

ECOLE POLYTECHNIQUE DE L'UNIVERSITÉ FRANÇOIS RABELAIS DE TOURS

Département Informatique

64 avenue Jean Portalis

37200 Tours, France

Tél. +33 (0)2 47 36 14 14

polytech.univ-tours.fr

Projet Recherche & Développement 2016-2017

Outil d'autoformation pour ingénieur en informatique

Tuteur académique
Sébastien AUPETIT

Étudiant
Valérien MENIN (DI5)

26 avril 2017



Liste des intervenants

Nom	Email	Qualité
Valérian MENIN	valerian.menin@etu.univ-tours.fr	Étudiant DI5
Sébastien AUPETIT	aupetit@univ-tours.fr	Tuteur académique, Département Informatique



Avertissement

Ce document a été rédigé par Valérian Menin susnommé l'auteur.

L'Ecole Polytechnique de l'Université François Rabelais de Tours est représentée par Sébastien Aupetit susnommé le tuteur académique.

Par l'utilisation de ce modèle de document, l'ensemble des intervenants du projet acceptent les conditions définies ci-après.

L'auteur reconnaît assumer l'entière responsabilité du contenu du document ainsi que toutes suites judiciaires qui pourraient en découler du fait du non respect des lois ou des droits d'auteur.

L'auteur atteste que les propos du document sont sincères et assument l'entière responsabilité de la véracité des propos.

L'auteur atteste ne pas s'approprier le travail d'autrui et que le document ne contient aucun plagiat.

L'auteur atteste que le document ne contient aucun propos diffamatoire ou condamnable devant la loi.

L'auteur reconnaît qu'il ne peut diffuser ce document en partie ou en intégralité sous quelque forme que ce soit sans l'accord préalable du tuteur académique et de l'entreprise.

L'auteur autorise l'école polytechnique de l'université François Rabelais de Tours à diffuser tout ou partie de ce document, sous quelque forme que ce soit, y compris après transformation en citant la source. Cette diffusion devra se faire gracieusement et être accompagnée du présent avertissement.



Pour citer ce document

Valérien Menin, *Outil d'autoformation pour ingénieur en informatique*, Projet Recherche & Développement, Ecole Polytechnique de l'Université François Rabelais de Tours, Tours, France, 2016-2017.

```
@mastersthesis{
  author={Menin, Valérien},
  title={Outil d'autoformation pour ingénieur en informatique},
  type={Projet Recherche \& Développement},
  school={Ecole Polytechnique de l'Université François Rabelais de Tours},
  address={Tours, France},
  year={2016-2017}
}
```

Table des matières

Liste des intervenants	a
Avertissement	b
Pour citer ce document	c
Table des matières	i
Table des figures	iv
Introduction	1
1 Acteurs, enjeux et contexte	1
2 Objectifs	2
3 Bases méthodologiques	2
1 Description générale	4
1 Environnement du projet	4
2 Caractéristiques des utilisateurs	4
3 Fonctionnalités du système	6
4 Structure générale du système	7
2 Etat de l'art / veille	10
1 Learning Management System	10
2 Langages	11
3 Interface graphique	11
4 Sécurité	12
5 Email	13
6 Base de données	14

7	Java Management Extension.....	14
8	Choix technologiques	14
3	Analyse et conception	15
1	Types de contenu	15
1.1	QCM.....	17
1.1.1	QCU	19
1.2	Texte	19
1.2.1	Texte libre	19
1.2.2	Langage structuré	20
1.3	Multimédia	22
1.3.1	Son.....	23
1.3.2	Image.....	23
1.3.3	Vidéo.....	23
2	Modules spécialisés	24
3	Priorisation de développement des contenus	24
4	Architecture du système	27
5	Mise en œuvre	30
1	Installation de Grails 3	30
1.1	SDKMAN	30
1.2	groovy	30
1.3	gradle	31
1.4	grails.....	31
2	Débuter un projet	31
2.1	Création du projet.....	31
2.2	Lancer le projet	32
2.2.1	Tuer un serveur.....	33
3	Modifier l'API	33
3.1	Modifier les logs	33
3.2	Ajout de dépendances.....	34
3.3	Architecture de l'API	35
3.4	Répertoire de configuration	35
3.5	Ajouter des domaines	35
3.6	Ajouter des contrôleurs, services et vues	38
3.7	Définir un proxy	38
3.8	Ajouter Spring Security	39
3.8.1	Créer des Role et User	39
3.8.2	Ajouter des droits d'accès par objet	40

4	Interagir avec l'API.....	41
4.1	Modifier des données depuis BootStrap	41
4.2	Modifier des données depuis POSTMAN	42
5	Tester l'application	45
6	Modifier le client Angular2.....	45
6.1	Architecture du client Angular2	45
6.2	Ajout de modules.....	45
6	Bilan	48
7	Annexes	51
1	Rapport DI3 - Ludification.....	51
2	Matrice de comparaison de différents frameworks java.....	72
3	Tests avec POSTMAN.....	75
	Comptes rendus hebdomadaires	93
	Webographie	97
	Bibliographie	98
	Glossaire	99
	Acronymes	100

Table des figures

Introduction

1	Heures de travail des professeurs - 2013 [2].....	1
---	---	---

1 Description générale

1	Diagramme d'un hiérarchie de groupes de sécurité.....	5
2	Diagramme de séquence - Inscription et deconnexion	7
3	Création de question	8
4	Réponse à un question avec création de quiz	8
5	MindMap - Axe de réflexion sur les modules à développer - DOT	9

3 Analyse et conception

1	Interface graphique - Création d'une question avec choix du type	17
2	IU - Création d'une question - QCM / QCU	18
3	IU - réponse à une question - UML	21
4	Interface graphique - réponse à une question - matching.....	22

4 Architecture du système

1	Architecture général.....	27
2	Diagramme de séquence d'une connexion JWT avec une API.....	28
3	Technologies incluent dans Grails 3	29

5 Mise en œuvre

1	Répertoire de travail du projet.....	32
2	Répertoire de l'API	36

3	Diagramme de classe	37
4	Manager d'environnements sous POSTMAN	43
5	Collection de requêtes sous POSTMAN.....	44
6	Répertoire du client Angular2	47
6	Bilan	
1	Diagramme de Gantt passé et prévisionnel.....	48
2	Diagramme de Gantt effectif.....	49

Introduction

1 Acteurs, enjeux et contexte

Le projet d'outil d'autoformation pour ingénieur informatique est un projet proposé par l'Ecole Polytechnique de l'Université de Tours représentée par monsieur Sébastien Aupetit.

Nous avons comme objectif de pouvoir faciliter l'accès à l'information et surtout à la formation. Nous avons donc voulu faire une plateforme complémentaire aux cours faits à l'école avec une formation en autonomie.

Nous faisons également le constat qu'il est très long de pouvoir former des personnes. Le temps de correction d'un exercice est clairement du temps perdu. Ainsi, un besoin commence à émerger : l'auto-correction / auto-formation.

Average number of 60-minute hours lower secondary education teachers report having spent on the following activities during the most recent complete calendar week¹

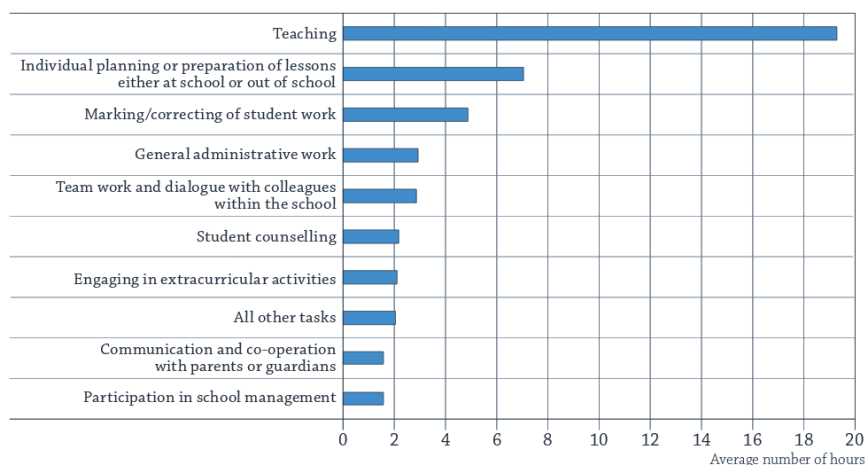


Figure 1 – Heures de travail des professeurs - 2013 [2]

La **Figure 1** nous montre (pour les lycées) les heures passées par professeur et par semaine à corriger des copies. D'autres études montrent également les délais de retour d'exercices qui sont de plus d'une semaine.

Dans cette optique de gain de temps et du "tout, tout de suite", un **Learning Management System (plateforme d'apprentissage) (LMS)** semble être une bonne solution pour une retour de correction plus rapide.

De plus, nous n'ajoutons pas plus ou moins de correction à nos professeurs, nous leur apportons en fait un complément d'enseignement. Ainsi, les professeurs peuvent se focaliser (en cours) sur des exercices à plus forte valeur ajoutée pour laisser les étudiants faire des questions d'entraînement sur le plateforme.

2 Objectifs

L'objectif du projet est la mise en place d'un système d'autoformation (en ligne) orienté ingénieur informaticien. Pour cela, nous souhaitons pouvoir proposer des exercices à l'utilisateur. Les exercices peuvent prendre des formes différentes et nécessiter des types de réponses différentes. Les réponses peuvent être sous la forme de **Question à Choix Multiple (QCM)** (cas simple, courant et pour lequel il existe de nombreux logiciels), sous la forme de diagramme, sous forme numérique, sous forme de code source... Pour permettre l'autoformation, il est essentiel que l'utilisateur puisse obtenir une évaluation de ses réponses le plus rapidement possible. Cela se traduit par une correction quasi instantanée. A la différence d'un exercice papier, le système sera capable de dire à l'étudiant si sa réponse est correcte ou non et ainsi de lui donner une note pour l'aider à suivre ses avancées. Cet outil est pédagogique.

Notre objectif est de créer une application web permettant de s'autoformer sur divers sujets.

Le système est un **Learning Management System (plateforme d'apprentissage)**. Le but est de pouvoir avoir un outil de formation quasi autonome. Les étudiants pourront ainsi se connecter sur l'application web et répondre à des questionnaires pour les aider dans leur apprentissage. A plus long terme, il faut prévoir d'élargir à n'importe quel utilisateur (étudiants, entreprises, particuliers).

L'application web doit être accessible par tous. Il nous faut donc simplifier au maximum les interfaces et créer des tutoriels d'utilisation ou un guide. Si la plateforme est trop complexe, nous savons que les étudiants ne voudront pas s'y former et que les professeurs ne voudront pas non plus perdre du temps à comprendre comme la plateforme fonctionne. Nous voulons donc une interface graphique aussi simple que possible et avec des options plus complètes disponibles si l'utilisateur le souhaite. Les options doivent donc pour la plupart fonctionner par convention sans que l'utilisateur est à les renseigner.

Le projet sera amené à être repris sur plusieurs années. Nous utilisons donc un git pour pouvoir gérer nos sources et faire du versionning. L'intégration continue nous permet également de ne pas faire de récession d'une version à l'autre. Nous attendons également un projet bien commenté, documenté et qui adopte des conventions de programmation (meilleures pratiques) pour conserver l'unité au cours du temps.

A la fin de l'année scolaire 2016-2017, le projet doit être fonctionnel en ce qui concerne la gestion des comptes utilisateurs avec connexions à l'API. Les modifications de droits d'utilisateurs et la création de questions ainsi que la réponse à celles-ci.

3 Bases méthodologiques

Nous utilisons l'intégration continue pour permettre le versionning des sources.

Chaque semaine fera l'objet d'un compte rendu sur les avancées, les difficultés rencontrées et les questions pour lesquelles il faut trouver une réponse.

Le projet doit faire l'objet d'un découpage en tâches avec chiffrage de la tâche en jour homme et importance de la tâche dans le projet. Ainsi, le projet s'effectuera en suivant cette trame principale. C'est ce document qui permet la conception du diagramme de Gantt et de pouvoir facilement rendre compte des avancées du projet.

Avant d'écrire le code source, les fonctionnels seront écrits. La couverture se fera à hauteur de 80%. Les tests sont en lien avec les tâches à effectuer. Le fait de valider les tests valide les tâches au fur et à mesure de l'avancée du projet. En plus de ces test unitaires et fonctionnels, nous souhaitons faire tester à des utilisateurs nos interfaces pour être certain que les besoins des utilisateurs sont respectés.

Lors du projet, nous essayons d'utiliser des sources libres de droit pour ne pas être dépendant d'un environnement propriétaire. Nous ne développerons pas d'éléments déjà implémentés par d'autres. Nous voulons également avoir un niveau d'abstraction important pour ne pas perdre du temps en configurations basiques qui peuvent être automatisées. Ce dernier point signifie que nous souhaitons réduire au maximum les configurations à faire. Nous utiliserons donc des langages assez haut niveau et ne souhaitons pas descendre au base de données quand nous pouvons manipuler des objets au lieux de requêtes SQL.

1

Description générale

1 Environnement du projet

Nous voulions utiliser un maximum d'outils libres pour le projet. Nous travaillons donc dans un environnement dérivé d'Ubuntu Linux qui est Kubuntu pour effectuer notre développement. Le serveur sur lequel sera déployé le projet est un serveur basé sur Linux possédant un JDK 8 oracle. L'utilisation de jdbc est également imposée.

Au début du projet, bien que les choix de développement n'étaient pas encore fixés, nous pensions développer en java avec le framework Vaadin et en utilisant Spring. Nous avons donc utilisé Eclipse qui est open-source. Cependant, nous avons ensuite découvert le langage Groovy qui est un langage haut niveau basé sur java. Le Groovy, couplé au framework Grails permet d'intégrer toutes les bibliothèques Spring (Spring boot, security, ...), hibernate et bien d'autres en ajoutant une surcouche. Cela permettrait de pouvoir gagner un temps important dans la conception de nos modèles et également dans l'écriture du code car l'écriture est simplifiée en Groovy. Le débogage sera donc également plus facile. Cependant, la documentation est moindre que pour d'autres technologies plus répandues. Pour ce qui est de l'IDE, nous avons essayé avec Eclipse et avec IntelliJ Idea de développer en groovy et il n'y a pas de différences si ce n'est l'interface graphique. Cependant, la partie interface graphique est en Angular2 mais nous avons préféré l'utilisation d'Eclipse pour celle-ci car la visualisation des sources y est plus intuitive.

2 Caractéristiques des utilisateurs

Notre système comporte deux grandes familles d'utilisateurs : les créateurs de contenu et les personnes répondants aux questions. Nous pouvons également ajouter les administrateurs mais ceux-là ne devraient que faire de la gestion logiciel pour ajouter des modules ou de la gestion matériel pour les mises à jour serveur et la maintenance.

Nos familles d'utilisateurs ne sont cependant pas aussi séparées que le seraient un système basique tel que Celene l'est.

Dans notre système, chaque utilisateur appartient à un groupe. Par exemple le groupe Polytech. Dans le groupe Polytech, on pourrait trouver les groupes Informatique, Mécanique, Electronique et Aménagement.

Les utilisateurs appartiennent tout de même à des groupes Elèves ou Professeurs. Les sous-groupes Elèves seront par exemple Licence 1, Master 1, Doctorat 1, ...

Il nous est important de pouvoir définir de nombreux rôles à nos utilisateurs et de pouvoir leur associer des droits. Par exemple la **Figure 1** nous montre un exemple type de ce que pourrait être les groupes d'accès.

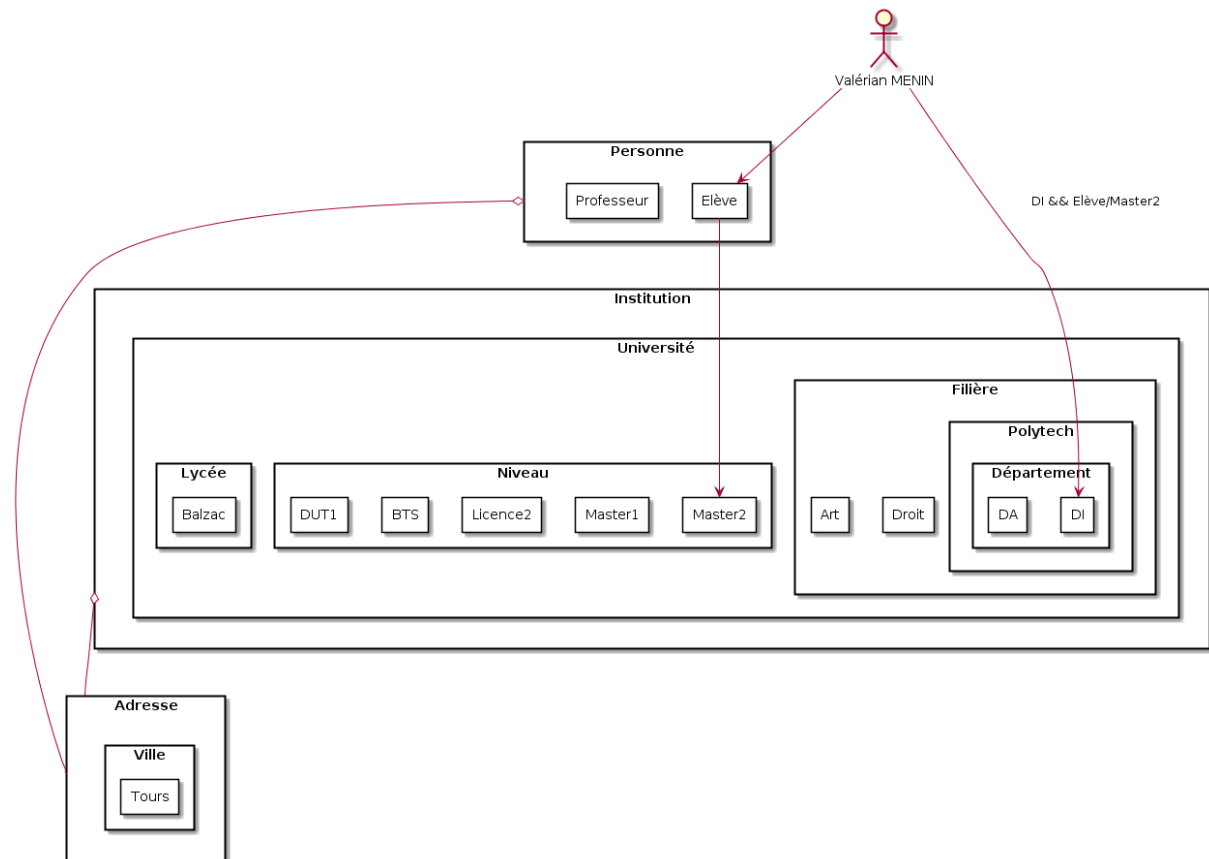


Figure 1 – Diagramme d'une hiérarchie de groupes de sécurité

Les utilisateurs appartiennent à différents milieux. Ils peuvent venir d'une entreprise, de Polytech, être en IUT, ...

Dans la **Figure 1**, nous représentons les grands principes des groupes. Ainsi, une personne pourrait être dans plusieurs établissements à la fois. Chaque groupe possède sa propre sécurité pour l'exécution, l'édition, la correction ou la création de contenus. La gestion est la même que celle des packages en programmation. Cette gestion est assez intuitive et permet de nombreuses possibilités.

La structure globale est gérée par les administrateurs mais on s'aperçoit vite qu'il est essentiel que la partie professeur soit capable de gérer les droits des utilisateurs affiliés à leur champ d'expertise. Les professeurs de Polytech peuvent donc gérer les accès aux cours qui les concernent, de faire hériter des droits ou d'ajouter de nouveaux groupes de la même façon que la création d'une liste de diffusion dans une boîte de messagerie. Les professeurs seront ainsi eux-mêmes limités selon leur échelon à un groupe d'utilisateurs. Cela se traduit en fait par des groupes d'autorisation pour chaque objet. Ainsi les utilisateurs auront (ou non) le droit de voir telle ou telle question et d'y répondre voir de la modifier ou d'en créer une nouvelle.

Les professeurs peuvent créer de nouveaux contenus (des quiz) et à terme, même les élèves devraient pouvoir créer des quiz si le droit leur a été attribué. On pourrait penser à un système de gain de droits dans un répertoire après avoir réussi un certain nombre de succès (par exemple

avoir terminé 90% des cours du répertoire avec 90% de réussite). Le système de badge est un principe phare de la ludification qui permet aux utilisateurs d'avoir un but de progression.

Concernant la ludification, la lecture de mon rapport de stage de DI3 [1] disponible [Section 1](#) (Chapitre 7) montre la multitude d'actions possibles à mettre en place pour améliorer l'expérience utilisateur sur le site et également pousser les utilisateurs à être de réels consommateurs de contenus.

Il y a plusieurs profils d'utilisateurs :

- administrateurs
- testeurs (étudiants)
- testeurs (étudiants)
- créateurs de contenus (professeurs)

Mais cela n'est qu'une vision simplifiée car nos "encadrants" auront eux aussi un rôle limité dans le système. Un Encadrant1 ne doit pas forcément avoir accès aux données de l'Encadrant2 ou de l'Etudiant1. Egalement, il doit être possible de dire qu'un Etudiant1 puisse lui aussi créer des quiz. C'est ce que l'on appelle un [Access Control List \(liste de contrôle d'accès\) \(ACL\)](#) (nous en reparlerons plus tard dans les parties techniques).

Pour ces raisons, il nous faut un gestionnaire de droits plus développé. Cela est en fait géré par l'aspect sécurité présenté à la [Figure 1](#) mais avant cette gestion, il faut pouvoir se créer un compte puis se connecter à l'application web comme présenté à la [Figure 2](#).

Basiquement, pour s'inscrire, les utilisateurs suivent le diagramme de séquence présenté dans la [Figure 2](#).

3 Fonctionnalités du système

Chaque utilisateur fait partie d'une dynamique globale de travail collaboratif. Il y a donc un travail à faire pour qu'un premier groupe d'utilisateurs actifs fasse venir d'autres utilisateurs. Il faut permettre une vraie conversation et de vrais échanges entre les différents acteurs. Ainsi, il faut un forum et un moyen de communiquer entre membres. C'est une communauté qui doit pouvoir ajouter du contenu et répondre à des questions de façon régulière.

Les utilisateurs vont donc pouvoir se créer un compte via une adresse électronique. Si ces utilisateurs font partie d'une organisation particulière, alors ils doivent également fournir une adresse mail liée (par exemple @etu.univ-tours.fr ou @univ-tours.fr). Cela leur assure un droit de regard sur les groupes au sein de leur organisation. Les groupes doivent être accessibles sur ajout (ou acceptation après demande) d'un responsable de groupe ou par clé d'authentification dans le cas d'un cours.

Un utilisateur demande donc des accès à un responsable de groupe. Ce dernier lui donne des droits de consultations, d'édition, de correction ou de création de contenu.

En utilisant ces droits, nous pouvons obtenir ce genre d'enchaînement :

Les [Figure 3](#) et [Figure 4](#) présentent les différentes interactions entre le serveur et le client lors de la création de question et lors de l'[exécution](#) de quiz.

Toute cette structure de données est stockée dans une [BDD](#). Les [questions](#) sont stockées dans des [Base De Données](#) situées sur le serveur. Les [quiz](#) sont aussi stockés dans une [BDD](#) qui comporte une liste de références sur des [questions](#).

La [Base De Données](#) contient également les informations liées à la sécurité et aux utilisateurs. Un système de logs (différent de l'historique des [quiz](#) effectués) permet aussi de voir ce qui a été exécuté sur le serveur. Ce système est particulièrement utile vis-à-vis des [dockers](#) sur lesquels

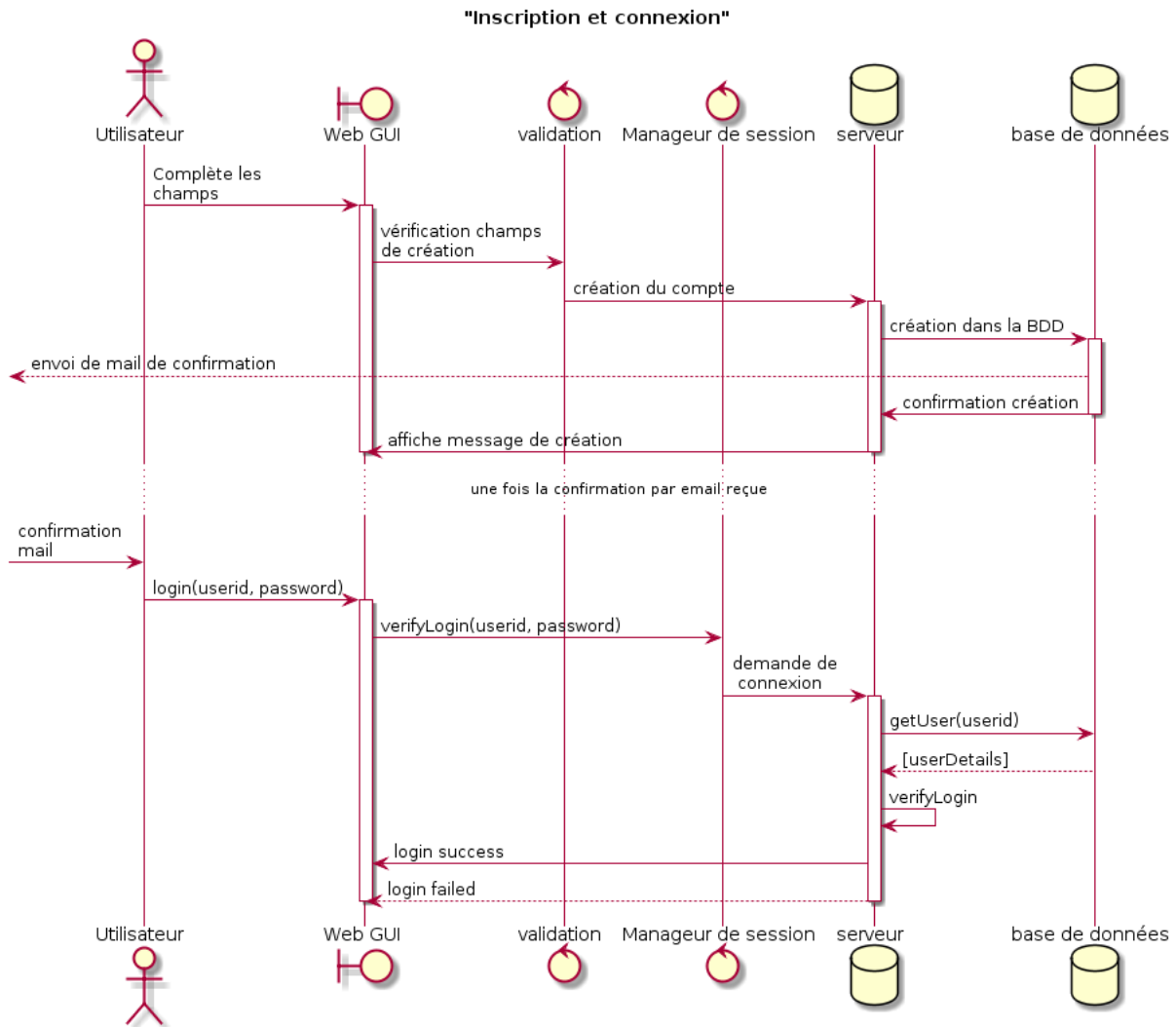


Figure 2 – Diagramme de séquence - Inscription et deconnexion

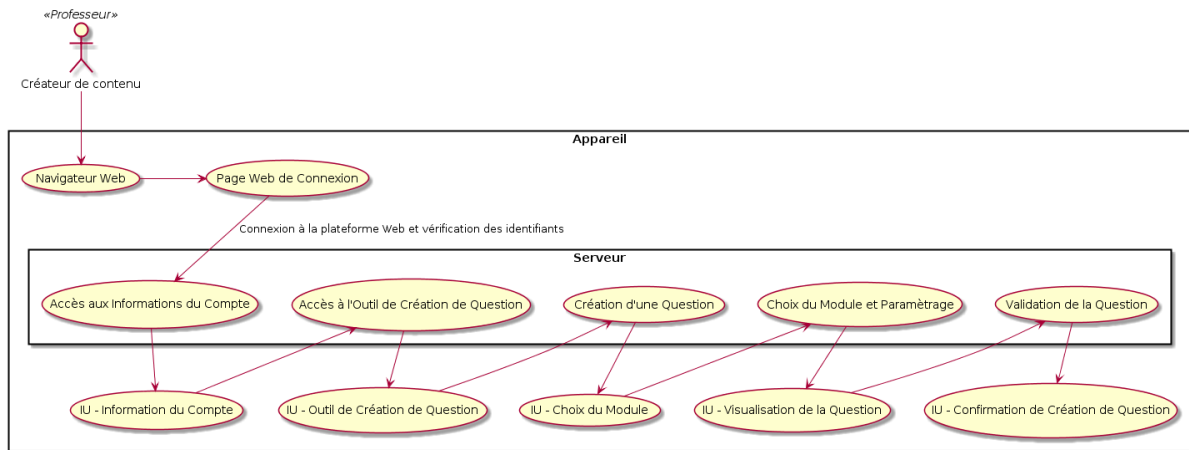
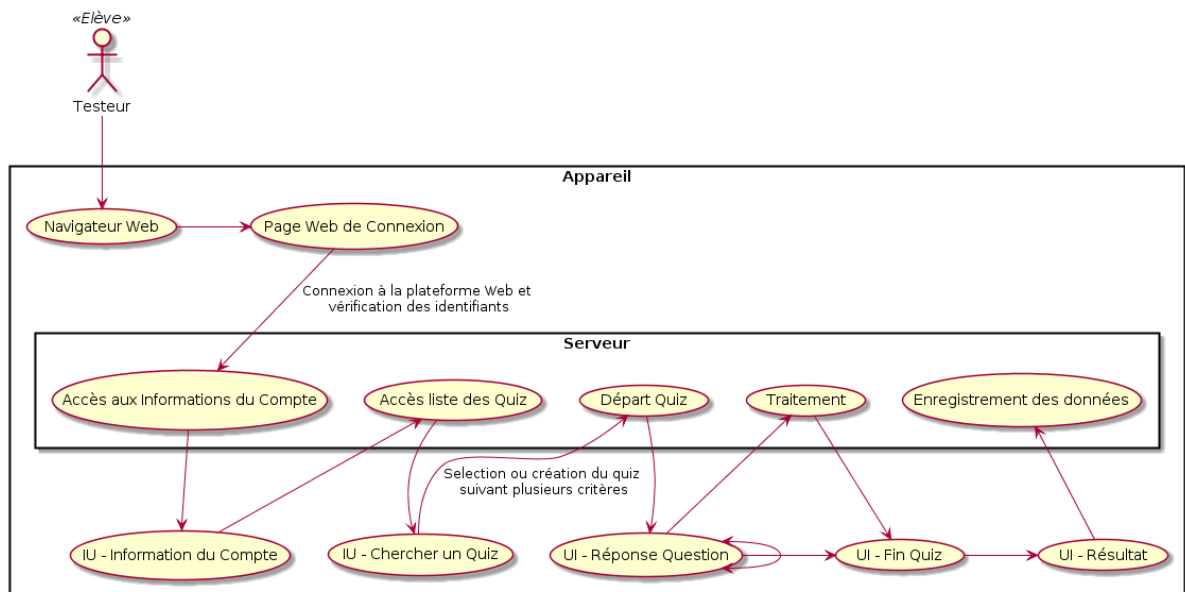
du code source sera amené à être exécuté. En cas d'attaque du serveur via injection (ce qui ne devrait normalement pas arrivé étant donné que nous utilisons docker), il faut pouvoir voir ce qui s'est exactement passé sur le serveur. Nous avons dans un premier temps mis en place un système de log qui s'écrit dans un seul fichier et souhaitons essayer une autre possibilité que Docker avec des workers Java Platform. Ces systèmes embarqués peuvent en fait vérifier à intervalle de temps régulier si le serveur à des réponses à évaluer. Cela permettrait de ne pas effectuer l'évaluation (en particulier celles sensibles avec du code source ou celles gourmandes en ressources) sur le serveur mais plutôt de le faire faire à ces plateformes java. Une mécanisme de plateformes qui se réveillent les uns après les autres suivant la change de travail serait également intéressant à mettre en place.

4 Structure générale du système

Le projet doit être modulaire pour permettre l'ajout / suppression ou simplement modification de parties sans impacter le reste du projet.

Il nous faut :

- gérer les créations de comptes avec la gestion de droits utilisateurs

Figure 3 – Création de *question*Figure 4 – Réponse à un question avec création de *quiz*

- manipuler des **modules** de **questions** de différents types qui peuvent être ajoutés au fil du temps comme présenté dans la Figure 5 et le Chapitre 3.
- noter chaque **réponse** avec un robot ou déléguer à un humain si le système n'est pas capable de valider la réponse avec un taux de certitude suffisant.



Figure 5 – MindMap - Axe de réflexion sur les *modules* à développer - DOT

La Figure 5 présente de façon visuelle les grandes familles de *modules* qu'il serait intéressant d'avoir ainsi que certaines idées de thèmes d'exercices.

Dans notre projet, nous cherchons à maximiser la modularité du système. en séparant les différents types de *questions*. La séparation des *questions* est en effet dans la continuité de la logique de plugins utilisé par grails 3.

2

Etat de l'art / veille

Pour commencer, nous nous sommes intéressé à la possibilité d'amélioration d'un outil libre qui pourrait répondre à certaines de nos attentes comme Auto Multiple Choice mais celui-ci est très limité. Par la suite, nous nous sommes intéressé aux possibilités technologiques à notre disposition avec des frameworks qui pouvaient nous faire gagner du temps de développement et de la lisibilité dans notre code source.

1 Learning Management System

De nombreux **Learning Management System (plateforme d'apprentissage)** existent déjà sur le marché. Cependant, la plupart d'entre eux sont propriétaires et ne permettent donc pas de modifications profondes. De plus, ces **LMS** sont très coûteux et ne répondent pas à toutes nos attentes[[WWW1](#)]. A ce jour, les **LMS** les plus performants ne permettent pas la création de correcteurs sophistiqués. La limitation est donc vite atteinte.

D'autres **LMS** sont libres mais sont difficilement personnalisables. C'est le cas de notre système sur l'**Environnement Numérique de Travail (ENT)** avec **Moodle**. Cependant, **Moodle** est une plateforme intéressante mais non pensée pour recevoir de telles modifications. Développer un plugin sur **Moodle** est intéressant mais pour l'intégrer dans avec le système de l'université, cela est très difficile. Le service informatique de l'université ne permet pas la connexion au service universitaire. Le problème est en fait sur le long terme car une fois un projet ajouté à **Moodle**, la maintenance revient ensuite au service informatique de l'université. Nous avons tout de même essayé de nous connecter à l'ENT via CAS mais n'ayant pas de clé d'accès, nous ne pouvons pas effectuer de connexion.

En bref, malgré une offre assez fournie en **LMS Open Source**, aucun ne répond à nos besoins et matières de vision modulaire et spécialisation. Il nous faut donc développer un tel outil. Notre plateforme doit contenir une gestion des utilisateurs, des droits de gestion par objets, un service mail et la possibilité d'ajouter des modules de questions facilement. Ces conditions n'ont été trouvées dans aucun **LMS** existant. Même le CMS Drupal ne possède qu'un début de solution avec seulement certains exercices simples et il est difficile d'en ajouter de nouveaux dans le CMS.

2 Langages

Un aspect essentiel du projet est la modularité du système. Il faut que tout notre projet soit développé par **modules**. Il y a plusieurs raisons à cela :

- éviter les problèmes de forte dépendance
- remplacer un module sans avoir à modifier tout le code → encapsulation des parties comme des plugins
- ajouter / supprimer des modules → avoir des modules indépendants
- travailler avec plusieurs équipes → gitlab (pour faciliter les projets collaboratif sur ce projet)
- encapsuler les modules pour mieux les tester → groovy / java le permettent
- ne pas exécuter de code externe directement sur le serveur → docker / Java Platform

L'une des contraintes du projet est l'utilisation du JDK 8 oracle et donc de java. Java permet une portabilité importante du code et possède de nombreux plugins.

Nous pensions donc travailler en java mais nous avons découvert le groovy qui est basé sur du java. Groovy possède une grande similarité avec java mais il est plus léger en terme d'écriture de code. De plus, de nombreux plugins sont disponibles pour ajouter des fonctionnalités à notre applications. Groovy est pensé pour être très modulaire et portable, c'est d'ailleurs la raison principale pour laquelle Samsung l'utilise pour ses objets **smarththings** ([documentation](#)). Groovy est totalement compatible avec java.

La plus grande différence entre le java et le groovy est que groovy est basé sur des conventions contrairement au java dans lequel il faut expliciter ses choix. De plus, le framework Grails 3 permet l'utilisation de plugins multiples.

Nous avons donc choisi d'utiliser groovy pour le projet d'outil formation. Nous sommes également amené à utiliser d'autres langages comme décrit plus bas.

Groovy est utilisable avec Gradle qui est un moteur de production et Grails 3 qui est un framework également libre. Gradle fonctionne par conventions plutôt qu'avec de longues configurations. Gradle permet de gérer des plugins de manière quasi transparente. Le système "plug and play" de Grails 3 est très puissant. De plus, Grails est basé sur Spring Boot. Grails s'adapte parfaitement à notre optique modulaire car il est basé sur cet aspect modulaire (comme montré dans cette [vidéo](#) explicative).

3 Interface graphique

Pour la partie frontend du projet, plusieurs possibilités s'offrent à nous. Nous avons dans un premier temps fait la liste des framework web les plus importants en utilisés

- vaadin
- swing → non spécialisé en web
- javafx → non spécialisé en web
- jsp → pour j2ee
- gsp → utilisé par défaut en groovy, système de plugins

Une matrice de comparaison de Frameworks java est disponible [ici](#) et simplifiée dans le chapitre annexes.

Cette matrice reprend les principales technologies existantes pour développer l'**Interface Utilisateur (IU)**. Nous pouvons voir les avantages de Vaadin assez clairement, seulement, Vaadin est incompatible avec un model MVC Vaadin n'est pas la solution retenue.

Selon ce comparatif (fait par Vaadin), Vaadin est bien meilleur que JSF, PrimeFaces, Wicket, Angular2, GWT, GXT, et JavaFX. Il y a une communauté très importante des perspectives de développement très intéressantes. De plus le projet Vaadin est maintenu pour s'intégrer comme un module ce qui va dans l'optique du projet. Cependant, Vaadin on Grails nous empêche d'exploiter certains avantages de grails comme le système MVC qui est le plus couramment utilisé. Il serait contre-productif d'utiliser un outil qui nous permet de faire de belles interfaces mais de ne pas pouvoir maintenir ou développer le système comme nous le souhaitons.

On pourrait toutefois s'intéresser à Griffon qui s'inspire de Grails. C'est un framework open-source qui utilise en grande partie Grails avec des ajouts poussés concernant Swing. Cependant, j'écarte la possibilité car la communauté est très peu active et Griffon n'est pas du tout dédié aux interfaces web.

Flex étant fait en Flash a très vite été abandonné comme option possible.

Si nous n'utilisons pas Vaadin, nous pouvons toujours utiliser l'interface utilisateur de base qui est SiteMesh. Autrement Bootstrap et KickStart restent une possibilité (voir un mélange des deux derniers).

Au final, utiliser grails semble la possibilité la plus logique et facile à mettre en place. La logique de plugins nous permet même d'ajouter Spring Boot à notre projet (qui n'est pas disponible de base en particulier pour avoir des version très légères pour les objets connectés). Notre projet est donc un projet grails (donc gradle et Spring Boot). Grails permet aussi l'utilisation de **Groovy Object Relational Mapping (GORM)** qui se base sur hibernate.

Il existe bien d'autres **frameworks groovy** mais grails 3 est celui le plus connu (et documenté) avec la meilleure gestion des dépendances.

Nous avons choisi d'utiliser Angular2 pour notre interface graphique. Cependant, Angular2 ne va pas permettre d'avoir une Modèle-Vue-Contrôleur. Angular2 va cependant nous permettre de totalement différencier le front-end du back-end. En cas de besoin, nous pourrions donc changer le front-end sans même toucher à l'API.

Après cette section, il est important de noter que nous avons définitivement choisi groovy avec grails 3 pour notre projet.

4 Sécurité

Notre mode de fonctionnement est en fait assez simple pour ce qui est de la création de comptes et les connexions d'utilisateurs. Cependant, il existe des solutions permettant de gérer ces aspects avec en plus une gestion des permissions.

En matière de sécurité, deux grandes possibilités s'offrent à nous avec grails 3 : Shiro ou Spring Security.

Shiro possède l'avantage d'être très customisable contrairement à Spring Security. Les deux services permettent en de nombreux points les mêmes choses. On peut même utiliser une synthèse des deux.

Leurs principales différences sont que le premier est plus lourd à mettre place contrairement au second. Spring Security gère ses permissions par rôles données aux utilisateurs. Shiro peut quand à lui gérer ses utilisateurs par rôle, groupe ou encore directement par utilisateur. La finesse est donc plus facile à gérer avec Shiro. Cependant des dépendance Spring peuvent être ajoutés au projet pour obtenir la même granularité La gestion des groupes de sécurité et les connexions par réseaux sociaux sont cependant plus facile avec Shiro mais ces manques de Spring Boot sont comblés par l'ajout de plugins permettant d'avoir ces fonctionnalités.

D'une manière générale, Shiro semble être plus complet que Spring Security. Cependant Spring Security est fortement maintenu alors que le projet Shiro a été arrêté pendant plusieurs années.

Il a enfin été reprise dans le projet Nimble pour ajouter plus de fonctionnalités mais on peut toujours se poser des questions sur son avenir.

Spring Security possède une documentation très fournie et une communauté importante. Nimble étant plus nouveau, la communauté est toujours en train de grandir mais n'atteint pas encore les sommets de Spring Security.

Dans une logique préventive, je conseillerais de prendre Spring Security.

Il y a cependant un nouveau moyen d'avoir les avantages de la robustesse de Spring Security avec la gestion plus facile de Shiro. Grails permet d'utiliser le plugin Spring Security Shiro depuis 3 ans. Le plugin permet de tout gérer comme l'un, comme l'autre ou de faire un mélange des deux tirer partie des deux. Les authentifications utilisent Spring Security et les gestionnaires de sessions utilisent Shiro mais là encore, la documentation est très peu fournie.

Nous avons donc deux outils parallèles tous deux fonctionnels. Bien que Spring soit plus lourd à mettre en place, il reste deux autres avantages à Spring Security :

- Spring Security est beaucoup plus facilement intégrable à Spring Boot que Shiro
- La communauté et la documentation Spring Security sont très importantes contrairement à Shiro

Nous avons donc choisi d'utiliser Spring Security pour sa robustesse et sa communauté importante. De plus, Spring est de très loin la dépendance la plus utilisée pour les projets web avec Grails. L'avantage d'utiliser Spring est de pouvoir ajouter d'autres dépendances en plus du core et de spring security. Nous avons donc ajouté spring ACL pour permettre la gestion des droits par objets. Ainsi, chaque objet va posséder ses propres droits. Spring restfull est également utilisé pour notre API.

La liste des plugins Spring disponibles avec grails 3 est très grande comme nous pouvons le voir sur le site de [grails](#).

On y retrouve les principales dépendances comme :

- [spring-security-acl](#)
- [spring-security-cas](#)
- [spring-security-core](#)
- [spring-security-facebook](#)
- [spring-security-rest](#)

5 Email

Plusieurs plugins sont disponibles pour la gestion des mails.

Nous avons trouvé deux solutions principales pour la gestion des mails :

- Novamail
- Spring (message asynchrone)

Novamail ne comporte pas de documentation très importante alors qu'il est facile de trouver de nombreux exemples d'utilisation sur Spring "asynchronous mails". De plus, nous avons déjà choisi de nombreuses dépendances Spring, il est donc plus facile de trouver des exemples avec ces différentes dépendances ensembles.

La solution retenue est donc l'utilisation basique de Spring avec l'envoi d'emails asynchrones. Cette solution est faite disponible de base dans le core spring.

6 Base de données

La partie **Base De Données (BDD)** n'est pas la plus fondamentale car nous souhaitons interagir avec elle le moins possible pour ne pas être dépendant de celle-ci. De plus notre utilisation de celle-ci est limité à la création de **Question**, le lien entre les **Question** pour faire des **Quiz**. Les autres données stockées sont liées aux utilisateurs et son assez basiques (information utilisateur, groupe de droits, mails, suivi de statistiques, historique ou encore customisation de l'**IU**).

La raison pour laquelle la base de données n'est pas essentielle est que nous utilisons l'ORM groovy. GORM permet la gestion des domaines (qui sont en fait nos classes objets) en tant qu'objets. GORM se charge de communiquer avec le **plugin hibernate 5** qui va lui-même gérer les données dans la base de données.

Nous avons choisi d'utiliser H2 comme base de données. C'est celle utilisée de base dans un nouveau projet grails 3.

Pour faire des modifications sur notre base de données, c'est Spring-Security qui se charge de vérifier les autorisations dans les contrôleurs. Nous faisons ces changements dans des transactions pour être certain d'avoir des données valides.

Le **plugin flyway** permet également de pouvoir migrer facilement notre base de donnée (H2, SQL server, Oracle, SQLite, MySQL, ...). Flyway permet également de faire de la configuration de base de données par convention.

7 Java Management Extension

A terme, nous souhaitons pour voir utiliser **Java Management Extensions (JMX)** pour gérer nos ressources. **JMX** est désormais dans la librairie standard de Spring.

Cette fonctionnalité ne sera cependant pas développée cette année. **JMX** nous permettra de gérer nos logs pour certaines actions. L'un des points critique est l'évaluation de code source. Si une question porte sur du code source, nous allons devoir exécuter ce code source sur notre serveur, nous voulons donc être certain de savoir ce qui s'est passé pendant cette exécution.

8 Choix technologiques

Cette section permet un rappel des différentes technologies utilisées pour le projet.

- grails 3
- gradle
- Spring Boot
- Spring Security
- spring-security-acl
- spring-security-core
- spring-security-rest
- Spring asynchronous mails
- **Groovy Object Relational Mapping (GORM)**
- Hibernate5
- **Java Management Extensions (JMX)**
- Angular2

3

Analyse et conception

1 Types de contenu

Nous sommes amené à décrire plusieurs types de contenus si nous voulons pouvoir prévoir un maximum de cas. Dans cette partie, nous décrirons les principaux types qui existent. Nous avons défini un certain nombre de types de questions. Certaines questions peuvent être hiérarchisées ou utiliser des éléments d'une autre question. Basiquement, nous appelons un type de question un **module**.

Type de développement

Une question importante est de savoir si nous souhaitons développer des **Module** très généraux puis descendre en granularité ou alors de développer une multitude de **Module** parallèles très indépendants les uns des autres. La première solution permet de pouvoir créer des idées très générales et de permettre l'utilisation de bibliothèques qui seront développées au fil des années. Il suffirait ensuite de redéfinir des fonctions dans les extensions de ces modules pour y ajouter les particularités du type. L'inconvénient / difficulté est de devoir prévoir une globalisation la plus poussée. La seconde solution est de créer des modules totalement indépendants et donc de permettre d'ajouter son module sans même connaître les autres modules qui existent puisqu'ils n'interagissent pas entre eux. Cependant, cela implique une redondance de code importante. Nous optons donc pour la première solution : faire des groupes de types pour maximiser les interactions et éviter la redondance de code. Cela permet la création d'une bibliothèque de types. La metaprogrammation permet de pouvoir gérer des exceptions pour lesquelles un module de type C étant lui-même de type A devrait utiliser une méthode du type B qui n'est pas relatif à A. Groovy permet cette gestion dynamique avec les fermetures, les `ExpandMetaClasses` et les catégories (un exemple complet peut être trouvé [ici](#)).

En fait, notre système va essayer de faire des modules indépendants les uns des autres autant que possible mais nous gardons l'idée d'étendre certains types de questions. Cependant, il faut éviter de trop descendre à plus de trois niveaux d'extension car les extensions se font avec des jointures entre les tables SQL et groovy. Or les jointures ne sont pas des actions rapides, il faut donc éviter de trop les multiplier.

Dans un premier temps, nous nous intéressons aux **Question à Choix Multiple (QCM)** / **Question à Choix Unique (QCU)**. Ces types sont en fait simplement binaires et un **QCU** est un cas particulier d'un **QCM**.

Les textes : il en existe des très nombreux sous-types. Ainsi, un type texte permettra la saisie de texte qui devront obligatoirement posséder un sous-type de texte. Ces extensions de texte seront soit du code source ou du texte linguistique.

Les textes libres sont de types communs et permettent donc le travail sur des langages parlés (français, anglais, latin ...) donc des langues vivantes.

Les codes source sont des langages structurés. Il peuvent donc être de type java, c, c++, LaTeX et autre types de codes de programmation mais également de types descriptifs tels que de l'**UML** ou encore des mathématiques.

Les **Question** sont organisées par types (**Figure 5** (Chapitre 1)), catégories et mots clés. La création d'une **Question** nécessite donc d'être dans une catégorie précise (exemple : Polytech / Informatique / JavaPerformance). Elle possède d'ailleurs un droit d'accès (test/édition/correction) définissables par le créateur de contenu (ces droits doivent être ceux d'un groupe et non d'une seule personne pour éviter les problèmes du type "ne travail plus ici"). L'historique de modifications (nominative et temporelle) doit être implémentée.

Il faut pouvoir créer des liste de **questions**. Ces listes sont appelés des **quiz**. Les quiz sont des objets qui possèdent également des droits liés à l'objet pour le partage, l'édition et la visualisation et l'évaluation des questions de la liste. Cela est particulièrement utile pour créer des examens ou s'entraîner sur des sujets précis.

Un quiz possède des options tels que la date de disponibilité (ouverture / fermeture) un pool de **questions**, des temps limites, le nombre de **questions** par page, l'autorisation ou non de revenir en arrière, l'autorisation ou non de reprendre plus tard, l'autorisation ou non d'ouvrir plusieurs sessions, l'autorisation ou non d'enregistrer son travail, la visualisation ou non de son score, la visualisation ou non des réponses, le nombre de point à ajouter/retirer par question, le correcteur référant, des autorisations d'accès. On devrait aussi pouvoir choisir un degré de difficulté (dans un premier temps défini par le créateur puis modifié en analysant les taux de réussite à la **question**).

La **Figure 1** nous montre l'**Interface Utilisateur** dessinée pour la création de **questions**. On commence par définir un nom à la **question** (ce nom est en fait la question en elle-même) puis on choisi le type de **question** parmi la liste des différents **modules** disponibles.

On entre ensuite un certain nombre de **tags** permettant d'améliorer la visibilité de la question dans le système et ainsi de pouvoir effectuer une recherche par **tags**.

On entre ensuite les droits de modification de la question puis les droits d'accès (qui sont les autorisations de visibilité / **exécution** lié à la **question**. Ces droits sont à renseigner avec des connecteurs logiques. On peut donc définir des "et" notés "&&", des "ou" notés "||" et hiérarchiser les priorités avec des paires de parenthèses "()".

La section de spécificités prend en paramètre une seule chaîne de caractère qui n'est autre que la concaténation de nombreuse options. Ces spécificités portent sur la **question** (et non le **évaluation** de celle-ci), elles sont les suivantes :

- retour en arrière possible
- activation du chronomètre
- affichage du chronomètre
- autorisation de visualisation de la réponse
- autorisation de visualisation de la note

Figure 1 – Interface graphique - Création d'une question avec choix du type

- limite de temps
- nombre d'essais maximum
- ne pas répondre à la question

Ensuite, on choisit si la question est prête à être publiée et si l'on souhaite définir une fenêtre de disponibilité de la question (par exemple dans l'optique d'un examen).

Globalement, la **réponse** est-elle correcte ?

- Si oui : +point
- Si non : -point
- Si pas de réponse : -point

Les points sont donnés par le créateur de contenus. Chaque **question** possède un certain nombre de points à ajouter ou à retirer.

Un problème est lié à l'attribution de la note. Par exemple, pour du code source, on peut passer 80% des tests avec succès et échouer le reste. Il faut donc prendre en compte ce point. Chaque **réponse** de **question** retourne donc un résultat et on choisit l'importance de cette réponse par rapport aux autres. Par exemple, un **QCM** peut rapporter 1 point alors que l'écriture d'un code source en rapporte 5. Le calcul de la note final doit donc être une combinaison des résultats obtenus à chaque questions et non une simple addition. Ce problème sera en fait géré par le créateur de **quiz** qui pourra définir la pondération par **question**.

1.1 QCM

Pour **QCM**, il faut déjà savoir quelle(s) bonne(s) réponse(s) sont disponibles dans la liste des propositions.

Les **QCM** sont gérés de manières booléennes avec le système de points interne au **module**.


Dans l'**Interface Utilisateur** de création d'une question (**Figure 2**), on distingue trois parties :

- le rappel du nom de la question (encore éditable)
- les différentes réponses possibles
- les spécificités liées au **évaluation**

Question nom de la question

Liste réponses

Réponse	Module	Vrai	Commentaire
Proposition 1	texte ▼	<input checked="" type="checkbox"/>	Vrai car ...
Proposition 2	texte ▼	<input type="checkbox"/>	Faux car ...

Ajouter une réponse 

Spécificités

```
toutBon=5;parBon=proportionnel;parErreur=-proportionnel*2;faux=-2;sansReponse=0;
```

Figure 2 – IU - Création d'une question - QCM / QCU

Un **module QCM** possède évidemment un nombre limité de **réponses**. Ces **réponses** comportent un nom qui n'est autre que la proposition de **réponses**. Cette proposition peut être présentée sous forme de texte, d'image, de vidéo, de son ou encore d'autres **modules** tels que les **modules** plantUML ou éditeur de code source avec coloration syntaxique.

Un **QCM** donne un score suivant ses options d'**évaluation**. On peut choisir d'enlever ou non des points pour chaque erreur ou pour une erreur. On peut également choisir si chaque case cochée vaut autant de points qu'une autre ou choisir au cas par cas.

Pour la **Figure 2**, on observe les spécificités suivantes :

- toutes les réponses sont bonnes : je donne des points
- par réponse bonne, je donne des points par rapport au ratio nombre de points à donner sur nombre de bonnes réponses
- les erreurs enlèvent deux fois plus de points qu'une bonne **réponse**
- ne pas répondre à la question ne fait pas perdre de points

Aussi, les deux propositions sont de type texte.

Pour savoir si les **réponses** sont bonnes, nous allons définir un objet de k entrées avec k le nombre de propositions. Pour chaque vrai, la proposition est correcte. On vérifie donc qu'il y a égalité ou non entre l'objet des propositions et celui de la correction. De très nombreuses méthodes différentes sont néanmoins possibles. En utilisant les spécifications, on calcul le nombre de points obtenus à la question.

On y retrouve le principe du logiciel **Auto Multiple Choice (ACM)**.

Le mode de visionnage de résultats reprend simplement le nom de la question, les propositions, la colonne vrai, un code couleur suivant si l'utilisateur a bien répondu ou non, et enfin un commentaire explicatif si le créateur de contenu en a défini un.

L'**évaluation** de la **question** est faite sur le serveur. On garde l'historique d'**évaluation** dans les logs du serveur si besoin. Nous pouvons également penser à effectuer des analyses statistiques par utilisateur, par questions et par **tag**, cela permettrait de mieux connaître les attentes de nos utilisateurs et leurs besoins.

1.1.1 QCU

Le **QCU** n'apporte pas de modification au **QCM**. Il limite seulement son utilisation. Un **QCU** est un cas particulier de **QCM**.

Un créateur de contenu peut choisir d'autoriser ou non la sélection de plusieurs propositions alors que c'est un **QCU**. L'interface graphique va seulement afficher un sélecteur unique au lieu d'un sélecteur multiple. Pour le serveur, l'**évaluation** est rigoureusement la même.

1.2 Texte

Pour un teste, il nous faut définir une grammaire ainsi qu'une syntaxe. Ce sont les bases d'un langage.

Chaque texte possède une grammaire et une syntaxe qui devront impérativement être définies dans chaque extension.

Ce **module** est très important en terme d'utilité mais également en terme de taille. En effet, toutes les parties code source et linguistique sont incluses dans celui-ci. Il est donc nécessaire de pouvoir catégoriser différents sous-modules dans ce dernier.

Coté création de contenu, le **module** linguistique fait appel aux autres **modules** pour l'édition d'un texte, l'ajout de contenu multimédia...

La visualisation du résultat présente tous les aspects évalués avec l'efficacité pour chaque critère et une note globale. Si toutefois l'**évaluation** n'a pas pu être automatisée, alors on indique seulement les critères évalués avec des notes provisoires.

1.2.1 Texte libre

La **réponse** s'effectue via une zone de texte pour laquelle on autorise ou non des spécificités comme la vérification orthographique, la mise en forme du texte ou les copier/coller.

Pour l'**évaluation**, on pourra utiliser des dictionnaires. Il faudra (ou non) permettre aux navigateurs web la vérification de ce qui est saisi. Les tests peuvent porter sur l'orthographe, la grammaire, du vocabulaire, une traduction / récitation / résumé / description avec un système de mots clés et un pourcentage de similarité.

On pourrait utiliser un système comme **Bayesian Essay Test Scoring sYstem** (BETSY) [WWW10] [WWW3] pour noter des essais.

On peut analyser la vitesse de frappe et le nombre d'erreurs.

Le créateur de contenu est amené à donner la formule qu'il souhaite utiliser pour pondérer les points qu'il souhaite noter.

Dans le sous-module linguistique, on pourra définir des options de langage. Cela permettra de définir une langue d'usage.

- français
- anglais
- latin
- ...

Le rendu de la note se fera de façon automatique (autant que possible). Chaque critère noté comportera un indicateur de certitude qui représente l'acceptation d'un caractère. En cas de grande différence, alors cela veut dire que le critère n'est pas reconnu dans le texte et donc que ce critère n'est pas évaluable. Il y a donc une demande de correction manuelle à un administrateur de cette **question** à la fin de l'**évaluation**.

1.2.2 Langage structuré

Il nous faut une coloration syntaxique (activable ou non par le créateur).

Un simple test est déjà de savoir si le code source compile ou non. Ce code sera testé dans un docker ou une plateforme java pour éviter d'avoir des problèmes sur le serveur. Le problème est lié à l'injection de code source malveillant. Si un code malveillant est amené à être exécuté sur le serveur, mieux vaut le faire dans un conteneur bien encapsulé plutôt que directement sur le serveur.

La notation peut donc être donnée via une vérification de la compilation (succès ou non), les warnings, fuites mémoires, convention de nommage, redondance de code, analyse AST (Abstract Syntax Tree), ... On pourrait donc calculer une partie de la notation en analysant ces logs de compilation / analyse de code (Valgrind, Sonar, ...).

On utilise en général les éléments suivants pour évaluer la qualité d'un code source :

- commentaire
- convention de nommage / règles de codage
- tests unitaire
- duplication de données
- complexité (spatiale et temporelle)
- bugs potentiels
- architecture

Des tests intéressants seraient d'exécuter un code source et de vérifier que l'on satisfait un certain nombre de tests unitaires ou / et assertions.

Chaque question pourra (ou non) permettre d'afficher les différents résultats liés aux tests internes (exemple : redondance, tests unitaire non passés, ...).

On commence par sélectionner le sous-module de code source utilisé. Comme précédemment, on définit un titre à la **question** puis on définit la ou les propositions.

Langages de programmation : java, C, ...

En langage de programmation, le créateur de contenu est amené à saisir du contenu qu'il peut choisir comme étant des tests, des assertions, du texte, un algorithme, et / ou du code source du même type que celui demandé en entrée, des spécifications peuvent être données pour par exemple donner des consignes sur les noms de fonction à utiliser ou à ne pas utiliser. Ensuite, il faut définir la fonction de pondération pour noter la **réponse** à la **question**.

En entrée utilisateur, on accepte une zone de texte dans laquelle une coloration syntaxique est ou non active (selon les spécificités autorisées). On pourrait également directement ajouter des fichiers sources avec les options de compilation. Les sorties seront affichables ou non. Le testeur pourra tester son code en générant des contenus (texte, ou multimédia) avant de valider la **question** (si il est autorisé à le faire par les spécificités de la **question**).

LaTeX

On peut ajouter un compilateur \LaTeX et noter sur le fait qu'il n'y ait pas d'erreurs ou faire de l'analyse de texte sur les titres, paragraphes pour noter la qualité du document avec **BETSY** et comparer la ressemblance avec le rendu désiré. On pourrait utiliser ShareLatex pour cela.

UML

Pour l'**UML**, nous utiliserons PlantUML qui est un outil **Open Source** qui nous permet de traduire du texte en un diagramme **UML**. Le testeur pourra utiliser PlantUML pour pouvoir

visualiser son UML dans une image. La visualisation en cours de réponse se fait de la même manière que sur le site [plantText](#). Nous pourrions même effectuer des comparaisons entre deux projets [WWW2].

Ce type de question est évidemment difficile à évaluer de manière automatique. Même en imposant des noms, il est très difficile de pouvoir évaluer la qualité du travail avec une certitude assez élevée. pour être certain de la note attribuée.

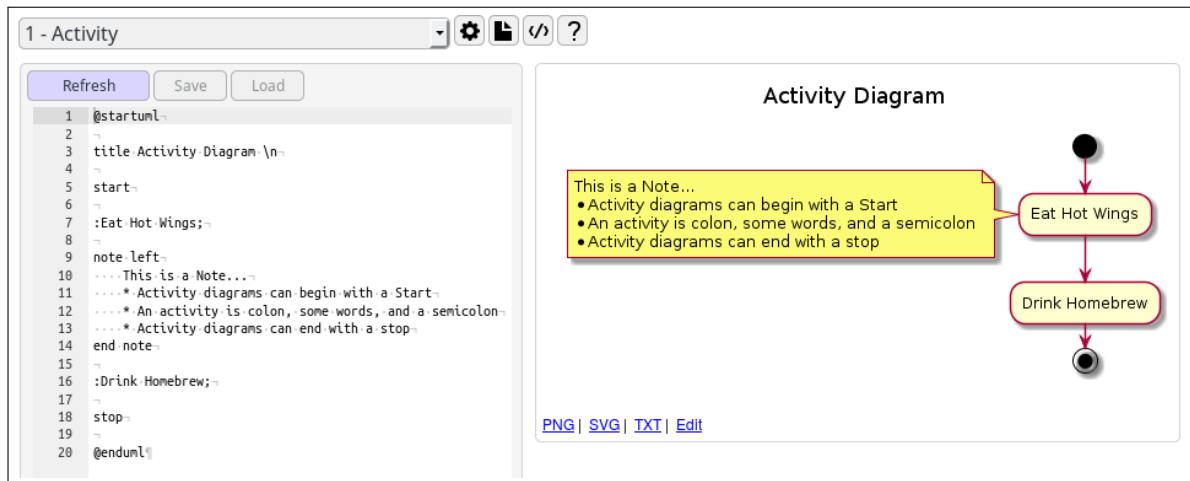


Figure 3 – IU - réponse à une question - UML

La Figure 3 est une capture d'écran du compilateur en ligne [plantText](#). La prise de décision sur la validité du diagramme étant peu certaine, un message sera envoyé à la personne responsable du quiz pour corriger la question de la personne ayant répondu à cette question.

Ecrire l'UML en PlantUML et pouvoir le comparer à une solution proposée pour l'évaluer.

Cette outil pourrait également être utilisé pour les examens ou pour faire proposer une architecture à un groupe de travail. Ainsi, chaque personne propose son architecture et il est ensuite possible de choisir la meilleure.

Graphe

Les graphes peuvent facilement être utilisés une fois plantUML opérationnel. En effet, on peut créer des graphes via [plantUML DOT](#). Il suffit de faire un `@startdot` puis un `@enddot` avec entre ces balises du DOT.

Autrement, nous pourrions utiliser la [librairie boost](#) pour rentrer des graphes et effectuer des calculs dessus. Un graphe fait avec boost est facile à traduire en DOT. Pour dessiner, plantUML est suffisant mais nous souhaitons effectuer un traitement important pour l'évaluation de la question, nous pourrions considérer l'utilisation de boost comme une bonne solution pour effectuer ces traitements.

Base de données

La création d'une Base De Données est faite via la création d'un docker contenant une BDD.

La BDD doit être simple mais si l'on travail sur de grosses bases de données, alors il faut pouvoir signaler que l'on travail sur une base de données externe.

Les requêtes et créations sont exécutées sur le docker.

On pourra (ou non) voir les retours de la BDD.

La notation consiste en la comparaison entre une BDD de référence et la BDD après l'exécution des commande ou au test d'un ensemble de commandes à lancer sur la BDD. On pourra donc saisir un certain nombre de commandes avec une notation pondérée suivant plusieurs critères.

Shell

Intégrer un Shell en ligne pour permettre à l'utilisateur de manipuler la console dans un docker se trouvant sur le serveur.

La notation serait faite par vérification en tapant automatique certaines commandes et en comparant la sortie attendue avec celle obtenue.

Mathématique

De la même manière que SciLab avec **Scilab rollapp**. On a une application SciLab lancée sur le serveur et l'on cherche à répondre à un certain nombre de questions. On pourra écrire des scripts et ces scripts seront exécutés sur le serveur pour faire des tests comme pour un langage classique.

Matching

Le matching est le fait de faire correspondre deux objets (ou plus) entre eux. On pourrait le faire sous forme de code même si une "bonne" façon de le faire serait plutôt de développer une interface graphique permettant de tracer des lignes ou de déplacer des objets dans la page.

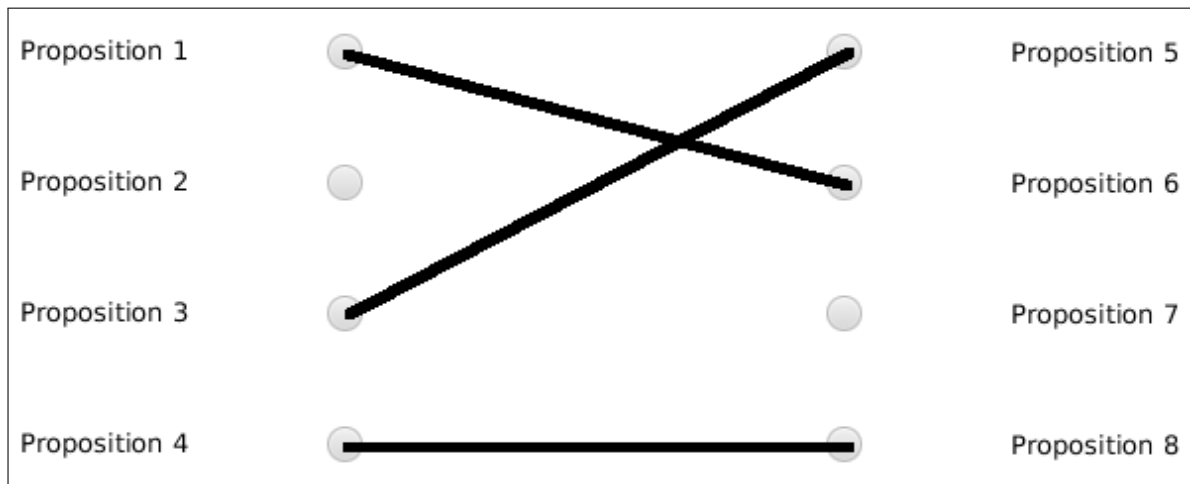


Figure 4 – Interface graphique - réponse à une question - matching

Dans ce **module**, il faut donc pouvoir utiliser n'importe quel rendu d'un type différent pour le lier à un autre. Evidemment, certains **modules** ne sont pas adaptés d'un premier coût d'œil mais ils faut bien penser que nous parlons du rendu final ou alors de l'entrée d'un module. Par exemple, pour un Shell, on parle de son entrée ou de sa sortie et en aucun cas son docker. Au final, le rendu utilisé sera un texte (avec possibilité de coloration syntaxique), une image, un son, une vidéo [...] qui sera à relier à une autre proposition. La **Figure 4** nous montre le principe général.

Par exemple, nous pouvons faire des associations entre des mots de même thèmes, entre des traductions, entre une image et un texte, entre deux images, ...

1.3 Multimédia

L'objet doit être stocké. On doit pouvoir le jouer une ou plusieurs fois selon l'option définie par le créateur. La **réponse** à un **module** de son est en fait un autre **module** choisi par le créateur. La **réponse** peut donc se faire sous n'importe quelle forme.

On peut également ajouter un média (micro, caméra ou fichier).

La création d'une **question** de type multimédia est faite via un fichier, ou un flux multimédia mais peu aussi se faire par un autre **module**.

On demande au testeur de fournir une entrée multimédia avec si souhaité une partie commentaire. Lors de ce rendu, les spécificités de la **question** autorisent ou non d'avoir un nouvel essai, de voir / écouter le média rentré ou même de l'éditer si un outil en permet l'édition (comme la retouche ou modification). Une fois un tel outil disponible, on pourrait modifier un média donné et pour lequel l'exercice serait de la modifier.

Suite à l'**évaluation**, les spécificités de la **question** permettront ou non de visualiser les critères de notations et la note obtenu.

1.3.1 Son

Les **questions** liées au son peuvent être multiples. On pourrait par exemple effectuer de la reconnaissance de musique par écoute (comme **shazam** ou **TrackID**). Ceci est faisable dans les deux sens :

- la musique se lance, il faut donc trouver le titre / auteur / année / note de musique ...
- on joue la musique, le micro écoute et on doit jouer assez bien pour que TrackID fasse un "match" / reproduire la musique

Nous pourrions citer d'autres exercices tels que

- trouver la note de musique jouée
- jouer une note de musique
- écoute de sons comme un sous-marin. Entraînements pour les ouïes d'or avec le sonar
- trouver l'objet et la distance (besoin d'un casque haut qualité)
- accordeur / reconnaissance / écoute

1.3.2 Image

Concernant les entrées d'images, on pourra donner un fichier ou prendre une photo via un périphérique. L'entrée est donc plutôt simple. Cependant, un **module** image n'est pas forcément une image en dans sa partie de création. Cela peut être un texte qui donne les instructions à l'utilisateur pour que celui-ci entre une image.

La partie de **réponse** est donc une image. Cette image peut être entrée de la même façon que pour la création de la question.

Les **questions** peuvent porter sur plusieurs critères

- retouche
- analyse
- comporte un objet (Reconnaissance de Forme, k Plus Proche Voisins, ...)

1.3.3 Vidéo

Le comportement du **module** de vidéo est identique aux deux **modules** précédents.

On y retrouve une création de glsquestion quelconque avec une entrée de **réponse**.

2 Modules spécialisés

Nous avons parlé des principaux types de questions auquel nous pouvons penser. Cependant, de nombreux types spécialisés existent.

- frise chronologique → édition d'images avec texte associé à l'image
- Laboratoire de langue → média
 - prononciation (prononcer en fonction du mot écrit ou de sa décomposition sous forme de prononciation)
 - écoute (donner l'écriture ou la décomposition de la prononciation)
- modélisation 2D/3D
 - équipements mécanique, Catia
 - plans (2D/3D pour architecture, dessins ou pièce mécanique)
- gestion d'un parc informatique
 - Windows Server
 - Packet Tracer (avec création de trames par exemple) haut ou bas niveau
- carte géographique interactive
 - faire un lien entre un point et ce qu'on écrit
 - donner le noms des pays
 - jeu de sélection des pays
 - trouver un pays / capital / langue ...
- analyse / création d'algorithmes
 - comptage d'après une photo (par exemple nombre de poissons sur une photo)
- Chimie
 - réactions chimiques
- Physique
 - Atomes et interactions
 - Art
 - média
 - dessins (à partir d'un planche à dessiner ou tactile)
 - ...

Dans cette liste de **modules** susceptibles d'être développés, les types de question ne vont pas toujours correspondre à une catégories précise mais plutôt à un mélange entre plusieurs. D'autre **modules** seront trop particuliers pour être assez globalisés dans un **module** générique.

D'une manière générale, les **modules** de base définis dans ce chapitre sont suffisants pour la majeure partie des **modules** spécialisés. Cependant leur utilisation sera certainement plus complexe et traitera certainement plusieurs objets pour une seule **question**.

Le problème de l'utilisation d'autres types de questions dans une question est le fait que les modules ne sont plus indépendants les uns des autres.

3 Priorisation de développement des contenus

Les **modules** seront développés au fil des années suivant les besoins.

Priorité	Module	Fait en 2017	Chapitre	Référence	Page
1	QCU	oui	Chapitre 3	Section 1.1.1	19
2	QCM	oui	Chapitre 3	Section 1.1	17
3	matching	non	Chapitre 3	Section 1.2.2	22
4	linguistique	non	Chapitre 3	Section 1.2.1	19
5	langage de développement	non	Chapitre 3	Section 1.2.2	20
6	UML	non	Chapitre 3	Section 1.2.2	20
7	graphe	non	Chapitre 3	Section 1.2.2	21
8	BDD	non	Chapitre 3	Section 1.2.2	21
9	mathématique	non	Chapitre 3	Section 1.2.2	22
10	shell	non	Chapitre 3	Section 1.2.2	22
11	image	non	Chapitre 3	Section 1.3.2	23
12	son	non	Chapitre 3	Section 1.3.1	23
13	vidéo	non	Chapitre 3	Section 1.3.3	23
14	L ^A T _E X	non	Chapitre 3	Section 1.2.2	20
15	modules spécialisés	non	Chapitre 3	Section 2	24

Le tableau précédent présente l'évolution du projet au fil des années. En ce qui concerne l'année 2016-2017, nous avons pour objectif les fonctionnalités suivantes :

Priorité	Fonctionnalité	Fait
1	architecture du projet	oui
2	définition des besoins	oui
3	choix technologiques	oui
4	gestion de la base de données H2 avec GORM	oui
5	accès à l'API	oui
5	créer des dossiers	oui
6	créer des dossiers si l'on a les droits dans le dossier	oui
7	ajouter des question dans un dossier	oui
8	ajouter des questions dans un dossier si l'on a les droits	oui
9	afficher la liste des question	oui
10	afficher la liste des questions d'un dossier	oui
11	afficher la liste des questions que l'on peut modifier	oui
12	afficher la liste des questions que l'on peut tester	oui
13	ajouter des utilisateurs	oui
14	ajouter des utilisateurs si l'on est admin	oui
15	ajouter des utilisateurs ayants un rôle de base	oui
16	ajouter des questions	oui
17	ajouter des quiz	oui
18	connexion sécurisé	oui
19	gestion des droits par objet	oui
20	gestion des QCM	oui
21	évaluation des QCM	oui
22	CRUD de question de type QCM	oui
23	faire des tests unitaires	oui
24	interface graphique	oui

4

Architecture du système

Le système est fait pour permettre d'enlever, d'ajouter ou de modifier des éléments sans impacter le reste du système. On distingue donc la vue (Angular2), le serveur (groovy), la base de données (H2) et les plateformes java (Java Platform). Le diagramme de la **Figure 1** nous montre ces principaux groupes.

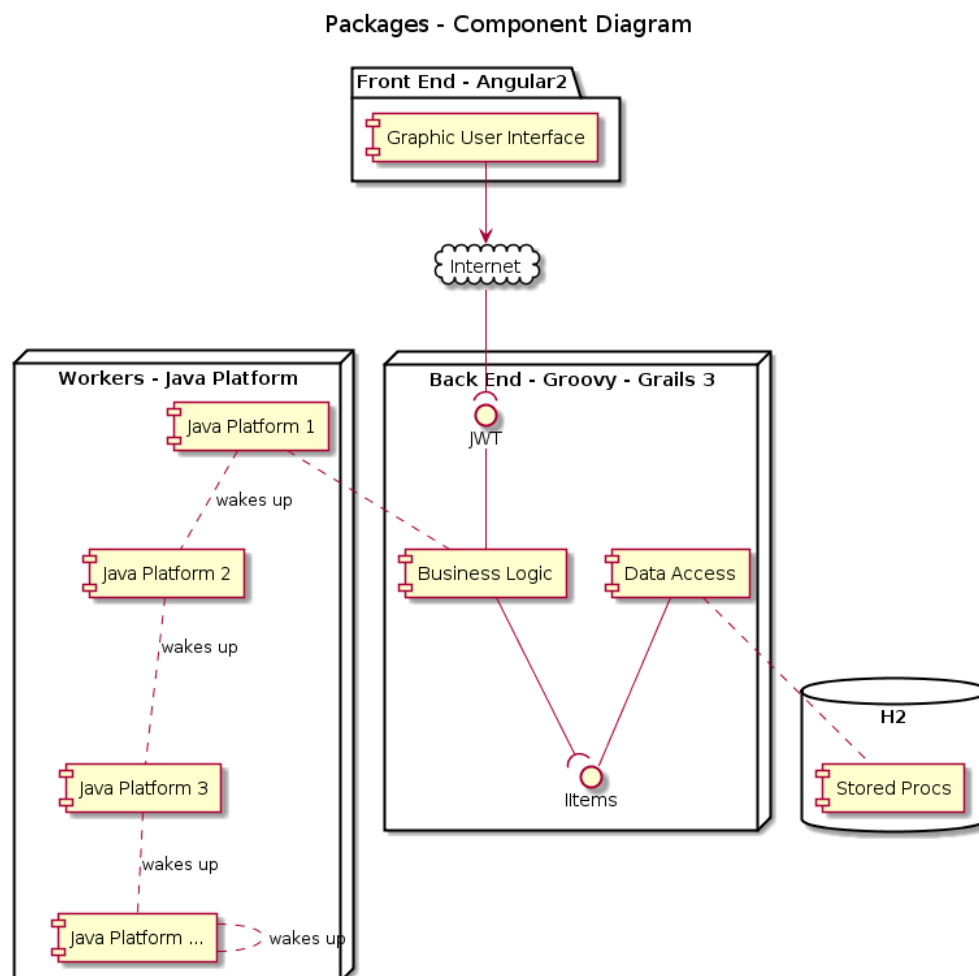


Figure 1 – Architecture général

Dans la **Figure 1**, on retrouve l'interface graphique qui sera faite en Angular2, la connexion avec un JsonWebToken au serveur groovy, le serveur qui est un serveur RESTful avec une connexion des workers à celui-ci pour évaluer les réponses en attentes de traitement. Dans ce système de Workers, on trouve le principe qui est de réveiller un autre worker si la charge de travail est trop importante pour un seul. Enfin, nous pouvons voir la connexion à la base de données H2 qui repose en fait sur Hibernate et GORM.

Ayant dissocié le parties front end et back end, nous avons Angular2 qui se charge de faire des opérations logiques (chez le client) et d'afficher les contenus des pages de l'API. Les échanges entre Angular et le serveur REST se font en json.

Le fichier json permet de transporter des objets à travers une connexion sécurisée (voir **Figure 2**). Le json représente mieux les objets et le contexte de l'application qu'un fichier XML, c'est à raison pour laquelle nous avons choisi le json. JWT nous permet de créer une connexion sécurisé entre notre client et le serveur REST [WWW6].

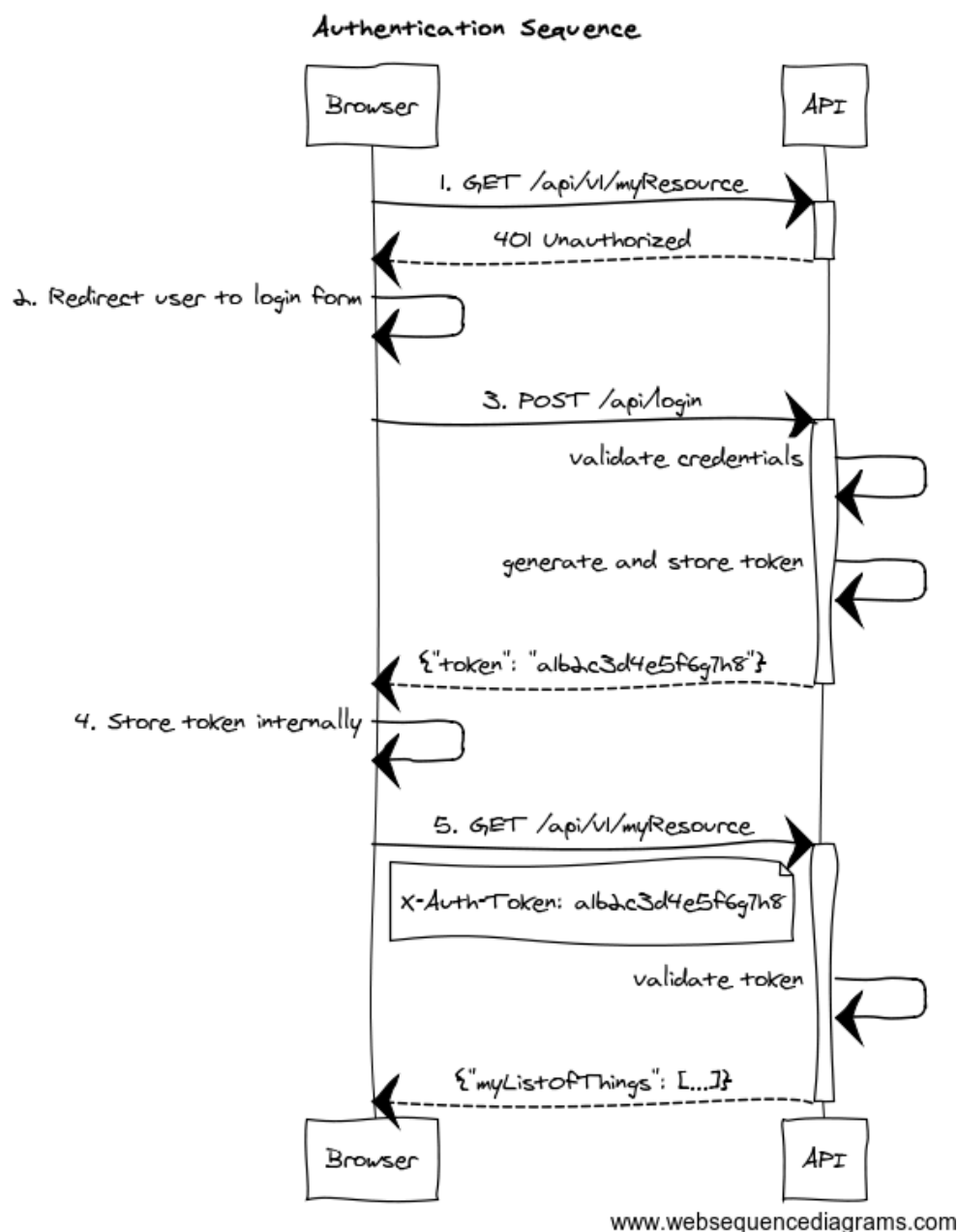


Figure 2 – *Diagramme de séquence d'une connexion JWT avec une API*

La **Figure 2** décrit donc les étapes d'une connexion entre le client (interface Angular2) et notre API. Le client n'étant au début pas connecté, celui-ci n'est pas autorisé à accéder aux ressources du système. Il doit donc se connecter à l'API via un `POST /api/login`. Ce POST est sécurisé et sera traité par l'API. L'API, une fois les identifiants de connexion vérifiés, retourner un json avec un token.

Un token est en fait la clé d'accès au système. Le token n'est valide que pendant un certain temps. Une fois la limite de vie du token atteinte, l'utilisateur devra demander un nouveau token avec un refresh token qui lui a été donné lors de sa connexion à l'API.

Lorsque l'utilisateur souhaite ensuite accéder à des données du système, il lui suffit de faire sa requête avec, en plus, son token dans l'en-tête de cette dernière.

Nous aurions également pu faire en sorte stocker les informations de connexions directement dans le cache de l'utilisateur mais cela n'est pas sécurisé. Stocker ces informations telles qu'elle permettrait de faire du vole de session (**XSRF** - cross-site request forgery - session hijacking). La gestions des token permet d'éviter ce problème avec le système de vérification du token.

L'utilisation de token nous permet également de pas enregistrer sur le serveur l'état du client. C'est le client qui stock son propre état et non le serveur ce qui évite de devoir gérer les sessions de connexion.

C'est spring security qui fait le lien entre JWT et le serveur REST. Cette gestion est fait par le **plugin spring-security-core** depuis la version 3.x de celui-ci.

Nous avons donc décrit les différentes parties du projet. Ces parties sont pour la plupart des plugins intégrées à grails 3. La **Figure 3** nous montre comment Grails 3 ne pose en surcouche de java. Avec groovy et le framework Grails 3, nous sommes donc à même d'utiliser Spring avec toutes ses dépendances, hibernate 5 (qui est lui-même englobé par GORM). Enfin, il est important de rappeler que le groovy est totalement basé sur du java, nous pouvons donc écrire la totalité de notre code source en java si nous le souhaitons.

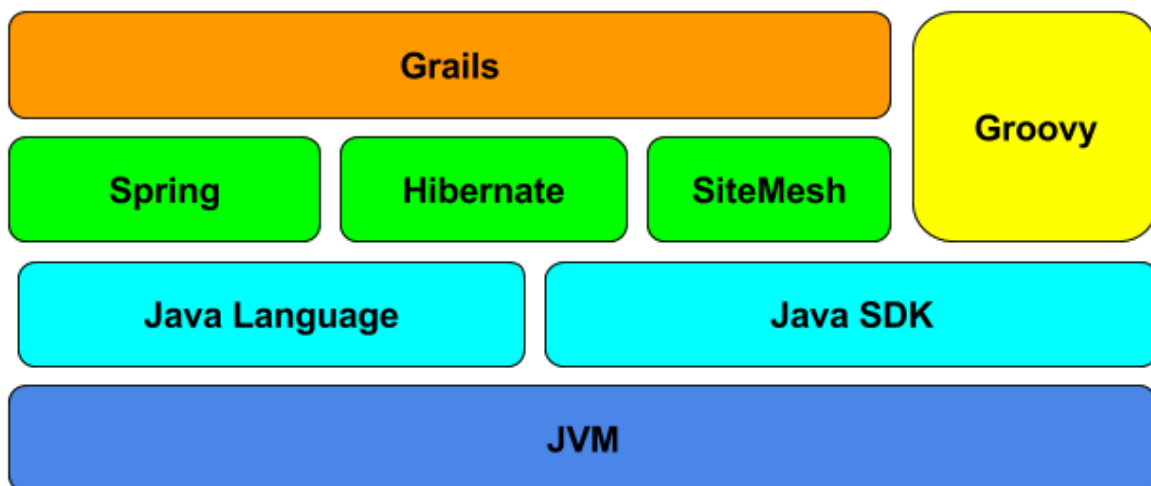


Figure 3 – *Technologies incluent dans Grails 3*

5

Mise en œuvre

Dans ce chapitre, nous cherchons à expliquer comment les développements ont pu être réalisés. A la manière d'un tutoriel, nous allons décrire comment créer un projet, comment le gérer et comment le lancer.

La mise en œuvre est donc un recueil entre le guide du développeur et les réflexions liées au projet.

1 Installation de Grails 3

Grails 3 est un framework du langage groovy. Nous considérons que le développeur possède déjà le JDK 8 oracle installé sur sa machine.

Que ce soit pour groovy ou grails, l'installation est simple. Il suffit de les installer avec SDKMAN.

1.1 SDKMAN

L'installation d'SDKMAN est elle aussi très simple comme le montre le [site officiel](https://get.sdkman.io) d'SDKMAN.

```
1 \ $ curl -s "https://get.sdkman.io" | bash
```

Une fois le téléchargement terminé et sdkman installé, l'installation se termine par ceci :

```
1 \ $ source "$HOME/.sdkman/bin/sdkman-init.sh"
```

On peut ensuite vérifier la version d'SDKMAN avec

```
1 \ $ sdk version
```

Cette commande va nous renseigner sur la version installée.

1.2 groovy

Le [site officiel](#) de groovy explique les différents moyens d'installer groovy. Avec SDKMAN, il suffit de faire un simple

```
1 \ $ sdk install groovy
```

pour installer groovy.

1.3 gradle

Le framework Grails dépend de gradle, il est donc essentiel de l'avoir sur sa machine. Le [site officiel](#) de gradle permet de télécharger le framework depuis SDKMAN comme cela :

```
1 \ $ sdk install gradle
```

1.4 grails

De même que pour groovy et gradle, le [site officiel](#) de grails permet de télécharger le framework depuis SDKMAN comme cela :

```
1 \ $ sdk install grails
```

Il est également possible d'installer java depuis SDKMAN.

Une fois ces installations faites, il est conseillé de vérifier les versions installées avec la commande

```
1 \ $ sdk current
```

qui devrait retourner les lignes suivantes (jeudi 30 mars 2017) :

```
1 Using:
2
3 gradle: 3.4.1
4 grails: 3.2.8
5 groovy: 2.4.8
```

Pour faire une mise à jour d'SDKMAN et des différentes bibliothèques installées, il suffit de faire

```
1 \ $ sdk upgrade
```

2 Débuter un projet

2.1 Création du projet

Pour commencer un projet grails, il faut se placer dans le répertoire dans lequel on souhaite créer son projet.

```
1 /home/\ $
```

Dans ce répertoire, nous allons entrer la commande suivante :

```
1 /home/\ $ grails create-app project --profile angular2 -features
    json-views,hibernate5,security,asset-pipeline,less-asset-pipeline
```

Cela signifie que nous créons un projet grails 3 qui s'appelle *project*.
On distingue ensuite deux options.

-profile est le type de projet créé. Nous pourrions par exemple utiliser le profile *rest-api* ou *angularjs* mais nous avons choisi le profil *angular2* qui est le dernier profile disponible. Ce profile est en fait basé sur le profile *rest-api*. Le principe de ce profile est de créer deux serveurs. Le premier serveur *client* est le serveur Angular2 sur lequel le client va pouvoir se connecter, le second serveur est *server* qui va héberger notre serveur REST groovy.

La seconde option de notre commande est *-features*, c'est une liste de fonctionnalités que nous souhaitons ajouter à notre serveur. En cas d'oubli, ces fonctionnalités se rajoutent très facilement dans notre projet. Il n'est donc pas nécessaire de les utiliser dans la commande de création du projet.

```

/home/project$
├─ build/ ..... LE RÉPERTOIRE DE BUILD
├─ client/ ..... ANGULAR2
├─ gradle/ ..... LE WRAPPER
├─ server/ ..... GRAILS 3
└─ settings.gradle ..... LE FICHIER QUI FAIT LE LIEN ENTRE NOS DEUX SERVEURS

```

Figure 1 – Répertoire de travail du projet

La structure du projet est présentée dans le Figure 1. Elle est assez simple et permet de bien voir le principe REST avec deux serveurs distincts.

Pour faire fonctionner le projet, il nous faut également *angular-cli* installé sur le client Angular2. *angular-cli* nécessite au minimum Node 5 et NPM 3. Il faut donc les installer :

```

1 /project\$ gradle npmInstall
2 /project\$ curl -sL https://deb.nodesource.com/setup_6.x | sudo -E bash -
3 /project\$ sudo apt-get install -y nodejs
4 /project\$ npm install -g @angular/cli@latest

```

Ces commandes permettent d'installer NPM, NodeJS 6, et angular-cli.

Il est possible que certains packages NPM soient mal installés, dans ce cas, on peut les installer comme ci :

```

1 /project\$ apt-get install npm
2 /project\$ npm install graceful-fs
3 /project\$ npm install minimatch
4 /project\$ npm install tough-cookie
5 /project\$ npm install uuid
6 /project\$ npm ls graceful-fs
7 /project\$ npm ls minimatch
8 /project\$ npm ls tough-cookie
9 /project\$ npm ls uuid

```

J'ai eu plusieurs fois des problèmes avec les packages cités ci-dessus.

2.2 Lancer le projet

Pour lancer le projet, on utilise le fichier *gradlew*. Celui-ci prend en entrée les paramètres d'exécution. La commande

```

1 /project\$ ./gradlew bootRun --parallel

```

devrait normalement pouvoir lancer les deux serveurs. Cependant elle ne fonctionne pas. Ce bug sera certainement modifié dans les prochaines versions.

Pour pouvoir lancer les deux serveurs, il faut donc lancer chaque serveur dans une console différente :

```
1 /project/$ ./gradlew client:bootRun
```

```
1 /project/$ ./gradlew server:bootRun
```

On pourrait également lancer les serveurs avec les commandes suivantes

```
1 /project/server/$ grails run-app
2 /project/client/$ ng start
3 /project/client/$ ng serve --host 127.0.0.10 --port 4201 --live-reload-port 49153
```

mais cela n'est pas conseillé. Il est mieux d'utiliser gradle pour lancer les serveurs en modifiant les réglages depuis gradle.

Un fois les bootRun lancés, il est possible d'accéder à nos serveurs. On peut d'ailleurs voir les logs dans les consoles.

2.2.1 Tuer un serveur

Il est possible que l'arrêt d'un des serveurs ne se fasse pas bien. Une commande utile est donc la suivante :

```
1 /project/$ sudo kill $(sudo lsof -t -i:4200)
```

Par défaut, le serveur client est lancé sur le port 8080 et l'API sur le port 4200.

3 Modifier l'API

3.1 Modifier les logs

L'un des premiers changements à effectuer est d'augmenter la granularité des logs car la configuration d'origine n'apporte aucune aide à débogage du serveur.

Pour cela, il faut modifier le fichier *projet/server/grails-app/conf/logback.groovy*. Ce fichier est celui gérant les logs de notre API. Dans ce fichier, j'ai ajouté un certain nombre d'affichages en plus :

```
1 logger("org.springframework.security", DEBUG, ['STDOUT', 'FULL_STACKTRACE'], false)
2 logger("grails.plugin.springsecurity", DEBUG, ['STDOUT', 'FULL_STACKTRACE'], false)
3 logger("org.pac4j", DEBUG, ['STDOUT', 'FULL_STACKTRACE'], false)
```

Il est également possible de lancer le serveur avec l'option *full-stacktrace* ou *stacktrace* mais cela va afficher toutes les actions du serveur dans son intégralité. Je déconseille donc l'utilisation de ces options lorsque nous n'avons pas besoin de beaucoup de précisions. Si nous voulions afficher ces logs détaillés, nous pourrions ajouter ceci dans le gestionnaire de logs :

```
1 root(ERROR, ['INFO', 'FULL_STACKTRACE'])
```

Pour finaliser l'usage de ces logs et leur affichage dans la console, il faut modifier la partie *bootRun* du fichier *projet/server/build.gradle*. Ce fichier est très important dans le projet comme expliqué dans la [Section 3.2](#). Concernant les logs, nous allons changer le *bootRun* comme ceci :

```

1 bootRun {
2     // jvmArgs('-Dspring.output.ansi.enabled=always')// original version
3     // If System.console() return non null instance,
4     // we force ANSI color support with 'always',
5     // otherwise use default 'detect'.
6     jvmArgs = ['-Dspring.output.ansi.enabled=' + (System.console() ? 'always' : 'detect')]
7 }

```

Vous avez pu voir que nous avons activé les logs pour *springsecurity*. Nous allons maintenant expliquer comment vérifier et ajouter les dépendances spring.

3.2 Ajout de dépendances

Dans le fichier *projet/server/build.gradle*, nous venons de voir la partie *bootRun*. L'autre fonction de ce fichier est la gestion des dépendances.

Les dépendances, ce sont tous les plugins ou librairies dont nous avons besoin pour notre projet. Ce fichier permet donc de centraliser les informations pour tout gérer depuis ce dernier.

On y retrouve donc de nombreuses dépendances. Les dépendances sont téléchargées, installées et auto-configurées. Le répertoire de dépôt grails est visible comme ceci :

```

1 repositories {
2     mavenLocal()
3     maven { url "https://repo.grails.org/grails/core" }
4 }

```

Ces lignes veulent dire "pour les dépendances qui suivent, va les chercher dans ce répertoire".

On trouve ensuite un grand nombre de dépendances comme celles-ci :

```

1 dependencies {
2     compile "org.springframework.boot:spring-boot-starter-logging"
3     compile "org.springframework.boot:spring-boot-autoconfigure"
4     compile "org.grails:grails-core"
5     compile "org.springframework.boot:spring-boot-starter-actuator"
6     compile "org.springframework.boot:spring-boot-starter-tomcat"
7     compile "org.grails:grails-plugin-url-mappings"
8     compile "org.grails:grails-plugin-rest"
9     compile "org.grails:grails-plugin-codecs"
10    compile "org.grails:grails-plugin-interceptors"
11    compile "org.grails:grails-plugin-services"
12    compile "org.grails:grails-plugin-datasource"
13    compile "org.grails:grails-web-boot"
14    compile "org.grails:grails-logging"
15    compile "org.grails.plugins:cache"
16    compile "org.grails.plugins:hibernate5"
17    compile "org.hibernate:hibernate-core:\$hibernateCoreVersion"
18    compile "org.hibernate:hibernate-ehcache:\$hibernateEhcacheVersion"
19    compile "org.grails.plugins:views-json"
20    compile "org.grails.plugins:views-json-templates"
21    compile "org.grails.plugins:spring-security-rest:\$springSecurityRestVersion"
22    compile "org.grails.plugins:spring-security-acl:\$springSecurityAclVersion"//ACL for ←
    Domain Objects security
23    console "org.grails:grails-console"
24    compile "org.grails.plugins:spring-security-core:\$springSecurityCoreVersion"
25    // compile 'org.grails.plugins:spring-security-cas:3.0.0'//for CAS authentication
26    profile "org.grails.profiles:angular2"
27    runtime "com.h2database:h2"
28 }

```

Cette liste de dépendances reprend les plus importantes avec le core grails, les services, spring-framework, spring-security, ...

Il est donc facile d'ajouter des dépendances à notre projet en ajoutant les références du plugin. On peut trouver ces références sur le site officiel de grails. Cependant, il faut faire attention de prendre un plugin utilisable avec Grails 3+ [WWW5].

Ce fichier est en fait complété par *projet/server/gradle.properties*. Dans ce fichier, on peut définir des constantes comme les versions des dépendances utilisées. Pour notre projet, nous avons par exemple les versions suivantes :

```

1 grailsVersion=3.2.4
2
3 jsonViewsVersion=1.2.0
4
5 hibernateVersion=6.0.4
6 hibernateCoreVersion=5.1.2.Final
7 hibernateEhcacheVersion=5.1.2.Final
8
9 springSecurityRestVersion=2.0.0.M2
10 springSecurityAclVersion=3.1.0
11 springSecurityCoreVersion=3.1.1

```

Nous pouvons donc désormais ajouter des plugins comme nous le souhaitons et modifier les version rapidement.

3.3 Architecture de l'API

Avant de pouvoir ajouter des objets et gérer les comptes, arrêtons-nous à la structure de l'API. C'est le répertoire *grails-app* qui contient la plus grande partie de notre code source. A l'intérieur, on y trouve la configuration de l'API, les contrôleurs, les domaines (qui sont nos objets), le répertoire multi-linguistique, le répertoire d'initialisation du serveur, les services, les utilitaires dédiés à l'API (avec interactions des objets) et les vues.

3.4 Répertoire de configuration

Le répertoire de configuration de l'API comporte un fichier *application.groovy* si spring-security est activé dans les dépendances du projet. Nous en reparlerons lors de la configuration de la sécurité.

Le fichier *application.yml* est un document de configuration qui va être lu lors du lancement du serveur. Il permet de configurer grails, spring, le mapping, hibernate et l'environnement de travail (base de données).

Dans le cadre d'un simple projet, l'auto-configuration est souvent suffisante. Cependant, si nous ajoutons par exemple une connexion à celene (CAS), une fois la dépendance ajoutée, avec seulement une quinzaine de lignes de code, la page de connexion pourrait prendre en compte de moyen de connexion. Il en va de même pour une connexion à facebook ou gmail.

3.5 Ajouter des domaines

Un domaine est en fait un objet que l'on va pouvoir manipuler. A un domaine, on va assigner un contrôleur, un service et des vues.



Figure 2 – Répertoire de l'API

Pour notre projet, la **Figure 3** nous montre les différentes classes de notre projet. Dans nos exemples, nous allons traiter de cas concrets sur ce diagramme.

Pour créer un domaine Question dans le package projet.question, il suffit de faire commande suivante :

```
1 /project/server$ grails create-domain-class project.question.Question
```

Un domaine *Question.groovy* sera alors créé dans le répertoire correspondant. Un nouvel objet est créé, nous pouvons le compléter comme ceci :

```

1 package project4.question
2
3 import project4.question.qcm.Qcm
4 import project4.quiz.Quiz
5
6 class Question {
7     String name //le nom de la question
8     String theme //le thème abordé (peut être de type Tag
9     String body //une description plus détaillée de la question
10
11     boolean published //true si on choisi de publier la question
12     int chrono //le temps écoulé depuis que la question est apparue à l'écran
13     int timeLimit //le temps maximum (compte à rebours) pour répondre
14     int maximumTry //le nombre d'essais maximum
15     boolean seeScore //l'autorisation ou non de voir sa note
16     boolean seeAnswer //l'autorisation ou non de voir la réponse à la question
17     boolean goBack //l'autorisation ou non de retourner en arrière
18     boolean pass //l'autorisation ou non de passer la question
  
```

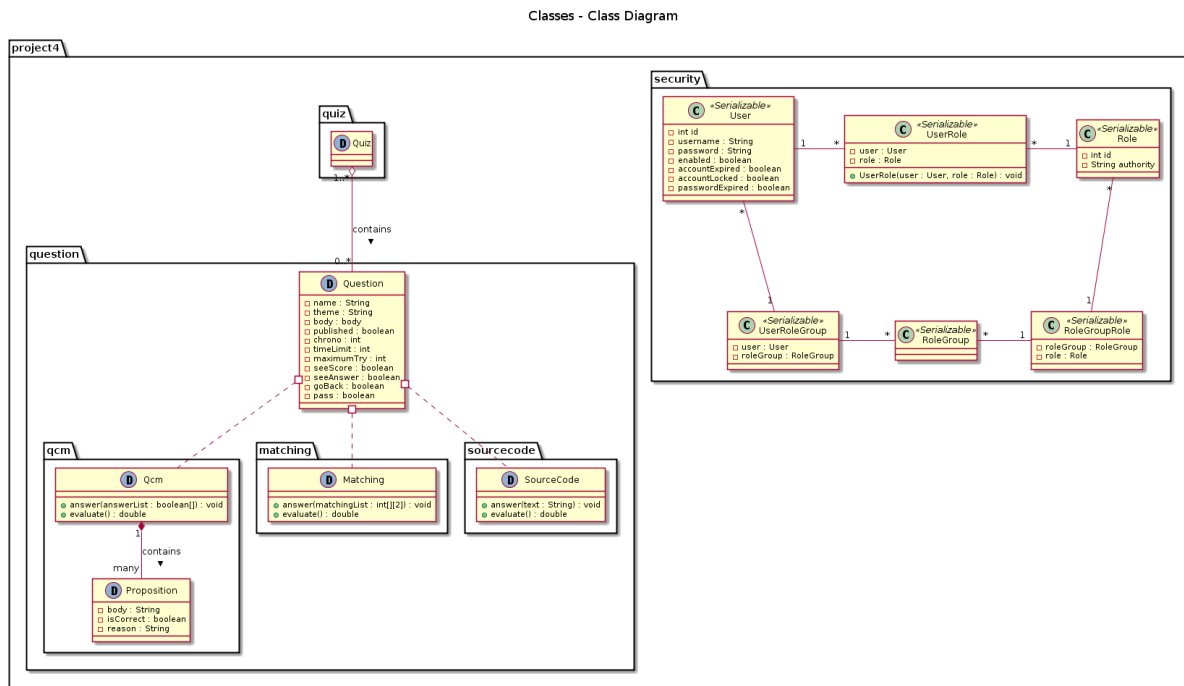


Figure 3 – Diagramme de classe

```

19
20 static belongsTo = [quiz: Quiz] //le lien d'une Question vers des Quiz
21
22 static constraints = {
23     name size: 5..255, blank: false //le nom de la question doit contenir entre 5 et 255 caractères
24     theme blank: false //il faut donner un tag à la question
25 }
26
27 static mapping = {
28     tablePerSubclass true //les extensions de la classe seront stockées dans d'autres tables
29 }
30 }

```

Dans ce domaine, nous avons déclaré un certain nombre d'attributs pour lesquels on se passera de commentaires. Ce qui est important est de voir la relations entre les objets comme la ligne 20. On y voit qu'une Question appartient à des Quiz. C'est une relation GORM [WWW4].

Les associations en GORM sont les suivantes :

```

1 static belongsTo = [a:A]
2 static hasOne = [b:B]
3 static hasMany = [cs: C]

```

On peut ensuite effectuer du mapping sur ces relations. Il est conseillé faire en sorte que c'est relations soient bi-directionnelles pour améliorer la qualité du code généré. Sauf cas particulier, les relations fonctionnent en cascades.

On constate également qu'il est possible de mettre des contraintes sur les attributs de l'objet.

Un domaine doit rester aussi clair que possible. Il se doit d'être bien documenté pour permettre une compréhension parfaite de l'objet.

Dans notre cas, nous avons étendu le domaine *Question* par les domaines *Qcm* et *Matching*

(comme montrer dans la [Figure 3](#)). Il suffit de faire un *extend* comme dans une application java traditionnelle.

3.6 Ajouter des contrôleurs, services et vues

Pour générer les vues et le contrôleur lié à un domaine, il faut faire la commande suivante

```
1 /project/server/$ grails generate-all project.question.Question
```

Les fichiers générés sont les suivants :

- `/project/server/grails-app/controllers/project/question/QuestionController.groovy`
- `/project/server/grails-app/views/question/_question.gson`
- `/project/server/grails-app/views/question/index.gson`
- `/project/server/grails-app/views/question/show.gson`

Si jamais la classe est *abstraite*, alors ces fichiers seront tous ignorés à la compilation.

Le premier fichier est le contrôleur de la classe. C'est là que nous allons gérer les accès avec les autorisations spring-security et les ACL. De base, le contrôleur auto-généré contient tout le service. Il est donc utile de pouvoir créer `/services/project/question/QuestionService.groovy` à la main. Le service doit pouvoir effectuer des actions sur l'objet (et donc la base de données) mais pas le contrôleur. Séparer le contrôleur du service fait partie des meilleurs protiques de développement. Cela permet de pouvoir différencier l'utilité de chaque partie de code source. Ainsi, la vue appelle le contrôleur qui va contrôler les accès et appeler des méthodes du service qui va lui-même interagir avec le modèle et la base de données. Ensuite, le service doit soit retourner l'information directement à la vue ou passer repasser par le contrôleur pour qu'il l'envoi au client qui va mettre à jour sa vue.

Les vues sont gson sont appelés pour créer un fichier json qui sera transmit au client Angular2. `_question.gson` est un template de vue concernant l'objet *Question*. `index` permet de donner la liste des instances de *Question* et `show` permet de montrer en détail une instance de *Question*.

On peut évidemment créer ses propres fichiers gson et fonctions. Dans notre applications, les services vont chercher les informations, les retournent aux contrôleurs qui vont ensuite les metrent dans le gson. Ces gson sont ensuite envoyés aux clients.

Grails fonctionne pas convention c'est grâce à cela que nous n'avons pas besoin de configurer les chemins entre les contrôleurs, domaines, services et vues. Grails va chercher dans des répertoire en fonction du package auquel appartient l'objet et par rapport à la classe de l'objet. Dans le dossier *controllers*, nous trouvons tout de même un mapping static permettant (si besoin) de changer les chemins par défaut. Par exemple, la page de bienvenue correspond à `www.projectsforge.org/`; ce dernier / peut être résolu comme ceci :

```
1 "/"(controller: 'application', action: 'index') //welcome page: default page
```

La vue disponible dans `views/application/index.gson` sera donc notre page par défaut.

Le mapping permet aussi de définir des groupes pour permettre à certains objets d'être fils d'un autre, de n'autoriser que certaines actions, ...

3.7 Définir un proxy

Pour permettre la redirection de certains URLs vers d'autres URLs (donc de faire un proxy), il suffit d'ajouter à la racine du client le fichier `proxy.conf.json` Ici, nous voulons rediriger toutes les URLs en `/api` et `/rest` vers notre API.

Voici à quoi ressemble le fichier `proxy.conf.json`

```

1 {
2   "/api": {
3     "target": "http://localhost:8080",
4     "secure": false
5   },
6   "/rest": {
7     "target": "http://localhost:8080",
8     "secure": false
9   }
10 }

```

Pour permettre au build de prendre en compte ce fichier, il suffit de modifier le fichier *package.json* se trouvant à la racine du serveur (client). Dans ce fichier, il faut modifier la commande du démarrage du serveur *"ng serve"* par : **"start" : "ng serve --proxy-config proxy.conf.json"**. Désormais, lors de l'exécution de la commande *ng start*, le build ira chercher la configuration proxy du projet.

3.8 Ajouter Spring Security

On demande aux utilisateur de se logger. L'approche sécuritaire adoptée est *pessimistic lockdown*, c'est-à-dire que toute l'application web est bloqué sauf l'authentification. Ainsi, chaque demande de page redirige vers la page d'authentification.

Notre partie sécurité est gérée dans *application.groovy*.

C'est là où l'on gère nos filtres de sécurité avec un filtre total sauf sur le */api*.

3.8.1 Créer des Role et User

Ajouter le package de sécurité

A la racine du serveur *"/api/"*, il suffit d'exécuter la commande suivante pour générer les différents composants de sécurité :

```

1 /project/server$ grails s2-quickstart project.security User Role ↩
   --groupClassName=RoleGroup

```

Les composants de sécurité sont les suivants (comme visibles sur la [Figure 3](#) :

- Role : l'autorité qui gère les accès
- User : l'utilisateur qui n'est autre qu'un compte utilisateur
- UserRole : le mapping many-to-many entre un Role et un User
- RoleGroup : le mapping many-to-many entre RoleGroupRole et UserRoleGroup (la liste des autorisation d'un groupe d'utilisateurs)
- UserRoleGroup : le mapping many-to-many entre User et RoleGroup
- RoleGroupRole : le mapping many-to-many entre Role et RoleGroup

```

1 /project/server$ grails s2-create-persistent-token project.security.PersistentLogin

```

Cela va créer un token persistant pour permettre l'utilisation d'un cookie *remember-me*.

Authentification

L'authentification se fait à l'adresse *http://localhost:8080/api/login*. On envoi un Json Web Token (JWT) de la forme suivante pour créer une connexion et générer un token. Une fois le *POST* fait,

le serveur va récupérer le token, chercher l'utilisateur et le mot de passe pour créer un token associé et le renvoyer.

Header :

```
1 Accept application/json
2 Content-Type application/json
```

Body :

```
1 {
2   "username": "admin123456",
3   "password": "123456789"
4 }
```

L'authentification retourne un json comme ceci :

```
1 {
2   "username": "admin123456",
3   "roles": [
4     "ROLE_ADMIN"
5   ],
6   "token_type": "Bearer",
7   "access_token": "eyJhbGciOiJIUzI1NiJ9.eyJwcm1uY2lwYWwJST0xZXdk2fQ.RoP_MbZdD",
8   "expires_in": 3600,
9   "refresh_token": "eyJhbGciOiJIUzI1NiJ9.eyJwcm1uY2lwYWw4ODQ3NjA5Nn0.1WS9Fih0"
10 }
```

Cela nous rappelle le *username*, ses *roles* et nous donne un token d'accès et un autre pour son rafraîchissement (tous deux de type *Bearer* par défaut).

JsonWebToken (JWT)

Le JWT est normalement auto-géré par springsecurity. Il peut cependant être customisé même si le type *Bearer* est fortement recommandé à l'utilisation puisqu'il fait parti des standard d'autorisation (voir la RFC 6750 [WWW7]).

3.8.2 Ajouter des droits d'accès par objet

Spring sécurité permet d'utiliser des annotations pour sécuriser des contenus. L'annotation `@Secured` permet de gérer les accès à certaines classes ou méthodes.

Par exemple, on pourrait sécuriser un contrôleur avec

```
1 @Secured('ROLE_ADMIN')
```

et sécurisé de façon différente l'un de ses méthodes via une *closure* (ici l'autorisation est garantie car on retourne vrai dans tous les cas)

```
1 @Secured(closure = {
2   println "\tauthentication.name: " + authentication.name
3   ctx.SecurityManagerService.check()
4   return true
5 })
```

Les fermetures (closures) sont très utilisées en groovy. Elles permettent d'exécuter du code supplémentaire en paramètre d'autres méthodes.

4 Interagir avec l'API

4.1 Modifier des données depuis BootStrap

Le fichier *BootStrap.groovy* est exécuté à chaque lancement du serveur. Il permet par exemple de faire certaines action suivant l'environnement lancé.

```

1  def init = { servletContext ->
2
3      switch (Environment.current){
4          case Environment.DEVELOPMENT:
5
6              //setup security groups and users
7              TestableSet.createSecurityProfiles()
8
9              //create several questions
10             TestableSet.createQuestion(5)
11             println "List of Questions"
12             println Question.list()
13
14             break
15         case Environment.PRODUCTION:
16             break
17
18         case Environment.TEST:
19             break
20     }
21 }
```

Dans l'exemple ci-dessus, si notre environnement est celui "DEVELOPMENT" alors nous allons appeler des méthodes présentes dans le fichier *TestableSet.groovy*.

Créer des User et Role

La fonction *createSecurityProfiles()* va donc créer des utilisateurs suivant une liste d'utilisateurs prédéfinis dans la fonction.

Nous créons en fait des utilisateurs

```

1  User userAdmin = new User(username: 'admin123456', password: '123456789', enabled: ←
    true).save()
2  User userTester= new User(username: 'userTester', password: '123456789', enabled: ←
    true).save()
```

et des rôles avec des noms d'autorité

```

1  Role roleSuperAdministrator = new Role(authority: 'ROLE_SUPER_ADMINISTRATOR').save()
2  Role roleAdmin = new Role(authority: 'ROLE_ADMIN').save()
```

puis nous créons des groupes de rôles

```

1  RoleGroup roleGroupAdmin = new RoleGroup(name: 'ROLE_GROUP_ADMIN').save()
2  RoleGroup roleGroupUser = new RoleGroup(name: 'ROLE_GROUP_USER').save()
```

et des groupes d'utilisateurs et nous mappons les groupes de rôles avec les groupes d'utilisateurs

```

1  RoleGroupRole.create roleGroupAdmin, roleAdmin
2  RoleGroupRole.create roleGroupAdmin, roleDeployer
```

En fait, pour spring, les notions de groupes de rôles et groupes d'utilisateurs n'existent pas car tout cela est ensuite traduit par une simple liaison *many to many* entre les utilisateurs et les rôles. Cependant, ils peuvent être intéressants d'exploiter cette fonctionnalité pour créer nos groupes (même si spring security ne les voit pas).

Créer des objets

`createQuestion(int numberOfQuestionToCreate)` est une méthode permettant de créer de façon automatique des *Questions* de type *Qcm* avec deux propositions :

```

1      numberOfQuestionToCreate.times {
2          Qcm test = new Qcm(//create a questions with name, theme and body related to ←
3              it's id number
4              name: "name\${it}",
5              theme: "theme\${it}",
6              body: "body\${it}",
7              published: true,
8              chrono: 0,
9              timeLimit: 100,
10             maximumTry: 5,
11             seeScore: true,
12             seeAnswer: true,
13             goBack: true,
14             pass: false,
15             propositions: [//add 2 propositions to the qcm's proposition list
16                 new Proposition(//add a proposition refering ←
17                     to the question's id
18                     body: "body proposition \${it}",
19                     reason: "reason proposition \${it}",
20                     isCorrect: true
21                 ),
22                 new Proposition(
23                     body: "body proposition \${it*10}",
24                     reason: "reason proposition \${it*10}",
25                     isCorrect: false
26                 )
27             ]
28         ).save()
29     }

```

Si `numberOfQuestionToCreate` est égale à 4, alors nous aurons 4 questions de type *Qcm* et 8 propositions. Pour rappel, *Qcm* étend *Question* ; en créant un *Qcm*, on crée donc automatiquement une *Question*.

La commande

```
1 /project/server\$ groovy console
```

permet aussi de pouvoir lancer des scripts sur le serveur alors que le serveur est en cours d'exécution.

C'est pour cela que nous avons ajouté

```
1 runtime 'org.grails.plugins:grails-console:2.0.8'
```

à la liste de nos dépendances.

4.2 Modifier des données depuis POSTMAN

Pour pouvoir utiliser notre API sans interface graphique, nous avons choisi d'utiliser POSTMAN. Cette application permet d'utiliser *curl* avec une interface graphique. Nous aurions également

pu le faire depuis Eclipse ou IntelliJ.

Pour installer POSTMAN, il faut se rendre sur leur [site officiel](#) et récupérer la dernière version. Lors de l'année 2016-2017, nous avons testé les version 4.8.2 à 4.10.2.

Pour utiliser POSTMAN pour notre projet, la première chose à faire est de créer un environnement dédié. Comme montré sur la [Figure 4](#), il faut également créer des variables dans cet environnement.

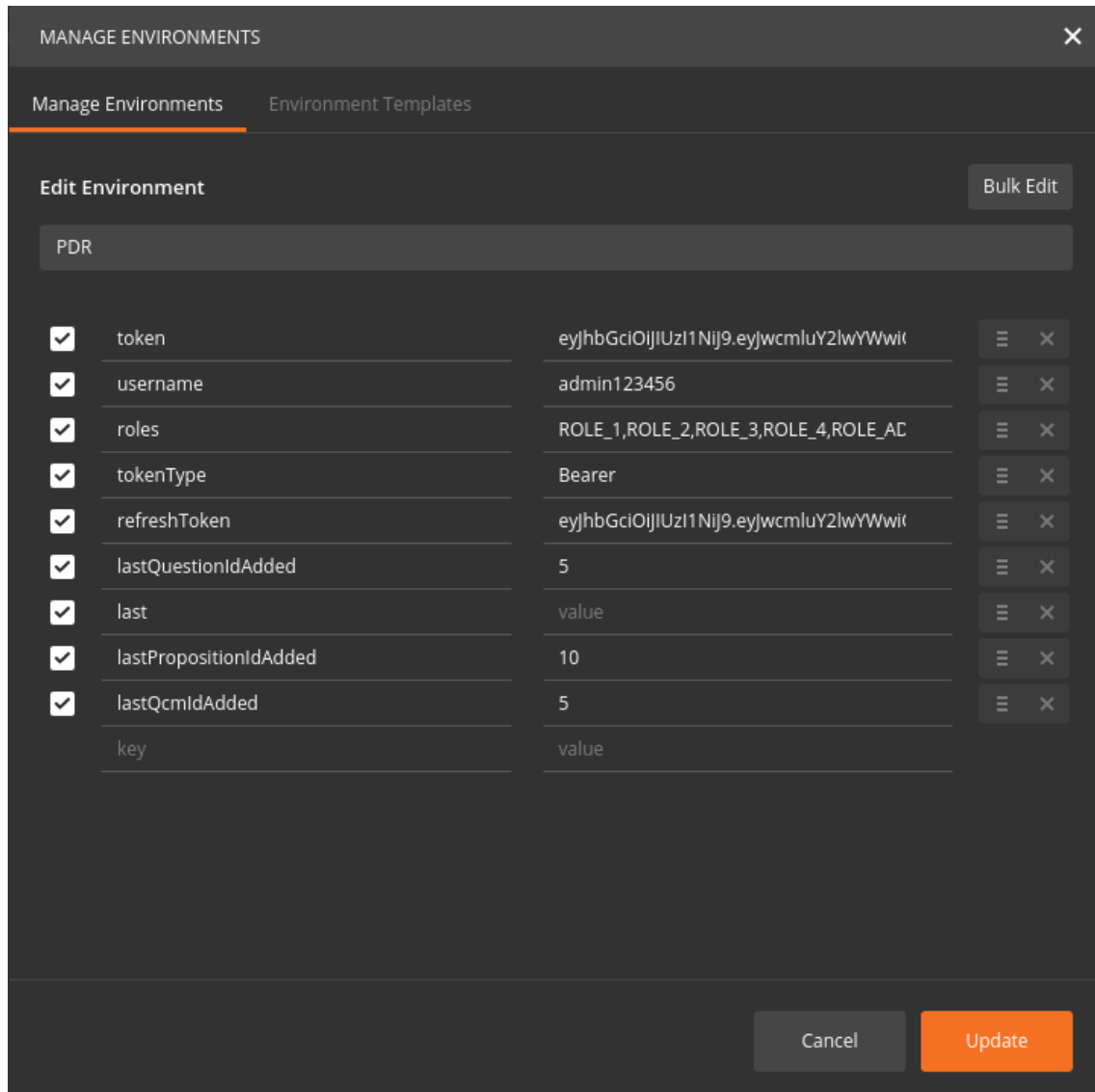


Figure 4 – Manager d'environnements sous POSTMAN

Une fois l'environnement créé (ici *PRD*), il faut ensuite créer des requêtes sur l'API.

Le [Section 3](#) (Chapitre 7) possède une archive de tests montrant comment POSTMAN traduit nos requêtes en cUrl.

```

1 curl --request POST
2   --url http://localhost:8080/api/login
3   --header 'accept: application/json'
4   --header 'content-type: application/json'
5   --data '{
6     "username": "admin123456",
7     "password": "123456789"
8   }'
```

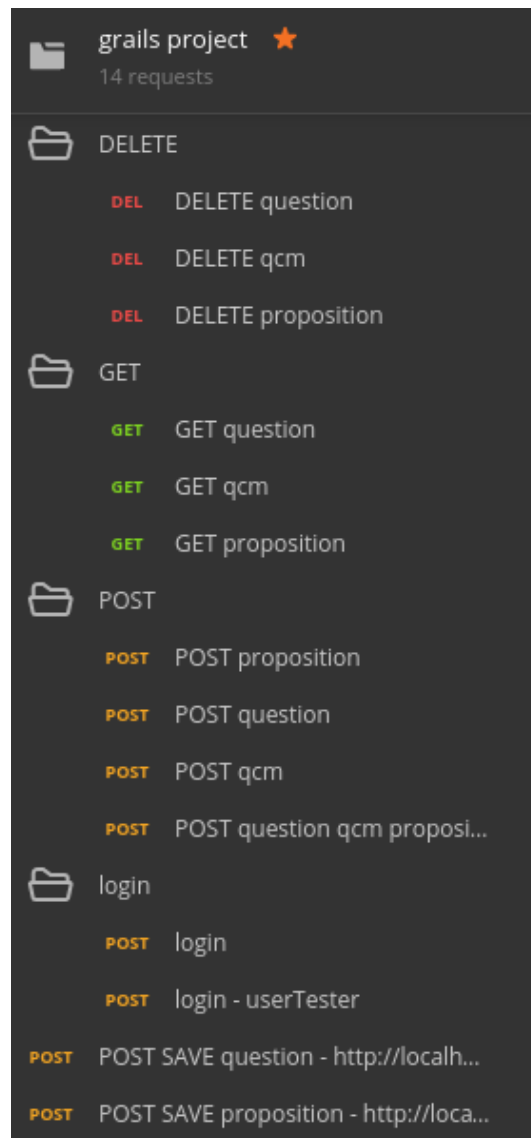


Figure 5 – Collection de requêtes sous POSTMAN

Il ne faut pas oublier qu’une fois notre application web sécurisée, il faut commencer par se connecter si l’on souhaite modifier certains contenus sécurisés.

Pour récupérer de façon automatique la réponse du login et ré-injecter ces attributs dans les autres requêtes, nous utilisons le script suivant :

```
1 var jsonData = JSON.parse(responseBody);
2 postman.setEnvironmentVariable("token", jsonData.access_token);
3 postman.setEnvironmentVariable("username", jsonData.username);
4 postman.setEnvironmentVariable("roles", jsonData.roles);
5 postman.setEnvironmentVariable("tokenType", jsonData.token_type);
6 postman.setEnvironmentVariable("refreshToken", jsonData.refresh_token);
```

Le script va donc se charger de parser le json reçu, d’en extraire les attributs et de les stocker dans les variables d’environnement préalablement définies. Pour par exemple utiliser le token reçu, il suffira d’écrire `{{token}}`.

5 Tester l'application

Dans la [Section 4.2](#), nous avons pu montrer un grand nombre de mécanismes permettant de tester notre API. L'annexe du [Section 3](#) (Chapitre 7) permet elle aussi de compléter les tests effectués sur postman.

Avec POSTMAN, nous avons effectué des tests de connexion, essayer d'effectuer des modifications sur des objets de l'API alors que nous n'avions pas les droits sur la méthode, envoyer des requêtes erronées ou encore tester l'envoi d'un ancien token pour vérifier que l'accès est toujours bloqué si la période de validité du token est dépassée.

Lorsque nous avons généré nos contrôleurs dans la [Section 3.6](#), nous avons également généré des tests unitaires et des tests d'intégrations dans les répertoires fils de `/server/src/`.

Les tests sont faits sur les domaines, les contrôleurs mais aussi sur les services. Les tests ne sont malheureusement pas encore complets. Et ne couvrent pas encore assez les classes pour garantir l'intégrité du projet.

Pour lancer l'API en mode test, il faut simplement faire la commande suivante :

```
1 /project/server/$ grails test-app
```

Attention à effectuer une copie des tests effectués si vous souhaitez refaire un *generate-all* car votre ancienne version sera écrasée. La [documentation de test](#) de Grails 3 est très complète et permet de comprendre comment effectuer des tests complets sur l'API. Cependant, ces tests ne testent pas les connexions, il est donc conseillé de compléter ces tests avec POSTMAN ou avec le client Angular2.

6 Modifier le client Angular2

Angular2 est un framework JavaScript basé sur le TypeScript. Pour un aperçu des grandes différences de performances et de styles de programmations entre plusieurs langages de frontend, deux articles du même auteur peuvent être lus [[WWW8](#)] et [[WWW9](#)].

Le [site officiel](#) d'Angular2 a publié *the Tour of Heroes tutorial* qui est un très bon guide pour prendre en main Angular2. J'ai suivi ce tutoriel pour comprendre comment fonctionne Angular2.

6.1 Architecture du client Angular2

Après avoir suivi plusieurs tutoriels et fait des essais avec plusieurs architectures, avec les conseils d'**Antoine Giret**, je suis arrivé à adopter l'architecture suivante ([Figure 6](#) pour structurer au mieux mon serveur Angular2).

6.2 Ajout de modules

Pour ajouter un module Angular2, il faut écrire la commande suivante :

```
1 /project/client/$ ng generate module monModule
2 /project/client/$ ng g module monModule
```

pour un sous-module :

```
1 /project/client/$ ng g module monModule/monSousModule
```

Ces commandes vont donc créer des modules Angular2 (@NgModule)

Il faut ensuite ajouter les routes. De base, les URLs sont redirigées vers index (qui est le module créé par défaut).

Pour ajouter la route d'un module, il faut entrer le ligne de commande suivante à la racine du serveur (client) :

```
1 /project/client/$ ng g module monModule --routing
```

Cela va créer les fichiers *monModule.module.ts* et *monModule-routing.module.ts*

ajouter l'import dans le *app.routes.ts* (ou *xxx-routing.ts*)

```
1 import { MyModuleComponent } from './monmodule/monmodule.component';
```

ajouter le path dans lequel on se trouve, la redirection. Cette redirection utilise le composant importé :

```
1 {path: 'monPath', redirectTo: 'monModule', pathMatch: 'full'},
2 {path: 'monModule', component: MyModuleComponent},
```

Une fois la route ajoutée, il faut importer le module ou composant au module père dans *app.module.ts* si il n'y a pas de module père. Il faut donc importer le module :

```
1 import { MyModuleComponent } from './monmodule/monmodule.component';
```

Puis l'ajouter aux modules Angular2 (@NgModule) dans la liste de déclaration des modules :

```
1 declaration: [
2     MyModuleComponent,
3     monModule2Component,
4     monModule3Component,
5     IndexComponent
6 ],
```

Après ça, si le serveur a été lancé avec *ng start*, la recompilation automatique se fait et l'on peut voir que si nous tapons *localhost :4200/monPath* alors nous sommes redirigé vers *localhost :4200/#/monmodule*

L'interface graphique a cependant été assez vite abandonnée pour pouvoir nous consacrer à l'API.

```

/home/project/client$
├─ angular-cli.json
├─ angular-cli.json
├─ build.gradle ..... POUR GRADLE SEULEMENT
├─ angular-cli.json
├─ build.gradle ..... POUR GRADLE SEULEMENT
├─ e2e ..... POUR LES TESTS END TO END
├─ ...
├─ package.json ..... DÉPENDANCES ANGULAR ET BOOTSTRAP ...
├─ protractor.conf.js
├─ proxy.conf.json ..... CONFIGURATION PROXY POUR LES REDIRECTIONS
├─ src
│   └─ app ..... LA 'RACINE' DE L'APPLICATION
│       ├── app.component.html
│       ├── app.component.ts ..... FAIT APPEL AU HTML/CSS
│       ├── app.module.ts .... IMPORTE LES MODULES ANGULAR (NgModule) UTILISÉS POUR LA
│       │   ROUTE ET LE COMPONENT
│       ├── app.routes.ts ..... DÉFINI LES ROUTES POSSIBLES DEPUIS LE MODULE
│       ├── module1
│       │   ├── components
│       │   │   └─ component1
│       │   │       ├── module1.component.css
│       │   │       ├── module1.component.html
│       │   │       └─ module1.component.ts
│       │   ├── services
│       │   │   └─ module1.services.ts
│       │   └─ models
│       │       └─ module1.model.ts
│       ├── user
│       │   ├── components
│       │   │   ├── login
│       │   │   ├── login.component.css
│       │   │   ├── login.component.html
│       │   │   └─ login.component.ts
│       │   ├── services
│       │   │   └─ login.services.ts
│       │   └─ models
│       │       └─ user.model.ts
│       ├── assets ..... LES ÉLÉMENTS EXTERNES
│       │   ├── images ..... LES IMAGES DU PROJET
│       │   │   ├── skin
│       │   │   │   └─ favicon.ico
│       │   │   └─ ...
│       │   └─ ...
│       ├── js ..... LES SCRIPTS DU PROJET
│       │   └─ ...
│       ├── index.html APPEL À LA VUE DE L'APPLICATION ET CHARGE LE HEADER AVEC LES SCRIPTS
│       ├── main.ts ..... PERMET DE DÉMARRER LE PROJET
│       ├── styles.css
│       ├── tsconfig.json ..... OPTIONS DE COMPILATION
│       └─ typings.d.ts

```

Figure 6 – Répertoire du client Angular2

6

Bilan

Lors des trois premiers mois de Projet de Recherche et Développement, j'ai pu découvrir de nombreuses technologies qui m'étaient jusqu'alors inconnues. J'ai pu découvrir un nouveau système d'exploitation, de nouveaux langages de programmation et de nouvelles techniques de développement.

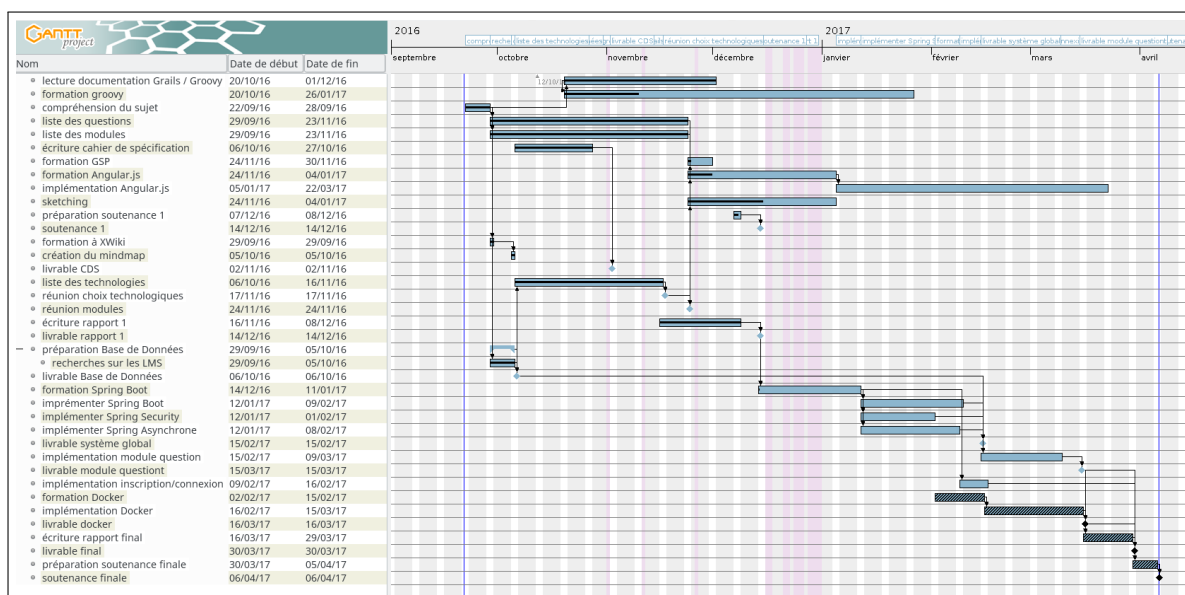


Figure 1 – Diagramme de Gantt passé et prévisionnel

Le diagramme de Gantt en Figure 1 montre le planning des premiers mois ainsi que les prévisions des derniers.

Le diagramme Figure 2 montre quant à lui la réalité du déroulement du projet sur toute l'année. Nous pouvons donc voir que certaines tâches ont été avortées voir annulées et que d'autres se sont rajoutées ou allongées.

Au premier semestre, le plus grand retard avait été apporté par l'utilisation de GSP qui s'était révélé complexe à prendre en main (en particulier à cause de sa documentation). Au second semestre, nous avons fait le choix de changer notre front end par Angular2 qui est plus documenté, possède une grande communauté et mieux supporté sur les terminaux mobiles.

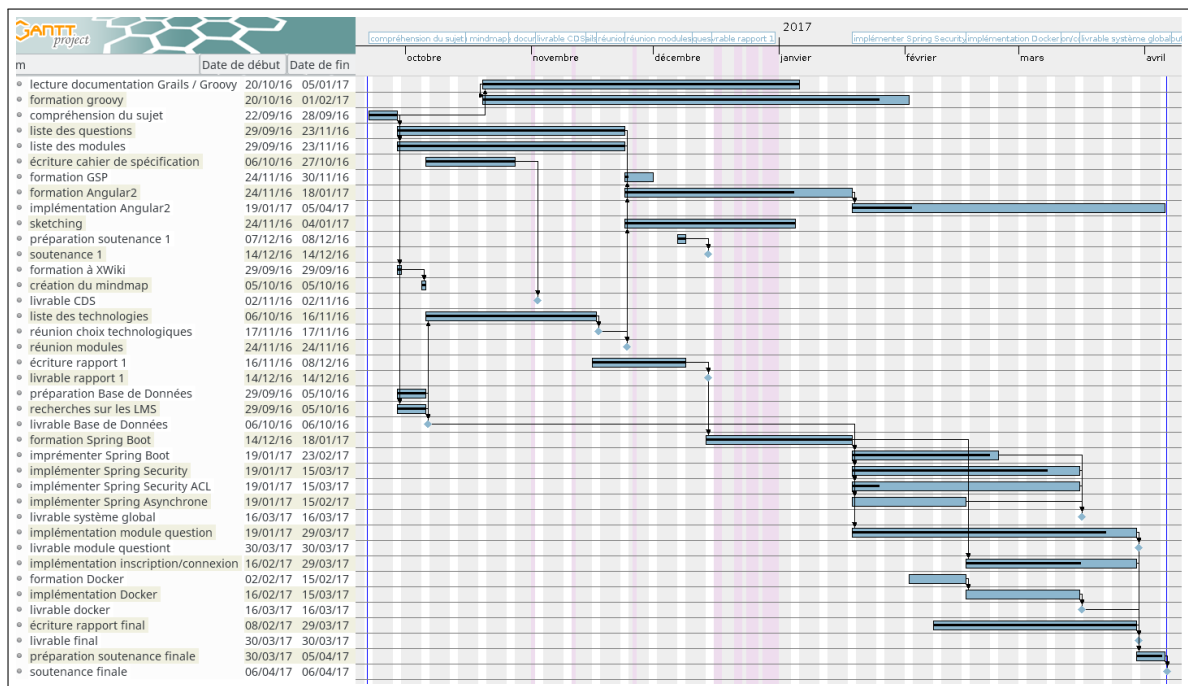


Figure 2 – Diagramme de Gantt effectif

Le problème d'avoir choisi Angular2 avec Grails 3 est que la documentation sur ce profile particulier est quasiment inexistante. J'ai donc commencé par suivre les tutoriels Angular2 pour ensuite les adapter au profile. J'ai perdu trop de temps sur le front end alors que j'aurais dû plus rapidement comprendre que la priorité n'était pas le front end mais le back end. Début février j'ai donc laissé Angular2 pour me consacrer exclusivement au back end. Je comptais encore revenir au front end plus tard même si je n'ai au final pas eu le temps de le faire.

Je me suis aussi dispersé sur plusieurs tâches en même temps alors qu'il aurait fallu me focaliser sur une seule tâche à la fois. J'ai voulu tout faire en même temps et cela m'a fait perdre du temps. Les deux derniers mois ont été les plus productifs. J'avais pu lire de très nombreuses documentations sur Grails 3 et comprenait enfin les rouages permettant de pouvoir développer plus sereinement ma solution.

Les deux tâches étant les plus importantes concernent les objets avec GORM et la sécurité avec Spring Security et Spring Security ACL. J'ai rencontré des difficultés que nous n'avons pas réussi à résoudre sur l'affichage des objets qui sont étendus. Il semble que ce problème soit lié à un bug du plugin *json-view*.

La partie Docker n'a pas été traitée car je n'avais pas le temps de la faire et surtout, nous avons convenu qu'il serait plus judicieux d'utiliser des workers Java Platform pour réaliser ce traitement.

La plus grosse difficulté a été de devoir passer des jours à lire et comprendre des documentations. C'était la première fois que je devais consulter autant de documents et les utiliser tous ensemble pour mon projet. Cependant, ces lectures ne sont une perte de temps car j'ai pu comprendre et apprendre de nombreuses choses sur Grails 3, java, Angular2 et certains protocoles.

Ce projet est très enrichissant mais j'aurais dû mieux définir le cadre pour ne pas m'éparpiller. Avoir essayé de faire le front end et le back end avec des technologies que je ne connaissais pas du tout n'était tout simplement pas possible. Le projet bien que non opérationnel pour l'année 2017-2018 possède tout de même de bonnes bases pour être continué. J'ai pu constituer une documentation / tutoriel permettant de décrire le langage et les attentes du projet. Les informations recueillies vont permettre au prochain étudiant reprenant le projet de commencer

plus rapidement son travail.

J'aurais tout de même aimé avoir un projet test fonctionnel, c'est une déception sur ce point mais les bases sont posées et j'ai beaucoup appris de ce projet.

7

Annexes

1 Rapport DI3 - Ludification

ÉCOLE POLYTECHNIQUE DE L'UNIVERSITÉ FRANÇOIS RABELAIS DE TOURS
Spécialité Informatique
64 avenue Jean Portalis
37200 TOURS, FRANCE
Tél +33 (0)2 47 36 14 14
www.polytech.univ-tours.fr

Report of the 3rd year internship 2014-15

Gamification design pattern of Android/iOS applications

Institution:

Beijing Institute of Technology
No. 5, Zhongguancun Nandajie, Haidian District,
Beijing 100081, P. R. China

**Institution tutor:**

ZHAO Fengnian
Teacher

Student:

MENIN Valérian
Promotion 2017

Organization tutor:

BOUQUARD Jean-Louis

Outline

Introduction

- I. Presentation**
 - a) What is gamification?**
 - b) Why are we doing research about it?**
- II. Research, classification and applications**
 - a) Researches and classification**
 - b) Applications**
- III. Our application**

Conclusion

Thanks
Bibliography

Introduction

3 billion hours; this is the amount of time humanity is playing videogames every week.

Nowadays, everyone is using a computer, laptop, tablet, smartphone ... and the number of gamers is always increasing, and that's the reason why we are studying how games attract us so much and so easily.

99% of boys and 94% of girls under 18 report playing videogames regularly. When you think about this and that you see that it's harder and harder to catch students' attention and to keep them concentrated, then gamification becomes a goal. This goal is the answer to "how to learn something without feeling bored", "how to spend more times on studying" ...

After considering that facts and our needs, it is becoming urgent to explore these questions and to do some researches about one of the main educational problems of the last years all around the world. Gamification could be the answer.

First of all, we will set out what is gamification. On a second time, we will talk about the researches that have been done and the classifications that already exist. To finish, we will introduce the application that we have developed.

I. Presentation

a) What is gamification?

To begin by talking about gamification, let's first talk about "what is gamification". The term gamification has been created in 2002 but that doesn't mean that it hasn't been used before. A simple definition of gamification would be:

- Make nongame activities more attractive and fun, make them more like games -

The idea is really modest but it is now becoming the main purpose of commercials, education, work ... everyday's life.

As said before, the term "gamification" has been created recently but the idea exists for a long time. In 1973, Charles Coonradt formed The Game of Work. Charles Coonradt had for refrain "profitability through productivity" and this is exactly the main purpose of gamification. He is now called the "Grandfather of Gamification".

It is only since 2007 that we are really trying to do some research about it.

b) Why are we doing research about it?

The point is that games are really smart. We could easily pass the entire day playing if it's good enough. We could do this and don't even feel hungry, look at how much time we have been playing or even don't go to toilet because we are about to finish an exciting part of the game.

What we are now trying to recreate that envy to continue and to do something well in a community. Not as a simple game but as a real useful thing.

It is generally agreed that nowadays students aren't as concentrated as before and are often playing videogames. But ...

The average game player is **31** years old

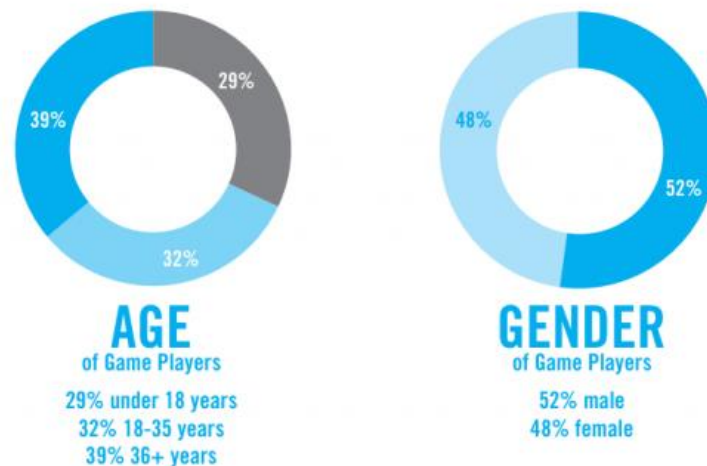


Figure 1 - Entertainment Software Association | game players age and gender

Yes, videogames are not only about teenagers and young people. Everyone is now playing videogames. From your 3 years old kid to your grandmother, everyone is now able to play and use electronic device. No matter we are male or female, gamification is for everyone.

However the word is new, it is easy to see that it exists for a long time and the best example are business and marketing. A good marketing makes you buy a product rather it is good or not.

Surrounding 2005, gamification became an important subject, especially for governments that saw a good way for their students to study more and better. Then came factory that saw a good way to increase their productivity (a nice return to Charles Coonradt and his book *The Game of Work* published in 1983). Business and marketing has always been using it without naming it.

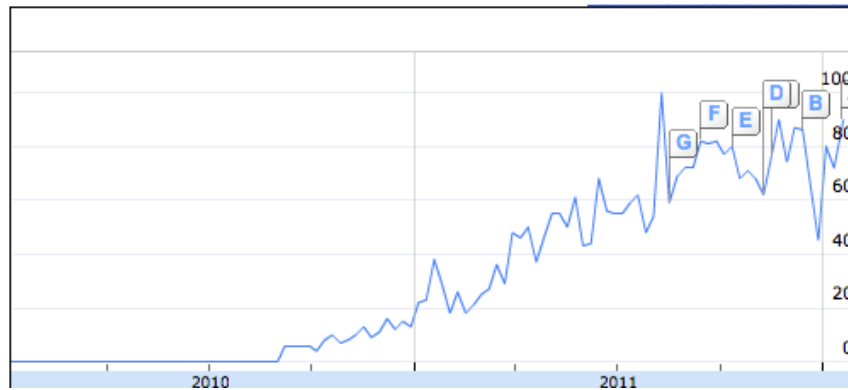


Figure 2 - search graph about gamification

If the number of research of increasing every years since the last 5 years, it is because all operators perceived their profits. At least, all studies are about the same topic which is gamification technics.

II. Research, classification and applications

a) Researches and classification

As aforementioned, many research has been done about gamification during the last years. Most of them has one main purpose. This ambition is to find the gamification technics; in other terms, it is to uncover how to make an attractive application that could be used by many people.

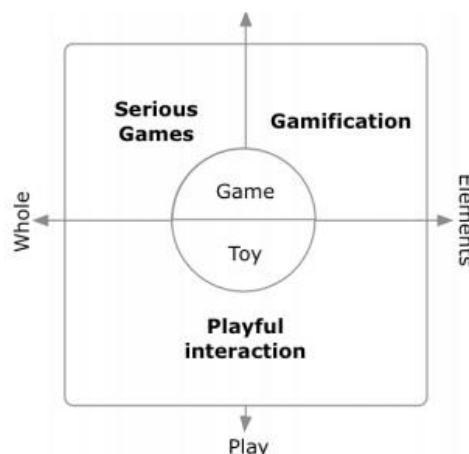


Figure 3 - Defining Gamification

The figure 3 is showing what kind of subject we should study to know more deeply what gamification is.

A Serious game is a game designed for a primary purpose other than pure entertainment (Wikipedia). It includes categories such as educational games and

advergames (advertising), political games, and training game (also known as game-learning).

The term “Persuasive game” is introduced in the title book “Persuasive Games, The Expressive Power of Video games” by Ian Bogost. Persuasive games mostly use psychology by employing design and social influence to change attitudes or behaviours of the user.

Gabe Zichermann shown on his book *Game Based Marketing* that “Games are the only force in the known universe that can get people to take actions against their self-interest, in a predictable way, without using force” and one of his favourite example is the FunFair. People are looking for this terrible but so good experience. He also added “Game mechanics and the psychological conditions exploit are powerful tools that marketers can use, and they are a lot cheaper ... than cash in the long run”. This last quote makes everyone understand easily why more than half studies about gamification are made by some provide business.

This last point has been the main purpose of why gamification has to follow some rules such as data privacy. Because gamification is not that far from game addiction and when you are addict to a game, you won't pay attention on your money, private information... you become really easy to manipulate and this is actually what commercial are trying to do to make you buy their own product.

Researchers had to make an ethic choice.

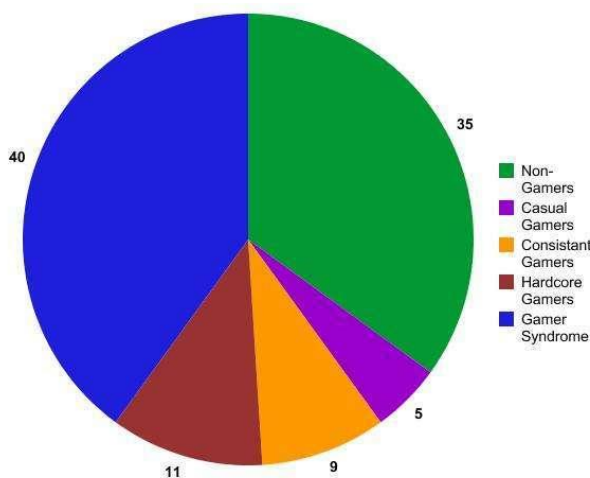


Figure 4 - type of players

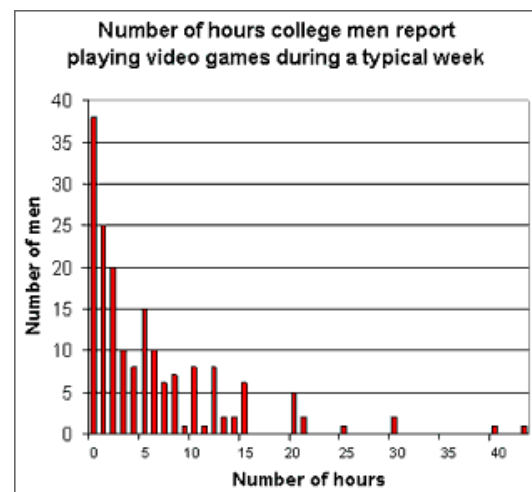


Figure 5 - number of hours men per week

In fact, in US, 5 million gamers are spending more than 40 hours a week playing games, the equivalent of a full time job.

So as Jane McGonigal is asking herself on her book “how do we know when we’re playing a good game and when would we’d better off doing something ‘real’”?

MMORPG (Massively Multiple Online Role-Playing Games) are actually the most played and the most addicted. Some studies shown that nearly 40% of WOW (World Of Warcraft) players are addicted to that game.

In France, WOW and Dofus are the most played. Both are using nearly the same rules and technics. Researchers thinks that MMORPG are the best gamified games. But here, we are back the addiction problem.

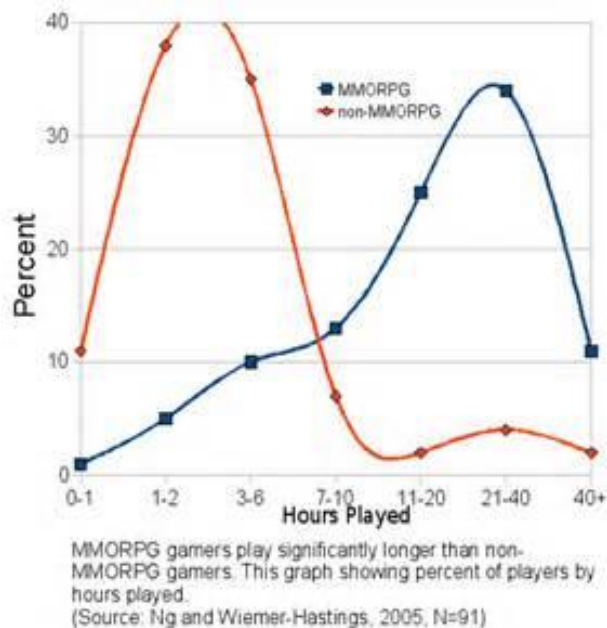


Figure 6 - number of hours played MMORPG / non-MMORPG

Investigations indicate different ways and technics of how it works. The main thing about MMORPG is to keep a clear goal and tasks, reinforcing feedback and increasing challenge. MMORPG are using both intrinsic and extrinsic motivation (<https://vimeo.com/88939322>).

It is important that there is no final victory or loss conditions in MMORPG, therefore, the game continues to evolve. As the game is always evolving, the player won't get tired of playing it.

MMORPG are also using emotional proximity to their character (e.g. Dofus in which you can only have a limited number of characters). The character becomes a type of avatar of the player in the game environment.

Another important point is that role playing fosters opportunities for collaboration across time and space, both within and across classrooms.

Quest: bounty quest, Fed Ex quest, collection quest, escort quest, goodwill quest and messenger quest. Quest is always a good way to make someone do something. It is a kind of manipulation. When you have to do something, you could be really motivated if you feel that you do it for free but you know that you will have a good thing and someone gonna be grateful in return. It becomes some kind of happy job.

The levelin up is also crucial for the player. It is a goal, a way to compare with others, to see his advancement ... Choice, control, personal design, collaboration, challenge and achievement are keys.

Also, both challenge and uncertainty are key elements into MMORPG gameplay experience.

Intrinsic motivation	MMORPG design
Choice	Character design Traits, skills, attributes, and adornment Narrative environment Choice of small quests (with limits place on the amount of small quests a player can adopt) Option to drop or delete small quest selected
Control	Narrative environment Quest selection Order of completion Strategies employed
Collaboration	Character design Social capital associated with player's character Narrative environment Chat and communication tools Collaborative quests
Challenge	Character design and narrative environment Quests equivalent to current level of skills combined with attributes
Achievement	Character design and narrative environment Marked progress indications Elevated status Advanced skills Enhanced attributes Bounty

Figure 7 - Summary of intrinsic motivation and MMORPG design

Often, the player has to get some new small quest and knowledge domains: declarative, procedural, strategy, metacognitive knowledge...

Knowledge domains	Small quests
Declarative knowledge	<ul style="list-style-type: none"> • Collection quests • Goodwill quests
Procedural knowledge	<ul style="list-style-type: none"> • Fed Ex quests • Messenger quests
Strategic knowledge	<ul style="list-style-type: none"> • Bounty quests • Escort quests
Metacognitive knowledge	<ul style="list-style-type: none"> • Bounty quests • Escort quests • Goodwill quests

Figure 8 - Categorizing small quest-types by knowledge domains

As told, MMORPG are a real source of inspiration for gamified applications but it is far from being the only one. The point is that, yes, it is probably one of the best but it is really difficult to make a simple app look like an MMORPG. It is too much long time work and not an interesting investment. Nobody wants to play a game and kill some creature to access to his mail or planning.

Even if gamification is about game, it is also about funny ways of doing something. Our purpose is first and foremost to teach something to students. Some frameworks for Game-Based Learning exists such as UniGame probably the most used because it is to help a teacher applying GBL in his/her classes, FDF for a 4 dimensional framework mostly use in science. Applied Behavioral Analysis is a framework/method of teaching to design a good game it is full of tips about gamification for when you are creating your application.

Game elements	
Game mechanics	Game dynamics
Points	Reward
Levels	Status
Trophies, badges, achievements	Achievement
Virtual goods	Self expression
Leaderboards	Competition
Virtual gifts	Altruism

Figure 9 - Game mechanics and game dynamics (adapted from Bunchball (2010))

We have seen that gamification is a big undertaking with many different possibilities of doing the same thing buy using different ways. But more than theory, we are now going to study some specific parts of several applications.

b) Applications

All bid brands are now using gamification for the purpose of marketing. One of the first gamification provider was Bunchball in 2005. Bunchball belongs to Alphabet and is a gamification platform. It has been the most used for some years, until gamification has been really famous and used by everyone.

One of the first important gamification project has been introduces by Starbuck. The company created a member card in which you gain point if you often go to the same shop but also if you often go to many different Starbuck. It has levels, feedback, community, badges ... Starbuck is also providing free games. After the beginning of that action, Starbuck saw its popularity increase and a lot of media has been created.



Figure 10 - Starbucks

In a different way, Milka used gamification.

Milka remade all its factory's mould to make a "missing" piece. Actually, the weight was the same. Everyone who send some information to Milka would receive for free the missing piece. It costed a lot to Milka but this was only an investment and as an investment, the brand get much more money using it. Milka get thousands of contacts and personal information from all that people who send them a message.



Figure 11 - Milka

Many restaurants are now using gamification in different ways to win the loyalty of their customers and get some new customers.

Social networks also use it a lot such as Weibo, WeChat, Line, Facebook, Twitter or Yahoo who all have their own app store, games, friends, points ...

For Dropbox and later Google Drive, the mechanic has been to use your contacts and share online files with co-workers, friends...

LinkedIn, the professional social media also joined this movement. LinkedIn is using game's knowledge. We can "improve" our profile, ask things to our community, see our friends, who is connected and more important: we have a progress bar that aims to motivate us to complete it.



Figure 12 - LinkedIn



Figure 14 - Scratch cat

This app is not a game but a funnier way to learn. Gamification on education is mostly about make something becoming fun. In here, programming has become fun to learn to do.

The Scratch project began in 2002 to be open to everyone on 2010.

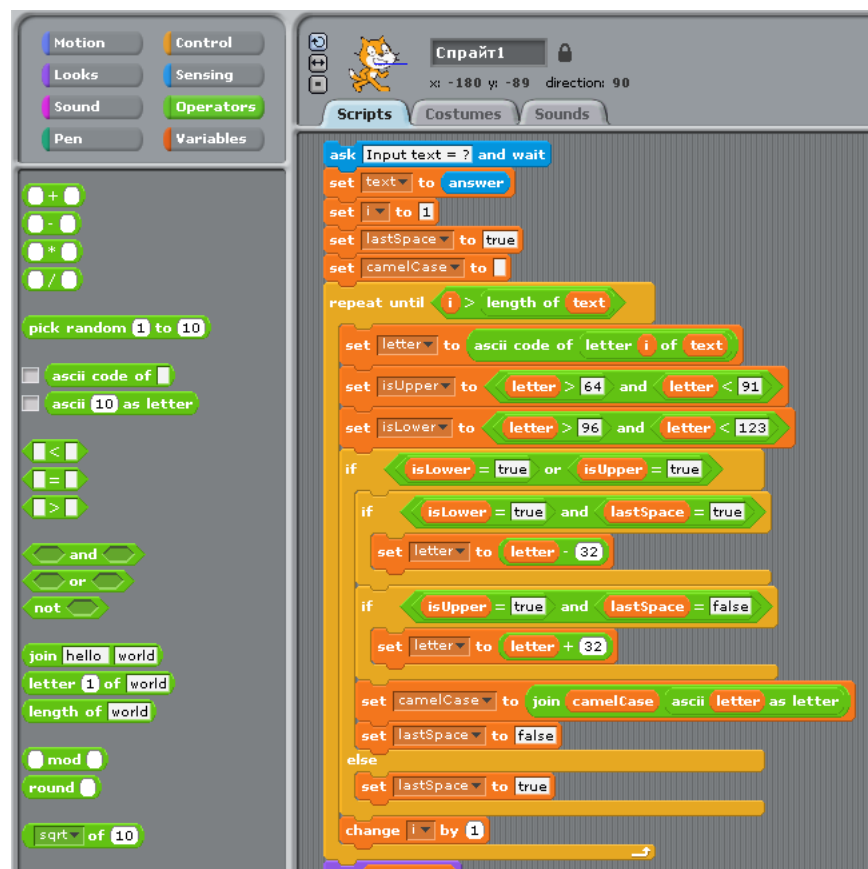


Figure 13 - Scratch from the MIT

For IT, we could also take a look at Stack Overflow. A gamified website which has been created as a forum. You also have badges and levels.

At Polytech Tours, we are now using Celene. On this platform, it is possible to create some quiz and to animate forums. We have been using it this year with some teachers. It is a good way to practice, to try again and again until we get the best score. And when we don't know something, other students are here to help us by giving us tips. This is creating link between students that might have not talk if they were in the same room.

That kind of method is now using in some business.

III. Our application

During this internship, I had, at first, to learn what is gamification and how works. Then, I had to program an app. Unfortunately, the amount of documents of the first part was really more important than expected and I didn't have enough time to program any working application.

On that part, I will introduce what I am now programing even after the training period.

With my teacher at Beijing Institute of Technology, we decide to program a gamified planning.

Some application of that kind already exist but most of them are really complex so we decided to create a simpler one. The one that already exist and that we have been working on are Get It Done, Mini Project, Goal Tracker and Jiffy.

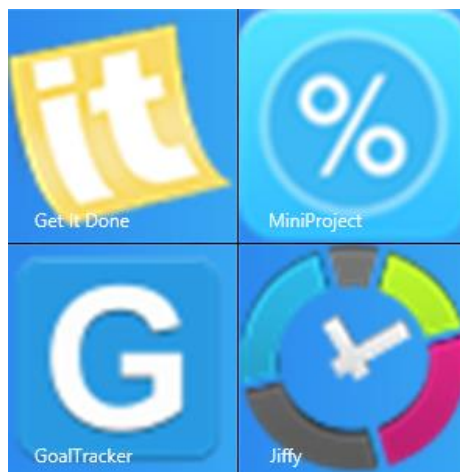


Figure 15 - bright idea

One difficult point is that the two best one only exist in Chinese. It has been difficult to understand and to use it.

I decided to call my application “Happy Schedule”. Because it is exactly what we are trying to do: make our schedule a bit happy.



Figure 16 – Our application

Our application will have different functions and menus.

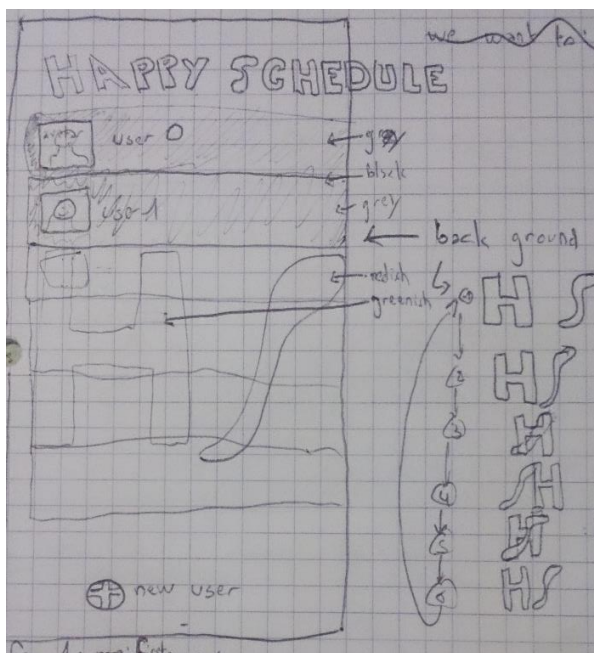


Figure 16 - first menu

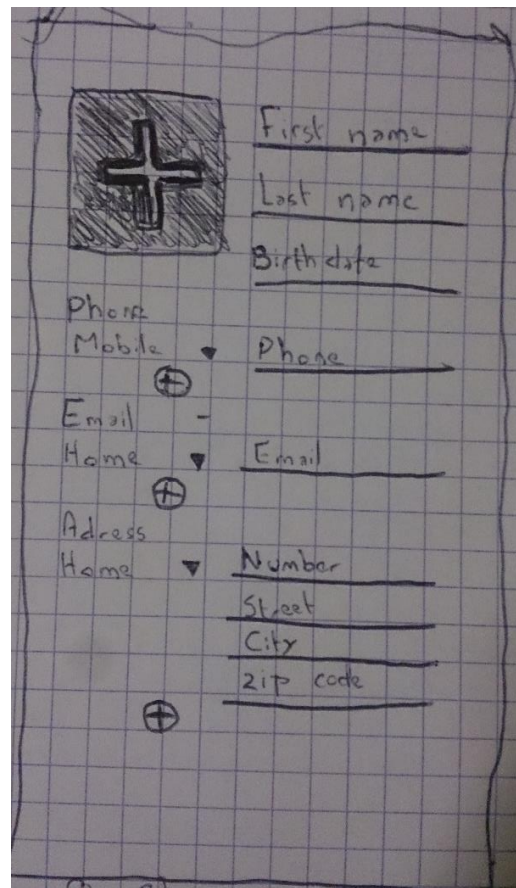


Figure 17 - new user

Here is some functions that our application has:

- Import / export a calendar and schedule (.csv file)
- User: create / delete / manage
- Task: create / delete / manage step by step / share / copy
- Steps: create / delete / manage / copy
- Order tasks by
 - Date of creation
 - Date of last reached
 - Date of issue
 - Most advanced (%)
 - Importance
 - All this order could be shown from first to last or from last to first
- Display tasks
 - Current task
 - Not began
 - Finished

That's all for the simple functions.

We now have what a basic scheduling application is supposed to do. We are now going to introduce the functions and facts that makes the application a gamified one:

- Left / right handed
This is really simple. Only some button on left or right but that makes the user happy of choosing and also makes him relaxed (the setting button could go from up-left to up-right ...).
- Change date format
After some months in Canada and China, I can tell it's really important to change it if you want your application to be international. By using different format you can select the more convenient for you. You won't have to make a conversion.
- Change language
It is basic but as I saw, most Chinese app are not translated into English. What a pity because some of them are really better than our English one. It is better for all users to have a list of different languages such as English, Chinese, Spanish, and French. After all, it is only some few words that we have to translate.
- Animated background
The background is made of our logo (two letters) turning around itself. The user is not bored anymore when watching his screen. It makes the app dynamic. A dynamic app becomes an app that we like to use.
- All tasks has steps
The user can add as many steps as he wants if he thinks it is an important part in the task.

- Users account
We can manage several users and customize their profile by choosing the main colour theme, its avatar, and its profile name. In the same way as a game, we try to make everything unique for each user to have his own experience.
- Share
A user can choose to make his profile public / private or customs. He can choose to share with only one group of friends some tasks and some others with different persons. This is how we create a community.
Away from games, all users can see / share their news such as levelling up, achievements, friends ...
- Points
Every time a user finish a task, he get some experience and points. With some point he can buy some new custom design. He could also send some to his friends. His experience helps to levelling up and to reach new levels and unlock achievements.
- Statistics
All users can see their statistics of the day / week / month and compare it with themselves or with friends. It is a good way to keep the link between people.
- Help
Users has to help others and will earn points if they provide a helpful answer but also the helped player. That make user not shy to ask even if they think it might be a dumb question it's better for them to ask.
For everything the user can do, we should make a short animation showing how it works with a simple example.

Even though our app is still not working, we had many idea to make “simple planning” becoming a gamified application and to make it dynamic, colourful and full of joy.

Conclusion

Gamification is now everywhere. Sometimes we see it, sometimes we can't but it is. Some people call it manipulation and want to prohibited it, on the other hand, some people are trying their best to use it. I thing gamification is like everything in science, not good or bad but only the way it is. It is only how we use it that could create an ethic problem.

During my internship, I have learnt how to search and analyse many researches. It has been really interesting to figure out that many people have been working on something without knowing it. But also stimulating of the fact that they worked on it. It is not easy to read a lot and summarize thousands of pages to create an application.

Gamification design pattern in a real important key of this century to motivate people and increase productivity but also recreate a link between people. We often say that computing networks are destroying communities but I think that with some good application, we could change and create that link.

It is also essential to answer to the educational problem. It is probably a good answer to the gaming phenomena.

List of references

<http://arstechnica.com/>
<http://www.socialfresh.com/>
<http://vator.tv/>
<https://en.wikipedia.org>
<http://www.ted.com/>
<http://img.seriousgamesinternational.com/>
<http://www.seriousgames.net/>
<https://badgeville.com>
<http://www.gameofwork.com/>
<http://gamersyndrome.com/>
<http://www.ucsf.edu/>
<http://pedropineda1.blogspot.com/>
<http://www.pcworld.com/>
<https://class.coursera.org>

- Sharpe, R., Beetham, H. & de Freitas, S. (Eds) (2010) Rethinking Learning in the Digital Age, London & New York: Routledge.
- Our Health: Using digital technologies for therapy and awareness raising around health issues
- Educational Technology Research & Development of Springer Science & Business Media B.V.



Fiche d'auto-évaluation des compétences – DI5

Nom et prénom de l'élève ingénieur : MENIN Valérian Spécialité : Informatique

Année universitaire : DI3 - 2014-15

TB : Très bien, expertise ; B : Bien, maîtrise ; M : Moyen, application ; I : Insuffisant, notion ; SO : sans objet

Compétences évaluées dans des situations représentatives du poste confié à l'élève ingénieur stagiaire Évaluez à l'aide d'une croix par ligne sur l'échelle proposée	TB	B	M	I		SO
1- Maîtrise des domaines scientifiques et techniques						
• Capacité d'analyse / compréhension des problèmes	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		<input type="checkbox"/>
• Mise en œuvre de ses connaissances	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		<input type="checkbox"/>
• Aptitude à acquérir de nouvelles connaissances	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		<input type="checkbox"/>
2- Maîtrise des méthodes et des outils de l'ingénieur						
• Methodologie / organisation du travail, gestion de projet	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>		<input type="checkbox"/>
• Synthèse et communication des résultats, maîtrise des outils de communication	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>		<input type="checkbox"/>
3- Conduite de l'action et prise de décision						
• Réalisation des objectifs - Qualité du travail réalisé	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		<input type="checkbox"/>
• Autonomie - initiative / créativité / ouverture d'esprit	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		<input type="checkbox"/>
4- Intégration dans une organisation et capacité d'animation						
• Capacité à s'intégrer dans une équipe	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		<input checked="" type="checkbox"/>
• Communication sur ses activités et capacité à rendre compte	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		<input type="checkbox"/>
• Prise en compte des enjeux métiers et économiques - Respect des procédures (qualité, sécurité, santé, sécurité...)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		<input checked="" type="checkbox"/>
5- Respect des valeurs sociétales, sociales et environnementales						
• Appropriation des valeurs, codes, et de la culture de l'équipe et de l'organisation	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		<input type="checkbox"/>
• Attitude / assiduité / ponctualité	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		<input type="checkbox"/>
6- Suivi du stage						
• suivi tuteur entreprise	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		<input type="checkbox"/>
• suivi tuteur enseignant	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		<input type="checkbox"/>
7- Compétence spécifiques à la spécialité Informatique						
• La connaissance et la compréhension d'un large champ de sciences fondamentales et la capacité d'analyse et de synthèse qui leur est associée.	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		<input type="checkbox"/>
• L'aptitude à mobiliser les ressources d'un champ scientifique et technique liées à la spécialité Informatique	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		<input type="checkbox"/>
• La maîtrise des méthodes et des outils de l'ingénieur: identification, modélisation et résolution de problèmes même non familiers et non complètement définis, l'utilisation des outils informatiques, l'analyse et la conception de systèmes.	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		<input type="checkbox"/>
• La maîtrise de l'expérimentation, dans un contexte de recherche et à des fins d'innovation et la capacité d'en utiliser les outils: notamment la collecte et l'interprétation de données, la propriété intellectuelle.	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>		<input type="checkbox"/>
• L'esprit d'entreprise et l'aptitude à prendre en compte les enjeux économiques, le respect de la qualité, la compétitivité et productivité, les exigences commerciale, l'intelligence économique.	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		<input checked="" type="checkbox"/>
• La capacité à s'insérer dans la vie professionnelle, à s'intégrer dans une organisation, à l'animer et à la faire évoluer: exercice de la responsabilité, esprit d'équipe, engagement et leadership, management de projets, maîtrise d'ouvrage, communication avec des spécialistes comme avec des non-spécialistes, voire la gestion d'entreprise innovante.	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		<input checked="" type="checkbox"/>
• La capacité à se connaître, à s'autoévaluer, à gérer ses compétences, (notamment dans une perspective de formation tout au long de la vie), à opérer ses choix professionnels.	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>		<input type="checkbox"/>



8- Appréciation globale sur le stage

Interesting internship with a good subject and many things to do. Maybe a bit too much because it took nearly the all training time to read old researchs and I didn't have enough time to do a real application. I had to stop at the beginning of it.

Signature de l'étudiant



Date

23/08/2015

Document à retourner à : Département Informatique Polytech Tours, Service Scolarité, 64 avenue Jean Portalis, 37200 TOURS

Gamification design pattern of Android/iOS applications

Abstract:

The main purpose of this internship has been to make research about gamification. The point is that many people are now trying to make “the best application” and education app as well. We tried to find real numbers on what we are studying to know exactly what we are talking about. The most important part of that internship has been to find what others researchers already did around the world for the last 50 years. After doing that, we took an inventory of its researches and about the different ways of creating a good application by using gamification. After having that catalogue, we were able to create our own application which has been chosen to be a scheduling app. In order to reach, I had to learn to learn how to program on an Android device. Finally, I made a paper first draft of the app and began to code even though the application is not operational for now.

Keywords:

Gamification, android, learning, analysis, educational, social, guideline, classification

Institution:

Beijing Institute of Technology
No. 5, Zhongguancun Nandajie, Haidian District,
Beijing 100081, P. R. China



Institution tutor:

ZHAO Fengnian
Teacher

Student:

MENIN Valérian
Promotion 2017

Organization tutor:

BOUQUARD Jean-Louis



2 Matrice de comparaison de différents frameworks java

	Vaadin Framework	JSF	PrimeFaces	Wicket	AngularJS	GWT	
Developed by	Vaadin Ltd	Java Community Process	PrimeTek Informatics	Various individuals. Apache project.	Google Inc.	GWT Steering Committee	
Framework description	A server-side Java UI framework	Java specification for building component-based user interfaces for web applications	Component suite for Java Server Faces 2.0	Component-based web application framework for Java	A structural framework for client-side web apps	Set of tools that allows web developers to create and maintain complex JavaScript front-end applications in Java	
Initial release		2001	2004	2009	2004	2009	2006
License	Apache 2.0	Implementation dependent	Apache 2.0	Apache 2.0	MIT	Apache 2.0	
Free for opensource use	x	Implementation dependent	x	x	x	x	
Free for commercial use	x	Implementation dependent	x	x	x	x	
Website	[vaadin.com] (https://vaadin.com)	[jcp.org/...] (https://jcp.org/en/jsr/detail?id=372)	[primefaces.org] (http://primefaces.org)	[wicket.apache.org] (http://wicket.apache.org)	[angularjs.org] (https://angularjs.org)	[gwtproject.org] (http://www.gwtproject.org)	
Maintenance releases per year		Implementation dependent		7	9	22	3
Committers in last 6 months		Implementation dependent		7	9	25	17
Latest official release	8.0 (21.2.2017)	Implementation dependent	5.3 (18.3.2015)	7.2.0 (20.1.2016)	1.4.9 (21.1.2016)	2.7.0 (20.11.2014)	

Philosophy

Built for single-page web apps	x	-	-	x	x	x
Built for web pages	-	x	x	x	-	-
Server-driven architecture	x	x	x	x	-	-
Client-side architecture	-	-	-	-	x	x

Environment

Java server-side APIs	x	x	x	x	-	-
Java client-side APIs	x	-	-	-	-	x
Declarative APIs	x	x	x	-	x	x
Deployment environment	Java Servlet Container or Java Portlet	Java Servlet Container or Java Portlet	Java Servlet Container or Java Portlet	Only Java Servlet	Web server	Web server
Programming language(s)	Java or any other JVM language	JSF Markup and Java or any other JVM language	JSF Markup and Java or any other JVM language	HTML with special syntax and Java or any other JVM language	HTML with special syntax and JavaScript	Only Java with a subset of the java.lang and java.util packages classes

All Java 8 features supported	x	x	x	x	N/A	x
Scala wrappers available	x	-	-	-	-	x

FEATURES

Documentation	[vaadin.com/docs] (https://vaadin.com/docs)	[jcp.org...] (https://www.jcp.org/en/jsr/detail?id=314)	[primefaces.org...] (http://www.primefaces.org/documentation)	[wicket.apache.org...] (http://wicket.apache.org/learn/)	[docs.angularjs.org...] (https://docs.angularjs.org/guide)	[gwtproject.org...] (http://www.gwtproject.org/doc/latest/DevGuide.html)
---------------	---	---	---	--	--	--

Abstraction over complex web technologies	*****	***	***	**	*	***
---	-------	-----	-----	----	---	-----

Automated server-browser communication	x	x	x	x	-	-
--	---	---	---	---	---	---

Usable without browser plugins	x	x	x	x	x	x
--------------------------------	---	---	---	---	---	---

Built-in Push (Websocket) support	x	-	x	x	-	-
-----------------------------------	---	---	---	---	---	---

Built in WAI-ARIA support	x	-	x	-	-	x
---------------------------	---	---	---	---	---	---

State securely on the server	x	x	x	x	-	-
------------------------------	---	---	---	---	---	---

Browser history / "back-button" support	x	Implementation dependent	x	x	x	x
---	---	--------------------------	---	---	---	---

i18n support	x	x	x	x	x	x
--------------	---	---	---	---	---	---

Localization support	x	x	x	x	x	x
----------------------	---	---	---	---	---	---

Built-in RTL support	-	Implementation dependent	x	x	x	x
----------------------	---	--------------------------	---	---	---	---

UI code architecture using MVC or MVP	x	x	x	x	x	x
---------------------------------------	---	---	---	---	---	---

Testing

Automated acceptance testing tool	x	x	x	x	x	x
-----------------------------------	---	---	---	---	---	---

1st party pixel level UI testing tool	x	-	-	-	-	-
---------------------------------------	---	---	---	---	---	---

Load testing with standardized tools (JMeter, Gatling)	x	x	x	x	x	x
--	---	---	---	---	---	---

UI code testable with JUnit	x	x	x	x	-	x
-----------------------------	---	---	---	---	---	---

Look 'n' feel

3 Tests avec POSTMAN

Le premier document est le fichier json à importer pour avoir l'environnement "PRD" de créé dans Postman.

Le second document montre ce qui est fait dans les différents fichiers. Bien qu'on ne puisse pas y voir les tests écrits en javascripts on voit les traductions des fichiers postman en leur équivalent cUrl.

```

1 {
2   "id": "61069b2b-5180-a4b4-195c-5bf9ecbcf36f",
3   "name": "PDR",
4   "values": [
5     {
6       "enabled": true,
7       "key": "token",
8       "value": "eyJhbGciOiJIUzI1NiJ9.eyJwcm1uY2lwYWwiOiJlbnhJQUFBQUFBQUFBBS1ZUUFdcl1RRQmkraEpS
9       VORZUVdxVWdNWmFGc3lGR1R3cElwY1EwcWN1d29IME9MQkxyWTFcL1RhODUyNU83ZkpnaXkxCMETFSXF
10      JVGdMXC9TZndNSVBRREN3ZG1ibFBhZXVFNWFLbSs1OTd2SHpQdStIe jg3UmpKS28ycGVZTW1WRkxPNV
11      RicWxJVXQ1WHhJOGwxVU1yVmtRR1JDZU1wd214Q3dnYW4xd2U1Un1VcDRGR3Q1MT1mSWpMRFBOKzJld
12      nRfMVwvWEJoS1ZoT3hmS081S0hKSwpJUSTzUzIXZlNES1ZJS1BPZmM2a jJSMjBpSDFmeEZ5N2d0dUR
13      pRW9TN0tDRkRITOVmMkNnS1I5ZUN0Y1VNelZKb1NVYz14Z0pIR1RFc2Q0VGtKVVNwZE0c2RsWVUxWn
14      VFMTF6MEZ5RWxRSjNcLzFUUzFzYTZ1VGMyT1ZUd0NyMUdoVUdVZ3dPOWUyQ29sdEd4TmdSa1VEVZYS
15      zEwZVNnQ3VrdE5jdEFmTGJcL1wvZnZKbDFNMGPcRDE1ZVBVM0dYNTNIWTIrdnZoekwybDB6dGZvem9U
16      MWpGWJST0JtTVZQdVNHsX1cL1wva1VcL0hCN1wvdmI1TmNoc0dFXC8rZng0cj1Zdk9EVGRFR0dHSnR
17      aaV1FY2d1RmVBK0QrTHJWNHvVXhoYWJSCEdQTUJHY1UyQ3l4U1pNS1Jia01LbFwvZGJvZXN0e jdkZX
18      J5Z0NsTkt4TWg5WHBjQzBONTVPd3Z0b11jbE9vTk1aY3o5MXV1TjEyQ3Q5TTRFMjc2WGpiZG10RmJ5U
19      m93M08zT2w1cm11bzE3Vlo5QWkwbfWfNZHVkK3lXc1YzRVFVa jVhcVc2OXVneD1LaVVqTWpzbHVSTJ
20      LemozKysrbmR6XC9DZ1UrUXpPSG1NVUV0bVFoSTdseDJDUH16ZG5wY3Zlanc1R2szK21cLz14ZHFvOW
21      1FdndNQUBPT0iLCJzdWIiOiJhZG1pb jEYmzQ1NiIsInJvbnVzI jpbI1JPTeVfMSIsI1JPTeVfMiIsI
22      1JPTeVfMyIsI1JPTeVfNCIsI1JPTeVfQURNSU4iLCJST0xFOX0FOT05ZTU9VUyIsI1JPTeVfREVQTE9Z
23      RVi1cLJST0xFOX01PTk1UT1IiLCJST0xFOX09RVJBVE9SiwiUk9MRV9URVNURVIiXSwiZXhwI joxNDk
24      wODkzZmk3LzJpYXQ1O jE00TA0Dk20Td9.riLtzVvc5HAB6PJJIROBd0baPUm3QNOKJ9hyDX4aVP4",
25     "type": "text"
26   },
27   {
28     "enabled": true,
29     "key": "username",
30     "value": "admin123456",
31     "type": "text"
32   },
33   {
34     "enabled": true,
35     "key": "roles",
36     "value": "ROLE_1,ROLE_2,ROLE_3,ROLE_4,ROLE_ADMIN,ROLE_ANONYMOUS,ROLE_DEPLOYER,
37     ROLE_MONITOR,ROLE_OPERATOR,ROLE_TESTER",
38     "type": "text"
39   },
40   {
41     "enabled": true,
42     "key": "tokenType",
43     "value": "Bearer",
44     "type": "text"
45   },
46   {
47     "enabled": true,
48     "key": "refreshToken",
49     "value": "eyJhbGciOiJIUzI1NiJ9.eyJwcm1uY2lwYWwiOiJlbnhJQUFBQUFBQUFBBS1ZUUFdcl1RRQmkraEp
50     SVORZUVdxVWdNWmFGc3lGR1R3cElwY1EwcWN1d29IME9MQkxyWTFcL1RhODUyNU83ZkpnaXkxCMETFSXFJVGdM
51     XC9TZndNSVBRREN3ZG1ibFBhZXVFNWFLbSs1OTd2SHpQdStIe jg3UmpKS28ycGVZTW1WRkxPNVRicWxJVXQ1W

```

```

52      HhJ0GwxVU1yVmtRR1JDZU1wd214Q3dnYW4xd2U1Un1VcDRGR3Q1MT1mSWpMRFB0KzJ1dnRFMVwvWEJoS1ZoT3
53      hmS081S0hKSWpJUSztzUzIxZ1NES1ZJS1BPZmM2a jJSMjBpSDFmeEZ5N2d0dURpRW9TN0tDRkRIT0VmMkNnS1I
54      5ZUN0Y1VNelZKb1NVYz14Z0pIR1RFc2Q0VGtKVVNwZE d0c2RsWVUxWnVMTF6MEZ5RWxRS jNcLzFUUzFzYTZ1
55      VGMYT1ZUd0NyMUd0VUdVZ3dPOWUyQ29sdEd4TmdSa1VEVVZYSzEwZVNNQ3VrdE5jdEFmTGJcL1wvZnZKbDFNM
56      GpCRDE1ZVBVM0dYNTNIWTIrdnZoekwybDB6dGZvem9UMWpGWWJST0JtTVZQdVNHSX1cL1wva1VcL0hCN1wvdm
57      I1TmNoc0dFXC8rZng0cj1Zdk9EVGRFR0dHSnRaaV1FY2d1RmVBK0QrTHJWNHVuVXhoYWJSCEdqtUJHY1UyQ314
58      U1pNS1Jia01LbFwvZGJvZXN0ejdKZXJ5Z0NsTkt4TWg5WHBjQzBONTVPd3Z0b11jbE9vTk1aY3o5MXV1TjEyQ
59      3Q5TTRFMjc2WGPiZG1ORmJ5Um93M08zT2w1cm11bzE3V1o5QWkwbWFnZHVkK31Xc1YzRVFVa jVhcVc20XVneD
60      1LaVVqTWpz bHVVSTJLemozKysrbmR6XC9DZ1UrUXpPSG1NVUV0bVfoSTdseJDUH16ZG5wY3Z1anIrT2szK21c
61      Lz14ZHFv0W1F dndNQUBPT0iLCJzdWIiOiJhZG1pbjEyMzQ1NiIsInJvbGVzIjpbI1JPTEVfMSIsI1JPTEVfMi
62      IsI1JPTEVfMyIsI1JPTEVfNCIsI1JPTEVfQURNSU4iLCJST0xFOX0FOT05ZTU9VUyIsI1JPTEVfREVQTE9ZRVIi
63      LCJST0xFOX01PTk1UT11iLCJST0xFOX0QRVJBVE9SiwiUk9MRV9URVNURVIiXSwiaWF0Ijo xNDkwODg5Njk3f
64      Q.Dow1ih9qm6AF1rfosglRIqTbn2X8Ymf3FWguABq17zU",
65      "type": "text"
66    },
67    {
68      "enabled": true,
69      "key": "lastQuestionIdAdded",
70      "value": "5",
71      "type": "text"
72    },
73    {
74      "enabled": true,
75      "key": "last",
76      "value": "",
77      "type": "text"
78    },
79    {
80      "enabled": true,
81      "key": "lastPropositionIdAdded",
82      "value": "10",
83      "type": "text"
84    },
85    {
86      "enabled": true,
87      "key": "lastQcmIdAdded",
88      "value": "5",
89      "type": "text"
90    }
91  ],
92  "timestamp": 1490889697882,
93  "_postman_variable_scope": "environment",
94  "_postman_exported_at": "2017-04-26T19:06:49.257Z",
95  "_postman_exported_using": "Postman/4.10.2"
96 }

```

One-click live docs for your teams collections [Try for 30 days and share with your team \(https://www.getpostman.com/pricing#cloud-free-trial-30\)](https://www.getpostman.com/pricing#cloud-free-trial-30)

Language

cURL

grails project

DELETE

DELETE DELETE question

http://localhost:8080/question/1

HEADERS

Accept

application/json

Content-Type

application/json

Sample Request

DELETE question

GRAILS PROJECT	
LAST UPDATED 30 MAR, 2016	
Introduction	
DELETE	
DELETE DELETE question	
DELETE DELETE qcm	
DELETE DELETE proposition	
GET	
GET GET question	
GET GET qcm	
GET GET proposition	
POST	
POST POST proposition	
POST POST question	
POST POST qcm	
POST POST question qcm proposition	

Authorization {{tokenType}} {{token}}
on
BODY
<pre>{ "name": "", "theme": "theme1", "body": "body1", "published": true, "chrono": 0, "timeLimit": 2, "maximumTry": 10, "seeScore": true, "seeAnswer": true,</pre>

DELETE DELETE qcm
<div>http://localhost:8080/qcm/1</div>
HEADERS

Accept	application/json
---------------	------------------

<pre>curl --request DELETE \ --url http://localhost:8080/question/1 \ --header 'accept: application/json' \ --header 'authorization: {{tokenType}} {{token}}' \ --header 'content-type: application/json' \ --data '{ "name": "", "theme": "theme1", "body": "body1", "published": true</pre>

Sample Request

DELETE qcm

GRAILS PROJECT	
LAST UPDATED 30 MAR, 2016	
Introduction	
DELETE	
DELETE question	
DELETE qcm	
DELETE proposition	
GET	
GET question	
GET qcm	
GET proposition	
POST	
POST proposition	
POST question	
POST qcm	
POST question qcm	
POST proposition	

Content-Type	application/json
Authorization	{{tokenType}} {{token}}
Body	<pre>{ "name": "", "theme": "theme1", "body": "body1", "published": true, "chrono": 0, "timeLimit": 2, "maximumTry": 10, "seeScore": true, "seeAnswer": true, }</pre>

DELETE DELETE proposition
http://localhost:8080/proposition/1

curl --request DELETE \
--url http://localhost:8080/qcm/1 \
--header 'accept: application/json' \
--header 'authorization: {{tokenType}} {{token}}' \
--header 'content-type: application/json' \
--data '{
"name": "",
"theme": "theme1",
"body": "body1",
"published": true

Sample Request
DELETE proposition

GRAILS PROJECT

LAST UPDATED 30 MAR, 2020

Introduction

DELETE

DEL DELETE question

DEL DELETE qcm

DEL DELETE proposition

GET

GET GET question

GET GET qcm

GET GET proposition

POST

POST POST proposition

POST POST question

POST POST qcm

POST POST question qcm

proposition

HEADERS

Acceptapplication/json

Content-Typeapplication/json

Authorization{{tokenType}} {{token}}

BODY

```
{
  "name": "",
  "theme": "theme1",
  "body": "body1",
  "published": true,
  "chrono": 0,
  "timeLimit": 2,
  "maximumTry": 10,
  "seeScore": true,
  "seeAnswer": true,
}
```

```
curl --request DELETE \
  --url http://localhost:8080/proposition/1 \
  --header 'accept: application/json' \
  --header 'authorization: {{tokenType}} {{token}}' \
  --header 'content-type: application/json' \
  --data '{
    "name": "",
    "theme": "theme1",
    "body": "body1",
    "published": true
  }'
```

GRAILS PROJECT
LAST UPDATED 30 MAR, 2016
Introduction
DELETE
DELETE DELETE question
DELETE DELETE qcm
DELETE DELETE proposition
GET
GET GET question
GET GET qcm
GET GET proposition
POST
POST POST proposition
POST POST question
POST POST qcm
POST POST question qcm proposition

GET

GET GET question

http://localhost:8080/question/

HEADERS	
Accept	application/json
Content-Type	application/json
Authorization	{{tokenType}} {{token}}

GET GET qcm

Sample Request

GET question
<pre>curl --request GET \ --url http://localhost:8080/question/ \ --header 'accept: application/json' \ --header 'authorization: {{tokenType}} {{token}}' \ --header 'content-type: application/json'</pre>

Sample Request

GET qcm

GRAILS PROJECT	
LAST UPDATED 30 MAR, 2016	
Introduction	
DELETE	
DELETE	DELETE question
DELETE	DELETE qcm
DELETE	DELETE proposition
GET	
GET	GET question
GET	GET qcm
GET	GET proposition
POST	
POST	POST proposition
POST	POST question
POST	POST qcm
POST	POST question qcm
	proposition

http://localhost:8080/qcm	
HEADERS	
Accept	application/json
Content-Type application/json	
Authorization {{tokenType}} {{token}}	

GET GET proposition	
http://localhost:8080/proposition	
HEADERS	
Accept	application/json

```
curl --request GET \
--url http://localhost:8080/qcm \
--header 'accept: application/json' \
--header 'authorization: {{tokenType}} {{token}}' \
--header 'content-type: application/json'
```

Sample Request

GET proposition
<pre>curl --request GET \ --url http://localhost:8080/proposition \ --header 'accept: application/json' \ --header 'authorization: {{tokenType}} {{token}}' \ --header 'content-type: application/json'</pre>

GRAILS PROJECT	
LAST UPDATED 30 MAR, 2016	
Introduction	
DELETE	
DELETE	DELETE question
DELETE	DELETE qcm
DELETE	DELETE proposition
GET	
GET	GET question
GET	GET qcm
GET	GET proposition
POST	
POST	POST proposition
POST	POST question
POST	POST qcm
POST	POST question qcm
	proposition

Content-Type	application/json
Accept	
Authorization: {{tokenType}} {{token}}	
on	
POST	
POST POST proposition	
http://localhost:8080/proposition/	
HEADERS	
Accept	application/json
Content-Type	application/json
pe	

Sample Request
POST proposition
curl --request POST \
--url http://localhost:8080/proposition/ \
--header 'accept: application/json' \
--header 'authorization: {{tokenType}} {{token}}' \
--header 'content-type: application/json' \
--data '{
"body": "body proposition {{lastPropositionIdAc
"isCorrect": true,
"reason": "reason proposition {{lastProposition
}

GRAILS PROJECT	
LAST UPDATED 30 MAR, 2016	
Introduction	
DELETE	
DELETE	DELETE question
DELETE	DELETE qcm
DELETE	DELETE proposition
GET	
GET	GET question
GET	GET qcm
GET	GET proposition
POST	
POST	POST proposition
POST	POST question
POST	POST qcm
POST	POST question qcm proposition

Authorization {{tokenType}} {{token}}	
BODY	
<pre>{ "body": "body proposition {{lastProposition}}", "isCorrect": true, "reason": "reason proposition {{lastProposition}}", }</pre>	
POST POST question	
<pre>http://localhost:8080/question/</pre>	
HEADERS	
Accept	application/json
Content-Type	application/json

Sample Request
POST question

<div><div>GRAILS PROJE</div><div>LAST UPDATED 30 MAR, 20</div><div>Introduction</div><div>DELETE</div><div>DELETE DELETE questio</div><div>DELETE DELETE qcr</div><div>DELETE DELETE propositio</div><div>GET</div><div>GET GET questior</div><div>GET GET qcm</div><div>GET GET propositior</div><div>POST</div><div>POST POST propositior</div><div>POST POST question</div><div>POST POST qcm</div><div>POST POST question qcm</div><div>proposition</div></div>	<div><div>Authorizati{{tokenType}} {{token}}</div><div>on</div><div>BODY</div><div><div><div>{</div><div>"body": "body question {{lastQuestionIdAc</div><div>"chrono": 0,</div><div>"goBack": true,</div><div>"maximumTry": 10,</div><div>"name": "name{{lastQuestionIdAdded}}",</div><div>"pass": true,</div><div>"published": true,</div><div>"questionType": {</div><div>"id": {{lastQcmIdAdded}}</div></div></div></div>	<div><div>curl --request POST \ --url http://localhost:8080/question/ \ --header 'accept: application/json' \ --header 'authorization: {{tokenType}} {{token}}' \ --header 'content-type: application/json' \ --data '{ "body": "body question {{lastQuestionIdAdded}}", "chrono": 0, "goBack": true, "maximumTry": 10</div></div>	
	<div><div>POST POST qcm</div><div><div>http://localhost:8080/qcm/</div></div><div>HEADERS</div><div>Acceptapplication/json</div></div>	<div><div>Sample Request</div><div>POST qcm</div></div>	

GRAILS PROJECT	
LAST UPDATED 30 MAR, 2018	
Introduction	
DELETE	
DELETE question	
DELETE question	
DELETE proposition	
GET	
GET question	
GET question	
GET proposition	
POST	
POST proposition	
POST question	
POST question	
POST question question	

Content-Type	application/json
Accept	application/json
Authorization	{{tokenType}} {{token}}
Content-Type	application/json
BODY	
{ "propositions": [{ "id": {{lastPropositionIdAdded}} }, { "id": 4 }, { "id": 5 }] }	

POST question question
proposition
http://localhost:8080/question/

```
curl --request POST \  
  --url http://localhost:8080/qcm/ \  
  --header 'accept: application/json' \  
  --header 'authorization: {{tokenType}} {{token}}' \  
  --header 'content-type: application/json' \  
  --data '{  
    "propositions": [  
      {  
        "id": {{lastPropositionIdAdded}}  
      }  
    ]  
  }'
```

Sample Request

POST question qcm proposition

<h1>GRAILS PROJE</h1> <p>LAST UPDATED 30 MAR, 20</p> <h2>Introduction</h2> <h3>DELETE</h3> <p>DEL DELETE questio</p> <p>DEL DELETE qcm</p> <p>DEL DELETE propositio</p> <h3>GET</h3> <p>GET GET question</p> <p>GET GET qcm</p> <p>GET GET propositior</p> <h3>POST</h3> <p>POST POST propositior</p> <p>POST POST question</p> <p>POST POST qcm</p> <p>POST POST question qcm proposition</p>	<div>HEADERS</div> <table><tr><td>Accept</td><td>application/json</td></tr><tr><td>Content-Ty</td><td>application/json</td></tr><tr><td>pe</td><td></td></tr><tr><td>Authorizati</td><td>{{tokenType}} {{token}}</td></tr><tr><td>on</td><td></td></tr><tr><td colspan="2">BODY</td></tr></table>		Accept	application/json	Content-Ty	application/json	pe		Authorizati	{{tokenType}} {{token}}	on		BODY		<pre>curl --request POST \ --url http://localhost:8080/question/ \ --header 'accept: application/json' \ --header 'authorization: {{tokenType}} {{token}}' \ --header 'content-type: application/json' \ --data '{ "body": "body question {{lastQuestionIdAdded}}", "chrono": 0, "goBack": true, "maximumTry": 10, "name": "name{{lastQuestionIdAdded}}", "pass": true, "published": true, "questionType": { "qcm":{ "propositions": [{ "body": "body proposition {{lastPro "isCorrect": true, "reason": "reason proposition {{las }, { "body": "body proposition {{lastPro "isCorrect": true, "reason": "reason proposition {{las }] } }, "seeAnswer": true,</pre>
	Accept	application/json													
	Content-Ty	application/json													
	pe														
	Authorizati	{{tokenType}} {{token}}													
on															
BODY															

GRAILS PROJECT	
LAST UPDATED 30 MAR, 2016	
Introduction	
DELETE	
DELETE	DELETE question
DELETE	DELETE qcm
DELETE	DELETE proposition
GET	
GET	GET question
GET	GET qcm
GET	GET proposition
POST	
POST	POST proposition
POST	POST question
POST	POST qcm
POST	POST question qcm proposition

```
{
  "body": "body question {{lastQuestionIdAdded}}",
  "chrono": 0,
  "goBack": true,
  "maximumTry": 10,
  "name": "name{{lastQuestionIdAdded}}",
  "pass": true,
  "published": true,
  "questionType": {
    "qcm": {
```

```
    "seeScore": true,
    "theme": "theme{{lastQuestionIdAdded}}",
    "timeLimit": 100
  }
}
```

login

POST login

```
http://localhost:8080/api/login
```

Login

HEADERS

Accept application/json

Sample Request

```
login

curl --request POST \
  --url http://localhost:8080/api/login \
  --header 'accept: application/json' \
  --header 'content-type: application/json' \
  --data '{
    "username": "admin123456",
    "password": "123456789"
  }'
```

GRAILS PROJECT	
LAST UPDATED 30 MAR, 2016	
Introduction	
DELETE	
DELETE DELETE question	
DELETE DELETE qcm	
DELETE DELETE proposition	
GET	
GET GET question	
GET GET qcm	
GET GET proposition	
POST	
POST POST proposition	
POST POST question	
POST POST qcm	
POST POST question qcm proposition	

Content-Type application/json	
Body	
BODY	
<pre>{ "username": "admin123456", "password": "123456789" }</pre>	
POST login - userTester	
<pre>http://localhost:8080/api/login</pre>	
Login	
HEADERS	
Accept	application/json
Content-Type application/json	
Body	

Sample Request

login - userTester
<pre>curl --request POST \ --url http://localhost:8080/api/login \ --header 'accept: application/json' \ --header 'content-type: application/json' \ --data '{ "username": "userTester", "password": "123456789" }'</pre>

<div>GRAILS PROJECT</div> <div>LAST UPDATED 30 MAR, 2018</div> <div>Introduction</div> <div><div>DELETE</div><div>DELETE question</div><div>DELETE qcm</div><div>DELETE proposition</div><div>GET</div><div>GET question</div><div>GET qcm</div><div>GET proposition</div><div>POST</div><div>POST proposition</div><div>POST question</div><div>POST qcm</div><div>POST question qcm proposition</div></div>		<div>BODY</div> <div><pre>{ "username": "userTester", "password": "123456789"}</pre></div> <div>HEADERS</div> <div><div>Acceptapplication/json</div><div>Content-Typeapplication/json</div><div>AuthorizationBearer {{token}}</div></div>		<div>Sample Request</div> <div>POST SAVE question - http://localhost:8080/qcm</div> <div></div>	
---	--	---	--	---	--

GRAILS PROJECT	
LAST UPDATED 30 MAR, 2016	
Introduction	
DELETE	
DEL DELETE question	
DEL DELETE qcm	
DEL DELETE proposition	
GET	
GET GET question	
GET GET qcm	
GET GET proposition	
POST	
POST POST proposition	
POST POST question	
POST POST qcm	
POST POST question qcm	
POST POST proposition	

BODY	
<pre>{ "body": "body6", "chrono": 0, "goBack": true, "maximumTry": 10, "name": "name6", "pass": true, "published": true, "questionType": { "id": 1 } }</pre>	
HEADERS	
Accept	application/json
Content-Type	
application/json	

curl --request POST \	
--url http://localhost:8080/question \	
--header 'accept: application/json' \	
--header 'authorization: Bearer {{token}}' \	
--header 'content-type: application/json' \	
--data '{	
"body": "body6",	
"chrono": 0,	
"goBack": true,	
"maximumTry": 10	
}'	
Sample Request	
POST SAVE proposition - http://localhost:8080	
/proposition/	
curl --request POST \	
--url http://localhost:8080/proposition \	
--header 'accept: application/json' \	
--header 'authorization: Bearer {{token}}' \	
--header 'content-type: application/json' \	
--data '{	
"body": "body proposition 1",	
"reason": "reason proposition 1",	
"isCorrect": true	
'	

<h1>GRAILS PROJECT</h1> <p>LAST UPDATED 30 MAR, 2020</p> <h2>Introduction</h2> <h3>DELETE</h3> <p>DELETE question</p> <p>DELETE qcm</p> <p>DELETE proposition</p> <h3>GET</h3> <p>GET question</p> <p>GET qcm</p> <p>GET proposition</p> <h3>POST</h3> <p>POST proposition</p> <p>POST question</p> <p>POST qcm</p> <p>POST question qcm proposition</p>	
Header	<p>Authorization: Bearer {{token}}</p>
Body	<pre>{ "body": "body proposition 1", "reason": "reason proposition 1", "isCorrect": true}</pre>

Comptes rendus hebdomadaires

Compte rendu n°1 du 23/09/2016

Lors de ces deux premiers jours de projet, j'ai commencé à me familiariser avec le sujet. J'ai pu voir les débuts du projet réalisé par un ancien PRD grâce au rapport de l'étudiant.

J'ai commencé la recherche de publications liées à la correction automatique d'exercices (qui sont en fait pour la plupart des examens de QCM ou choix multiples).

La préparation de la machine virtuelle Ubuntu reste un problème. Par trois fois, l'exécution du script d'auto-configuration a totalement bloqué ma machine virtuelle. Je cherche encore un moyen de réussir l'installation. Si la VM ne fonctionne pas, je ferais un dual boot.

Pour la semaine prochaine, je compte continuer à lire des publications pour savoir plus précisément ce qui existe sur les corrections automatique.

Je vais également commencer Spring boot une fois la machine virtuelle fonctionnelle.

Compte rendu n°2 du 29/09/2016

Kubuntu est enfin installé.

J'ai continué à lire des rapports d'études sur les quiz mais la littérature sur le sujet est très peu fournie. Je ne crois pas qu'il y ait de réelle technologie existante pour nous aider au façonnement du projet. J'ai tout de même vu deux outils qui me semblent intéressants :

- TAO qui est une plateforme opensource pour les quizzes et qui est compatible avec Moodle
- Moodle qui comprend un projet important sur l'autoformation et même le passage d'examens en ligne Pour les examens, il y a également SEB (Safe Exam Browser) qui est intéressant, même si ça s'éloigne de notre sujet je pense que c'est intéressant de garder à l'esprit qu'il existe.

J'ai continué ma liste de types de quiz et sujet qui peuvent s'y lier.

D'une manière générale, le QCM et Choix Multiple sont les principales demandes car c'est le premier type de test qui vient à l'esprit des gens et la correction "à la main" est vraiment une perte de temps. Mais en cherchant plus, je suis en train de me faire un panel de questions types ou exercices types par secteurs. Des demandes liées à la musique, à l'imagerie au droit ou aux mathématiques possèdent des singularités intéressantes. Pour tout cela, il nous faudra des modules différents mais même si nous allons prévoir un projet très modulable, il reste important de noter que la plupart des test peuvent être traduits sous forme de Choix Multiples. Aussi, il faut garder à l'esprit que nos utilisateur ne sont pas forcements en informatique

donc il faudra développer une interface particulièrement facile à prendre en main (autant pour les testeurs que pour ceux qui vont proposer leurs questions) car si en informatique nous avons l'habitude de nous adapter à une notation différente, d'autres personnes n'ont pas forcément ces automatismes. Par exemple, la représentation d'un graphe est facile pour nous en informatique avec du DOT mais pour quelqu'un qui vient d'un secteur tel que la chimie, la logistique ou l'aménagement le DOT est incompréhensible et les utilisateurs ne prendront pas le temps de comprendre. Il faudra donc des mécanismes de création de noeuds/arrêtes ... pour gérer les graphes d'une façon visuelle. Nous devons nous détacher du bas niveau pour aller au maximum vers du graphisme même si cela nécessite ensuite de retraduire ces informations dans un langage plus bas niveau (ie : traduire nos noeuds/arrêtes en DOT automatiquement). Le problème de graph n'est qu'un exemple simple mais il y en a d'autres bien plus complexes.

Le traitement des réponses QCM et Choix Multiples est très fait avec l'application Auto Multiple Choice qui fonctionne avec LaTeX ou des fichiers texte en entrée. Je pense que cet outil sera utilisable pour notre projet. Je vais continuer à voir les possibilités d'auto gestion de l'application. De plus, cette application est probablement maîtrisée par M. Aupetit et M. Monmarché car je crois que c'est avec qu'ils ont préparés nos examens il y a quelques années. Je suppose qu'un traitement des réponses a donc été développé à Polytech, ce qui nous permettra de gagner du temps de développement.

Questions :

- Nos examens ont-ils été réalisés avec Auto Multiple Choice ? (Je pense que oui) Comment s'effectue le traitement après réponses ?
- Moodle fait-il partie de Celene ?

Problème :

Depuis l'exécution du script sur Kubuntu, impossible de booter sur Windows. J'ai formaté mes disques durs et testé une réinstallation de windows mais l'installation bloque au moment de sélectionner le disque dur d'installation (mes disques sont désormais accessibles depuis Kubuntu)

Compte rendu n°3 du 06/10/2016

Continuation du listage des différents quiz qui pourraient être proposés puis traduction du modèle littéraire vers un graph des différentes entrées/sorties de notre système (le "quoi"). Continuation de la liste complémentaire du "comment" avec différentes méthodes de tests et analyse de texte.

En parallèle, j'ai continué à lire des articles sur ce qui existe sur l'autoformation (self-assessment / self-reflection). Je continue de constater qu'il y a peu de littérature sur des sujets hors "QCM". Les études pour étudiants en informatique sont peu fournies et surtout peu poussées.

J'ai commencé la rédaction du Cahier De Specification du projet.

Compte rendu n°4 du 13/10/2016

Pendant la semaine, j'ai lu des articles sur la gestion de QCM et de TDD (Test-Driven Design). Nous avons pu définir une première ligne directrice pour les recherches à effectuer. Il faut donc que je pousse plus les recherches sur les algorithmes/gestion de quiz. Que ce soit les QCM ou d'autres méthodes, je dois plus pousser mes recherches.

Je dois également faire une analyse plus fine des questions que pourront comporter notre application. Cette analyse pourra permettre de faire différents cadres d'utilisation et donc de faire ressortir de nouvelles problématiques ou de mieux définir nos besoins. Le MindMap doit donc être plus fin pour ensuite faire des cas d'utilisation et développer notre UML.

Le framework Grails pourrait être intéressant autant pour sa gestion simplifiée de certains modules que pour son maintien dans le temps. Il reste donc à voir si Grails est un réel gain.

Compte rendu n°5 du 20/10/2016

J'ai commencé à regarder la documentation Grails pour étudier plus en profondeur ce qu'il est possible de faire avec. J'ai également essayé de créer des premiers tests en Groovy.

Après avoir installé Grails 3 avec sdkman, je n'ai pas réussi à gérer les plugins Eclipse. Le support Eclipse n'étant plus actif, nous allons donc changer d'IDE pour utiliser IntelliJ Ultimate qui est l'IDE qui semble le mieux intégrer le plugin Grails 3. En contrepartie, il faudra réinstaller à la main tous les plugins utiles d'Eclipse sur IntelliJ.

L'écriture du cahier de spécification n'est pas encore terminée, je liste des composants et fais des diagrammes pour illustrer. Il sera terminé (dans sa première version) vers le milieu/fin de la pause pédagogique. Je dois traduire mes idées en description détaillées et reproduire mes UML en version numérique.

Je n'ai pas eu le temps de travailler en dehors des horaires normaux car nous avons beaucoup de travail en dehors.

Le travail de la semaine sera donc la rédaction du cahier de spécification et l'installation/configuration d'IntelliJ avec Grails.

Compte rendu n°6 du 27/10/2016

J'ai continué le cahier de spécification demandé par Monsieur Ragot. Celui-ci est remis mais ne comporte pas de termes très techniques. Il met à plat ce que nous cherchons à faire avec l'application.

Compte rendu n°7 du 03/11/2016

Le cahier de spécification est remis. Il y aura des ajouts et modifications (UML principalement) mais le principal y est.

J'ai continué la liste technologique pour le projet.

Oui, Grails sera bien utilisé avec Spring Boot. Il nous faudra choisir d'un point de vue sécurité (avec gestion des comptes) entre Spring Security, Shiro + Spring Security ou Nimble.

Du point de vue, interface utilisateur. Soit nous utilisons Grails de base (et nous pouvons modifier les templates de base ou en rajouter) soit nous passons par Vaadin.

Il nous faudrait également voir l'accès à la Base de Données et donc choisir le type de base de données. Un simple SQL avec Hibernate serait certainement le plus simple à gérer et à faire évoluer.

Aussi, il faut choisir un système de gestion de Mails. Spring en propose un, je pense qu'il est suffisant pour ce que nous allons en faire.

Enfin, faut-il prévoir de faire des mises à jour du système avec un outil comme JMX (Java Management Extension)? Je pense que ce serait bien de le prévoir.

Compte rendu n°8 du 10/11/2016

Cette semaine, j'ai pu continuer la liste technologique avec celle des différents modules à développer. J'ai par ailleurs commencé mes lectures sur Dockers et avancé dans la documentation Groovy.

Compte rendu n°9 du 16/11/2016

Après notre réunion concernant les choix technologiques, j'ai pu lister la liste de choix :

- Grails 3.2 (groovy 2.7)
- Spring boot
- GORM 6 (Hibernate + H2 BDD)
- Emails avec Spring
- Dockers
- Java 1.8

Notre seconde réunion de travail concernant les modules de test à développer nous a permis de voir les attentes concernant la création/édition/test des questions ainsi que des quiz.

A la suite de cette dernière réunion, j'ai commencé à faire du sketching pour nous donner une idée plus précise des fonctionnalités nécessaires. Pour continuer à apprendre le groovy, j'ai voulu réaliser cette partie avec gsp (Groovy Server Page). Malheureusement j'ai de grandes difficultés à comprendre les liens entre les vues (gsp), les contrôleurs et les services.

Si possible, j'aimerais donc que nous puissions nous revoir pour faire un point sur ce qu'il est possible de faire pour gérer les vues car je suis bloqué à ce niveau là.

La semaine prochaine, nous avons une réunion de PRD mercredi de 10h30 à 12h30.

Aussi, jeudi se déroulera le forum des entreprises, je vais donc chercher une salle différente pour travailler (probablement dans l'ancienne salle d'impression 3D au 1er étage).

Compte rendu n°10 du 23/11/2016

Après les difficultés liées au gsp que j'ai pu rencontré, j'ai voulu essayé Angular.js comme vous m'en aviez parlé. Je préfère son fonctionnement. De plus de nombreuses personnes de la communauté sont passées sous Angular.js car l'application web est directement responsive. J'espère mieux réussir avec Angular qu'avec gsp.

Jeudi, j'ai participé au forum des entreprises toute la journée. Je n'ai donc pas avancé le projet.

Compte rendu n°11 du 30/11/2016

Angular.js n'est pas facile mais je pense mieux m'en sortir que gsp. Toutefois, j'ai choisi de réaliser mon sketching avec SceneBuilder que je connais déjà. La date de rendu du rapport approche, je préfère donc avancer plus rapidement sur des problèmes que je pourrais régler pendant les vacances. Je consacre donc cette semaine ainsi que la semaine prochaine à l'écriture de rapport.

Compte rendu n°12 du 08/12/2016

Dernier rapport avant la première soutenance. J'ai perdu du temps à essayer de configurer ma bibliographie et le glossaire mais le rapport sera rendu à temps.

Pour ce qui est du Gantt, je n'ai pas de référentiel, donc j'ai donné une estimation que j'espère pouvoir respecter.

J'ai commencé ma présentation pour la soutenance.

Webographie

- [WWW1] URL : <https://www.ceeol.com/search/journal-detail?id=574>.
- [WWW2] Mikael BARBERO et Laurent GOUBET. *EMF Compare*. 2014. URL : https://wiki.eclipse.org/EMF_Compare/UML_Compare.
- [WWW3] Hongbo CHEN et Ben HE. *Automated Essay Scoring by Maximizing Human-machine Agreement*. 2013. URL : http://www.aclweb.org/website/old_anthology/D/D13/D13-1180.pdf.
- [WWW4] GORM. *Documentation*. URL : <http://gorm.grails.org/6.0.x/hibernate/manual/>.
- [WWW5] GRAILS. *Plugins*. URL : <http://plugins.grails.org/>.
- [WWW6] Julien HERPIN. *Introduction aux JSON Web tokens*. URL : <https://www.ekino.com/introduction-aux-json-web-tokens/>.
- [WWW7] IETF. *RFC 6750 - The OAuth 2.0 Authorization Framework : Bearer Token Usage*. URL : <https://tools.ietf.org/html/rfc6750>.
- [WWW8] Shawn McKAY. *Comparing Front End Frameworks : Code Style*. URL : <http://www.shmck.com/comparing-front-end-frameworks-code-style/>.
- [WWW9] Shawn McKAY. *Comparing Performance of Blaze, React, Angular-Meteor and Angular 2 with Meteor*. URL : <https://blog.meteor.com/comparing-performance-of-blaze-react-angular-meteor-and-angular-2-with-meteor-c650c913d3f8>.
- [WWW10] WIKIPEDIA. *Automated Essay Scoring*. 2016. URL : https://en.wikipedia.org/wiki/Automated_essay_scoring.



Bibliographie

- [1] Valérian MENIN. « Gamification design pattern of Android/iOS applications ». Rapport de stage DI3. Tours, France : Ecole Polytechnique de l'Université François Rabelais de Tours, 2014-2015.
- [2] OECD. *Education at a Glance 2014 : OECD Indicat*. Sous la dir. d'OECD. Page 479. OECD, 2014. URL : <http://www.oecd.org/edu/Education-at-a-Glance-2014.pdf>.



Glossaire

exécution L'exécution d'une question est le fait d'y répondre. 6, 16

module Un module est une entité regroupant un groupe. Par exemple, la création / édition / exécution d'un QCM est un module différent de celui accordeur. iv, 8, 9, 11, 15–19, 22–24

moodle Moodle est une plateforme d'apprentissage Open-Source. 10

Open Source Libre de droit, dont les codes sources sont disponibles et modifiable librement. 10, 11, 20

question Une question contient une sortie (texte, image, son, ...) qui demande à l'utilisateur de fournir une entrée (texte, image, son, ...) pour traiter cette entrée comme étant la réponse à la demande faite en sortie. iv, 6, 8, 9, 14, 16–20, 23, 24

quiz Un quiz est un recueil de questions. iv, 6, 8, 14, 16, 17

réponse Equivalent du traitement. iv, 8, 17–23

tag Mots clés utilisé pour améliorer le référencement et donc augmenter la lisibilité. 16, 18

évaluation L'évaluation est le fait de corriger l'entrée d'une question. C'est le mécanisme qui va dire si l'entrée est correcte, n'est pas correcte ou s'il faut une intervention humaine pour traiter le problème. 16–19, 23



Acronymes

ACL Access Control List (liste de contrôle d'accès). 6

ACM Auto Multiple Choice. 18

BDD Base De Données. 6, 14, 21, 25

BETSY Bayesian Essay Test Scoring sYstem. 19, 20

ENT Environnement Numérique de Travail. 10

GORM Groovy Object Relational Mapping. 12, 14

IU Interface Utilisateur. iv, 11, 14, 16–18, 21

JLX Java Management Extensions. 14

LMS Learning Management System (plateforme d'apprentissage). 2, 10

QCM Question à Choix Multiple. iv, 2, 16–19, 25

QCU Question à Choix Unique. iv, 16, 18, 19, 25

UML Unified Modeling Language. iv, 16, 20, 21, 25

Outil d'autoformation pour ingénieur en informatique

Valérien Menin

Encadrement : Sébastien Aupetit

Constat

L'apprentissage dans les livres et sur les forums est pratique mais qu'en est-il de connaître son niveau? Peu de solutions gratuites existent.

Il est important que les étudiants puissent se former en dehors des cours et donc puissent obtenir une notation automatisée sur des exercices.

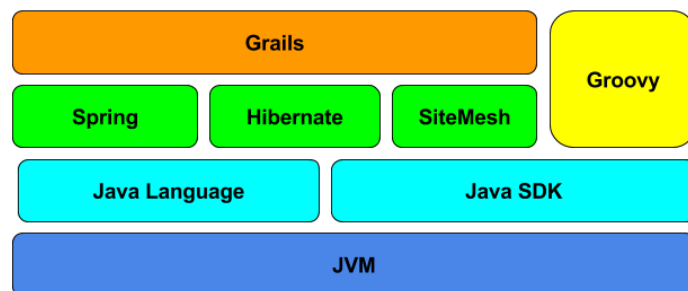
Solution

Un outil de correction automatique pour auto-former les élèves serait parfait pour gagner du temps en correction.

Nous avons donc choisi de développer une application web adaptée aux besoins des ingénieurs de Polytech.

Conclusion

Notre plateforme web d'autoformation permet aux élèves de pouvoir se former sur des sujets qu'ils choisissent suivant leurs affinités ou leurs lacunes. Avec une réponse rapide à de nombreux types d'exercices, ce projet permet d'aider les élèves dans leur apprentissage.



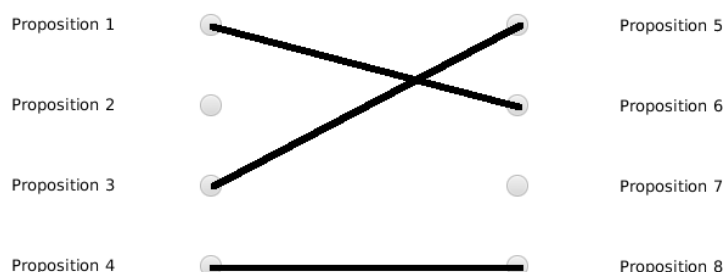
Les technologies sur lesquelles Grails 3 repose

Cette image montre l'interface utilisateur pour créer une question. Elle est divisée en plusieurs sections :

- Question** : un champ de texte pour le 'nom de la question'.
- Liste réponses** : un tableau avec quatre colonnes : 'Réponse', 'Module', 'Vrai' et 'Commentaire'.

Réponse	Module	Vrai	Commentaire
Proposition 1	texte	<input checked="" type="checkbox"/>	Vrai car ...
Proposition 2	texte	<input type="checkbox"/>	Faux car ...
- Ajouter une réponse** : un bouton vert avec un signe plus.
- Spécificités** : un champ de texte contenant des paramètres de notation : `toutBon=5;parBon=proportionnel;parErreur=-proportionnel*2;faux=-2;sansReponse=0;`

Création de question



Réponse à une question

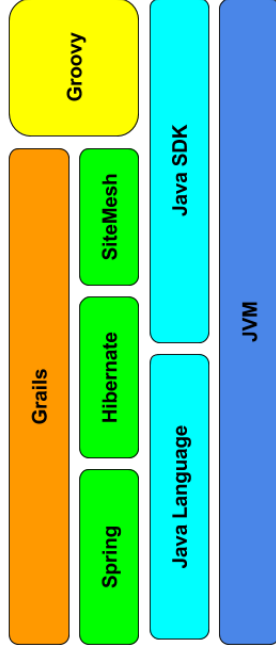
Outil d'autoformation pour ingénieur en informatique

Valérien Menin

Encadrement : Sébastien Aupetit

Constat

L'apprentissage dans les livres et sur les forums est pratique mais qu'en est-il de connaître son niveau? Peu de solutions gratuites existent.
Il est important que les étudiants puissent se former en dehors des cours et donc puissent obtenir une notation automatisée sur des exercices.



Les technologies sur lesquelles Grails 3 repose

Solution

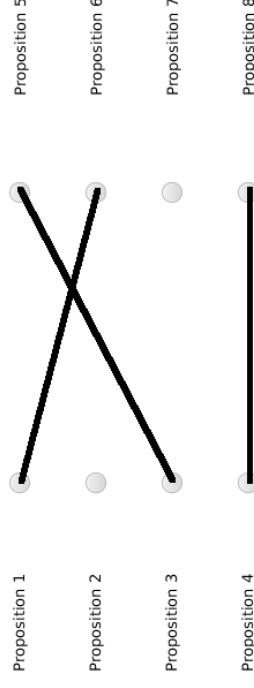
Un outil de correction automatique pour auto-former les élèves serait parfait pour gagner du temps en correction.
Nous avons donc choisi de développer une application web adaptée aux besoins des ingénieurs de Polytech.

Conclusion

Notre plateforme web d'autoformation permet aux élèves de pouvoir se former sur des sujets qu'ils choisissent suivant leurs affinités ou leurs lacunes. Avec une réponse rapide à de nombreux types d'exercices, ce projet permet d'aider les élèves dans leur apprentissage.

Question	nom de la question	Réponse	Module	Vrai	Commentaire
Liste réponses		Proposition 1	texte	<input checked="" type="checkbox"/>	Vrai car ...
		Proposition 2	texte	<input type="checkbox"/>	Faux car ...
Spécificités					
toutBon=5;parBon=proportionnel;parErreur=-proportionnel*2;faux=-2;sansReponse=0;					
Ajouter une réponse					

Création de question



Réponse à une question

Outil d'autoformation pour ingénieur en informatique

Résumé

Les élèves ingénieurs ont besoin de se former sur de nombreux sujets différents. Chaque étudiant possède ses propres besoins et avance à son propre rythme. Devoir aller en cours est utile mais un complément est souvent apprécié pour approfondir ses connaissances ou réviser. Pourtant, il est très difficile de trouver un outil permettant de se former automatiquement avec des sujets pertinents pour tous. Le projet vient donc du fait que nous souhaitons nous autoformer rapidement sans avoir à être dans un endroit précis ou avec une personne qui pourrait nous partager ses connaissances. Notre professeur n'est pas toujours avec nous pour nous enseigner quelque chose. L'outil d'autoformation pour ingénieur en informatique a pour objectif de répondre à ce besoin avec une gestion de comptes utilisateurs, une gestion des accès et surtout une communauté capable de s'agrandir seule ainsi que d'apporter ses propres connaissances pour faire évoluer l'outil. Ce projet cherche à créer un outil modulaire pour lequel il est possible de créer des questions, de les classer, de créer une sélection de questions pour faire un quiz et de suivre sa montée en compétences à l'aide de statistiques utilisateurs.

Mots-clés

autoformation, informatique, note, modulaire, question, quiz, formation, groovy, Angular2, sécurité

Abstract

Engineering students need to train themselves with plenty of different themes. Each student has its own needs and speed. Going to class is useful but it is often appreciated to have another way to learn or review. Nevertheless, it is very difficult to find a tool allowing to train oneself automatically with an efficient thematic. This project thus comes because we wish to train ourselves quickly without having to be in a precise place or with a person who could share its knowledge. The autotraining for IT engineer tool's goal is to answer this need with a management of user accounts, a management of the accesses and especially one community capable of getting bigger as well as of bringing its appropriate knowledge to develop the tool. This project tries to create a modular tool for whom it is possible to create questions, to classify them, to create a selection of questions to make a quiz and follow its rise in skills by means of users' statistics.

Keywords

autoformation, IT, scoring, modular, question, quiz, training, groovy, Angular2, security