenté une structure de parcours du cours qui va remplir le `MarkupBuilder` en fonction de si nous sommes dans une section/sous-section ou dans une unité de cours.

Dans le cas d'une section/sous-section, le `MarkupBuilder` ajoutera une entrée `<item>` dans laquelle sera ajouté le titre de la section/sous-section et ses enfants.

Dans le cas d'une unité, le `MarkupBuilder` ajoutera une entrée `<item>` dans laquelle sera ajouté simplement le titre de cette unité. Le contenu de cette unité sera ajouté comme une entrée `<resource>` dans la partie correspondante du manifest. Toutefois, un attribut `res` est ajouté à la balise `<item>`. La valeur de cet attribut est le nom de la ressource correspondant au contenu de l'unité. Cet attribut lie les deux éléments et permettra de retrouver le contenu de l'unité lors de la lecture du paquetage par une application tierce.

Viens enfin la partie correspondant aux ressources. Les ressources sont les fichiers de contenu de chaque unité de cours au format HTML. Les noms de ces ressources doivent correspondre à ceux renseignés dans les balises `<item>` des unités de cours.

Une fois notre `MarkupBuilder` complet, l'extraction du contenu en tant que chaîne de caractère se fait via le `StringWriter`. Avec cette chaîne construite, nous pouvons vérifier que sa construction est correcte en la validant avec le schéma XSD fournis par IMS GLC.

Si la construction n'est pas validée, l'export est annulé.

Si la construction est validée, l'export se poursuit. Les fichiers de contenu de cours au format HTML sont ajoutés à l'archive zip ainsi que les images référencées dans ces contenus.

Maintenant que nous avons une archive complète, nous pouvons la retourner à l'utilisateur dans une réponse HTTP en précisant `application/zip` comme `ContentType`.

3.2 WebDAV

3.2.1 Recherche et analyse

Comme pour l'IMS Content Packaging, il m'a d'abord fallu me renseigner sur ce qu'était WebDAV.

WebDAV est une extension du protocole HTTP. Il comprend un ensemble de nouvelles méthodes qui s'utilisent avec le protocole HTTP. Ces méthodes permettent de créer, modifier et supprimer des fichiers sur le serveur WebDAV. Elles permettent aussi de consulter/modifier

les propriétés de ces fichiers. Un système de gestion de version est aussi disponible via des commandes de verrou sur les fichiers pour protéger des écritures concurrentes.

Avant tout, j'ai cherché si un plugin Grails n'existait pas déjà gérant les échange WebDAV. Et il s'est avéré qu'un tel plugin existe sur un dépôt Maven. Or nous utilisons le système de dépendance de Maven pour les dépendances de notre projet. Ce plugin s'appelle Sardine, est codé en Java et est disponible sur GitHub [1]. J'ai donc enquêté sur la fiabilité de ce plugin en recherchant des avis et d'éventuels manques vis-à-vis du protocole. La fréquence des commits et les différents avis que j'ai pu recueillir m'ont convaincu quant à l'utilisation de ce plugin.

3.2.2 Solution mise en œuvre

Pour cette fonctionnalité, j'ai développé un service Grails qui interface le plugin Sardine et fournit des méthodes pour créer, mettre à jour, supprimer des fichiers sur un serveur WebDAV.

Pour tester le bon fonctionnement de cette fonctionnalité, j'ai également mis en place un serveur WebDAV sur ma machine.

Cependant, d'autres fonctionnalités prioritaires étaient à développer et intégrer avant celle-ci. Mon travail est donc dans une branche du git en attendant d'être complété et intégré au projet.

3.3 Artifact

3.3.1 Recherche et analyse

Le besoin d'avoir des artéfacts vient du fait que la transformation d'un cours en format de sortie pour l'utilisateur final est trop long pour une utilisation fluide de l'application. Ce temps de transformation sera encore plus évident quand nous aborderons la partie sur la diffusion d'un cours.

Un artéfact est une version précise d'un cours, transformé dans un format particulier. Le nom `Artifact` correspond à la collection/table qui sera stockée en base. Ce système de persistance permettra de faire office de cache et de retourner à l'utilisateur une version transformée du cours plus rapidement.

Il est également nécessaire d'ajouter une contrainte d'unicité sur les artéfacts afin de garder l'intégrité des données et n'avoir qu'un document à retourner à l'utilisateur pour une version de cours et un format donné. Cette contrainte portera donc sur le cours, sa version et le format de l'artéfact.

3.3.2 Solution mise en œuvre

Pour la mise en place de cette solution, il y a deux étapes.

La première étape est de créer la classe du domaine `Artifact` qui correspond aux champs qui seront stockés en base et aux contraintes sur ces paramètres comme la contrainte d'unicité exposée plus haut. Ensuite, j'ai créé une classe `ArtifactService` de service Grails qui assure les échanges transactionnels avec la base de données. Cette classe suit les mécaniques CRUD [2] de Grails. Elle fournit une méthode `save` pour ajouter une nouvelle entrée et des méthodes `findBy` pour récupérer un artéfact en fonction d'une ou plusieurs des composantes de la contrainte d'unicité.

La deuxième étape est d'ajouter une vérification de l'existence d'un artifact avant l'export d'un cours. La logique de ce cache avec les artéfacts est la suivante : Lorsqu'un utilisateur

1. `https://github.com/lookfirst/sardine`
2. **C**reate **R**ead **U**pdate **D**elete

demande l'export d'un cours dans un format particulier, si un artéfact correspondant à ce cours, dans cette version et dans ce format existe en base, on peut directement le renvoyer à l'utilisateur avec le bon `ContentType`. Par contre, si aucun artéfact ne correspond en base, on doit effectuer la transformation. Une fois cette transformation effectuée, on stocke le binaire généré en base avec le cours, sa version et le format correspondant. Ainsi, à la prochaine demande, l'élément sera dans un « cache » et récupéré directement.

Le développement de cette deuxième partie a montré une duplication de code avec l'intégration de ce système de cache. En effet, le code est le même pour chaque exporteur. Nous avons donc mis en place un système plus générique au niveau des exporteurs. La factorisation de ces exporteurs nous a permis d'isoler une classe abstraite principale `CourseExporter` et différentes implémentations : `ImsCourseExporter`, `PdfCourseExporter`, `OnlineSlideShowCourseExporter`, etc. De plus, nous avons mis en place une « factory » pour la création de ses exporteurs. Cette factory ajoute en fonction des besoins des décorateurs pour l'export au format zip ou la mise en cache. Ces décorateurs implémentent eux aussi la classe abstraite `CourseExporter` Ainsi, nous nous retrouvons avec une seule classe `ZipCourseExporter` et une seule classe `CacheCourseExporter`. En les connectant, on peut mettre en cache et empaqueter dans une archive zip n'importe quel format d'exportation avant de le livrer à l'utilisateur.

3.4 Diffusion

3.4.1 Recherche et analyse

La possibilité de diffuser un cours est une des fonctionnalités capitales d'Elaastic. Cette diffusion permettra aux étudiants d'accéder à des versions en « lecture seule » de ces cours dans les différents formats disponibles à l'exportation. Une diffusion correspondra donc à un cours dans une version particulière. Avec ce système en place, on peut rapidement imaginé une charge importante sur le serveur lorsque plusieurs groupes d'étudiants voudront récupérer la même version d'un cours en particulier. C'est avec des échelles comme celle-là que l'importance du système de cache avec les artéfacts se justifie pleinement.

Les diffusions seront stockées en base correspondant à un cours, sa version, un nom d'affichage, un nom normalisé pour l'URL et un type de diffusion. Pour l'instant, nous n'avons défini qu'un seul type de diffusion : `ELAASTIC_SIMPLE`. C'est celui mentionné d'ici là dans ce document. Il permet de récupérer un cours dans une version particulière et dans les formats disponibles à l'exportation.

3.4.2 Solution mise en œuvre

Mon travail sur cette fonctionnalité a principalement été de mettre une interface sur « externe » à l'éditeur pour obtenir un cours dans un des formats d'exportation. J'ai donc créé un contrôleur Grails, `PlayerController`, qui se charge de formater la sortie de la diffusion en fonction de la requête de l'utilisateur. Avec ce contrôleur, j'ai ajouté les entrées correspondantes dans le fichier `UrlMapping` de Grails et j'ai créé une vue pour lister les exports disponibles dans cette diffusion. L'URL est pensée pour être « human readable[3] ». Elle se construit comme suit : `http://<url de l'application>/<utilisateur>/<nom de cours normalisé>`. La figure 3.1 illustre une partie de la vue qui liste les formats disponibles. Les liens sur cette vue redirigent vers les exporteurs correspondant au format choisi.

3. Facilement lisible par un être humain

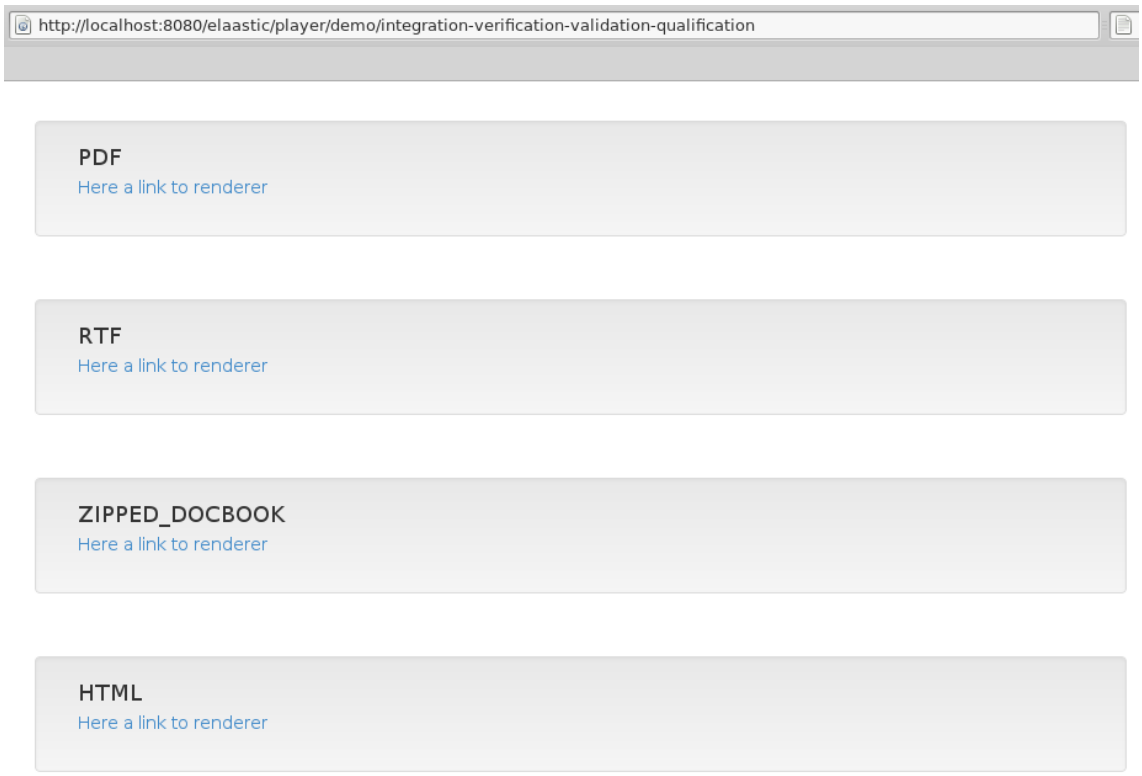

FIGURE 3.1 – Une diffusion de type `ELAASTIC_SIMPLE`

3.5 Question interactive

3.5.1 Recherche et analyse

Là encore, le langage Moodle GIFT a été une découverte pour moi. Il m'a fallu lire la documentation pour apprivoiser la syntaxe et les possibilités de ce langage. Cependant, Franck utilisait déjà un analyseur de code Moodle GIFT dans Tsaap-Notes. Il l'a extrait et en a fait un plugin Grails – appelé Tsaap-Questions – que j'ai pu intégrer à Elaastic. Ce plugin retourne un graphe d'objet Java correspondant à un `Quiz` composé de `Questions`, elles-mêmes composés de `TextBlocks` et d'`Answers`. Il m'a également guidé à travers ces concepts pour comprendre comment récupérer ce qui m'intéressait dans ces questions et me donner des pistes pur le rendu de ces questions dans les différents formats.

Par ailleurs, ces différents formats ne sont pas tous « interactifs ». Par exemple, le rendu ne pourra pas être le même pour un diaporama HTML, où les questions peuvent être des formulaires et les réponses retraitées pour établir des statistiques ou des retours pour les utilisateurs, que pour un PDF où chaque étudiant répondra de son côté à ces questions sans interaction avec les autres étudiants de la promotion. Il a fallu donc différencier les types « statiques » des types « dynamiques ». La séparation s'est faite comme suit :

- Types dynamiques :
  - Diaporama HTML ;
  - HTML empaqueté dans un zip (l'interaction se fera via une API REST) ;
  - IMS Content Packaging (le rendu étant du HTML, le fonctionnement est le même que pour le HTML zippé).
- Types statiques
  - Docbook ;
  - PDF ;
  - RTF.

3.5.2 Solution mise en place

Pour rendre la saisie de question possible, j'ai ajouté un type d'élément de cour. Ces types sont renseignés dans l'énumération `ElementType`. Chaque élément de cour est de type `ElementType`. La distinction se fait au niveau des exporteurs qui traitent les sections/sous-sections différemment des unités et maintenant des questions. Ainsi, lors de l'export d'un élément de type `QUESTION`, le plugin Tsaap-Questions est appelé pour convertir cette question en graphe d'objets, puis ces objets sont analysés et transformés en :

— un formulaire HTML pour les types dynamiques ;
— une liste à puce pour les formats statiques.

Chapitre 4

Bilan

4.1 Expérience professionnelle

Si j'ai choisi ce stage, c'est pour avoir une expérience professionnelle dans le web. Durant mon parcours je n'ai pas eu l'occasion de participer à un projet web d'envergure autre que ceux fait à l'université. Il m'a permis de cerner les enjeux et les contraintes lors de la réalisation d'un projet web. J'ai aussi eu un aperçu des éventuelles évolutions par lesquelles il peut passer et qu'il faut prendre en compte lors de la conception d'une nouvelle fonctionnalité.

Cette expérience m'a permis de voir l'organisation d'un framework web comme Grails et des avantages et inconvénients que cela implique d'utiliser une telle technologie pour son application web.

J'ai également pu pratiquer le développement en suivant une approche agile pour un projet professionnel concret et au sein d'une équipe de développement qui maîtrise ces pratiques. J'ai là encore pu voir les avantages que cela apporte au projet mais aussi parfois une certaine contrainte d'organisation et de rigueur pour les développeurs, nécessaire au bon fonctionnement de ces méthodes.

4.2 Apports personnels

D'un point de vue plus personnel, je n'avais fait du web que pour des projets personnels qui satisfaisaient plus ma curiosité que vraiment une nécessité technique ou fonctionnelle. Ils n'allaient donc par conséquent pas très profond dans la technique et ne suivaient pas forcément les bonnes pratiques de développement. Une autre de mes motivations est que nous ne faisons pas énormément de web à l'université alors qu'on en entend parler de plus en plus au niveau professionnel. Des entreprises se spécialisent dans le web, fournissent des services via internet. L'essor du « as a service » ne fait aucun doute et le développement utilisant une puissance de calculs et des outils dématérialisés se popularise. Il était donc important pour moi de me renseigner sur ce sujet et c'est pourquoi j'ai répondu à cette offre de stage.

Au terme de ces trois mois de stage, j'ai pu faire un point sur cette expérience dans mon parcours. Certes il tend à compléter mon CV avec une expérience dans le web mais il m'a aussi conforté dans mon idée que j'ai une préférence pour les applications de type client lourd local plutôt que client léger dématérialisé. Je conçois la puissance que sont par exemple les services REST et je ne les rejette pas complètement. Cependant, je garde une grande affection pour les langages natifs comme le C++ que j'ai eu l'occasion de pratiquer beaucoup durant mon parcours et que j'ai tendance à privilégier pour mes projets personnels.

Glossaire et acronymes

Back end Pour un logiciel informatique correspond à la partie qui fait office d'interface entre le logiciel et les applications et bibliothèques tierces qui sont invisible pour l'utilisateur – à l'opposé du *front end*.

ENT (**E**space **N**umérique de **T**ravail). Plateforme de travail qui permet aux enseignant de diffuser des contenus de cours ou des exercices aux étudiants en fonction de leur groupe, de leur options, etc.

Front end Pour un logiciel informatique correspond à la partie qui fait office d'interface entre le corps du logiciel et l'utilisateur. Ça correspond le plus souvent à l'interface utilisateur (graphique ou non) – à l'opposé du *back end*.

IDE (**I**ntegrated **D**evelopment **E**nvironment). Logiciel facilitant la création et la gestion de code source, le plus souvent au sein de projets correspondant à un logiciel cible particulié.

LMS (**L**earning **M**anagement **S**ystem). Equivalent d'ENT.

Mock objects (litéralement faux objets). Ce sont des objets donc leur comportement (corps de fonction, valeurs retournées, valeurs des attributs, etc.) sont entièrement simulés pour tester des intéractions avec d'autre objets réel en atttendant des résultats connus puisque simulés sans pour autant avoir à connaître une implémentation de ces faux objets.

PaaS (**P**latform **a**s **a** **S**ervice). Service cloud offrant une plateforme et plusieurs solutions logicielles accessible à distance. Il suit la logique des service fournit via le cloud comme les SaaS ou les IaaS.

Paquetage de Contenu IMS (ou IMS Content Packaging) est un format de paquet de contenu créé par l'IMS Global Learning Consortium. C'est un format standardisé qui permet l'échange – import, export, etc. – de contenu entre différents systèmes reconnaissant ce format là. Nous l'utiliseront car c'est un des format à partir duquel Moodle sait importer un cours.

SaaS (**S**oftware **a**s **a** **S**ervice). Logiciel disponible de manière distante via un service de cloud computing. Le entrées et sorties de l'application transitent à travers le réseau entre le client et le serveur hôte du logiciel. Tout le calcul est effectué à distance et est transparent pour le client.

WebDAV (**Web** **D**istributed **A**uthoring and **V**ersioning) est une extension du protocole HTTP qui facilite le partage et l'édition collaborative entre utilisateurs. Il permet de faire des opérations de création/édition/suppression de fichiers sur des dépôts partagés

par plusieurs utilisateurs tout en gérant les différentes versions de ces documents au moyens de vérrous. Il permet également une gestion des droits d'accès aux fichiers.

ZIP est un format d'archive normalizé contenant une arborescence de fichiers/répertoires compréssés selon un algorithme de compression sans perte de données.

Table des figures