HAUTE ÉCOLE
D'INGÉNIERIE ET DE GESTION
DU CANTON DE VAUD
www.heig-vd.ch

Travail de Bachelor

Mars - Septembre 2018

Création d'une application pour la gestion de stock centralisée et localisation des articles

Auteur :
BENJAMIN CURRAT

Mandant :
Arnaud JAQUIER (Kieed)
Conseiller :
Raphaël P. BARAZZUTTI

heig-vd
HAUTE ÉCOLE
D'INGÉNIERIE ET DE GESTION
DU CANTON DE VAUD
www.heig-vd.ch

Travail de Bachelor

Préambule

Ce travail de Bachelor est réalisé en vue de l'obtention du titre de Bachelor of Sciences en Ingénierie.

Son contenu, sans préjuger de sa valeur, n'engage ni la responsabilité de l'auteur, ni celles du jury du travail de Bachelor et de l'Ecole.

Aucune utilisation, même partielle, de ce travail ne peut être faite sans l'autorisation écrite préalable de la Direction. Toutefois, l'entreprise ou l'organisation qui a confié un sujet de travail de Bachelor peut utiliser les résultats du travail pour ses propres besoins.

Doyenne du
Centre Formation de Base

L. Larghi

Yverdon-les-Bains, novembre 2017

HAUTE ÉCOLE
D'INGÉNIERIE ET DE GESTION
DU CANTON DE VAUD
www.heig-vd.ch

Département **FEE**
Filière **Informatique**
Orientation **Logiciel**
Candidat **Currat Benjamin**
Responsable **Barazzutti Raphaël**

TRAVAIL DE BACHELOR 2017 - 2018

Création d'une application pour la gestion de stock centralisée et la localisation des articles

---

Entreprise Kieed.ch (Jaquier Arnaud)

Résumé publiable

Contexte

L'entreprise Kieed, active dans la création de T-shirts cherche à optimiser leur gestion du stock et la localisation des articles. L'inventaire de leur stock devenant difficile à suivre. Ils se retrouvent avec des décomptes de marchandises qui ne correspondent pas aux commandes chez leur fournisseur. La vente s'articule aux travers de différents canaux de distribution qui ne simplifient pas cette gestion. Dans le processus actuel, seuls les bons de commandes passés aux fournisseurs permettent de connaître les entrées et sorties de marchandises.

Objectif

Le but premier est de créer un service web central qui permettra la gestion du stock de marchandises de l'entreprise. Les données saisies dans l'application cliente devront se synchroniser avec le shop en ligne. Elle devra permettre le traitement des commandes réalisées à travers le site. Les marchandises commandées seront saisies par lots et dispatchées dans les différents canaux de vente. Il sera ainsi possible de localiser le flux des marchandises.

Technologies

Le client web est réalisé avec GWT (Google Web Toolkit). Le serveur est un service REST développé avec Spring. Les données sont stockées sur une base de données PostgreSQL. Toutes ces technologies utilisent le langage Java. Le travail se déroule avec un environnement d'intégration continue (git,docker et tests).

Résultat

L'application permet de créer différents modèles de produits. Le site met en avant des modèles de produits et aucun élément d'inventaire n'est traité sur celui-ci. Les produits sont ajoutés à l'inventaire par lots. Il est ainsi possible de localiser les déplacements par lots. L'architecture de l'application client-serveur est standardisée (MVP-MVC) afin de rendre l'implémentation des nouvelles fonctionnalités aisée.

Candidat

Currat Benjamin Date: _____ Signature: _____

Responsable

Barazzutti Raphaël Date: _____ Signature: _____

Kieed.ch

Jaquier Arnaud Date: _____ Signature: _____

Table des Matières

1. Introduction

1.1 Contexte

L'entreprise Kieed [1], active dans la création de T-shirts cherche un étudiant pour la réalisation d'une plateforme afin de centraliser la gestion du stock et la localisation des articles.

1.1.1 Problématique

La vente de T-Shirts par l'entreprise Kieed s'effectue à plusieurs endroits distincts. Ces endroits sont les suivants :

— Le magasin de Fribourg
— Le site internet
— Les marchés locaux
— les dépôts-ventes

1.1.2 Situation actuelle

L'inventaire de leur stock devenant difficile à suivre. Ils se retrouvent avec des inventaires qui ne correspondent pas à la marchandise commandée chez leur fournisseur (Tous endroits confondus). Dans le processus actuel, seuls les bons de commandes passés aux fournisseurs permettent de connaître les entrées et sorties de marchandises.

Il est dès lors difficile pour eux de cibler les raisons de différences entre les produits commandés chez le fournisseur, pour lesquels ils obtiennent un bon de livraison et les marchandises en dépôts-vente,

1. KIEED. *Magasin de T-Shirts en ligne*. URL : `http://kieed.ch`.

volées et vendues. Impossible dès lors de ressortir une quelconque statistique qui permettrait de cibler les améliorations à apporter au processus de vente.

1.2 Histoire de l'entreprise

Arnaud Jaquier (Enseignant au Cycle d'orientation de Jolimont à Fribourg), a décidé de créer une marque de T-shirts avec deux de ses anciens élèves. Créée en juillet 2012, l'entreprise Kieed a fait sa place dans le milieu de la création de T-shirts dans le canton de Fribourg. Des idées novatrices, sympathiques et populaires ont permis à Kieed de prendre son envol.

1.2.1 Le nom

Le nom de la marque Kieed est le résultat de la contraction des dernières lettres de chaque prénom des créateurs.

Anou**K** Mehd**I** Sidoin**E** Stéphan**E** Arnau**D**

1.2.2 La création

N'ayant pas le matériel nécessaire à l'élaboration des T-shirts, l'entreprise Kieed produit ses T-shirts chez "Couleursplus SA" à Villars-sur-Glâne. Suite à l'arrêt de l'entreprise "Graphein" en 2015. Dirigée par Jacques Favre, "Couleurplus SA" leur permet de poursuivre leur histoire en produisant des T-shirts de qualité. Les T-shirts sont exclusivement des marques "American Apparel" et "Clique".

2. Cahier des charges fonctionnel

Comme l'analyse l'a démontré lors de la pré-étude, le cahier des charges fonctionnel est conséquent. Nous avons donc convenu avec le mandant de fixer l'objectif au plus proche des fonctionnalités des epics [1] 1, 2, 3, 4 et 6. Celles-ci ont un peu évolué lors de l'analyse concrète de chaque partie. [2]

2.1 Fonctionnalités retenues

Les fonctionnalités visées pour l'exécution du travail, sont basées sur les epics à partir desquels on a défini les user stories [3]. Cependant dans les grandes lignes, ce qui a été décidé avec le mandant était dans un premier temps d'arriver à une implémentation CRUD [4] de chaque partie. Comme le produit continuera d'évoluer vers un système de gestion complet cela est apparu comme étant réalisable dans le temps imparti.

1. Création, modification et suppression des attributs de produits.
2. Création, modification et suppression des modèles de produits.
3. Création, modification et suppression des fournisseurs.
4. Création, modification et suppression des entrepôts.
5. Création, modification et suppression des lots de produits.

1. Une Epic (ou "épopée") est une définition de travaux qui peuvent être scindés en plusieurs petits chapitres.
2. ATLASSIAN. *Gestion de développement et de projets*. URL : `www.atlassian.com`.
3. Petite histoire qui contient des tâches pour un projet Agile
4. Create, Read, Update and Delete (CRUD)

2.2 Epics retenus

En partant d'une "Loi de Pareto" [5] de 60/40 afin de ne pas vouloir réaliser l'irréalisable, les chapitres ont étés puisés dans les epics cités lors de la pré-étude.

2.2.1 Gestion des produits

Identifiant	Epic 1
Titre	Gestion des produits
Description	En tant qu'utilisateur je veux pouvoir gérer les produits en un seul et unique endroit
Tâches associées	1. L'utilisateur doit pouvoir ajouter, modifier et supprimer des produits. 2. L'utilisateur doit pouvoir ajouter, modifier et supprimer des modèles de produits. 3. L'utilisateur doit pouvoir ajouter, modifier et supprimer des attributs de produits. 4. L'utilisateur doit pouvoir lier des attributs de produits à un modèle de produit. 5. L'utilisateur doit pouvoir ajouter, modifier et annuler des lots de produits. 6. L'utilisateur doit pouvoir lier un lot de produits à un entrepôt. 7. L'utilisateur doit pouvoir modifier l'entrepôt d'un lot en gardant l'historique. 8. L'utilisateur doit pouvoir vendre un produit. 9. L'utilisateur doit pouvoir reprendre un produit. 10. L'utilisateur doit pouvoir lier un modèle de produit à un fournisseur.
Importance	11
Complexité	7
Auteur	Benjamin Currat

5. Un phénomène empirique constaté dans certains domaines : environ 80 % des effets sont le produit de 20 % des causes.

2.2.2 Gestion des tiers

Identifiant	Epic 2
Titre	Gestion des tiers
Description	En tant qu'utilisateur je dois pouvoir gérer les différents acteurs tels que les utilisateurs, les fournisseurs et les entrepôts.
Tâches associées	1. L'utilisateur doit pouvoir ajouter ou modifier un fournisseur . 2. L'utilisateur doit pouvoir définir un fournisseur comme obsolète. 3. L'utilisateur doit pouvoir ajouter ou modifier un entrepôt . 4. L'utilisateur doit pouvoir définir un entrepôt comme obsolète. 5. L'utilisateur doit pouvoir ajouter ou modifier une adresse. 6. L'utilisateur doit pouvoir définir une adresse comme obsolète. 7. l'utilisateur doit pouvoir lier une adresse à l'entrepôt. 8. l'utilisateur doit pouvoir lier une adresse au fournisseur.
Importance	11
Complexité	3
Auteur	Benjamin Currat

2.2.3 Gestion des accès

Identifiant	Epic 3
Titre	Gestion des accès
Description	En tant qu'administrateur je dois pouvoir gérer les accès des différents utilisateurs de l'application
Tâches associées	1. L'administrateur doit pouvoir ajouter, modifier et supprimer des utilisateurs 2. L'administrateur doit pouvoir affecter des rôles aux utilisateurs 3. L'administrateur doit pouvoir affecter des vues de l'application aux rôles utilisateurs prédéfinis
Importance	5
Complexité	3
Auteur	Benjamin Currat

2.2.4 Gestion des commandes

Identifiant	Epic 4
Titre	En tant qu'administrateur, je dois pouvoir gérer les commandes faites à travers le magasin en ligne
Description	En tant qu'administrateur je dois pouvoir gérer les accès des différents utilisateurs de l'application
Tâches associées	1. L'administrateur doit pouvoir synchroniser les commandes effectuées par le magasin en ligne. 2. L'administrateur doit pouvoir traiter les commandes effectuées par le magasin en ligne.
Importance	5
Complexité	7
Auteur	Benjamin Currat

2.2.5 Gestion de l'inventaire

Identifiant	Epic 6
Titre	
Description	En tant qu'utilisateur, je dois pouvoir consulter l'état des stocks, rechercher un produit, un lot de produits et localiser leur position
Tâches associées	1. L'utilisateur doit pouvoir rechercher les produits en se basant sur plusieurs critères. 2. L'utilisateur doit pouvoir consulter les stocks et la répartition des produits.
Importance	11
Complexité	3
Auteur	Benjamin Currat

3. Cahier des charges technique

Le cahier des charges technique permet de mettre en évidence les technologies qui seront utilisées durant la réalisation du travail de Bachelor.

3.1 Technologies

Les technologies employées se divisent en trois grands groupes qui sont les langages, les outils et la persistance des données.

3.1.1 Langages

Le langage utilisé principalement est le langage Java. Cependant les frameworks [1] utilisés nécessitent aussi de bonnes connaissances web. GWT [2] permet d'écrire en Java mais le code compilé est du HTML, CSS et Javascript. Il est donc primordial de pouvoir comprendre ces langages lorsque le résultat après compilation n'est pas en phase avec le résultat escompté.

3.1.2 Outils

L'IDE "IntelliJ" est utilisé. Les frameworks Spring [3], GWT et GWTP [4] sont utilisés.

1. Désigne un ensemble cohérent de composants logiciels structurels.
2. Inc. GOOGLE. *Google Web Toolkit*. URL : `http://www.gwtproject.org/`.
3. SPRING. *Spring Framework MVC Java*. URL : `https://spring.io/guides`.
4. ARCBEES. *Framework MVP pour GWT*. [Google Web Toolkit Presenter]. URL : `https://dev.arcbees.com/gwtp/`.

3.1.2.1 IDE

Il propose une bonne implémentation des différentes technologies utilisées pour ce travail.

3.1.2.2 Gestion de dépendances

La gestion de dépendances est réalisée avec le gestionnaire de projet Maven [5] qui permet d'automatiser les tâches liées au projet comme la compilation, l'exécution ou encore la création des exécutables. Rédigée dans des fichiers POM, celle-ci permet en outre la gestion de dépendance de librairies tierces et automatise leur téléchargement.

Les plugins suivants sont utilisés :

— "maven-jar-plugin" de la fondation Apache
— "maven-compiler-plugin" de la fondation Apache
— "gwt-maven-plugin" de T. Broyer
— "spring-boot-dependencies" de Spring
— "spring-boot-maven-plugin" de Spring

3.1.2.3 CI/CD

L'environnement d'intégration continue et de déploiement continu utilisé est le logiciel GitLab [6] qui met aussi à disposition un environnement complet pour le versioning, les tests et les déploiements. Les piplines permettent de livrer en continu des versions "staging" [7] pour le client. Il est aussi possible par cet outil de déployer directement sur un serveur de production.

3.1.2.4 frameworks

— Spring
— Spring Boot
— GWT
— GWTP

5. SONATYPE. *Apache Maven*. [Dependancies management]. URL : `https://maven.apache.org/`.
6. GitLab INC. *The only single product for the complete DevOps lifecycle*. URL : `https://about.gitlab.com/`.
7. Environnement de simulation

3.1.2.5 Librairie

Pour une liste exacte de toutes les librairies utilisées, veuillez vous référer aux fichiers POM du projet. Voici cependant une liste non-exhaustive des librairies les plus connues utilisées.

— GIN [8]
— GUICE [9]
— Guava [10]
— GWT Material Design [11]
— Jackson [12]
— Common IO [13]

3.1.3 Persistance de données

La base de données PostgreSQL [14] sera utilisée pour sa robustesse et sa scalabilité. En soit l'utilisation de l'api JPA [15] avec Spring ne nécessite pas forcément de connaissances approfondies du langage.

3.2 Architecture

L'application est développée sur les bases d'une architecture trois tiers. C'est à dire, une base de données, une application et un client qui peuvent être hébergés séparément. La partie métier(serveur) est un service REST [16] qui fait le pont entre les demandes du client et la persistance des données. Celle-ci se base sur Spring et utilise une architecture MVC [17]. Le client lui est basé sur une architecture MVP [18] qui est un peu différente, mais reste tout aussi fiable.

3.3 Client

L'application est développée en langage Java basé sur le framework GWT utilisant une librairie GWT Material Design.

8. Google INC. *GIN*. URL : https://github.com/nishtahir/google-gin.
9. Google INC. *Documentation Guice*. URL : https://github.com/google/guice/wiki/Motivation.
10. GitHub COMMUNITY. *Google Guava*. URL : https://github.com/google/guava.
11. Google INC. *Librairie basée sur le Material Design*. URL : https://github.com/GwtMaterialDesign/gwt-material.
12. FASTERXML. *Jackson*. URL : https://github.com/FasterXML/jackson-docs.
13. APACHE. *Common IO*. URL : https://commons.apache.org/proper/commons-io/.
14. PostgreSQL est un système de gestion de base de données relationnelle et objet.
15. Inc. JBOSS. *Persistence des bases de données relationnelles*. URL : http://hibernate.org/orm/.
16. Representational State Transfer (REST)
17. Modèle Vue Contrôleur (MVC)
18. Modèle Vue Présenteur (MVP)

3.4 Serveur

La partie serveur est développée en langage Java avec le framework Spring. L'application serveur mettra à disposition une API [19] de type REST avec les les fonctions CRUD attendues pour chaque partie.

19. Application programming interface (API)

4. Réalisation

Pour les détails concernant la réalisation, je me base sur les sprints et me focalise sur les éléments qui méritent un commentaire ou une explication plus détaillée. J'estime qu'un niveau technique minimum est exigé afin de comprendre les sections suivantes. Seuls les éléments d'analyses qui ont nécessité une réflexion particulière sont décrits par esprit de synthèse.

4.1 Sprint 1 - du 9 au 22 juillet

Configuration de l'environnement CI/CD. Je pensais avoir le temps de mettre cela en place avant de commencer le travail. Ce ne fut pas le cas. C'est donc devenu la première tâche lors du premier Sprint, avec comme objectif de dégrossir le modèle de données et de commencer à attaquer la partie serveur REST. A l'issue de ce sprint, une coquille vide est montrée au mandant.

4.1.1 Intégration continue CI/CD

Installation de l'application GitLab en docker sur un serveur "bac à sable"

4.1.1.1 Problème rencontré

Difficultés à faire fonctionner le gitlab-runner en mode "Docker in Docker" [1].

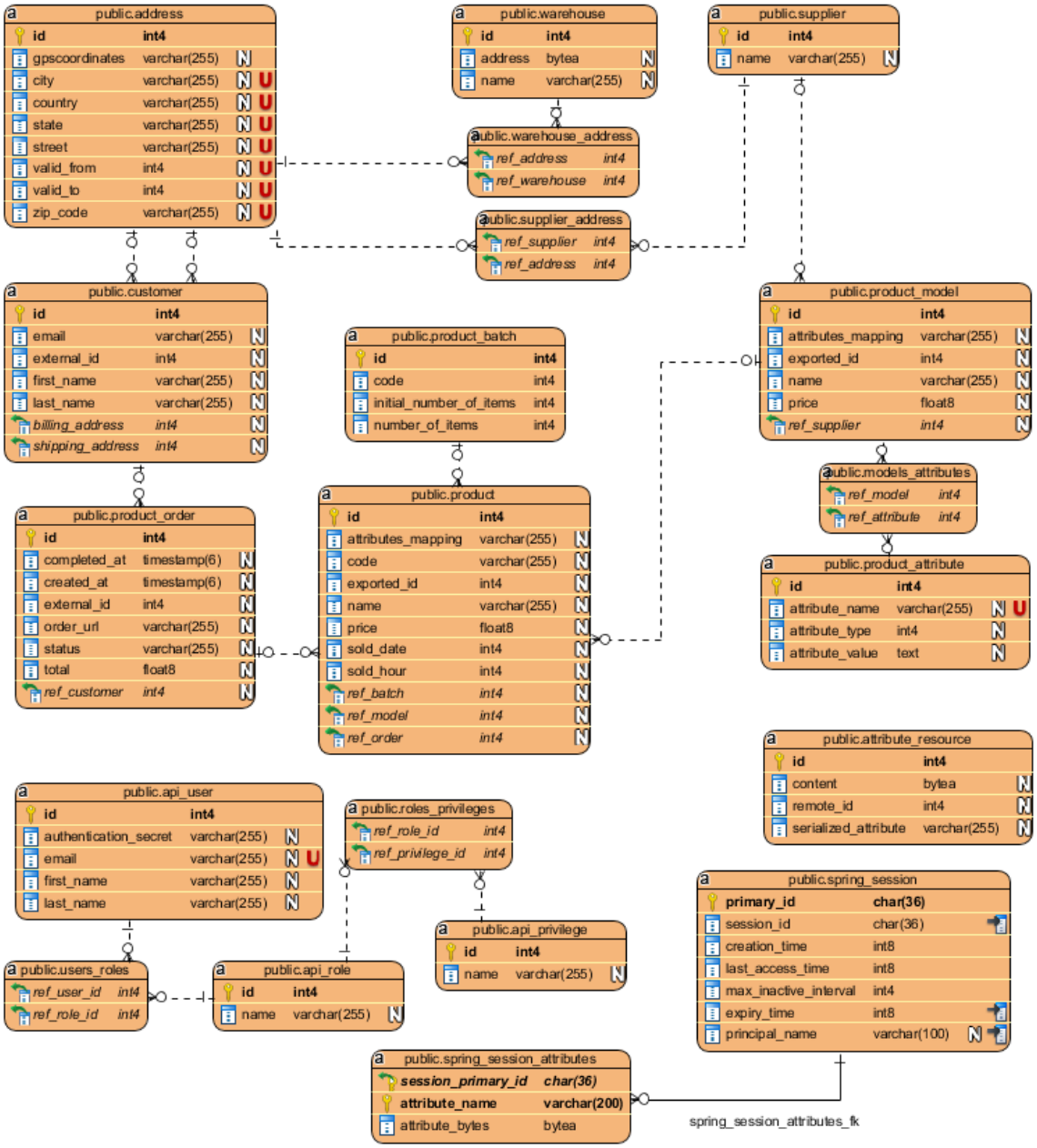
1. GITLAB. *Building Docker images with GitLab CI/CD*. URL : `https://docs.gitlab.com/ee/ci/docker/using_docker_build.html`.

4.1.2 Solution

Installation du gitlab-runner sur le serveur directement et ensuite le lier à l'instance Docker.

4.1.3 Analyse du modèle logique

Suite à l'analyse faite lors de la pré-étude, des éléments se sont ajoutés pour la partie utilisateur. L'analyse préliminaire ne prenait pas en compte la manière dont "Spring" gère les utilisateurs. Des détails concernant la session s'y sont aussi ajoutés pour les mêmes raisons.



4.1.3.1 Problème rencontré

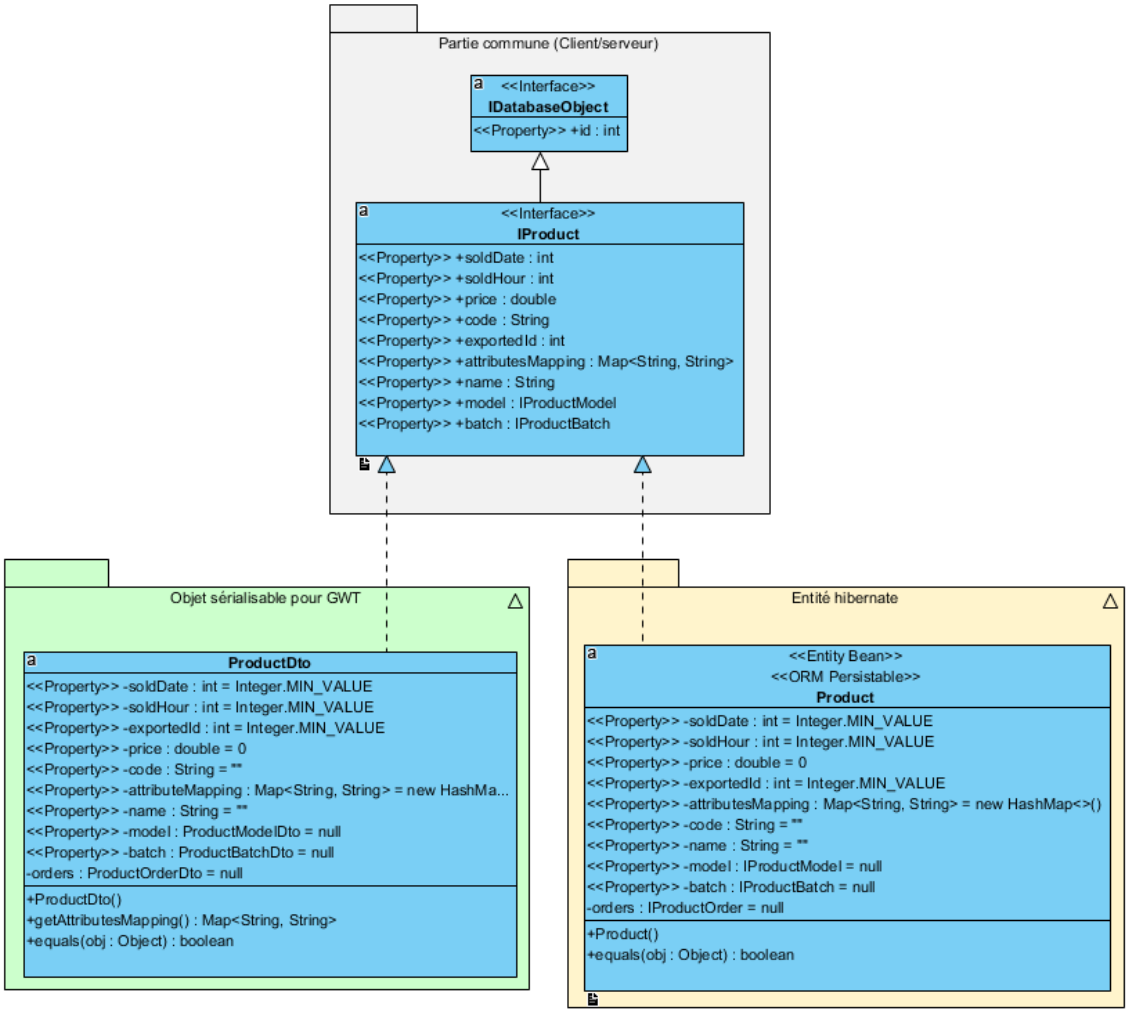
D'un point de vue relationnel, je me suis beaucoup posé de questions quant à la modélisation des attributs de produits. En voulant rester générique sur ceux-ci, je constate, en effet, que le modèle se complique fortement et je perds la philosophie de pouvoir intégrer plus tard d'autres types de commerce en ligne.

4.1.3.2 Solution

J'ai finalement décidé de garder une partie sérialisée des attributs afin de garder un modèle de données le plus à plat possible. Cela sera bénéfique, par exemple, lors de la recherche des termes contenus dans les attributs de produits.

4.1.4 Définition des entités communes

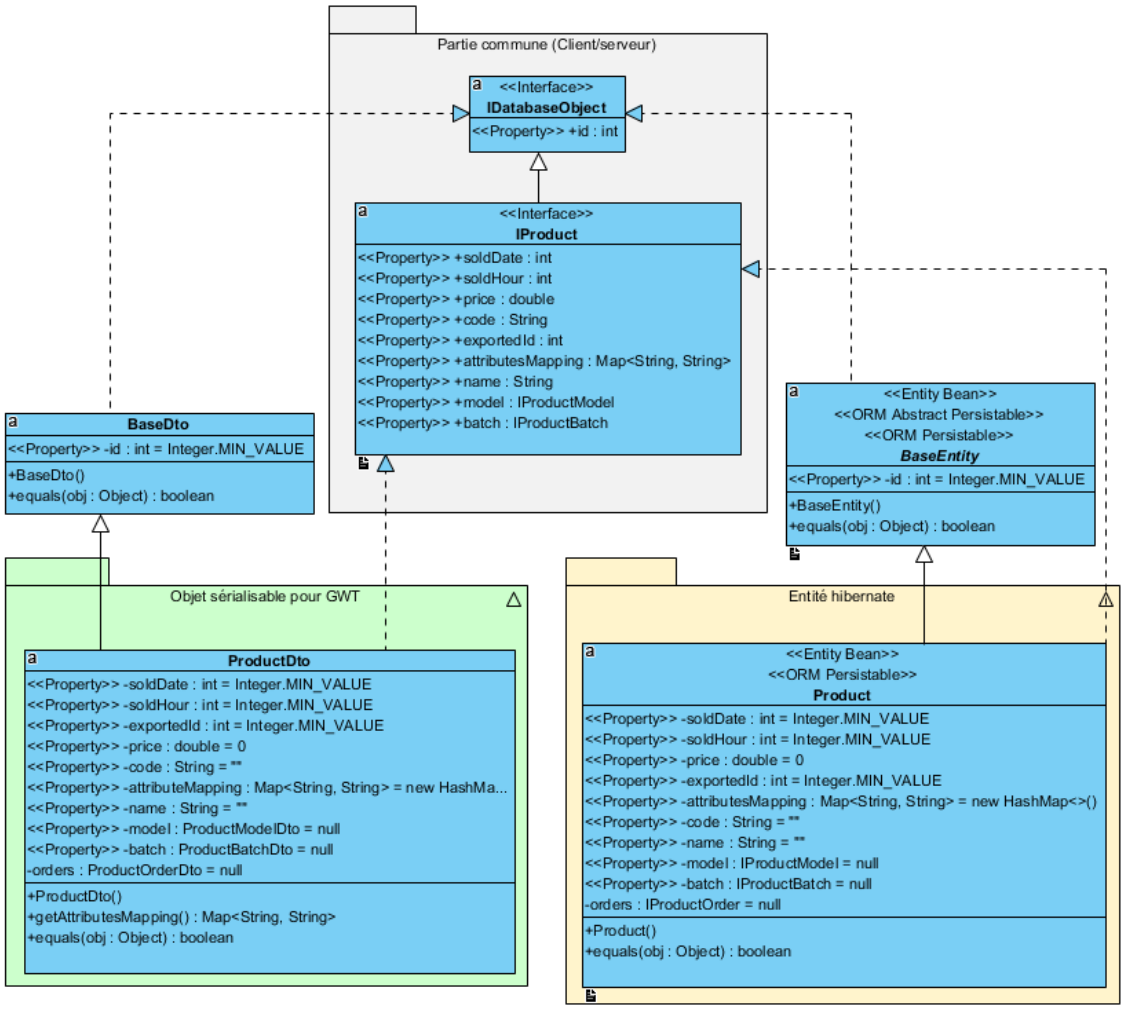
Une fois le schéma de base de données décidé, il faut créer les modèles "JPA/Hibernate" et leur version sérialisable pour GWT. J'ai opté en premier lieu de créer des interfaces communes qui définissent une structure minimale des modèles. Ce qui donne, par exemple pour les entités produits, un modèle comme ci-dessous.



Pour le package de couleur verte, on peut voir un modèle DTO [2] d'une entité produit qui partage une interface avec le modèle Java Persistence API (JPA)/Hibernate en orange.

2. Data Transfer Object (DTO)

Comme chaque objet utile est gardé en base de données, il est donc possible de généraliser un objet "BaseDto" et "BaseEntity" qui ne possède qu'un identifiant. Ce qui donne, en restant dans un exemple simple, le modèle de classe suivant.



4.1.4.1 Problème rencontré

Je ne suis pas entièrement satisfait du résultat, je trouve rébarbatif de devoir créer autant de modèles similaires. La contrainte étant imposée par le compilateur GWT pour la partie cliente car il ne supporte pas les annotations de type JPA/Hibernate.

4.1.4.2 Solution

Aucune solution n'a été découverte jusque là. Mais je continue d'y réfléchir. Une jolie proposition pourrait émaner de la librairie Jackson si elle intégrait les annotations "GwtCompatible" comme proposées par la librairie Guava de Google.

4.2 Sprint 2 - du 23 au 5 août

Les objectifs fixés à l'issue de ce sprint furent de pouvoir piloter par le navigateur ou le logiciel Postman le service REST ainsi que de récupérer les articles du site actuel afin de partir d'une base réaliste.

4.2.1 Création de la partie serveur REST

La création de la partie serveur, de manière générale, est la mise en place avec Spring de la mécanique MVC. Ce qui inclue la création des modèles JPA/Hibernate correspondants à l'analyse des données, la création de chaque contrôleur et son repository respectif.

4.2.2 Communication avec WooCommerce

Lors de la pré-étude une API semblait pouvoir faire l'affaire. Cependant dans le cas étudié, celle-ci n'a pas pu être utilisée. En effet l'API ne possède qu'une seule méthode d'authentification (OAuth) [3] qui s'avère ne pas fonctionner lorsque le site est en mode SSL [4]. Dans un tel cas de figure l'authentification Basic uniquement est autorisée par WooCommerce. J'ai, de ce fait, dû apprendre et utiliser les RestTemplate [5] proposés par le framework Spring.

Le shop en ligne du client a été cloné dans un espace de test à l'aide de docker. Ceci n'a pas été signalé lors de la pré-étude, cependant il apparait sage de ne pas perturber un environnement de production durant le développement.

4.2.2.1 Problème rencontré

La librairie préconisée ne gère pas le cas SSL [6]. Aucune autre librairie ne permet de le faire. Ce qui incombe une grosse perte de temps sur la planification initiale.

4.2.2.2 Solution

La solution a été d'écrire ma propre librairie pour communiquer avec WooCommerce. J'ai écrit des modèles d'entités WooCommerce que je convertis ensuite pour le modèle de données.

4.3 Sprint 3 - du 6 au 19 août

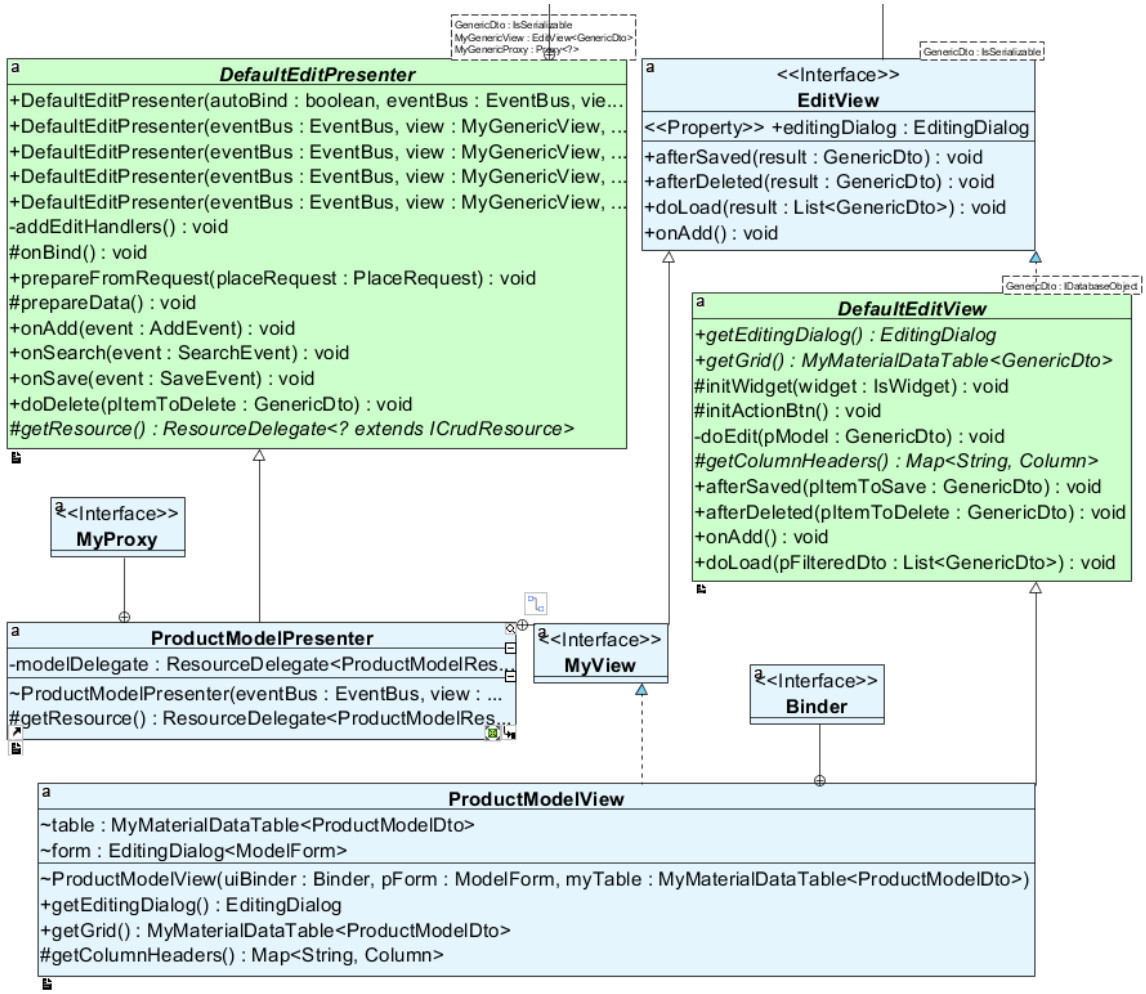
L'objectif fixé à l'issue de ce sprint est d'avoir une première vue CRUD fonctionnelle. Afin de pouvoir lors du sprint suivant clore le chapitre CRUD des vues attendues.

3. OAuth est un protocole sécurisé libre d'accès par délégation.
4. WOOCOMERCE. *Consuming a RESTful Web Service*. URL : `http://woocommerce.github.io/woocommerce-rest-api-docs/`.
5. SPRING. *Consuming a RESTful Web Service*. URL : `https://spring.io/guides/gs/consuming-rest/`.
6. Secure Socket Layer (SSL)

4.3.1 Définition d'un modèle de base MVP pour le client

MVP est un patron de conception bien connu, il existe de base avec GWT. Cependant l'implémentation proposée par GWT, est un peu légère et ne permet pas forcément de partir sur une base solide et l'architecture inhérente de cette proposition d'intégration me semble un peu faible à l'instar de GWTP.

GWTP permet une intégration relativement rapide de ce paradigme. J'ai donc opté pour cette solution. Dans mon cas, je constate que la plupart de mes vues nécessitent des actions assez similaires. J'ai donc décidé de faire une couche intermédiaire avec le fonctionnement initiale de GWTP. J'ai défini un "DefaultEditPresenter" et un "DefaultEditView" qui permettent de gérer des actions de bases utiles à la manipulation de mes modèles. Le résultat de cette analyse me pousse vers un modèle comme ci-dessous :



Cette sous-couche me permet de créer rapidement une vue dite d'édition pour un modèle quelconque.

4.3.2 Sérialisation des modèles de données

La sérialisation des modèles est un chapitre important avec le framework GWT. Effectivement, étant donné qu'il émule une machine virtuelle Java, cela implique que certaines options du langage ne sont pas implicitement présentes.

■ **Exemple 4.3.1** l'introspection de classes est remplacée par des générateurs. Il n'est donc pas possible d'instancier un objet en utilisant le "Class.forName('packageName')" communément utilisé dans le langage Java. ■

4.3.2.1 Les générateurs

Les générateurs permettent de créer des objets à la compilation. L'idée générale est de donner au générateur une grammaire ou une définition de "comment écrire l'objet que l'on désire" puis, celui-ci est matérialisé lors de la compilation.

Dans la configuration de GWT, nous pouvons ensuite donner un générateur de classe pour un type particulier. Il est donc possible par la configuration de donner une implémentation X ou Y d'un objet. L'objet est instancié par un "GWT.create(.class);" à la place du "new" habituel.

4.3.2.2 Jackson mapper pour GWT

Jackson mapper est une autre solution pour créer ces générateurs. Cette solution permet de ne pas devoir écrire un générateur spécifique à chaque cas d'utilisation. Il suffit pour cela de créer une interface qui étend l'objet "ObjectMapper<Type>" et de l'instancier avec "GWT.create(Mapper.class)".

Cette méthode d'instanciation activera les capacités du "Jackson mapper" pour générer des sérialiseurs et désérialiseurs de chacun des membres de l'objet initial. Cette solution semble bien moins fastidieuse que de devoir définir un générateur pour chaque modèle qui sera transformé en JSON [7]

4.3.2.3 GWTP - ResourceDelegate

GWTP met à disposition un délégué de ressource qui permet de simplifier l'utilisation d'un service REST avec GWT [8]. Celui-ci fonctionne sur les bases du "Jackson mapper" vu précédemment. J'ai opté pour celui-ci. Il permet de ne définir qu'une interface en donnant les informations de Path du service REST ainsi que le type de ressource retourné par la requête. Cela simplifie fortement la proposition faite par GWT qui utilise un "RequestFactory" [9].

7. Java Script Object Notation (JSON)

8. ARCBEES. *Resource Delegates*. URL : `https://dev.arcbees.com/gwtp/communication/resource-delegates.html`.

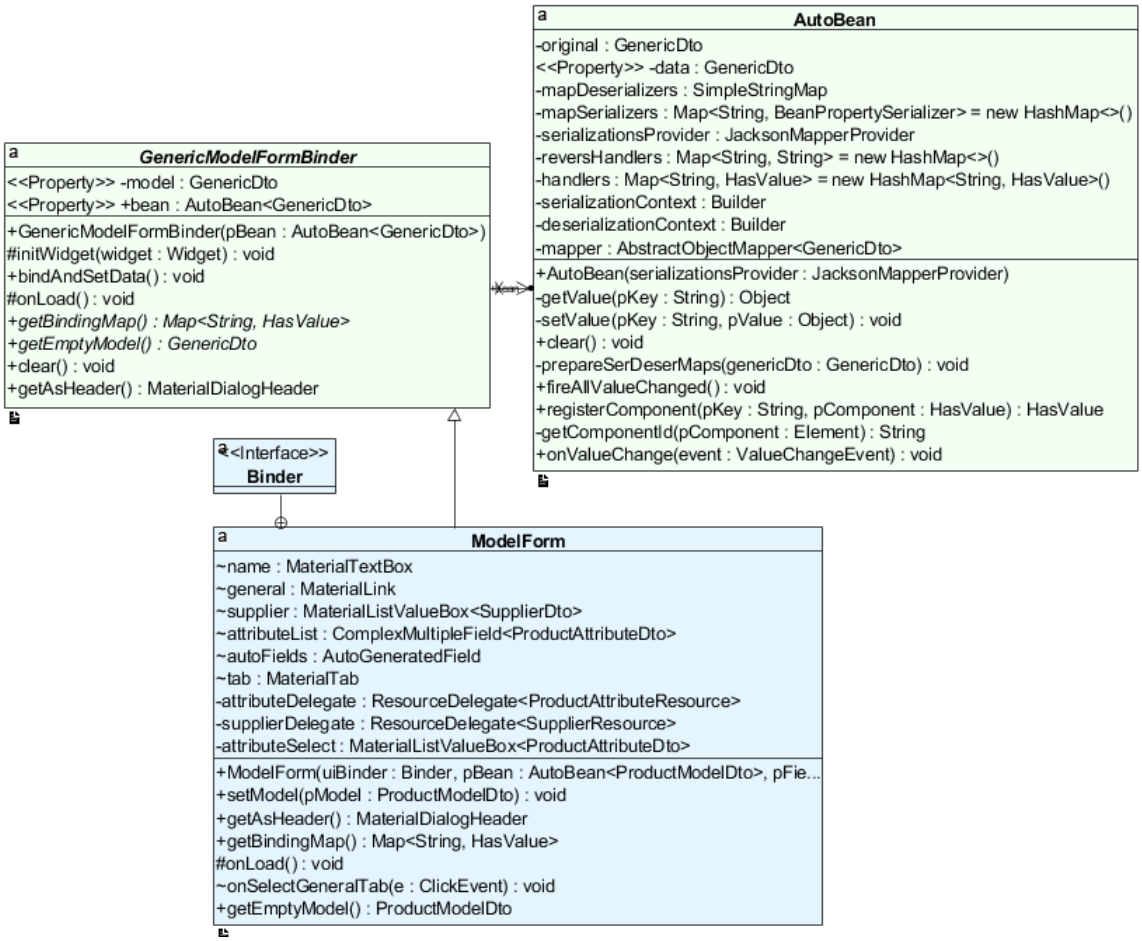
9. Google INC. *Coding with RequestFactory*. URL : `http://www.gwtproject.org/doc/latest/DevGuideRequestFactory.html`.

4.3.3 La liaison des données et des composantes du formulaire

Pour boucler la boucle, il ne me manque plus qu'une chose à généraliser afin de pouvoir efficacement créer n'importe quel type de vue d'édition en minimisant la répétition de code. Je pars du principe que chaque formulaire d'édition est représenté par un dialogue. Ce dialogue comporte les champs relatifs aux modèle de données. J'ai décidé de définir un objet "GenericModelFormBinder" dont le rôle est simplement de faire le lien entre les composantes du dialogue et leur modèle de données.

Chaque formulaire d'édition étend cette classe abstraite et fournit les composants formulaires par injection de dépendances. Ce qui permet de n'avoir que deux composants dans chacune des vues d'édition. Un composant tableau qui liste les entités et un composant formulaire qui contient les champs et le modèle de données en adéquation avec ceux-ci.

Chaque champ mis à jour doit mettre à jour le modèle de données, c'est pourquoi, à la place d'ajouter un handler par champ, j'ai décidé de créer un objet "AutoBean". Cet objet a deux rôles principaux. Le premier est de réagir sur un événement du champ lorsque celui-ci change. Le second est de mettre à jour le modèle lorsque celui-ci a changé. Ce qui mène vers un modèle de formulaire comme ci-dessous :



4.3.3.1 Problèmes rencontrés

Le premier problème rencontré a été la maîtrise des injections de dépendances du côté GWTP. En effet cette injection se fait grâce à la librairie GIN et celle-ci peut s'avérer quelque peu récalcitrante.

Le second problème, causé par les composants formulaire de GWT Material Design s'est avéré être un défi non attendu. Il n'est pas possible de différencier une instance d'une autre donc, les événements captés ne permettent pas de distinguer la source avec certitude. Le troisième problème rencontré fut l'absence d'introspection pour la classe "AutoBean".

4.3.3.2 Solutions

Le première solution est apparue par la documentation sur le sujet, comprendre le mécanisme de fonctionnement de GIN puis le maîtriser. Celui-ci fonctionne sur le principe de fournisseur et injecteur. Ce qui veut dire qu'un élément non-connu de GIN doit pouvoir trouver un fournisseur de celui-ci, soit une politique d'instanciation. GWTP utilisant des modules Google INjection (GIN), il est possible de mettre à disposition un package directement pour qu'il puisse déduire un fournisseur par lui-même. Il n'est cependant pas possible d'injecter sans fournisseur des structures de données génériques.

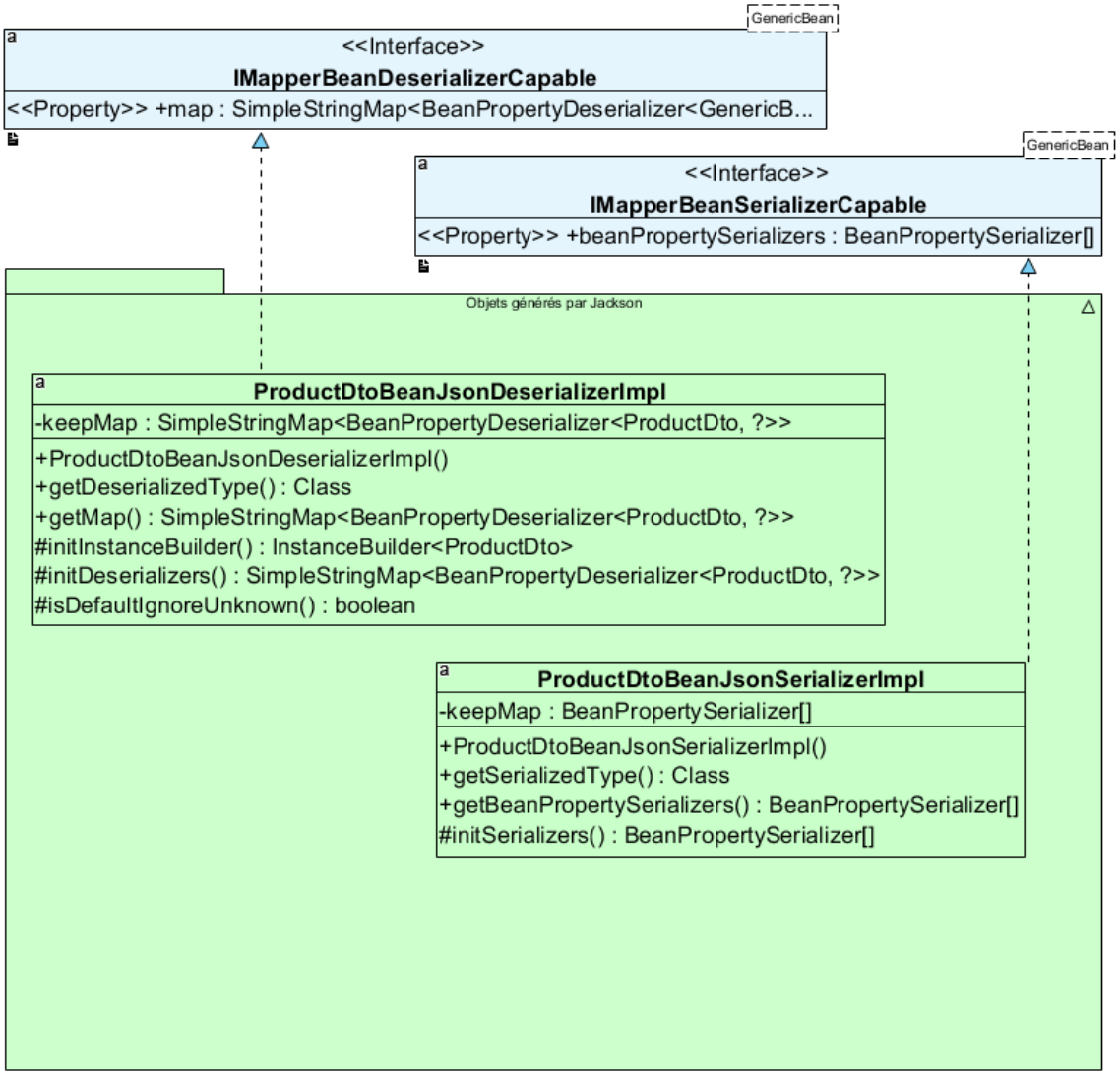
■ **Exemple 4.3.2** Un objet de type "Map<String,String>" devra obligatoirement avoir un fournisseur qui implémente l'interface "Provider" de GIN sans quoi il ne lui sera pas possible de l'injecter. ■

La seconde solution a été découverte grâce au debugger et résolue par une méthode d'enregistrement de composant dans la classe "AutoBean". Si le composant en question ne contient pas d'identifiant, il faut générer un identifiant unique pour celui-ci. Il a fallu faire très attention à ce que le composant soit bel et bien celui passé lors d'un événement de type "ValueChangeEvent".

La dernière solution a nécessité, à mon grand désarroi, une copie partielle du générateur de base proposé par GWTP afin de pouvoir récupérer les sérialiseurs et désérialiseurs générés par la librairie Jackson qui permettent d'activer les "setter" et "getter" appropriés par le nom des membres de classe. Dans la partie client, un package "rebind" s'est avéré nécessaire afin de donner aux objets générés la capacité de retourner une "Map" de sérialiseurs pour les "getters" et un tableau de désérialiseurs pour les "setters".

Je voulais, dans un premier temps, ne changer que les générateurs des classes "BeanJsonSerializer-Creator" et "BeanJsonDeserializerCreator" pour les rendre disponibles. Mon incompréhension fût de découvrir qu'il n'était pas possible de le faire aussi élégamment. Soit les membres de classes ou méthodes étaient privées, soit elles étaient protégées mais finales. J'ai dû reprendre l'architecture complète pour y ajouter mes propres interfaces qui mettent à disposition les sérialiseurs et désérialiseurs de chaque modèle généré à travers Jackson.

Ce qui me permet de générer leurs implémentations de cette manière.



On peut voir dans le package en vert les élément générés par Jackson.

4.4 Sprint 4 - du 20 au 2 septembre

A la base du projet je pensais pouvoir, à cette étape déjà, intégrer une vue. Le sprint précédent ayant donné plus de difficultés que prévu, nous avons convenu avec le mandant de repousser l'aspect synchronisation des données avec le site en ligne hors du périmètre du travail de Bachelor. Il parait à ce moment complètement utopique de pouvoir tenir ce défi avec les problèmes rencontrés jusqu'ici. En terme de gestion de projet, l'issue de ce sprint apportera un rythme de croisière pour la fin du projet.

4.4.1 Vue "Attributs de produits"

La vue des attributs de produits n'est pas spécialement complexe en soit. Quoique la matérialisation d'un point de vue formulaire d'un attribut générique n'est pas du même acabit. En effet, j'ai limité les possibilités de ceux-ci à des pseudo-types prédéfinis tels qu'une liste permettant la multi-sélection, une liste permettant la sélection-simple, un type booléen et un type de ressource image.

Chacun de ces pseudo-types est représenté d'une manière générique. Ainsi un attribut peut être référencé par un modèle de produit.

■ **Exemple 4.4.1** Pour les T-Shirts, si on définit un attribut "Taille" qui peut avoir plusieurs valeurs possibles telles que ("S", "M", "L", "XL"). Les valeurs des attributs représentent à ce moment les "possibles" de l'univers nommé "Taille". On pourrait d'ailleurs l'assimiler aux tuples [10] dans les langages fonctionnels. ■

4.4.1.1 Problème rencontré

D'un point de vue des formulaires, il faut pouvoir rester souple quant aux valeurs autorisées. On doit pouvoir ajouter n'importe quelles valeurs à l'univers que l'on désire peupler. Ce cas de figure n'est bien entendu pas applicable à tous les pseudo-types prédéfinis. Mais dans le cas des listes, que la sélection puisse être multiple ou pas, il est nécessaire de pouvoir ajouter ce que l'on désire.

4.4.1.2 Solution

Comme ce paradigme risque d'arriver souvent, je décide de créer un composant générique formé d'un ou plusieurs champs qui constituent une entité. Un bouton d'ajout, un bouton de suppression et un tableau pour la représentation des données sont les éléments minimaux requis. Cette composante de formulaire peut dans le cas de l'univers "Taille" avoir un champ simple que l'on appellera "Nom" et un tableau qui affiche toutes les possibilités ajoutées.

Dans un cas plus complexe, par exemple, une adresse se constituera de plusieurs champs qui représenteront une entité plus complexe.

10. WIKIPAEDIA. *Tuple*. URL : `https://en.wikipedia.org/wiki/Tuple`.

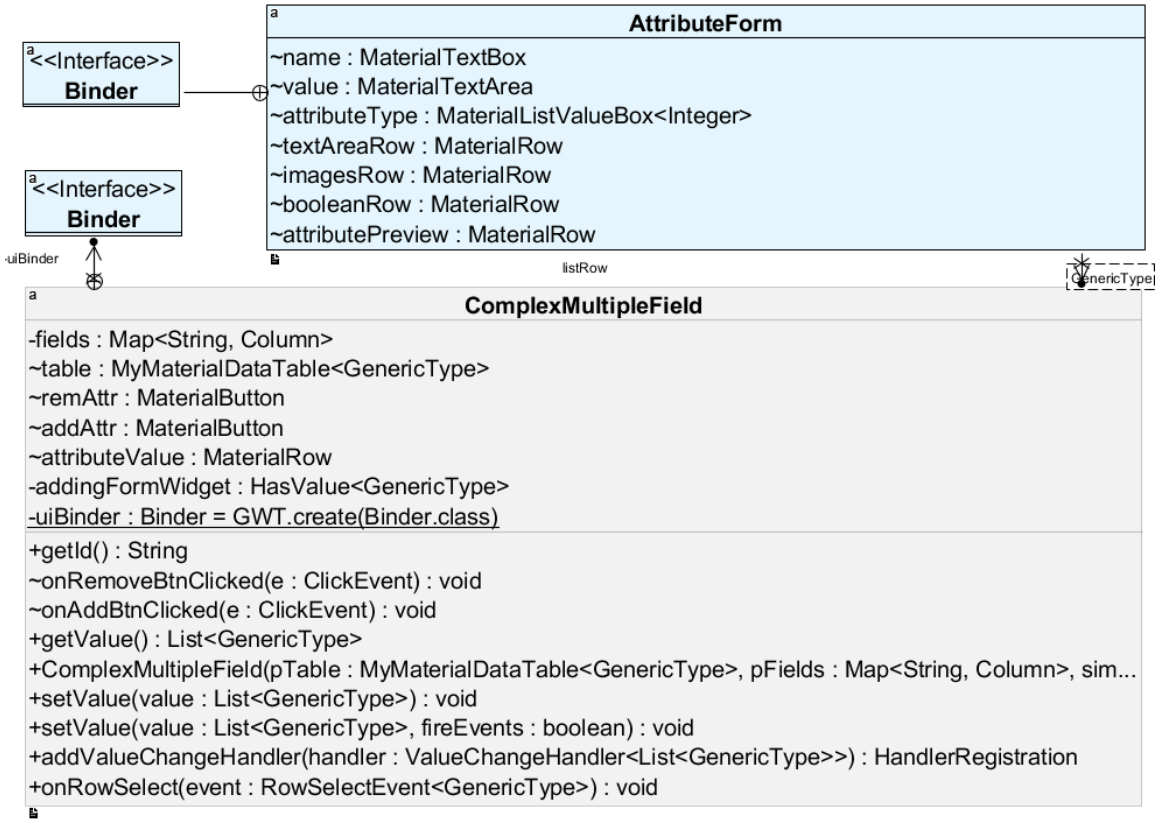
La composante devient dans ce cas un champ complexe que je nomme "ComplexMultipleField" avec un type générique.

```
public class ComplexMultipleField<GenericType> extends
        Composite implements HasValue<List<GenericType>>,
            RowSelectHandler<GenericType> {

    private final Map<String, Column> fields;
    ...

    @Inject
    public ComplexMultipleField(MyMaterialDataTable<GenericType> pTable,
        Map<String, Column> pFields, HasValue<GenericType> simpleInput) {
        ...
    }
}
```

Cet objet sera représenté et utilisé par un formulaire de la manière suivante :



Cette composante permet à ce moment de matérialiser de manière générique les cardianalités de type 0 à n, tout comme 1 à n.

4.4.2 Vue "Modèles de produits"

Les modèles de produits sont liés aux attributs par une cardinalité 1 à n. Tout comme les attributs, cette vue utilisera aussi un "ComplexMultipleField<GenericType>". Contrairement aux attributs, il ne regroupe pas toutes les possibilités de l'univers "Taille" pris en exemple précédemment. C'est pourquoi il détient aussi un champ de donnée "AttributeMapping" qui verbalise quelles entités de l'univers "Taille" sont disponibles pour lui.

■ **Exemple 4.4.2** le couple ("M", "XL") qui sont des membres de l'attribut lié mais réduit. ■

Cette méthode à l'avantage de minimiser les schémas relationnels et d'optimiser les performances lors d'une recherche.

4.4.2.1 Problème rencontré

Encore une fois le formulaire se complique. Effectivement, nous pouvons lier les attributs de produits à la configuration d'un nouveau modèle. Cependant, non seulement un modèle a un aspect de liaison aux attributs mais aussi un aspect de génération de champs dynamiques selon la sélection des attributs effectuée ainsi qu'une sélection des entités de l'univers des possibles.

4.4.2.2 Solution

Pour ce qui est de la génération de champ dynamique, j'ai créé un composant "AttributeFieldFactory" qui est en soit un patron de conception de type "Factory" qui permettra de constituer une représentation des attributs de produits de manière dynamique.

```
public class AttributeFieldFactory extends Composite implements
    HasValue<Map<String, String>> {

    ...

    private Map<String, String> attributeMap = new HashMap<>();

    private final ResourceDelegate<AttributeResource> resourceDelegate;

    @Inject
    public AttributeFieldFactory(Binder uiBinder,
        ResourceDelegate<AttributeResource> resourceDelegate) {
        this.resourceDelegate = resourceDelegate;
        ...
    }

    private Widget generateField(ProductAttributeDto p) {
        Widget w = new TextWidget();
        ((TextWidget) w).setText("Not implemented yet");
        switch (p.getAttributeType()) {
            case IProductAttributeTypes
                    .BOOLEAN:
                ...
                break;
            case IProductAttributeTypes.IMAGE_RESOURCE:
                ...
                break;
            case IProductAttributeTypes.TEXT_AREA:
                ...
                break;
            case IProductAttributeTypes.SINGLE_SELECT:
            case IProductAttributeTypes.MULTI_SELECT:
                ...

        }

        return w;
    }

}
```

La valeur du champ sera une simple "Map<String, String>" que représente le champ de données "AttributeMapping" cité précédemment. Dans le couple clé/valeur de la map, la clé reste simple mais la valeur, elle, est souvent plus complexe. La chaîne de caractère stockée en base de données de cette valeur est un JSON.

Grâce au type de champ, ce composant est capable d'interpréter les valeurs contenues dans "AttributeMapping". Par exemple lorsque c'est une image, les valeurs sont des identifiants d'une ressource de type image (idéalement préchargée) ou lorsque c'est un type de sélection multiple il crée un nuage de boutons "on/off" activé ou non s'il est présent dans le mapping.

4.5 Sprint 5 - du 3 au 17 septembre

A l'issue de ce sprint, il serait bien d'avoir une version de démonstration des principaux CRUD sélectionnés.

4.5.1 Vue "Fournisseurs"

La vue fournisseurs n'est pas une vue très compliquée, outre le fait d'avoir une liaison vers des adresses. Cette liaison est une cardinalité "n à n", mais délimitée par une date de validité. Ce qui permettrait de retracer les adresses fournisseurs dans le passé.

Dans ce cas de figure il n'y a pas un grand intérêt à laisser cette possibilité. Cependant c'est une fonctionnalité dont j'aurai besoin plus tard dans le but de traquer les déplacements de marchandises. Si le principe fonctionne pour un cas aussi simple que le changement d'adresse d'un fournisseur, il n'y a aucune raison que cela ne fonctionne pas pour un déplacement de marchandises.

4.5.1.1 Complexité rencontrée

La seule complexité rencontrée a été le fait de devoir utiliser le composant "ComplexMultipleField". Comme cela a été réfléchi auparavant, il n'y a pas nécessité de plus ample analyse. Il a été possible d'utiliser le composant tel quel. Il a cependant été nécessaire de construire un composant "AddressField" pour faire fonctionner le "ComplexMultipleField" comme prévu à la base. Ce composant utilise l'API de Google afin de permettre une saisie rapide d'une adresse.

4.5.1.2 Observation

Au moment d'utiliser ce composant complexe, je commence à avoir envie de m'abstraire de toute construction de champ concret. J'ai fortement envie de généraliser une fabrique de champs et de garder les composants en base de données. Comme une sorte de descripteur de champ. Je pourrais ensuite lier ces champs à mes vues et utiliser l'objet "AutoBean" pour automatiser la mise à jour des données. L'idée me séduit, je la garde pour une amélioration future.

4.5.2 Vue "Entrepôts"

La vue des entrepôts est pour ainsi dire une copie conforme de la vue "Fournisseurs", les commentaires ne sont dans ce cas pas nécessaires.

5. Synthèse du projet

Malgré la pré-étude et le fait d'avoir conscience de la taille des epics pour la réalisation de ce projet, je suis resté d'un optimisme certain. Ce qui en découle, est un résultat bien connu dans les projets informatiques, l'estimation temporelle et les envies ne sont pas entièrement atteintes. Je suis malgré tout satisfait du déroulement du travail. Le sujet est passionnant, le mandant souple et un peu rêveur ce qui à pour effet sur le temps de ne pas se faire ressentir.

5.1 Point de vue du produit

Concernant le produit, je pense que c'est une bonne ébauche et non seulement pour ce cas de figure. Je pense réutiliser une partie des concepts pour mon prochain mandat.

5.2 Améliorations possibles

La plupart des améliorations ont été citées dans le chapitre de la réalisation. Je préfère arriver à un produit utilisable avant de refactoriser les concepts. Il reste en effet encore beaucoup de chapitres à aborder. Comme la portabilité sur les appareils mobiles et la possibilité d'utilisation en mode hors-connexion pour ne citer que les deux prochaines grosses pièces.

5.3 Pérennité du projet

J'ai l'impression qu'une fois ce projet arrivé à terme, sa durée de vie reste assez grande. Les concepts métiers étants bien séparés, rien n'empêche une implémentation avec un client différent que GWT.

Il continuera au-delà du travail de Bachelor, ce qui est à mon sens une preuve de satisfaction de la part du client.

5.4 Expérience

Grâce à ce projet j'ai pu explorer les limites de mes connaissances et les repousser en même temps que le projet avançait. La méthodologie Agile que je voulais découvrir sur le terrain contient des concepts très intéressants. Je suis cependant persuadé que ceux-ci ne sont pas adaptés à une petite équipe de développement.

Le principe d'inclure le client à chaque étape du projet permet d'avancer vers ce que le client désire réellement. Cette collaboration est pour moi un point essentiel qui permet de minimiser les dommages collatéraux induits par un manque de communication.

6. Dossier de gestion

6.1 Définition du contenu des sprints

Les sprints sont définis en collaboration avec le mandant et en adéquation avec l'avance du sprint précédent. Cette méthode permet d'avoir un regard constant sur l'objectif visé.

6.2 Rendez-vous

Les rendez-vous pris avec le mandant ou le conseiller sont inscrits dans le tableau ci-dessous. Les remarques servent de rappel du sujet traité.

Date	Participant	Remarques
10.07.2018	Arnaud Jaquier	Entretien pour validation du plan du premier Sprint
18.07.2018	Arnaud Jaquier	Demande d'ajout de fonctionnalité (Tél.)
13.07.2018	Raphaël Barazzutti	Point CI/CD
23.07.2018	Arnaud Jaquier	Validation du sprint 1 et définition du Sprint 2
06.08.2018	Arnaud Jaquier	Validation du sprint 2 et définition du Sprint 3
13.09.2018	Raphaël Barazzutti	Point sur les objectifs et problématiques techniques
19.08.2018	Arnaud Jaquier	Validation du sprint 3 et définition du Sprint 4
04.09.2018	Arnaud Jaquier	Validation du sprint 4 et définition du Sprint 5
20.09.2018	Raphaël Barazzutti	Point de situation (Tél.)
21.09.2018	Arnaud Jaquier	Présentation de la première release

6.3 Journal de travail

Le journal de travail consiste en une vision des user stories et tâches sélectionnées pour chaque Sprint.

6.3.1 Sprint 1 - du 9 au 22 juillet

6.3.1.1 Configuration de l'environnement de travail

Identifiant	Tâche 1
Titre	Environnement
Description	Mise en place des outils de développement
Tâches	1. création d'une instance docker avec l'image GitLab. 2. Installation et configuration de l'IDE IntelliJ. 3. Définition de la hiérarchie des fichiers POM pour Maven.
Auteur	Benjamin Currat

6.3.1.2 Modélisation des données

Identifiant	Tâche 2
Titre	Modélisation
Description	Je dois avoir une idée générale des modèles de données à traiter.
Tâches	1. Création d'un schéma de base de données. 2. Création d'un schéma de classe correspondant. 3. Création des interfaces communes au schéma.
Auteur	Benjamin Currat

6.3.2 Sprint 2 - du 23 au 5 août

6.3.2.1 MVC côté serveur

Identifiant	Tâche 3
Titre	Création de l'architecture de l'API
Description	Je désire implémenter le patron de conception MVC.
Tâches	1. Configuration de base de Spring. 2. Configuration d'un minimum de sécurité. 3. Création des contrôleurs. 4. Création des repositories. 5. Implémentation d'une authentification par session sans état.
Auteur	Benjamin Currat

Identifiant	Tâche 4
Titre	Création des modèles persistants
Description	Je désire pouvoir utiliser des entités JPA/Hibernate sans me soucier de la persistance.
Tâches	1. Création des schémas réfléchis lors de l'analyse. 2. Implémentation du contrat commun client/serveur.
Auteur	Benjamin Currat

6.3.2.2 Clonage du site marchand

Identifiant	Tâche 5
Titre	Clonage du site en production
Description	Afin de piloter le site, il faut un environnement de tests proche de la réalité.
Tâches	1. Créer une entrée DNS pour rediriger sur le serveur de développement. 2. Préparer une image docker avec un Wordpress. 3. Faire un dump de la base de donnée de production. 4. Copier la configuration du site Wordpress. 5. Monter l'instance docker avec un Let's Encrypt pour la communication SSL.
Auteur	Benjamin Currat

6.3.2.3 Import données WooCommerce

Identifiant	Tâche 6
Titre	Connexion à l'API WooCommerce avec un RestTemplate
Description	Arriver à s'authentifier sur l'application
Tâches	1. Créer une clé d'accès sur le clone du site. 2. Se documenter sur l'utilisation des RestTemplate. 3. Ecrire un service qui se connecte au site. 4. Ecrire un test de connexion.
Auteur	Benjamin Currat

Identifiant	Tâche 7
Titre	Créer des entités Woocommerce pour l'import
Description	Observer les réponses JSON du service et en créer des objets Java
Tâches	1. Lire la documentation de l'API. 2. Observer les réponses du service WooCommerce. 3. Créer les entités nécessaires à l'import.
Auteur	Benjamin Currat

Identifiant	User Story 1
Titre	Ecrire une méthode d'import de produit
Description	En tant qu'utilisateur, je désire pouvoir importer la base de produits présent sur le site.
Tâches	1. Créer une méthode d'import pour le service 2. Création d'une mécanique d'import le plus générique possible
Auteur	Benjamin Currat

6.3.3 Sprint 3 - du 6 au 19 août

6.3.3.1 MVP côté client

Identifiant	Tâche 8
Titre	Généraliser une "Vue d'édition"
Description	Je désire pouvoir utiliser une vue d'édition sans avoir à réécrire plusieurs fois le même concept.
Tâches	1. Analyse de l'implémentation GWTP (ViewImpl). 2. Modéliser le principe à implémenter. 3. Création de l'objet générique "DefaultEditView". 4. Implémenter les événements de base que doit gérer la vue.
Auteur	Benjamin Currat

Identifiant	Tâche 9
Titre	Généraliser les événements d'une "Vue d'édition"
Description	Je désire pouvoir utiliser les événements d'une vue d'édition sans avoir à réécrire plusieurs fois le même concept.
Tâches	1. Analyse de l'implémentation GWTP (Presenter). 2. Modéliser le principe à implémenter. 3. Création de l'objet générique "DefaultEditPresenter". 4. Implémenter les événements de base que doit gérer la vue.
Auteur	Benjamin Currat

6.3.3.2 Resource Delegates

Identifiant	Tâche 10
Titre	Créer les délégués pour les requêtes client
Description	Création des délégués correspondants aux services REST mis à disposition.
Tâches	1. Définition d'une interface CRUD commune. 2. Créer le délégué adresse. 3. Créer le délégué attribut de produits. 4. Créer le délégué fournisseur. 5. Créer le délégué produit. 6. Créer le délégué entrepôt.
Auteur	Benjamin Currat

6.3.4 Sprint 4 - du 20 au 2 septembre

6.3.4.1 Vue des attributs de produits

Identifiant	User Story 2
Titre	Créer la vue des attributs de produits.
Description	En tant qu'utilisateur, je désire pouvoir gérer les ajouts, suppression ou édition des attributs de produits.
Tâches	1. Créer la vue. 2. Créer la présentation. 3. Tester.
Auteur	Benjamin Currat

6.3.4.2 Vue des modèles de produits

Identifiant	User Story 3
Titre	Créer la vue des modèles de produits.
Description	En tant qu'utilisateur, je désire pouvoir gérer l'ajout, suppression ou édition des modèles de produits.
Tâches	1. Créer la vue. 2. Créer la présentation. 3. Tester.
Auteur	Benjamin Currat

6.3.5 Sprint 5 - du 3 au 17 septembre

6.3.5.1 Vue "Fournisseurs"

Identifiant	User Story 4
Titre	Créer la vue des fournisseurs.
Description	En tant qu'utilisateur, je désire pouvoir gérer l'ajout, suppression ou édition des fournisseurs.
Tâches	1. Créer la vue. 2. Créer la présentation. 3. Tester.
Auteur	Benjamin Currat

6.3.5.2 Vue "Entrepôts"

Identifiant	User Story 5
Titre	Créer la vue des entrepôts.
Description	En tant qu'utilisateur, je désire pouvoir gérer les ajouts, suppression ou édition des entrepôts.
Tâches	1. Créer la vue. 2. Créer la présentation. 3. Tester.
Auteur	Benjamin Currat

7. Annexes

— Bibliographie
— Glossaire
— Acronymes
— Pré-étude
— Affiche

HAUTE ÉCOLE
D'INGÉNIERIE ET DE GESTION
DU CANTON DE VAUD
www.heig-vd.ch

Centre Formation de Base

Travail de Bachelor

Modèle d'authentification

Authentification

Le soussigné, Benjamin Currat, atteste par la présente avoir réalisé seul ce travail et n'avoir utilisé aucune autre source que celles expressément mentionnées, si ce n'est les connaissances acquises durant ses études et son expérience acquise dans une activité professionnelle.

Fribourg, le .....

B. Currat

Bibliographie

APACHE. *Common IO*. URL : https://commons.apache.org/proper/commons-io/ (cf. page 9).

ARCBEES. *Framework MVP pour GWT*. [Google Web Toolkit Presenter]. URL : https://dev.arcbees.com/gwtp/ (cf. page 7).

— *Resource Delegates*. URL : https://dev.arcbees.com/gwtp/communication/resource-delegates.html (cf. page 18).

ATLASSIAN. *Gestion de développement et de projets*. URL : www.atlassian.com (cf. page 3).

COMMUNITY, GitHub. *Google Guava*. URL : https://github.com/google/guava (cf. page 9).

FASTERXML. *Jackson*. URL : https://github.com/FasterXML/jackson-docs (cf. page 9).

GITLAB. *Building Docker images with GitLab CI/CD*. URL : https://docs.gitlab.com/ee/ci/docker/using_docker_build.html (cf. page 11).

GOOGLE, Inc. *Google Web Toolkit*. URL : http://www.gwtproject.org/ (cf. page 7).

INC., GitLab. *The only single product for the complete DevOps lifecycle*. URL : https://about.gitlab.com/ (cf. page 8).

INC., Google. *Coding with RequestFactory*. URL : http://www.gwtproject.org/doc/latest/DevGuideRequestFactory.html (cf. page 18).

— *Documentation Guice*. URL : https://github.com/google/guice/wiki/Motivation (cf. page 9).

— *GIN*. URL : https://github.com/nishtahir/google-gin (cf. page 9).

— *Librairie basée sur le Material Design*. URL : https://github.com/GwtMaterialDesign/gwt-material (cf. page 9).

JBOSS, Inc. *Persistence des bases de données relationnelles*. URL : http://hibernate.org/orm/ (cf. page 9).

KIEED. *Magasin de T-Shirts en ligne*. URL : http://kieed.ch (cf. page 1).

SONATYPE. *Apache Maven*. [Dependancies management]. URL : https://maven.apache.org/ (cf. page 8).

SPRING. *Consuming a RESTful Web Service*. URL : https://spring.io/guides/gs/consuming-rest/ (cf. page 16).

— *Spring Framework MVC Java*. URL : https://spring.io/guides (cf. page 7).

WIKIPAEDIA. *Tuple*. URL : https://en.wikipedia.org/wiki/Tuple (cf. page 22).

WOOCOMERCE. *Consuming a RESTful Web Service*. URL : http://woocommerce.github.io/woocommerce-rest-api-docs/ (cf. page 16).

Glossaire

Agile Méthodologie de travail itérative. 28

Common IO Librairie de gestion d'entrées/sorties. 9

epic Une Epic (ou "épopée") est une définition de travaux qui peuvent être scindés en plusieurs petits chapitres.. 3, 27

framework Désigne un ensemble cohérent de composants logiciels structurels.. ii, 7, 8, 9, 16

GitLab Solution de déploiement. 8, 11

gitlab-runner Application de GitLab qui permet de lancer des tâches après un commit. 11

Google Le géant du web. 15

Guava Librairie proposée par Google proposant des outils ou structures de données complémentaires. 9, 15

GWT Material Design Librairie de composants GWT basée sur la spécification Material Design de Google. 9, 20

Hibernate Hibernate est un framework open source gérant la persistance des objets en base de données relationnelle.. 13, 14, 15, 16, 30

IntelliJ IntelliJ IDEA est un IDE Java commercial développé par JetBrains.. 7

Jackson Librairie de haute performance pour le JSON. 9, 15, 20

Let's Encrypt Une autorité de certification.. 31

Loi de Pareto Un phénomène empirique constaté dans certains domaines : environ 80 % des effets sont le produit de 20 % des causes.. 3

Maven Outils de gestion de production de projet. 8, 30

OAuth OAuth est un protocole sécurisé libre d'accès par délégation.. 16

Path Nom commun pour définir un chemin en informatique. 18

pipeline Concept d'enchaînement d'actions pour automatiser des tests ou un déploiement.. 8

PostgreSQL PostgreSQL est un système de gestion de base de données relationnelle et objet.. 9

Postman API de simulation de requêtes web. 15

RestTemplate Fonctionalité de Spring pour la communication avec un service REST. 16, 31

Spring Framework qui permet de définir l'infrastructure d'une applicaltion. 7, 8, 9, 16

Spring Boot Utilitaire qui permet l'intégration de la librairie Spring de manière rapide. 8
staging Environnement de simulation. 8
user story Petite histoire qui contient des tâches pour un projet Agile. 3, 29
WooCommerce Plugin de magasin en ligne pour Wordpress. iii, 16, 31

Acronymes

API Application programming interface. 9
CI/CD Continious Integration and Continious Deployment. ii, 8, 11
CRUD Create, Read, Update and Delete. 3, 9, 16, 26
DTO Data Transfer Object. 14
GIN Google INjection. 9, 19, 20
GUICE Librairie d'injection de dépendances. 9
GWT Google Web Toolkit. 7, 8, 9, 13, 16, 17, 18, 27
GWTP Framework basé sur GWT qui propose un patron de type MVP. 7, 8, 16, 17, 19, 20
IDE Integrated Developpment Environnement. ii, 7
JPA Java Persistence API. 9, 13, 14, 15, 16, 30
JSON Java Script Object Notation. 18, 25, 31
MVC Modèle Vue Contrôleur. 9, 30
MVP Modèle Vue Présenteur. iii, 9, 16
POM Project Object Model. 8, 9, 30
REST Representational State Transfer. iii, 9, 11, 16, 18
SSL Secure Socket Layer. 16, 31