



Projet de Fin d'Etudes

Développement d'un générateur
d'applications web

Préparé par : **M. BADR KACIMI**

Sous la direction de : **Mme. Rajaa SAIDI** (INSEA)
M. SENTISSI Salmane (HIVEPROD)
M. ZOULAY Abderrahim (HIVEPROD)

Soutenu publiquement comme exigence partielle en vue de l'obtention du

Diplôme d'Ingénieur d'Etat

Filière : INFORMATQUE

Devant le jury composé de :

- **Mme. Rajaa SAIDI** (INSEA)
- **M. OUAZZANI TOUHAMI** (INSEA)
- **M. ZOULAY Abderrahim** (HIVEPROD)

Dédicace

A ma mère,

Tu m'as donné la vie, la tendresse et le courage pour réussir. Tout ce que je peux t'offrir ne pourra exprimer l'amour et la reconnaissance que je porte.

A mon père,

L'épaule solide, l'œil attentif compréhensif et la personne la plus digne de mon estime et de mon respect.

Aucune dédicace ne saurait exprimer mes sentiments, que Dieu te préserve et te procure santé et longue vie.

A mes chers frères, Faissal, Taha et Yassine pour leur appui et leur encouragement,

Bader

Remerciements

Je tiens tout d'abord à remercier Dieu le tout puissant et miséricordieux, qui m'a donné la force et la patience d'accomplir ce Modeste travail.

En second lieu, Je tiens à remercier (mon encadrante à l'INSEA) Mme. Saidi Rajaa, Mr. Selmane SENTISSI (CEO de HIVEPROD) et Abderahim Zoulay (mon encadrant à HIVEPROD) et Ayoub Bakour pour leurs précieux conseils et leur aide durant toute la période du travail.

Mes vifs remerciements vont également aux membres du jury pour l'intérêt qu'ils ont porté à mon recherche en acceptant d'examiner mon projet et de l'enrichir par leurs propositions.

Enfin, Je tiens également à remercier toutes les personnes qui ont participé de près ou de loin à la réalisation de ce travail.

Résumé

Ce mémoire constitue une synthèse de mon projet de fin d'études effectué au sein de la société HIVEPROD. Ce projet a pour objectif de mettre en place une architecture web générique permettant de générer des applications web.

Pour une meilleure conduite du projet, J'ai suivi la méthodologie SCRUM. Après avoir mis le projet dans son contexte, J'ai en premier lieu étudié les approches classiques pour le développement des applications web pour relever ses défaillances afin d'offrir un moyen de les générer sans avoir à répéter les mêmes routines.

J'ai enchaîné par l'étude des différents composants constituant notre solution et par la sélection des outils. La réalisation de cette solution a été faite avec la plateforme JAVA. Ensuite, j'ai illustré un exemple complet de génération d'une application web.

Le résultat final de ce travail est sous forme des couches génériques indépendantes pouvant être utilisées dans toute application web pour éviter la répétition du code.

Mots-clés: Meta data, JSON, JSON Schema, JAVA, JSON Editor, Template.

Abstract

This dissertation is a synthesis of my project of end of studies carried out within the company HIVEPROD. This project aims to set up a generic web architecture to generate web applications.

For better project management, I followed the SCRUM methodology jointly. After putting the project in context, I first studied the classic approaches to developing web applications to address its failures to provide a way to generate them without having to repeat the same routines.

I went through the study of the different components of our solution and the selection of tools. The realization of this solution was made with the JAVA platform. Then I illustrated a complete example of generating a web application.

The result of our work is on the one hand independent generic layers that can be used in any web application to avoid code repetition, and on the other hand, the analysis of the possibility of integration of solutions based on the artificial intelligence to further minimize human intervention in the development and génération of web applications.

Table des matières

Dédicace	3
Remerciements	4
Résumé	5
Abstract	6
Liste des tableaux	8
Liste des figures	8
Liste des abréviations	11
Introduction générale	12
Chapitre 1 : Contexte général du projet	13
1. Organisme d'accueil	14
1.1 Présentation	14
1.2 Domaines d'activités	14
1.3 Organigramme et portefeuille de produits	14
2. Définition de projet	15
2.1 Problématique et motivation	16
2.2 Objectif de projet	16
3. Conduite de projet	17
3.1 Présentation de Scrum	17
3.2 Utilisation Scrum dans notre contexte	18
Conclusion	21
Chapitre 2 : Génération back-end et Front-end	22
1. Analyse de l'existence	23
2. Spécification fonctionnel de Projet	23
3. Génération coté client –Front End	24
3.1 Description	24
3.2 Approche classique	26
3.3 Approche adoptée	26
3.3.1 Description	26
3.3.2 Scénario	27
4. Génération coté serveur –Back End	28
4.1 Définition	28
4.2 Approche adoptée	29
4.2.1 Pattern MVC	29
4.2.2 Sécurité des web services-REST	31
5. Schémas technique globale	33
Conclusion	34
Chapitre 3 : Technologie et outils	35
1. Architecture de générateur	36
2. Grands choix techniques	37
Conclusion	40
Chapitre 4 : Réalisation	41
Template Flex	42
1. Coté client –Front End	42
1.1 Page d'authentification-JSON Editor	43
1.2 Création de menu	43

1.3 Création de la table.....	44
1.4 Création de formulaire.....	45
2. Coté client –Back End.....	46
2.1 Schéma de POJO	46
2.2 Saisie des attributs.....	46
2.3 Génération de POJO.....	47
3. Application web généré : Gestion de produit.....	51
3.1 Page authentification.....	51
3.2 Page d'accueil.....	52
3.3 Ajout/suppression d'un produit.....	52
Conclusion et perspectives	54
Bibliographie et webographie	55
Annexe	56

Liste des tableaux

Tableau 1 : Temps réel de la suite Fibonacci.....	19
Tableau 2 : Description textuelle du cas d'utilisation « Cote client Front-End».....	24
Tableau 3 : Description textuelle du cas d'utilisation « Coté serveur Back-End».....	24

Liste des figures

Figure 1 : Domaine d'activités	14
Figure 2 : Organigramme de l'organigramme	15
Figure 3 : L'architecture dirigée par les modèles	17
Figure 4 : Le processus Scrum	17
Figure 5: Sprints.....	20
Figure 6 : user story <<User Management>>	20
Figure 7 : Burndown Chart	21
Figure 8 : use case << Générateur d'application web>>	23
Figure 9 : Un Template	24
Figure 10 : composante d'un Template	25
Figure 11 : Génération de JSON Schéma- Meta data.....	27
Figure 12 : Scénario global de génération de Front-end d'une application web.....	28
Figure 13 : Extrait de JSON Schéma (modèle de données de POJO)	30
Figure 14 : Diagramme de séquence <<Scenario de génération d'un POJO>>	30
Figure 15 : Approche du style d'architecture REST	31
Figure 16 : jeton JWT.....	31
Figure 17 : JWT Authentification	32
Figure 18 : Schéma technique du générateur	33
Figure 19 : Schéma technique de l'application web générée	34
Figure 20 : Template Flex	42
Figure 21 : Page d'authentification	42
Figure 22 : JSON Editor – page accueil	43
Figure 23 : Menu gauche(SideNavDropDown).....	43
Figure 24 : SubMenu	44
Figure 25 : saisit les entêtes du tableau	44
Figure 26 : Buttons send ()	45
Figure 27 : Composante Forms	45
Figure 28 : Création de POJO.....	46
Figure 29 : Champs de Schéma POJO	46
Figure 30 : Attribut libelle.....	46
Figure 31 : Attribut prix	47
Figure 32 : Attribut quantité	47
Figure 33 : Génération POJO ().....	48
Figure 34 : REST schéma.....	48
Figure 35 : Saisie Méthode de REST-1	49

Figure 36 : Saisie Méthodes REST-2	49
Figure 37 : Génération de REST Controller.....	50
Figure 38 : Message de confirmation.....	50
Figure 39 : Client TS –REST Client.....	51
Figure 40 : Page d’authentification d’application web générée	51
Figure 41 : Gestion produit - menu gauche.....	52
Figure 42 : Ajouter un produit.....	52
Figure 43 : Formulaire pour ajouter et modifier	53
Figure 44 : Input HTML	56
Figure 45 : HTML généré	57
Figure 46 : JSON généré	57
Figure 47 : Composant généré	57
Figure 48 : Controller JSON Editor GenPOJO	58
Figure 49 : Valeurs JSON d'un POJO	58
Figure 50 : POJO généré	59
Figure 51: Swagger configuration de sécurité.....	59
Figure 52 : Authorization server.....	60
Figure 53 : Swagger authentication	60

Liste d'abréviations

Abréviation	Désignation
AWS	Amazon Web Services
API	Application Programming Interface
CSS	Cascading Style Sheets
DOM	Document Object Model
IHM	Interface Homme Machine
HTML	HyperText Markup Language
IDE	Integrated Development Environment
JAVA EE	Java Enterprise Edition
MVC	Model View Controller
POJO	Plain Old Java Object
UML	Unified Modeling Language
XML	eXtensible Markup Language
JSON	Javascript Object Notation
JWT	JSON Web Token
OMG	Object Management GroupModel Driven Architecture (MDA)
REST	Representational state transfer
TS / JS	Typescript / JavaScript
WAR	Web application Archive

Introduction générale

Face à la concurrence acharnée dans le secteur des nouvelles technologies, les sociétés de services informatiques se voient dans l'obligation d'améliorer leur productivité et la qualité de leurs services ceci est réalisé en cherchant des atouts et des méthodes efficaces pour pallier à la multitude de problèmes liés à la mauvaise gestion des travaux générant des pertes de temps et des coûts souvent exorbitants.

Etant une jeune société de services en ingénierie informatique et dans le souci de satisfaire ses clients/développeurs et améliorer sa marge, HIVEPROD a démarré un projet d'automatisation de son processus de production.

Dans ce cadre, la société a confié à une équipe projet, dont je fais partie, la mise en place d'un «Générateur d'application Web» qui a pour objectif d'automatiser l'implémentation du besoin non-fonctionnel (META-Conception, Meta-Développement). Le présent rapport illustre les différentes phases de ce projet :

Le premier chapitre est consacré à une présentation de l'organisme d'accueil et à la description de la problématique et les objectifs du projet. Ainsi que la conduite du projet.

Le deuxième chapitre traite le processus de génération de Front-End et du Back-end d'une application web.

Le troisième chapitre est consacré à la définition des outils et technologie d'implémentations de ce projet.

Le quatrième chapitre est consacré à la réalisation d'une application web générée complètement à base de Java EE et Angular 4.

En complément, les annexes présentent quelques des détails du générateur.

Chapitre *1* : Contexte générale du projet

Ce premier chapitre a pour objectif de décrire le contexte général du projet, qui consiste à mettre en place une architecture web générique, pour la société HIVEPROD.

Il sera composé de 3 sous parties :

- La présentation de l'organisme d'accueil HIVEPROD et son champ d'activité et le cadre dans lequel s'inscrit ce projet et les motivations qui ont poussé l'entreprise à le choisir.
- Les objectifs à atteindre durant le stage.
- La démarche suivie pour mener à bien ce travail.

1. Organisme d'accueil

1.1. Présentation



HIVEPROD Editeur de solution Web Collaboratif crée au Maroc depuis Janvier 2014, et opérant en France, depuis 2007.

Six ans après sa création, la société est épaulée par le ministère français de la recherche via l'incubateur PACA-Est de Sophia-Antipolis et labellisé, pour sa création Jetbee, par Oséo Innovation.

Aujourd'hui HIVEPROD, prend le relais au Maroc, basé sur les technologies WEB Java et destiné à créer la nouvelle génération de solutions de travail collaboratif et de gestion intelligente.

HIVEPROD est donc une startup marocaine essayant de bâtir sa propre identité dans la perspective de ce qui a été acquis par Anthil (En France). C'est une structure qui espère élargir son portefeuille client à autre que les clients Français demandeurs de prestation, et conquérir ainsi le marché marocain.

1.2. Domaines d'activités

HiveProd offre une large palette de prestations organisées autour des activités suivantes :

- Offrir des solutions de gestion et des outils de collaboration.
- Assistance à la maîtrise d'ouvrage

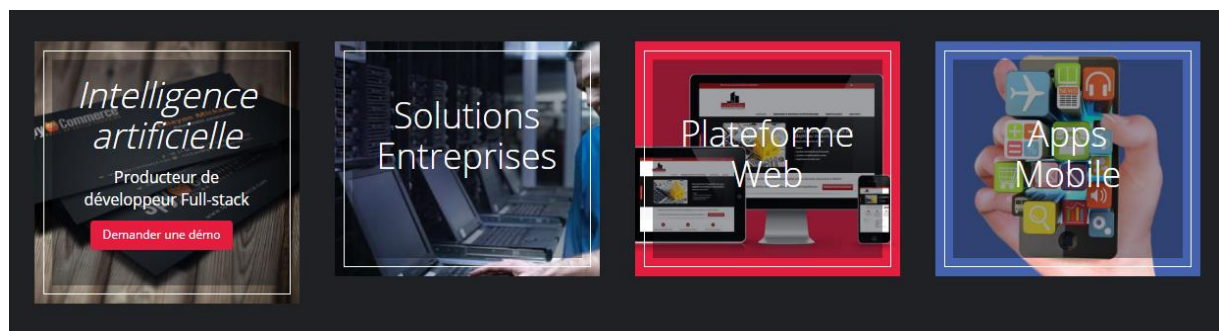


Figure 1 : Domaine d'activités

1.3. Organigramme et portefeuille de produits

HIVEPROD est, administrativement, un corps (cf. figure 2) indépendant de la société « mère » en France. C'est une startup en plein essor, cherchant à agrandir son équipe en reposant sur un planning stratégique de recrutement dépendant des besoins et variations du marché. Il était donc nécessaire, pour la boîte, d'avoir recours au recrutement de stagiaires en stage pré embauche afin d'en sélectionner les meilleurs éléments.

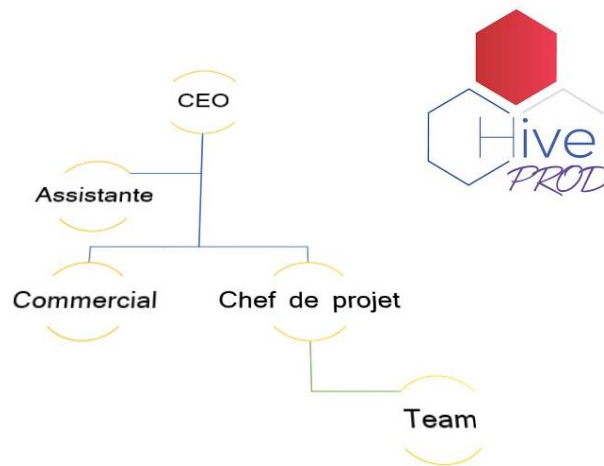


Figure 2 : Organigramme de l'organigramme

HIVEPROD dispose d'un Portefeuille de produits :



JETBEE est une plate-forme innovante de travail collaboratif permettant la création et la gestion simple d'intranets, extranets et portails d'entreprises au travers d'un simple navigateur et d'une interface web 2.0 la plus ergonomique du marché.

Ciblant les PME, collectivités et associations, JETBEE se décline en plusieurs packages adaptés à la taille de la structure du simple weboffice en ligne, à la plate-forme de création d'espaces collaboratifs, proposant des outils de bureautique, d'administration, agenda, widgets, webmail, CMS, CRM... en mode SaaS, accessibles depuis tout appareil communicant (PC, Mac, Linux, Smartphone, Netbook, Tablet...)

Il s'agit d'un système d'exploitation web permettant à un utilisateur d'accéder à ses données et à des applications distantes en utilisant un navigateur web.

Le principal avantage de ce bureau est d'être facile d'utilisation, disponible depuis n'importe quel ordinateur disposant d'une connexion internet.

Bien que certaines applications légères ne puissent rivaliser avec les mêmes applications bureau classique à cause des limites des navigateurs actuels néanmoins elles offrent l'avantage d'être toujours disponibles sans être installées ni mises à jour.

2. Description de projet

Ce stage a été réalisé dans le cadre du projet de mise en place d'une architecture web générique, qui facilite la génération des applications web en se basant sur les web services, dans le but d'améliorer la productivité et rendre la phase de développement plus rapide.

2.1. Problématique et motivation

Aujourd'hui, la réalisation des applications de gestion, demande un temps considérable de développement et de ressources, alors que le socle et les processus sont routiniers, le développeur se retrouve à répéter la même chose dans chaque application, ceci se situe dans différents endroits :

- **Côté serveur :** Dans les architectures conseillées aujourd'hui et utilisées par la majorité des frameworks, on utilise une architecture en couches pour bien séparer les traitements.

Une application de gestion typique contiendra une vue suscitant une action, un contrôleur, un service. Ce qui pose un problème en phase de maintenance, puisque un changement nécessitera de changer les entités, mais aussi d'intervenir dans toutes les couches.

- **Côté client :** Bien que la plupart des 'Server-side Framework' offre des composants serveurs qui sont rendus en HTML, une panoplie de Frameworks JavaScript ont vu le jour pour venir offrir des composants riches et gérer le parcours des éléments du DOM (interface pour examiner et modifier le contenu du navigateur web)².

Ces derniers sont de plus en plus convoités et quelques un, utilisés au sein de *HiveProd*, sont aussi adaptés pour être utilisés en MVC, ce qui pose le même problème au développeur, qui est amené à changer dans plusieurs parties lorsqu'il y'a une nouvelle fonctionnalité à ajouter, ou une modification à effectuer.

Pour pallier ce problème, on s'est de plus en plus appuyé sur la L'architecture dirigée par les modèles ou MDA (pour l'anglais model driven architecture) (cf. *figure 3*) pour essayer de maîtriser cette complexité, tant pour produire le logiciel (conception) que pour le valider (test). MDA propose de mettre à disposition des outils d'automatisation de génération de code à partir de logique métier.

« Vous ne faites plus partie du système. Vous êtes au-dessus du système, au sommet, au-delà. » Men In Black.

2.2. Objectifs du projet

Les objectifs de ce projet consistent à développer une solution permettant de générer et automatiser la création des applications de gestion tout en rendant générique le plus de parties possibles afin d'optimiser le code et améliorer la productivité des développeurs au sein de HIVEPROD, notamment en offrant les facilités suivantes :

- A partir du point d'entrée (Schémas : Modèles de données), l'application permettra de générer les vues finales et les mapper aux données et aux services.

- Insérer des briques génériques qui seront utilisées par toutes les applications, afin de résoudre le problème du codage des parties.

Laisser la possibilité d'extension de l'application avec du développement spécifique écrit à la main et la possibilité de faire du pré/ post traitement sur les briques dites génériques.

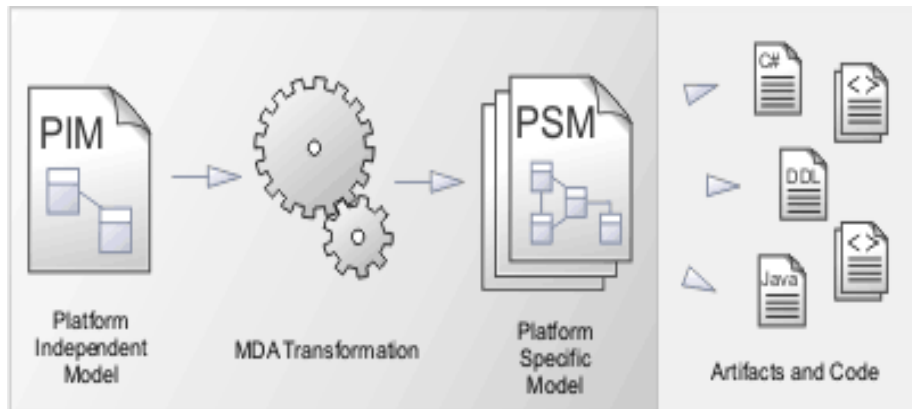


Figure 3 : L'architecture dirigée par les modèles

3. Conduite de projet

3.1. Présentation de SCRUM

Scrum est une méthode agile dédiée à la gestion de projet. « Une méthode agile est une approche itérative et incrémentale, qui est menée dans un esprit collaboratif avec juste ce qu'il faut de formalisme. Elle génère un produit de haute qualité tout en prenant en compte l'évolution des besoins des clients ». Scrum a pour objectif d'améliorer la productivité de son équipe.

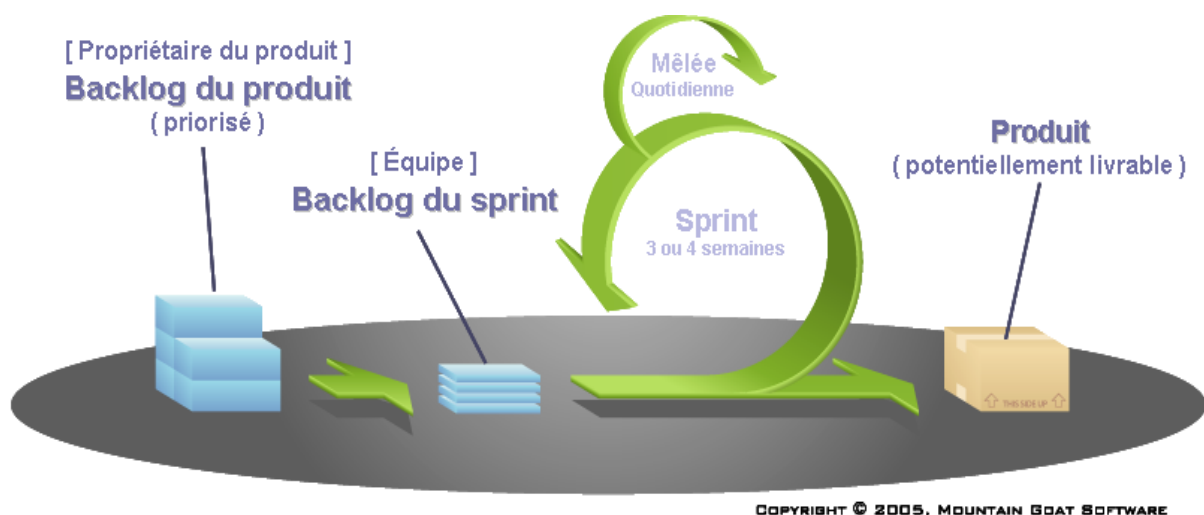


Figure 4 : Le processus Scrum

Le choix de Scrum comme une méthodologie de pilotage pour notre projet s'est basé sur les atouts de ce dernier. Il se résume comme suit :

- ✓ Plus de souplesse et de réactivité,
- ✓ La grande capacité d'adaptation au changement grâce à des itérations courtes,

- ✓ Et la chose plus importante, c'est que Scrum rassemble les deux cotés théorique et pratique et se rapproche beaucoup de la réalité.

3.2. Utilisation de Scrum dans notre contexte

❖ Equipe Hiveprod

L'équipe HiveProd, étant jeune et dynamique, se compose de 5 développeurs et un directeur technique.

- Pas de rôle bien déterminé : architecte, développeur, testeur.
- Tous les membres de l'équipe apportent leur savoir-faire pour accomplir les tâches.
- 6 personnes.

❖ Product Owner

- Expert métier, définit les spécifications fonctionnelles.
- Etablit la priorité des fonctionnalités à développer ou corriger.
- Valide les fonctionnalités développées.
- Joue le rôle du client.

❖ Product Backlog

Après avoir déterminé les besoins fonctionnels du projet, l'équipe s'est mis d'accord et constitué un backlog, regroupant les modules potentiellement livrables sous forme de fonctionnalités à réaliser. Ces grands modules seront par la suite décomposés sous forme de tâches pour constituer un backlog de sprint, beaucoup granulaire et atomique au niveau des tâches à réaliser durant ce dernier.

❖ Réunion de planification

Avant le début de chaque sprint, l'équipe se réunit et fixe le module à réaliser qui est le plus prioritaire. Suite à cela, on détermine les tâches à réaliser tout en estimant la durée de chacune avec la méthode du "Planning Poker".

❖ Poker Plannings

L'estimation des tâches est l'une des aspects les plus difficiles. Les membres ayant peur de surestimer ou sous-estimer une tâche, préfèrent se retenir au lieu de proposer leur estimation. De ce fait, le planning poker représente un avantage principal qui est de permettre à tous de s'exprimer librement. L'estimation serait meilleure parce que plusieurs personnes l'auront validée : des participants avec des niveaux d'expérience et d'expertise différents. De plus, cette technique favorise les échanges entre le responsable de produits et l'équipe de développement.

L'estimation des tâches est faite sur la base de la suite de Fibonacci :

Nombre de la suite	Durée réelle
1	2h
2	4h
3	6h
5	10h
8	16h

Tableau 1 : Temps réel de la suite Fibonacci

❖ Daily Scrum

Au début de chaque journée, une réunion de 10 min (2 minutes/membre) est programmé à 9h précise où chaque membre de l'équipe afin d'aborder les 3 points ci-dessous :
Ce qu'il a fait hier.

- ✓ Ce qu'il va faire aujourd'hui.
- ✓ Ce qui le bloque, afin de savoir si quelqu'un peut l'aider.
- ✓ Ceci permet une organisation et évaluation du travail fait jusqu'ici.

❖ Rétrospection de SPRINT

A la fin de chaque sprint, l'équipe se regroupe pour apporter une synthèse sur le sprint achevé, qu'est ce qui a bien marché, et qu'est ce qui devrait être amélioré par la suite. Comme ça, on commence à gagner en maturité et de sprints réussis.

❖ Outils

Pour ce qui est de l'outil utilisé pour suivre les tâches réalisées dans les sprints, on a opté pour l'outil JIRA.

Première raison d'utiliser JIRA Software quand il s'agit d'appliquer Framework Scrum : Il répond en effet à plusieurs exigences de ce Framework :

- Un carnet de produit
- Un carnet de sprint
- Un Scrum board qui facilite d'ailleurs les daily sprints
- La possibilité d'estimer vos tâches
- La possibilité de faire du reporting – sprint burndown (graphique d'avancement de sprint).

❖ Durée SPRINT et User Story

La durée de chaque Sprint est déterminée avant le début de celui-ci, la majorité des sprints ont duré entre 1 et 4 semaines. Cela dépend grandement de la fonctionnalité du backlog à réaliser, et aussi de l'ampleur des tâches qu'on effectue (cf. figure 5).

HPGEN-208	Create a JPA POJO from JsonEditor	Story	High	DONE
HPGEN-211	Validation des JsonSchema	Story	Low	DONE
HPGEN-217	Nexus deployment	Story	Medium	DONE
HPGEN-218	Create RESTService from JsonEditor	Story	Low	DONE
HPGEN-219	Update constraints JsonSchema of Flex's Components	Story	Highest	DONE
HPGEN-220	Initialisation du JsonEditor par deux params (jsonschema + json)	Story	Medium	DONE
HPGEN-221	enrichir les conditions de json schema pour chaque composant	Task	Medium	DONE
HPGEN-222	appeler une jsonSchema a partie d'une autre json Schema par reference	Task	Medium	DONE
HPGEN-223	Compléter le Json Schema ci-joint, en ajoutant les autres types du pojo	Task	Medium	DONE
HPGEN-224	generer un POJO from json Editor, (sans annotation JPA)	Task	Medium	DONE
HPGEN-225	Generer un pojo avec les annotations JPA (@Entity, @Id, @OneToMany ...)	Task	Medium	DONE

Figure 5: Sprints

Une User Story ("récit utilisateur" en français) est la description fonctionnelle utilisée dans les méthodes agiles pour spécifier le développement d'une fonctionnalité, en exprimant à qui elle s'adresse et en quoi elle apporte de la valeur. Elle est généralement rédigée par le Product Owner et divisée en tâches, afin de définir un besoin auprès des équipes de développement (cf. figure 6).

User management

Edit
Comment
Assign
To Do
In Progress
Done

Type: Story
Priority: Medium
Component/s: None
Labels: None
Sprint: HPGEN Sprint 18/19

Status: DONE (View workflow)
Resolution: Done

Assignee: Kacimi Badr
Reporter: Salmane Sentissi
Votes: 0 Vote for this issue
Watchers: 2 Stop watching this issue
Created: 23/Apr/18 12:00 PM
Updated: 16/May/18 9:49 AM
Resolved: 16/May/18 9:49 AM

Description

- les utilisateurs dans jsonEditor, doivent etre enregistrer dans une base de données mysql (aujourd'hui, les utilisateurs sont enregistré dans dynamoDB) (HPGEN-248 DONE)
- mettre une relation entre les POJO générer et l'utilisateur qui a generer ces pojos (HPGEN-249 DONE)
- l'utilisateur doit récupérer ses composants, lorsqu'il se connecte (HPGEN-250 DONE)

Attachments

Drop files to attach, or browse.

Issue links

relates to

HPGEN-248 les utilisateurs dans jsonEditor, doivent etre enregistrer dans une base de données ... DONE

Development

Create branch

Agile

Future sprint: HPGEN Sprint 18/19
View on Board

Figure 6 : user story <<User Management>>

❖ Burndown Chart (graphique d'avancement)

En méthode Agile, on utilise des outils pratiques pour suivre l'évolution de la charge de travail. Un de ces outils est le Burndown Chart. Il est utilisé en Scrum Un **Burndown Chart** (cf. *figure 7*) est un graphique simple qui indique le degré d'avancement dans la réalisation des tâches. En d'autres termes, c'est une représentation graphique de l'évolution de la quantité de travail résiduelle en fonction du temps, sur une période donnée.

HP-GEN / HPGEN board / Reports

Burndown Chart

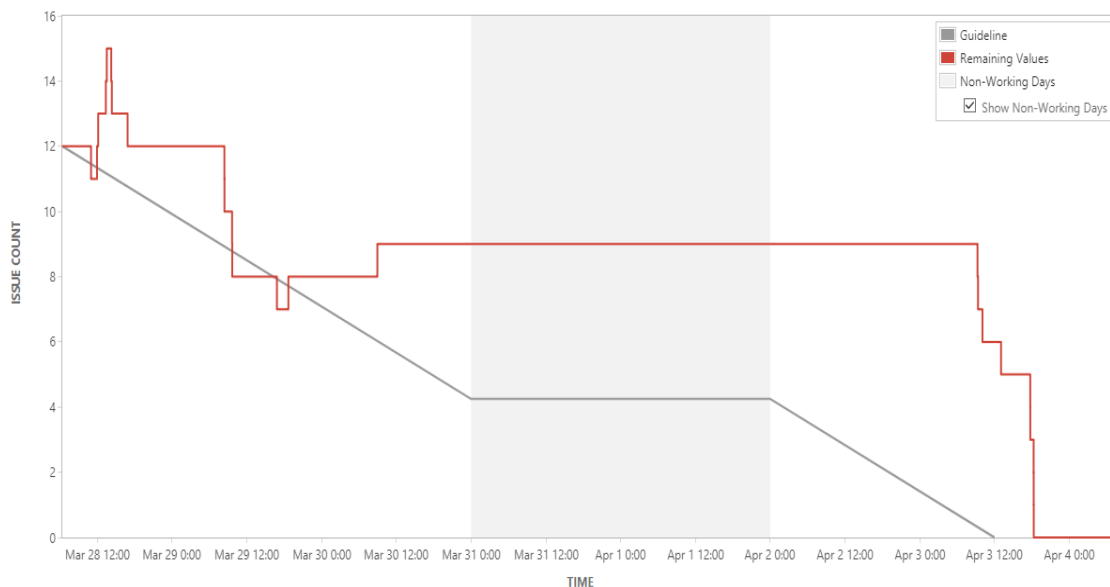


Figure 7 : Burndown Chart

Conclusion

Dans ce premier chapitre, j'ai décrit, en trois grandes parties, le contexte général de ce projet qui consiste en la mise en place d'une architecture web générique. La première partie a été consacrée pour la présentation de l'organisme d'accueil *HIVEPROD* et son champ d'activité. Dans la deuxième partie, j'ai traité le cadre fonctionnel dans lequel s'inscrit ce projet, tandis que la troisième partie a abordé les objectifs à atteindre durant le stage ainsi que la démarche suivie pour mener à bien ce travail.

Il convient maintenant d'entamer le premier volet du projet, concernant la génération de Back-end (coté serveur) et Front-End (coté client) tout en s'appuyant sur une étude préalable.

Chapitre 2 : Génération back-end et Front-end

Le but de ce deuxième chapitre est d'analyser les approches classiques du développement d'une application web coté client et coté serveur, et proposer une approche afin de rendre le processus de développement automatique et générique.

1. Analyse de l'existence

Procédure routinière : Qu'est-ce qui caractérise une application de gestion ?

Une application web de gestion, comme son nom l'indique, doit gérer des données. Ce qui implique l'utilisation d'une base de données (la plupart du temps relationnel). Cela implique également une interface pour permettre à des gestionnaires de voir et modifier ces données.

La majorité des applications de gestion requièrent des modules routiniers qui se répètent, ces derniers requièrent de copier les mêmes traitements d'une solution à une autre en modifiant les parties variables :

- Restriction des ressources et gestion des rôles (Sécurité).
- Validation des données répétées sur toutes les couches.

2. Les spécifications fonctionnels

Dans cette partie, je vais traduire les spécifications fonctionnelles en diagramme selon UML. (cf. figure 8)

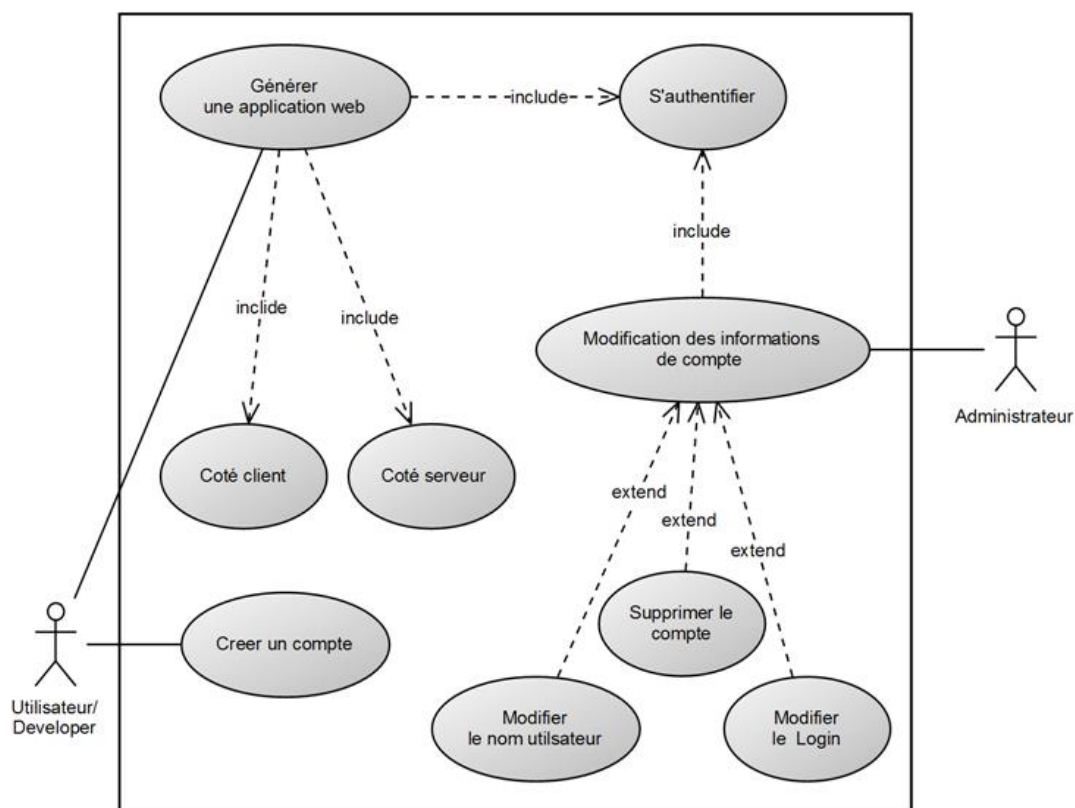


Figure 8 : use case << Générateur d'application web>>

Pour rendre notre diagramme des cas d'utilisation plus lisible et afin de décrire le comportement d'un système, les tableaux suivants présentent la description textuelle des cas d'utilisation (cf. Tableaux 2 et 3).

Cas d'utilisation	côté client
Acteurs	Utilisateur développeur
Précondition	Utilisateur authentifié
Post-condition	L'utilisateur choisit un Template
Scénario nominal	1- L'utilisateur choisit les composants voulus du Template 2- L'utilisateur remplit les champs 3- La Génération du côté client (Front-End)
Scénario alternatif	N/A

Tableau 2 : Description textuelle du cas d'utilisation « Cote client Front-End»

Cas d'utilisation	côté Serveur
Acteurs	Utilisateur développeur
Précondition	Authentification
Post-condition	L'utilisateur entre sa conception dans le système
Scénario nominal	1- L'utilisateur saisie les propriétés de POJO 2- Le système crée les classes /persistance en base de données 3- L'utilisateur saisie le REST API 4- La Génération de REST API (exposer les ressources)
Scénario alternatif	N/A

Tableau 3 : Description textuelle du cas d'utilisation « Coté serveur Back-End»

3. Génération coté client Front-End

3.1. Description

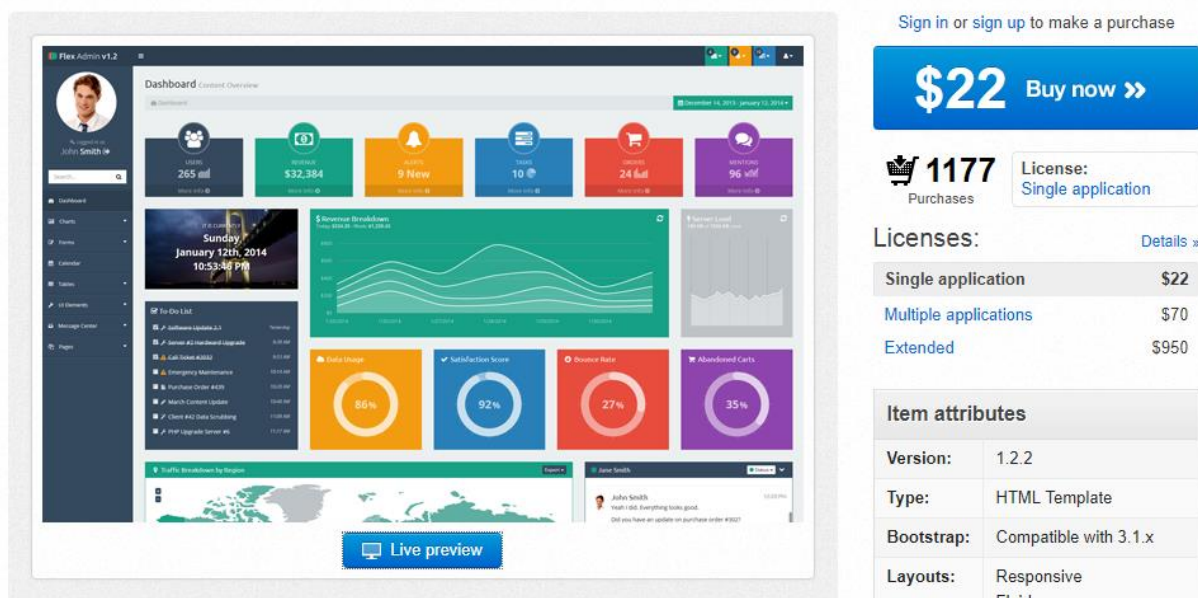


Figure 9 : Un Template

Dans un Template (cf. figure 9), la partie statique concerne la structure, caractères HTML des composants qui sont destinés au navigateur web et non aux utilisateurs humains. La partie dynamique concerne les données, les textes, les icônes et les couleurs qui contiennent des informations intéressantes pour les utilisateurs et les développeurs.

Ces informations qui majoritairement en format JSON, alors chaque composant contient de JSON data qui varie entre numéros de texte, dates, classe Bootstrap etc...

Exemple, le bloc suivant :

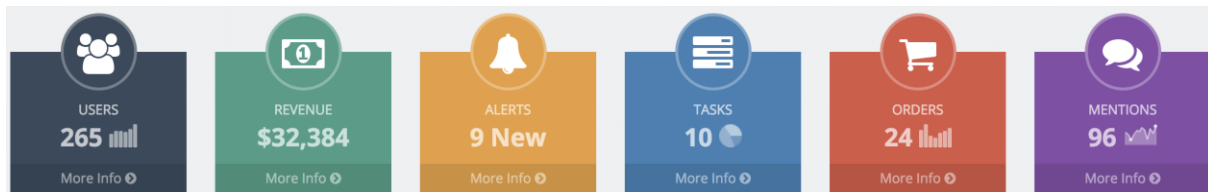


Figure 10 : composante d'un Template

Traduit par :

```
<div class="row">
  <div class="col-lg-2 col-sm-6">
    <div class="circle-tile">
      <a href="#">
        <div class="circle-tile-heading dark-blue">
          <i class="fa fa-users fa-fw fa-3x"></i>
        </div>
      </a>
      <div class="circle-tile-content dark-blue">
        <div class="circle-tile-description text-faded">
          Users
        </div>
        <div class="circle-tile-number text-faded">
          265
          <span id="sparklineA"><canvas width="29" height="24" style="display: inline-block;
width: 29px; height: 24px; vertical-align: top;"></canvas></span>
        </div>
        <a href="#" class="circle-tile-footer">More Info <i class="fa fa-chevron-circle-
right"></i></a>
      </div>
    </div>
  </div>
</div>
```

Les " vraies " valeurs qui intéressent un utilisateur ou un développeur dans ce HTML sont :

Les textes ["More Info", "265", "Users"]

Les icons ["fa-users", "fa-money", "fa-bell" ...]

Les couleurs ["dark-blue", "orange", "green" ...]

Tous les autres caractères HTML sont destinés au navigateur web et non aux utilisateurs !
Comment notre générateur pourra les détecter, les extraire et les transformer dynamiquement en des composants Angular ?

J'ai fait l'exercice manuellement, étape par étape, ensuite on apprendra à le faire automatiquement ...

Je vais donc transformer l'HTML en JSON en utilisant XPATH, on utilisant la même source HTML initiale avec le XPath suivant sur :

```
//div[contains(@class, "circle-tile-description")]
```

On obtient :

```
Element='<div class="circle-tile-description text-faded">Users</div>'
Element='<div class="circle-tile-description text-faded">Revenue</div>'
Element='<div class="circle-tile-description text-faded">Alerts</div>'
Element='<div class="circle-tile-description text-faded">Tasks</div>'
Element='<div class="circle-tile-description text-faded">Orders</div>'
Element='<div class="circle-tile-description text-faded">Mentions</div>'
```

Où contient alors les textes intéressants pour des utilisateurs et des développeurs.

Ces éléments sont "variables" d'un client utilisateur à un autre, donc l'Output du générateur doit être un composant Angular où il externalise ces "variables" et ces attributs : Title icon color

Comment le générateur d'applications web saura qu'il faut utiliser "circle-tile-description" et non une autre expression ?

La solution est la suivante :

- ⇒ Lister toutes les combinaisons dans une IHM et laisser l'utilisateur ou le développeur à faire son choix .

3.2. Approche classique

On veut personnaliser, modifier ou encore mettre à jour un site réalisé à l'aide d'un Template, on aura besoin de logiciels spécifiques. Parmi les plus faciles à utiliser, l'on retrouve Microsoft Frontpage. Certains outils, tels que Macromedia Dreamweaver ou Flash, nécessitent par contre des connaissances en langage HTML. Raison de plus pour passer auprès d'une agence de création de site web, en passant par la décomposition en partie statique et dynamique, ce qui devient plus en plus complexe avec le nombre et les composantes de chaque pages. Ainsi le résultat est n'est instantanée et prend du temps. Donc, il faut automatiser le processus afin qu'il soit générique et se concentrer le fonctionnel de business c.-à-d. un minimum de développement humain et un maximum de développement par la machine.

3.3. Approche adoptée

3.3.1. Description

Afin de laisser le client/développeur choisir les composantes de son interface et les modifier, l'idée s'était de travailler avec les métadonnées des composantes (cf. Figure 11) de Template pour générer des interfaces personnalisées pour chaque client en utilisant une interface conviviale IHM (Projet JSON Editeur), toute en limitant les entrées (input validation) grâce aux mécanismes de JSON Schéma.

Le projet JSON Editor est un outil puissant pour valider la structure des données JSON et visualiser les JSON Schéma, il est développé sous la technologie JAVA Framework Spring et Java Script par nous (team HIVEPROD) durant la période de stage.

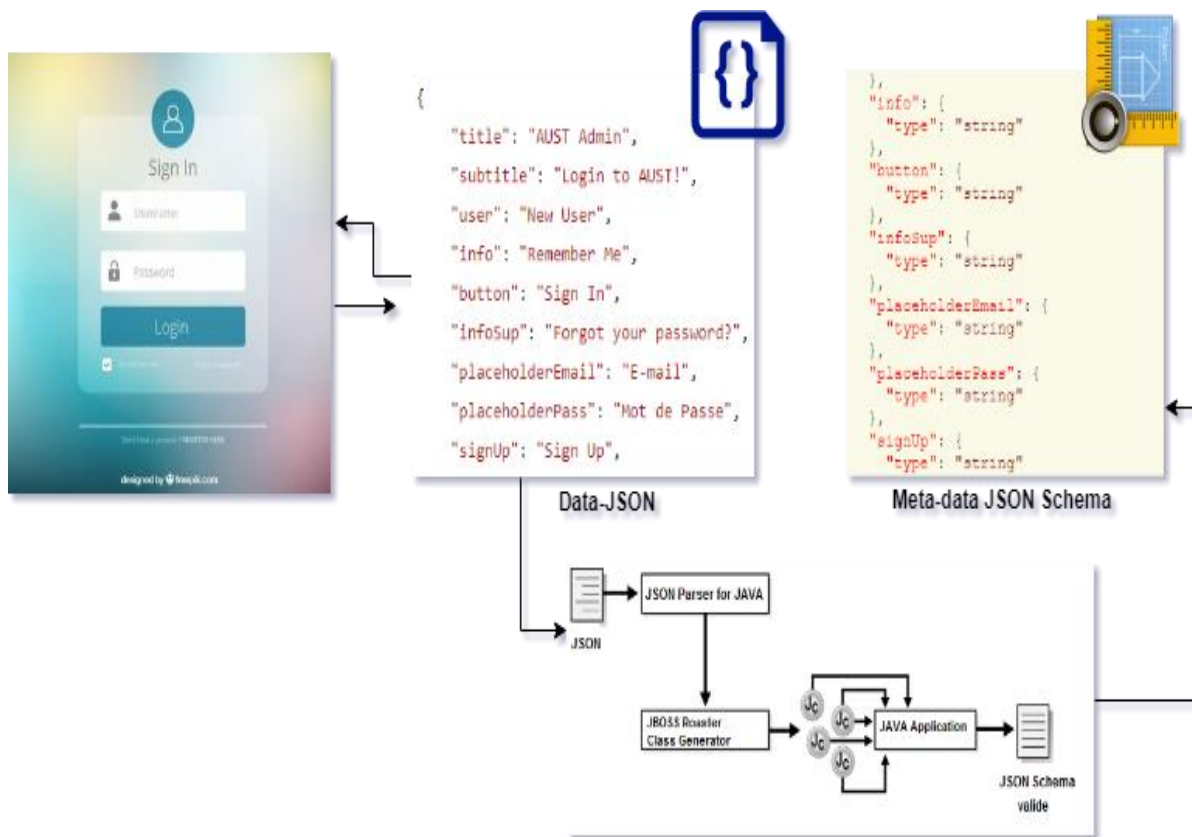


Figure 11 : Génération de JSON Schéma- Meta data

3.3.2. Scénario

- ⇒ Générer le JSON Schéma (Modèle de données de JSON) de toutes les composantes d'un Template (cf. figure 10) donnée ainsi l'enrichissement des conditions des JSON Schéma.
- ⇒ Utilisateur crée un compte dans JSON Editor pour définir ses composantes.
- ⇒ Génération d'un projet Angular dynamique (Front-End).

Chaque client (utilisateur de générateur) peut personnaliser son interface avec les composantes désirées (texte, couleur, forme, Tableaux, forms...) proposées par le Template choisie et peut changer l'interface après la génération de projet et consulter les modifications en real-time et l'application web est mise en service dynamiquement (grâce aux services web d'Amazon : AWS) dans les serveurs de l'entreprise.

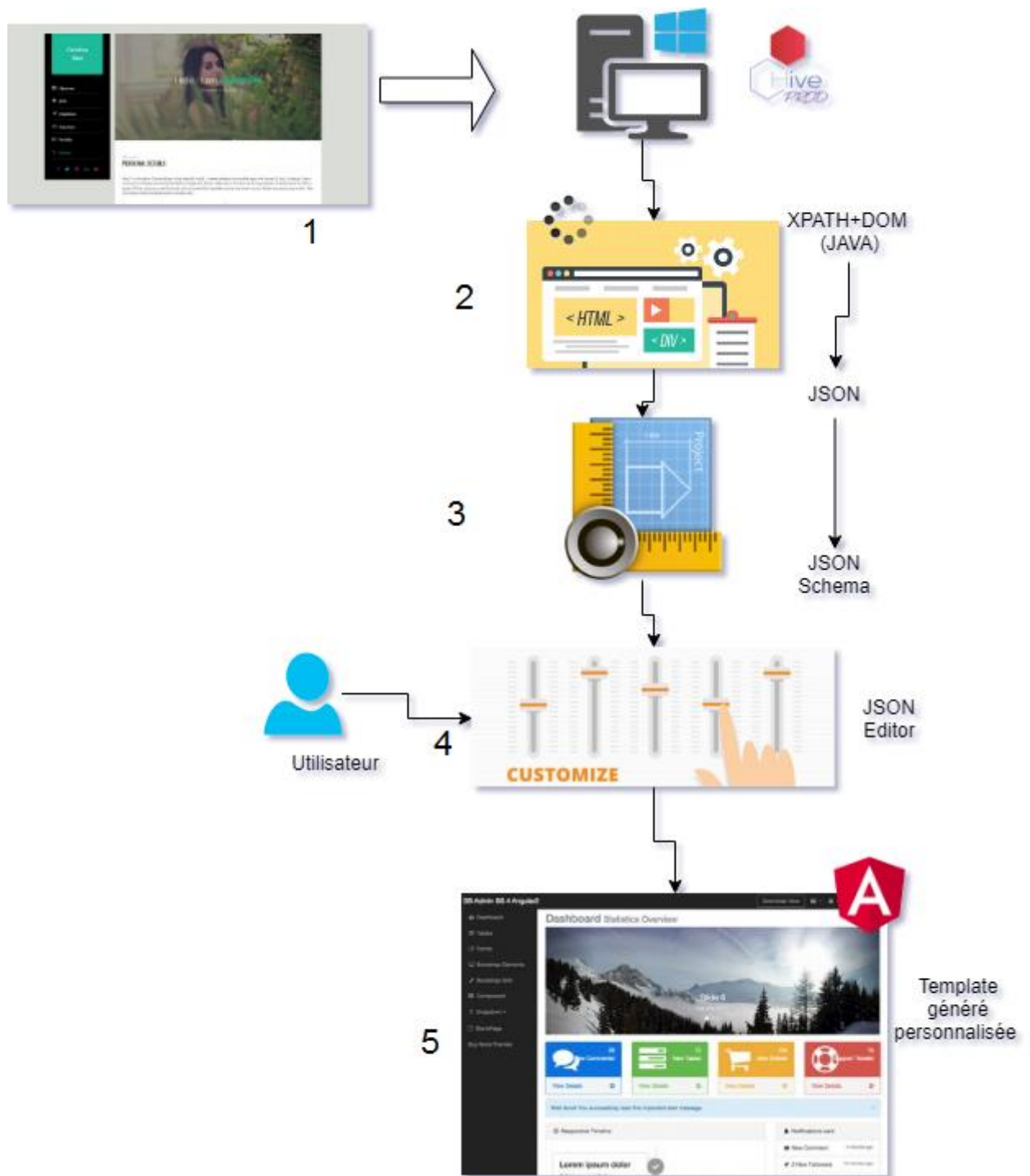


Figure 12 : Scénario global de génération de Front-end d'une application web

4. Partie Serveur Back-End

4.1. Définition

Back-end concerne la partie invisible d'une application web, il comporte ces éléments :

- Le serveur qui héberge le programme.
- La base de données relative à un site ou à l'appli (fichier client, fiches produits, etc.)

4.2. Approche adoptée

On va faire apprendre au générateur comment générer le Back-End de notre application web automatiquement. La génération d'un jeu de classes utilise un utilitaire fourni dans la distribution Java JBOSS : Roaster. En s'appuyant sur Schéma (modèle de données) d'un POJO pour faire comprendre au générateur comment générer des POJO à travers des JSON Schéma.

4.2.1. Modèle-vue-contrôleur :

Modèle-vue-contrôleur ou MVC est un motif d'architecture logicielle destiné aux interfaces graphiques lancé en 1978 et très populaire pour les applications web. Le motif est composé de trois types de modules ayant trois responsabilités différentes : les modèles, les vues et les contrôleurs.

- Un **modèle** (Model) contient les données à afficher.
- Une vue (View) contient la présentation de l'interface graphique.
- Un **contrôleur** (Controller) contient la logique concernant les actions effectuées par l'utilisateur.

Ce motif est utilisé par de nombreux Frameworks pour applications web tels que Ruby on Rails, Django, Spring qui le Framework de l'application web générée.

On 'a déjà généré la vue (view : Front-End), il reste maintenant le modèle et le contrôleur.

❖ Modèle

La première étape l'écriture d'une JSON Schéma de POJO de **model** (les types de variables, sérialisation, annotations JPA valables etc...) qui contient les règles de génération de POJO (cf. figure 12). Des conditions sont mis en place dans le JSON Schema ainsi dans le code de génération afin de valider l'input et respecter les standards JAVA.

Après authentification, l'utilisateur sait les informations nécessaires pour définit Un JSON en ouput contenant le nom de classe variable etc... pour l'exploiter dans la génération de POJO en utilisant le package Java Roaster (cf. figure 13).

```
{
  "description": "A POJO JsonSchema Description",
  "type": "object",
  "title": "POJO Constructor",
  "properties": {
    "className": {
      "type": "string",
      "minLength": 2,
      "pattern": "^[A-Za-z]*$"
    }
  },
  "attributes": {
    "type": "array",
    "minItems": 0,
    "maxItems": 999,
    "uniqueItems": true,
    "additionalItems": false,
    "items": {
      "title": "Field Name",
      "type": "object",
      "properties": {
        "ChoixType": {
          "description": "Java field type",

```

Figure 13 : Extrait de JSON Schéma (modèle de données de POJO)

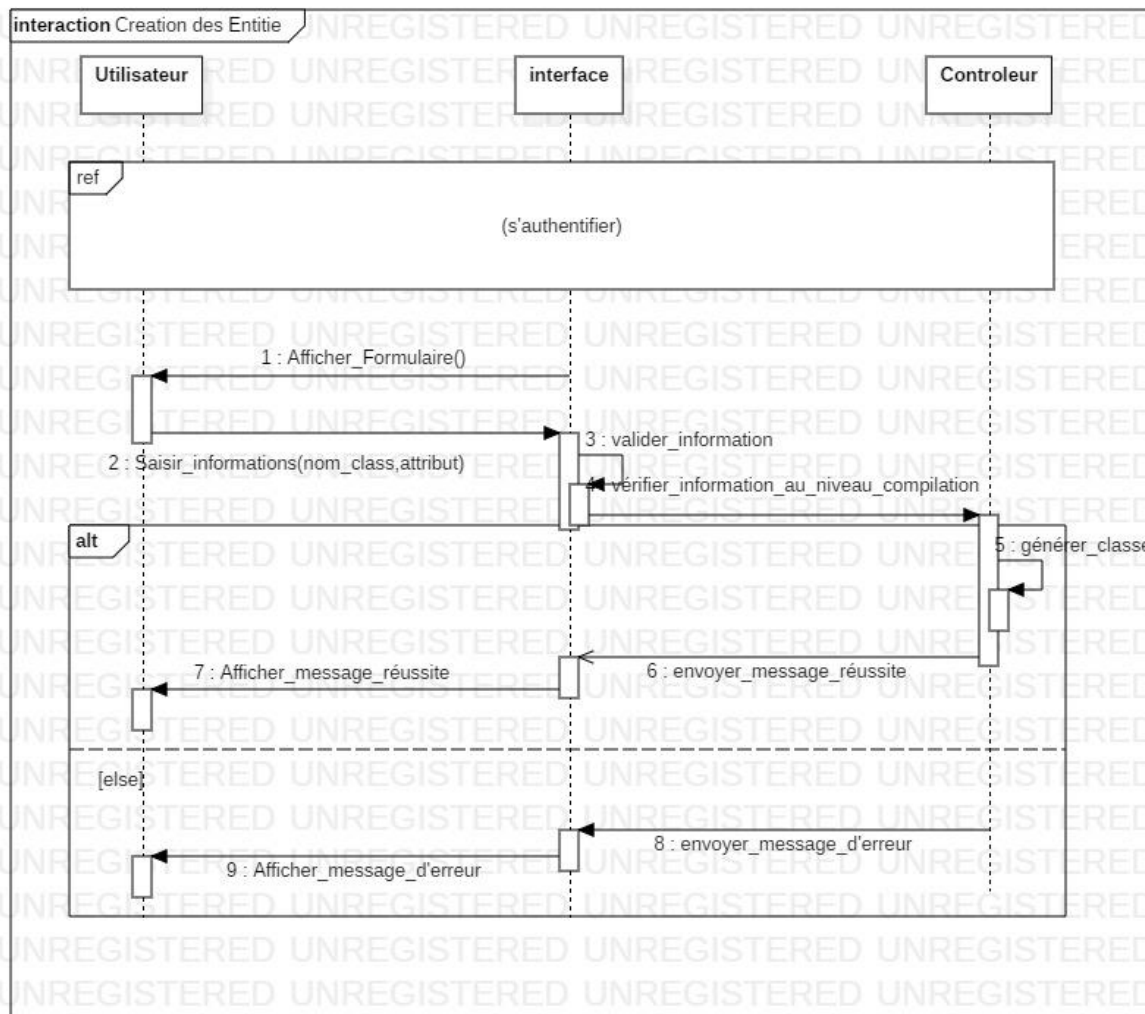


Figure 14 : Diagramme de séquence <<Scenario de génération d'un POJO>>

Les Pojo générées contiennent des annotations du spécification JPA et leurs tables associés persistent dans la base de données dès la première compilation du projet.

❖ contrôleur

J'ai utilisé l'architecture **REST** (cf. Figure 14 page 32) comme moyen d'obtenir des ressources externes ou d'exposer une partie des tiennes dans un format "brut" style JSON ou XML.

La génération d'un contrôleur Rest est faite de la même manière que le modèle avec une JSON schéma qui définit la structure et la grammaire des classes REST (@Rest Controller http protocole mapping etc...).

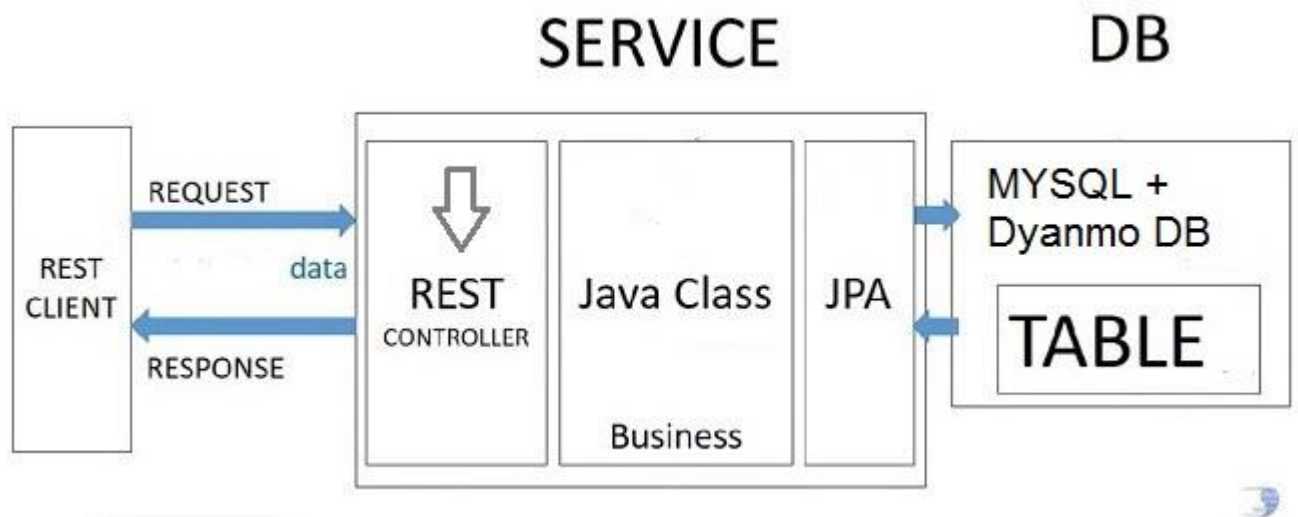


Figure 15 : Approche du style d'architecture REST

❖ Swagger

Swagger est un standard de description d'APIs REST, afin de rendre accessible et compréhensible de tous (robots et humains) les services mis à disposition et la façon de les utiliser sans avoir accès au code source.

Swagger va au delà d'une simple documentation technique car il permet à un consommateur (quel qu'il soit) l'utilisation à distance des services afin d'en explorer toutes les capacités.

On'a utilisé cet outil indispensable pour générer un contrat qui décrit le service REST et les ressources de l'application ainsi générer le code client (Rest client) Angular de Template à partir de ce contrat .

4.2.2 Sécurité des services web –Rest

Les Services Web, par nature ouverts et interopérables, constituent des points d'entrée privilégiés pour des attaques. Afin de pallier à ces vulnérabilités potentielles, un certain nombre de bonnes pratiques doivent être prises en compte. Elles s'articulent autour de 3 axes majeurs qui sont : la sécurisation côté client, la sécurisation de l'application et la sécurisation des échanges(cf. livre RESTful Java Patterns and Best Practices 2014 *page 58*).

Le format d'un jeton JWT) est particulièrement bien adapté pour circuler dans une uri(cf. Figure 17) et donc pour les applications REST.JWT est actuellement utilisé par beaucoup de développeurs

Un jeton JWT est composé de trois parties séparées par un point (cf. Figure 16), chaque partie est créée différemment. Ces trois parties sont :

Header Payload Signature

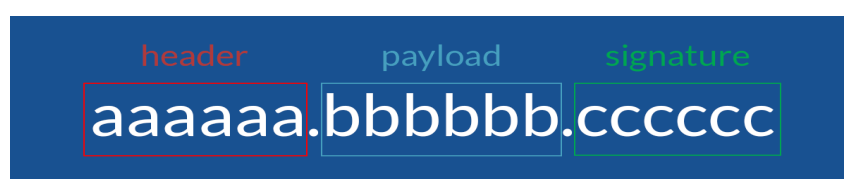


Figure 16 : jeton JWT

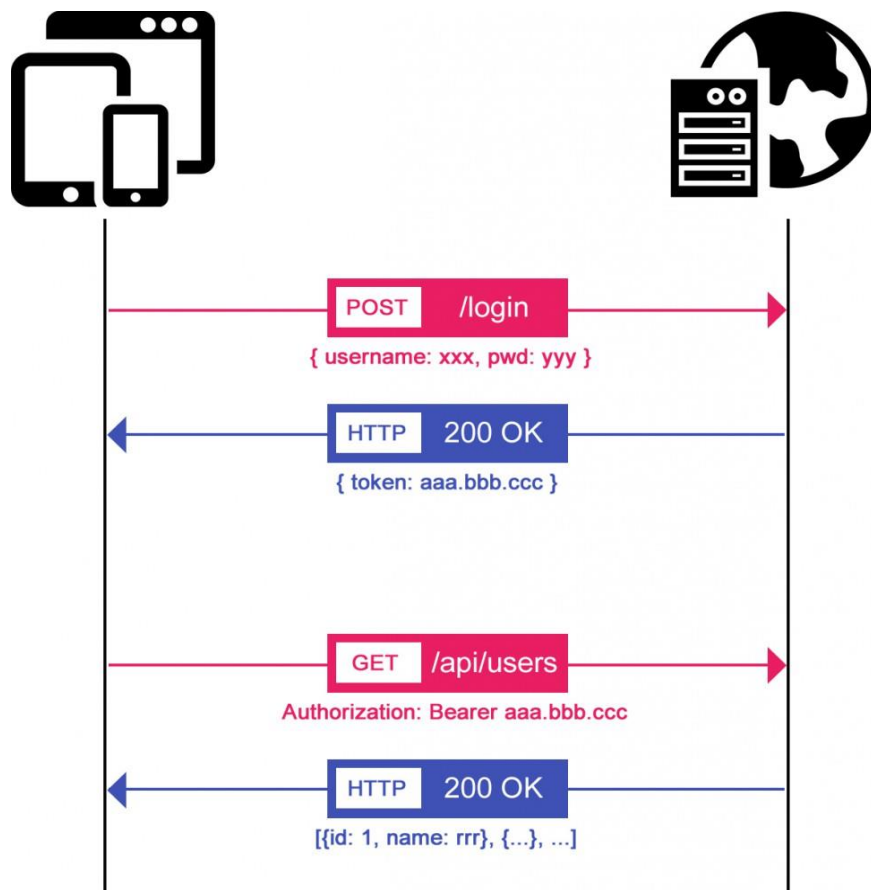


Figure 17 : JWT Authentication

➤ Header (ou en-tête) :

Le header (ou en-tête) est un document au format JSON, qui est encodé en base 64 et qui contient deux parties :

```

{
  "typ" : "JWT",
  "alg" : "MD5"
}
  
```

- Le type de token, qui est ici JWT
- L'algorithme de chiffrement à utiliser pour hasher le payload

➤ Payload (ou contenu) :

Le payload (ou contenu) est un document au format JSON, qui est encodé en base 64 et qui contient les informations à échanger entre le client et le serveur. Ces informations sont appelées claims ou revendications.

En règle générale, on fait transiter des informations sur l'identité de l'utilisateur, mais il ne doit absolument pas contenir de données sensibles.

➤ Signature :

La signature est composée d'un hash des éléments suivant :

Header +Payload+Secret


```
var encodedString = base64UrlEncode(header) + "." +
base64UrlEncode(payload) ;
HMACMD5(encodedString, 'secret');
```

Le secret est une signature détenue par le serveur. C'est de cette façon que le serveur sera capable de vérifier les tokens existant et d'en signer des nouveaux.

5. Schéma technique globale

Schéma technique globale du générateur :

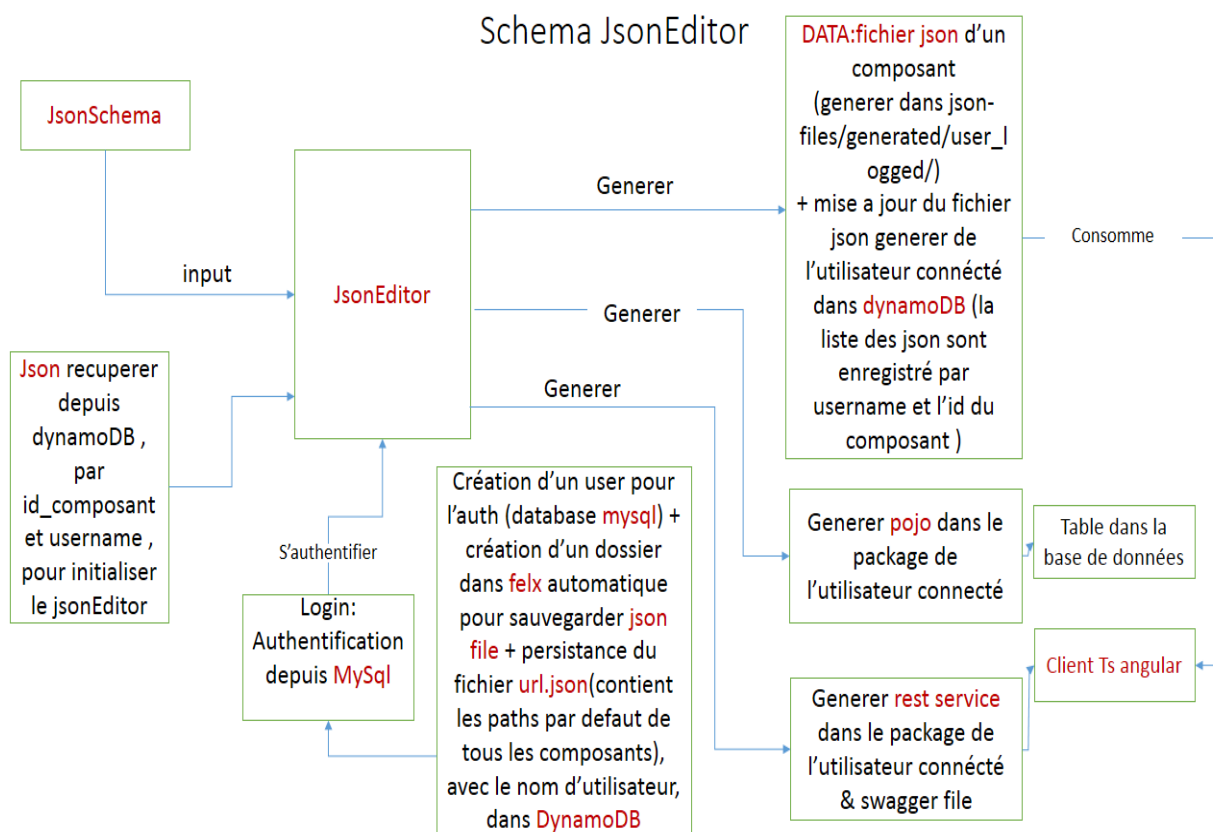


Figure 18 : Schéma technique du générateur

Ci-dessous, le schéma technique de l'application web générée et son fonctionnement générale. (cf. figure 19).

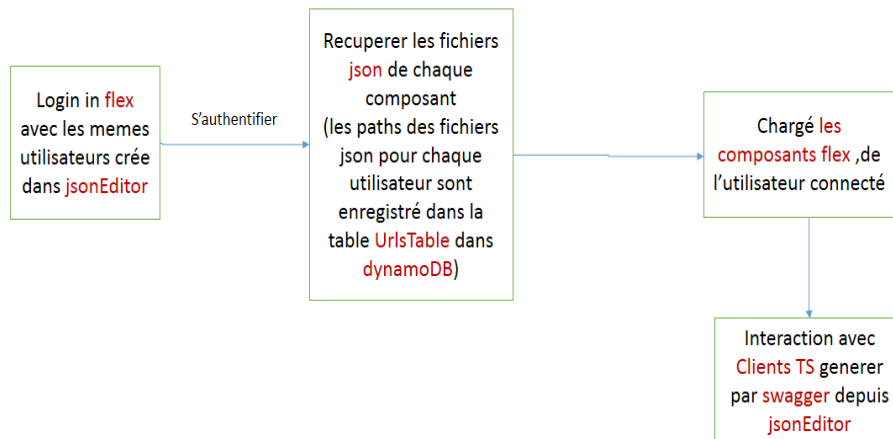


Figure 19 : Schéma technique de l'application web générée

Conclusion

Dans ce deuxième chapitre consacré à l'analyse et spécification, j'ai commencé par une analyse des architectures techniques existantes. J'ai présenté en un premier lieu les caractéristiques des applications de gestion adopté par HIVEPROD.

Ensuite, J'ai critiqué l'existant des différentes solutions existantes afin de ressortir les parties répétitives dans la phase de développement. Finalement, j'ai établi les exigences fonctionnelles afin de savoir exactement ce que la solution devrait faire.

Après l'étude des différentes approches en détail et les différentes démarches suivies pour la génération d'une application web en coté client et en coté serveur, le chapitre suivant est dédiée à la présentation des outils et technologies utilisées dans ce projet.

Chapitre 3 : Technologies et outils

Ce troisième chapitre abordera les différentes technologies utilisées afin de réaliser la solution. Il va présenter en premier lieu l'architecture technique de la solution. En deuxième partie, on présentera les outils et technologies utilisées pour les applications développées au sein de HIVEPROD, et de ce fait par notre solution.

1. Architecture de l'application

Le style d'architecture en niveaux spécifie le nombre de niveaux organisationnels où vont se situer les environnements d'exécution du système.

Dans l'architecture 3-tiers, il existe un niveau intermédiaire, c'est-à-dire que l'on a généralement une architecture partagée entre :

- Un client, c'est l'ordinateur demandeur de ressources, équipé d'une interface utilisateur (généralement un navigateur web).
- Le serveur d'application (appelé également middleware), chargé de fournir la ressource mais faisant appel à un autre serveur.
- Les serveurs de base de données, fournissant au serveur d'application les données dont il a besoin.

2. Grands choix techniques

Pour ce projet le choix des technologies à utiliser n'a pas été sujet d'une longue recherche ou bien benchmarking puisque les grands traits de la solution étaient déjà fixés dès le départ en tant qu'exigences. Sauf que cela ne nous a pas empêché de rechercher et de se documenter sur la solidité de ces technologies et avoir une justification de la part du directeur technique à leur sujet.

❖ Choix du langage JAVA

Bien que le choix du langage fût lui aussi imposé comme exigence, le choix de Java comme langage est dû à sa portabilité, sa disponibilité dans les offres Cloud et à la disponibilité des Framework open source, et ceci est d'autant plus important pour notre projet vu qu'il s'appuiera grandement sur la modification des apis afin de prendre avantage de la robustesse de ces dernières et une rapidité de développement accrue, au lieu d'avoir à réinventer la roue.

❖ Pourquoi Spring ?

HiveProd utilise le Framework java Spring pour ses développements puisqu'elle développait ses solutions dans l'ère où le standard J2EE était complexe à mettre en œuvre. Donc Spring était le candidat idéal du fait qu'il soit complet pour répondre aux besoins. Un autre facteur décisif, et qu'il est portable sur tous les serveurs d'application.

Environnement de développement



Spring Tool Suite est un environnement de développement basé sur Eclipse qui est personnalisé pour le développement d'applications Spring. Il prend en charge le ciblage des applications sur des serveurs locaux, virtuels et basés sur le cloud. Il est disponible gratuitement pour le développement et les opérations commerciales internes sans limite de temps, entièrement open-source.



WebStorm est un Puissant JavaScript IDE commercial développé par JetBrains pour le développement JavaScript moderne avec complétion de code et refactoring pour JavaScript, Type Script et les Frameworks web les plus populaires.



JavaScript est un langage de programmation de scripts principalement employé dans les pages web interactives mais aussi pour les serveurs² avec l'utilisation (par exemple) de Node.js³. C'est un langage orienté objet

Système de gestion de base de données



MySQL est un serveur de bases de données relationnelles SQL développé dans un souci de performances élevées en lecture, ce qui signifie qu'il est davantage orienté vers le service de données déjà en place que vers celui de mises à jour fréquentes et fortement sécurisées. Il est multi-thread et multi-utilisateur. C'est un logiciel libre, open source



Un service de base de données NoSQL propriétaire entièrement géré qui prend en charge les structures de données de valeur-clé et de document et est proposé par Amazon.com. DynamoDB utilise la réplication synchrone sur plusieurs centres de données pour une durabilité et une disponibilité élevées.

Versioning

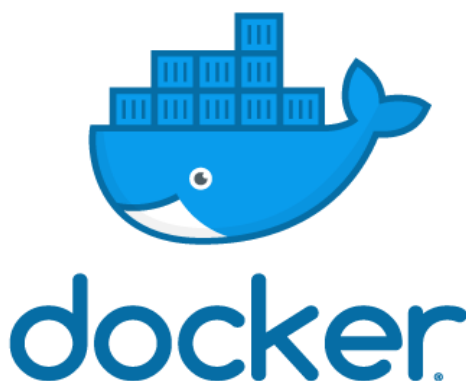


AWS CodeCommit est un service commercial de contrôle de version hébergé par Amazon Web Services que vous pouvez utiliser pour stocker et gérer en mode privé des ressources (telles que des documents, du code source et des fichiers binaires) dans le cloud.



Git est un système Open-Source gratuit de distribution de versions. Il est utilisé pour contrôler les versions des fichiers (enregistrement des changements).

Tests et déploiement



Docker est un outil qui peut emballer une application et ses dépendances dans un conteneur isolé, qui pourra être exécuté sur n'importe quel serveur. Ceci permet d'étendre la flexibilité et la portabilité d'exécution d'une application, que ce soit sur la machine locale, un cloud privé ou public.



Sonatype Nexus

Avec Nexus, vous pouvez contrôler complètement l'accès et le déploiement de chaque artefact de votre organisation à partir d'un seul emplacement. Vous devez utiliser Central Nexus comme proxy et gérer vos propres référentiels pour garantir la stabilité de votre organisation

Devops-Continuous integration



Jenkins est un outil open source d'intégration continue, fork de l'outil Hudson après les différends entre son auteur, Kohsuke Kawaguchi, et Oracle. Écrit en Java, Jenkins fonctionne dans un conteneur de servlets tel qu'Apache Tomcat, ou en mode autonome avec son propre serveur Web embarqué.

Framework



Angular (v2 et supérieur) est une plate-forme d'application Web frontale open source basée sur TypeScript et dirigée par l'équipe Angular de Google et par une communauté. Des individus et des sociétés. Angular est une réécriture complète de la même équipe qui a construit AngularJS.



Swagger est un framework logiciel open source soutenu par un large écosystème d'outils qui aide les développeurs à concevoir, construire, documenter et consommer des services Web RESTful. Sponsorisé par SmartBear Software.

Agile



Jira Software est un outil de développement logiciel pour les équipes agiles conçu de sorte que chaque membre d'équipe de développement puisse planifier, suivre et livrer d'excellents logiciels. : Il incorpore le framework Scrum.

Autres technologie et outils



Bootstrap est une collection d'outils utile à la création du design (graphisme, animation et interactions avec la page dans le navigateur ...etc.) de sites et d'applications web. C'est un ensemble qui contient des codes HTML et CSS, des formulaires, boutons, outils de navigation et autres éléments interactifs.



XPath est une technologie qui permet d'extraire des informations (éléments, attributs, commentaires, etc...) d'un document XML via l'écriture d'expressions dont la syntaxe rappelle les expressions rationnelles utilisées dans d'autres langages.

Conclusion

Dans ce troisième chapitre consacré aux outils ainsi qu'aux technologies utilisées, J'ai en premier lieu présenté l'architecture sur laquelle reposera la solution. Je vais ensuite cité les différentes technologies avec lesquelles j'ai travaillé avec justification du grand choix, le prochain chapitre va aborder la réalisation d'une application web à l'aide de générateur.

Chapitre 4 : Réalisation

Dans ce quatrième chapitre on va illustrer la génération par un exemple complet de génération d'une application web de gestion de produit

- D'abord on va générer Front-End de l'application (coté client)
- Passer au Back end
- Tester l'application générée

On a travaillé (Team HIVEPROD) sur un Template qui s'appelle Flex (cf. Figure 18) dans ce projet pour le test de générateur.

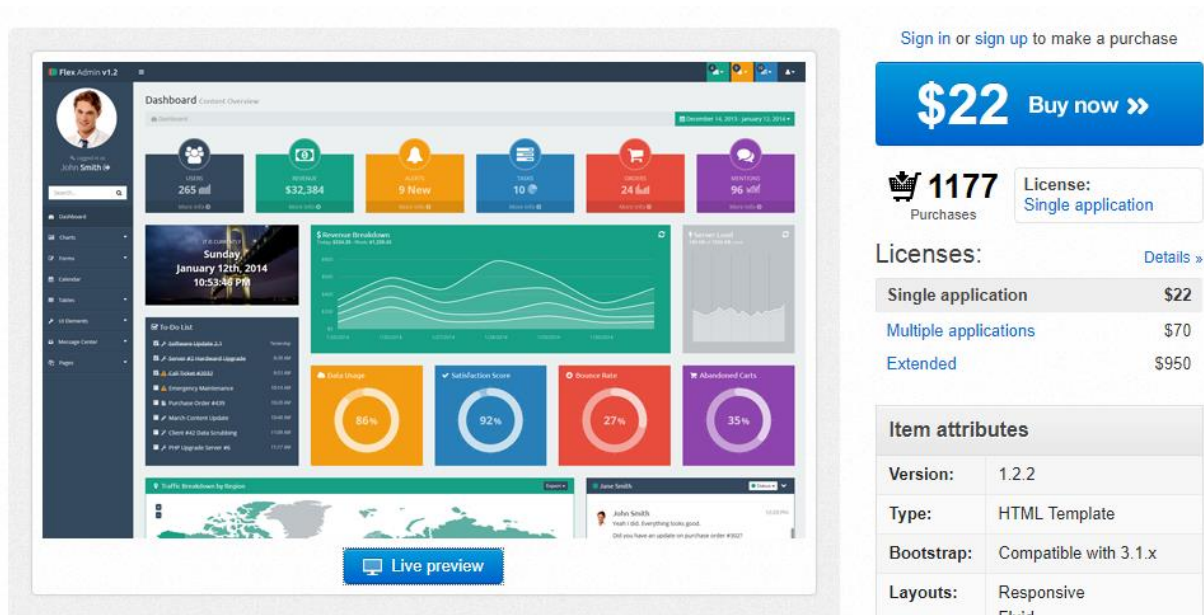


Figure 20 : Template Flex

L'étape de génération des JSON schéma des composantes de Template est faite pour qu'on puisse laisser à l'utilisateur la liberté de personnaliser son interface.

1. Coté client (Front-End)

1.1. Page authentication

-La première page de générateur est la page de l'authentification :

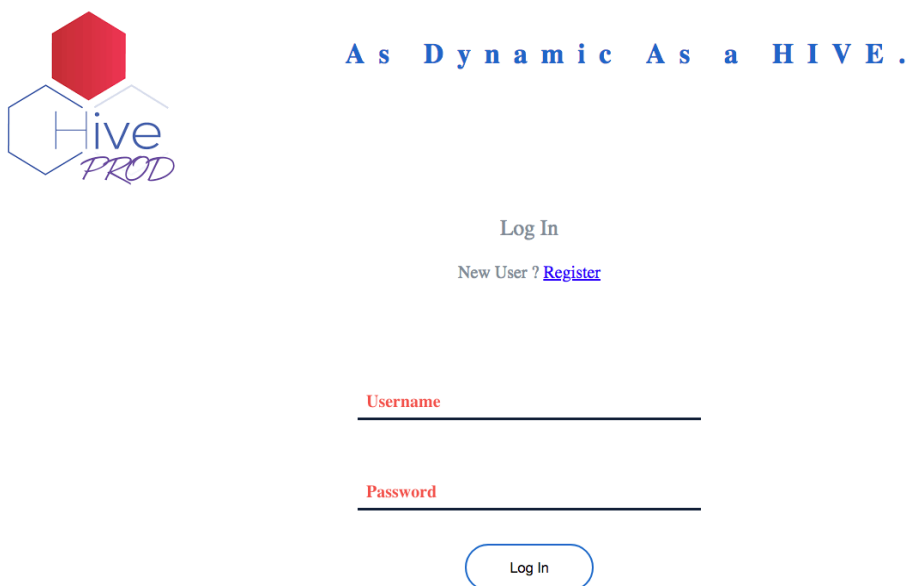


Figure 21 : Page d'authentification

-Le bouton Register : Créer un nouveau compte pour un nouvel utilisateur.

L'utilisateur connecté est l'administrateur :

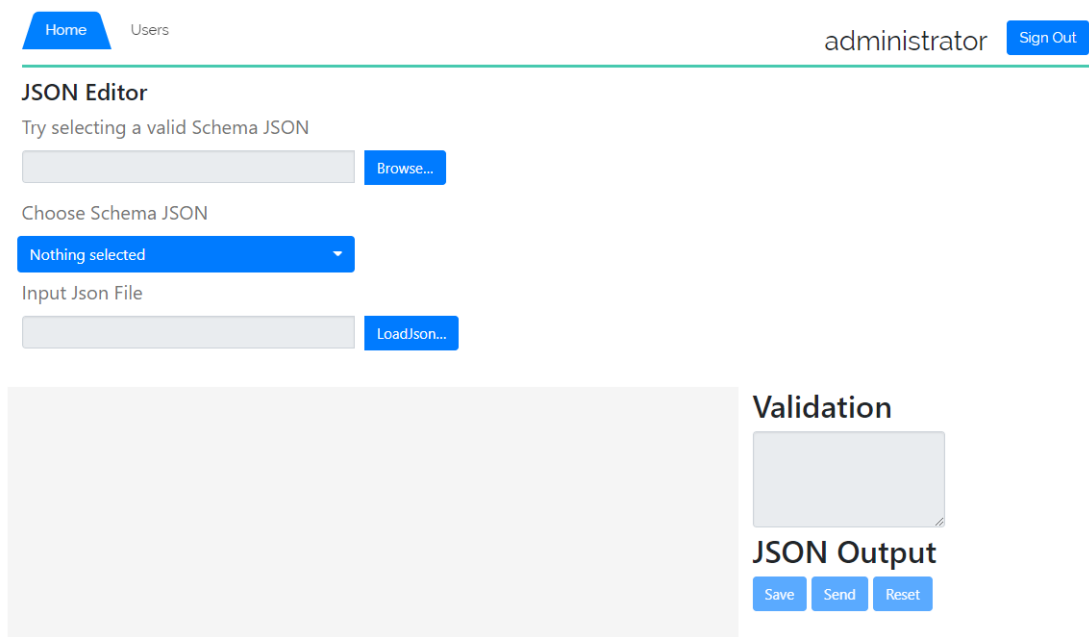


Figure 22 : JSON Editor – page accueil

1.2. Création de menu

⇒ La première étape est de créer un Menu « Menu gauche(SideNavDropDown) » à l'aide de JSONEditor (cf. Figure 21).

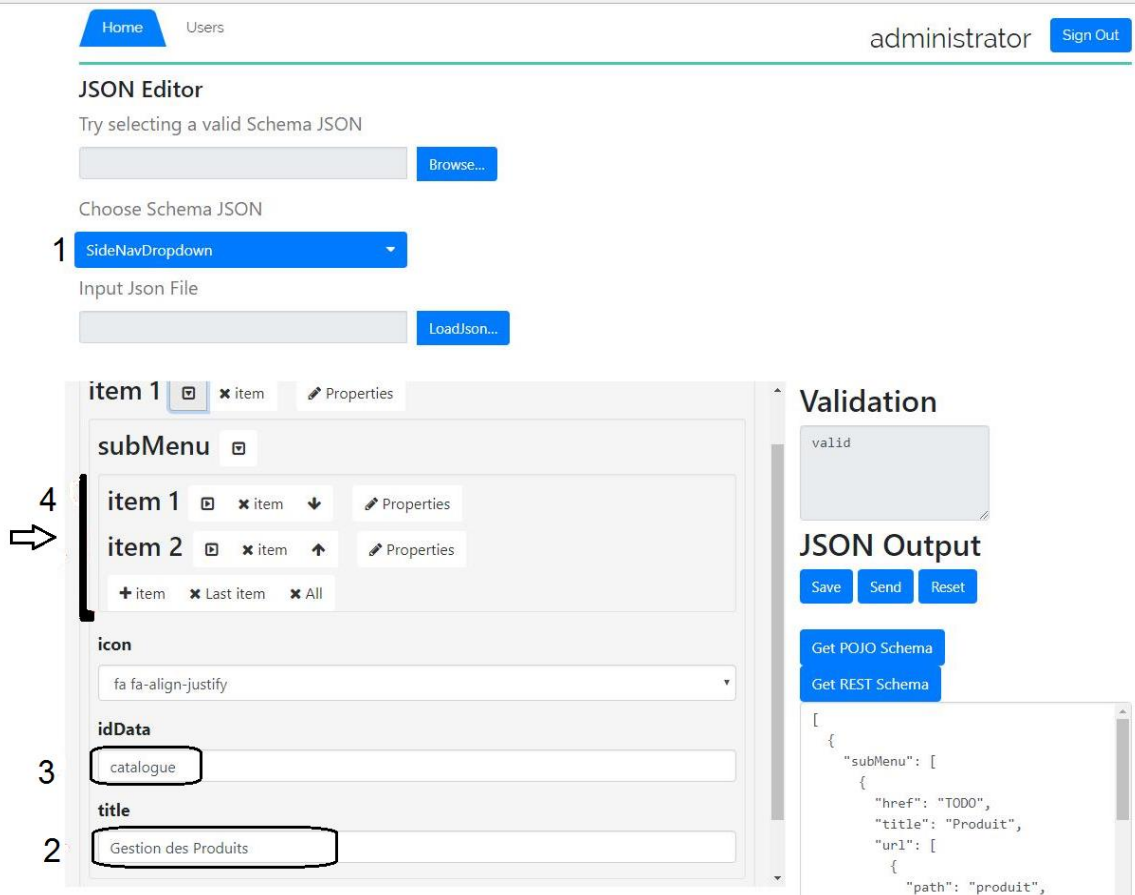


Figure 23 : Menu gauche(SideNavDropDown)

1. Sélectionner JSON Schéma du menu gauche « SideNavDropDown »
2. Saisir Title du menu (exp : Gestion Produits)
3. idData pour différencier les items du Menu (il doit être unique).
4. SubMenu : les différents éléments d'un item dans le menu :

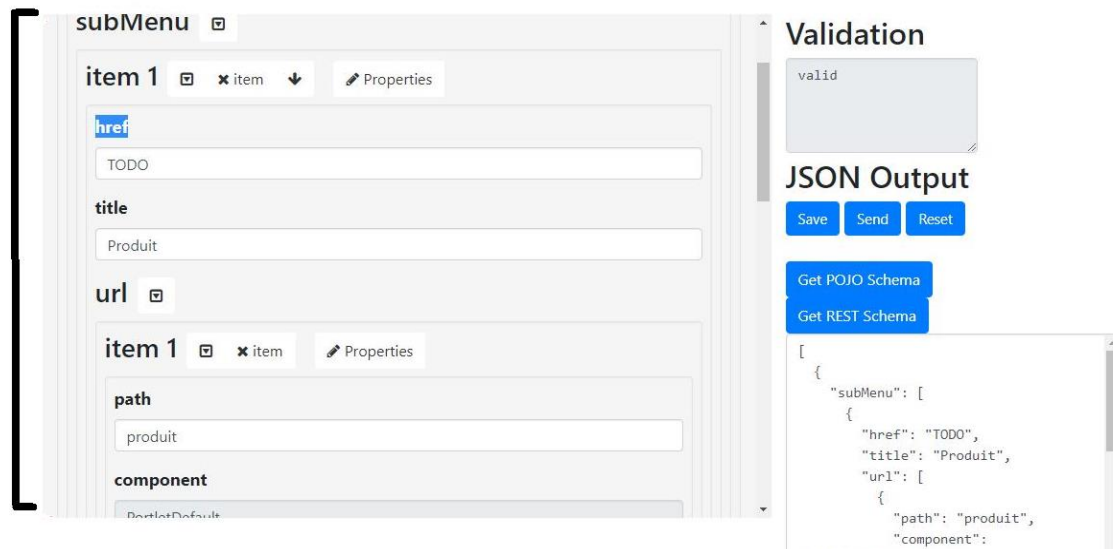


Figure 24 : SubMenu

- ✓ Href
- ✓ Title de l'item
- ✓ url : un tableau qui contient path (nom pojo) & component (route) qui va faire le map entre le menu et tableau correspondant.

-Puis appuie sur le button **send**.

1.3. Création de la table

La deuxième étape est de générer le tableau et le formulaire. Donc on choisit **new table** pour créer un nouveau tableau (cf. Figure 23) :

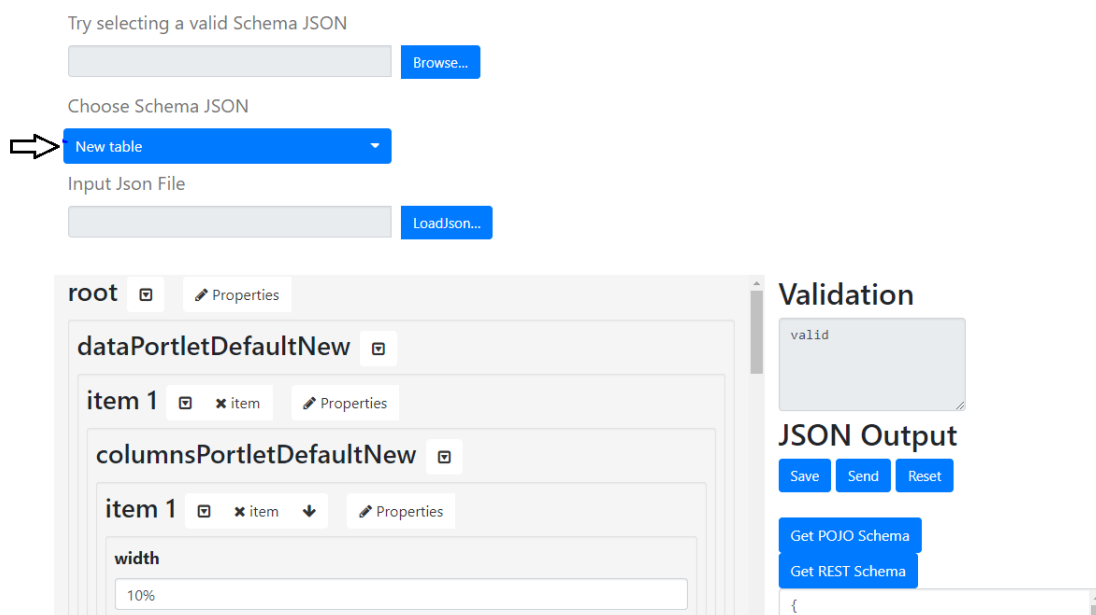


Figure 25 : saisit les entêtes du tableau

Ensuite, on saisit les entêtes de notre tableau, et on clique sur le bouton send (les entêtes du tableau doivent avoir les mêmes noms que les colonnes du POJO concerné).

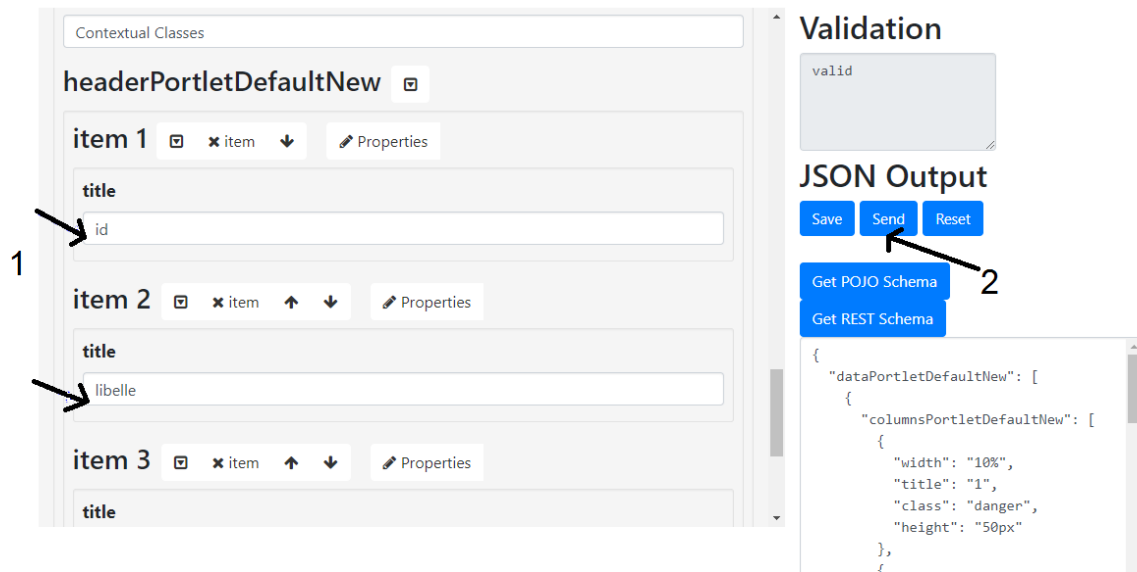


Figure 26 : Buttons send ()

Lors du clique en send button, le fichier JSON Output (cf. Figure 24) s'enregistre dans le projet Flex et précisément dans l'emplacement de projet Flex (Template) dans les serveurs de l'entreprise.

1.4. Création du formulaire

De la même manière on crée le formulaire avec les mêmes données qu'on veut modifier.

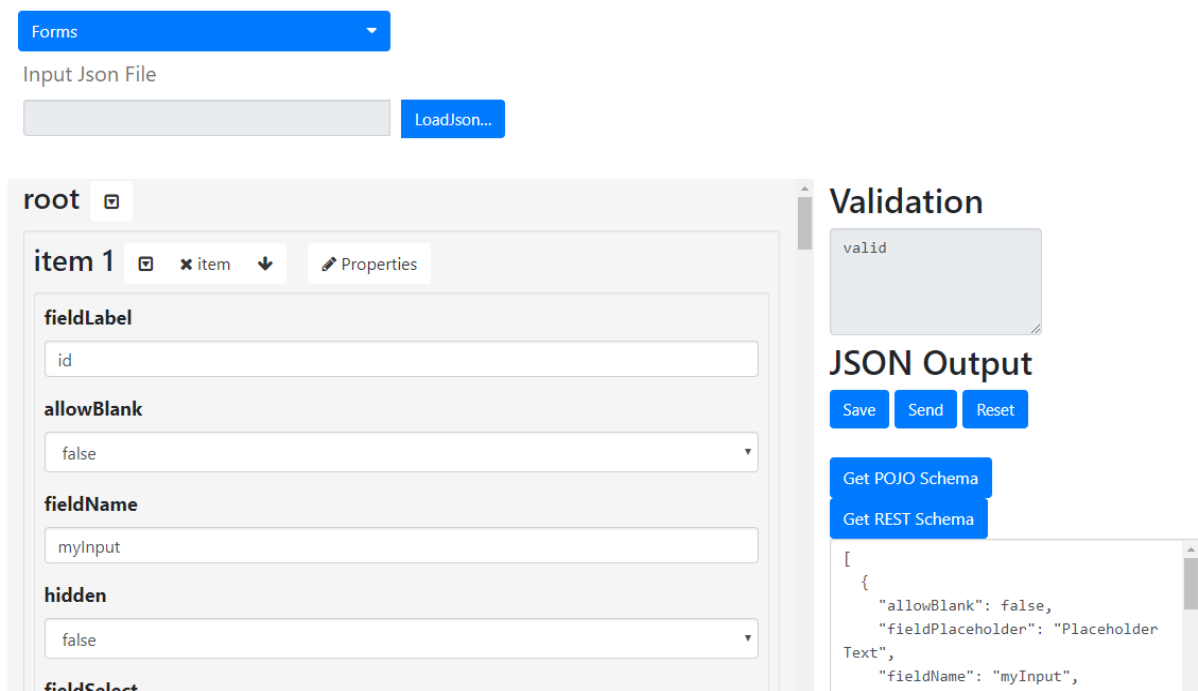


Figure 27 : Composante Forms

2. Coté serveur (Back-End)

-La deuxième étape est de créer le POJO, pour cela on clique sur le button **Get POJO Schema** pour importer le schéma.

2.1. Schéma de POJO

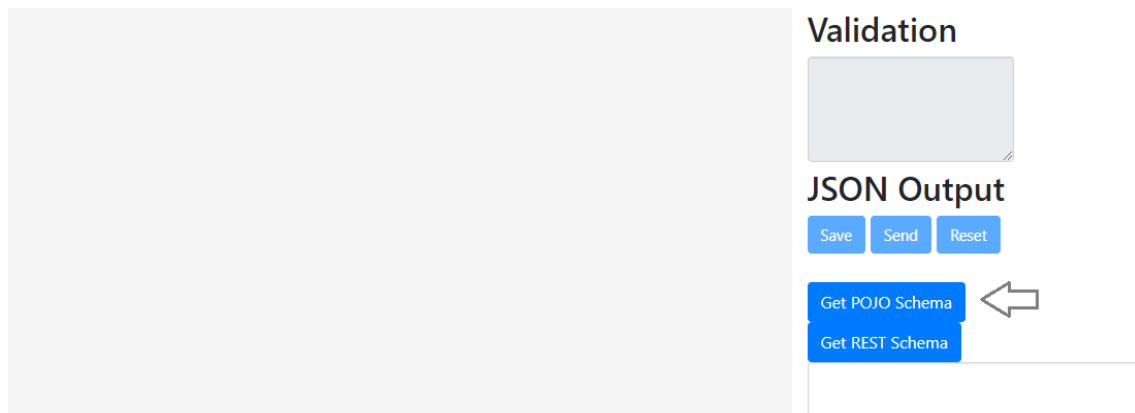


Figure 28 : Création de POJO

-Ensuite, on saisit les données de notre POJO (nom du POJO : Produit, les attributs avec leurs annotations JPA) :

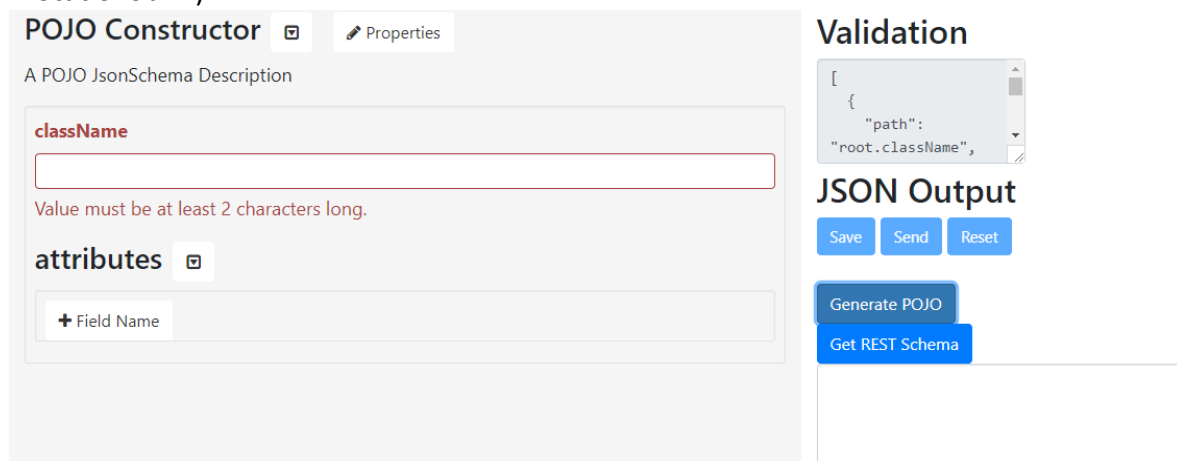


Figure 29 : Champs de Schéma POJO

2.2. Saisie des attributs

+ Attribut libelle

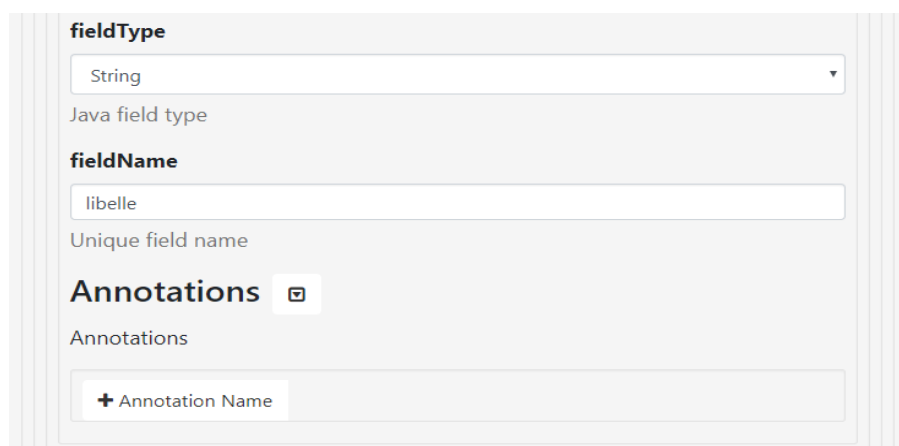
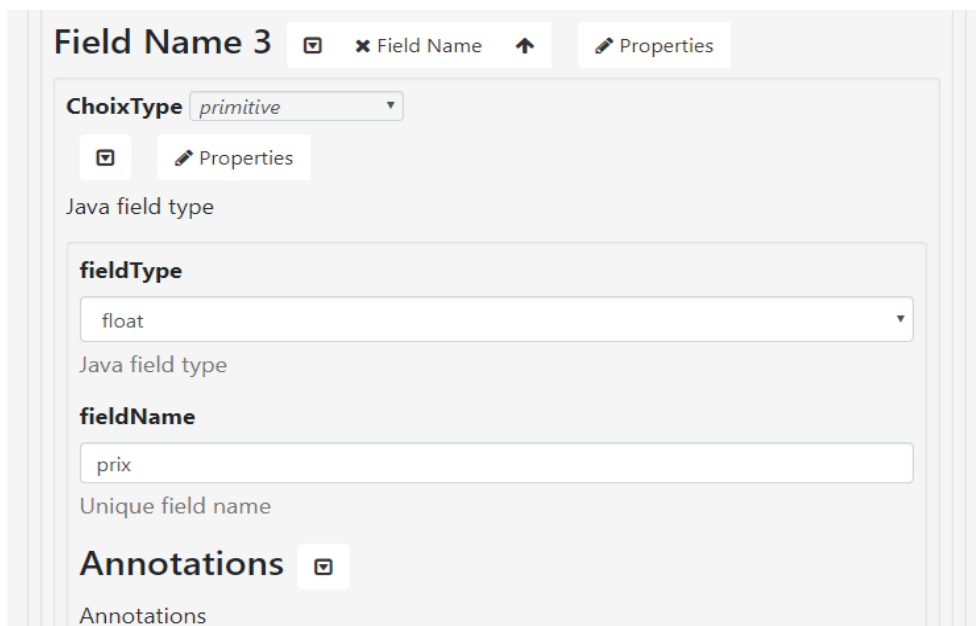


Figure 30 : Attribut libelle

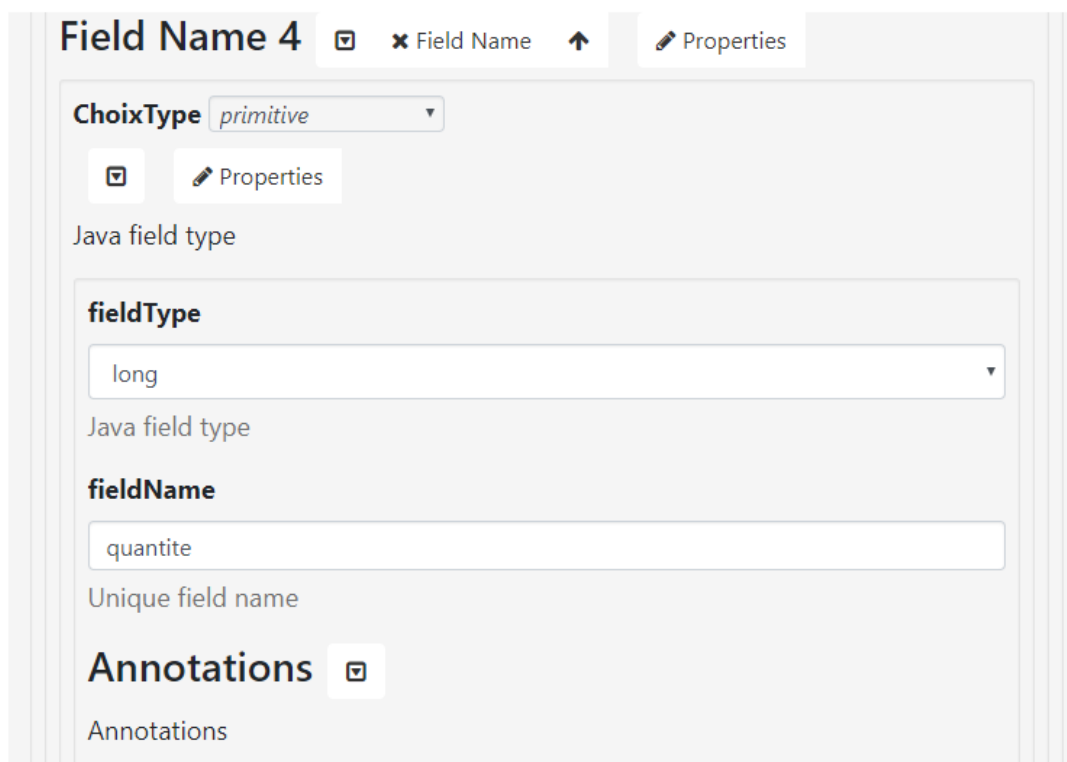
+ Attribut Prix



The screenshot shows the configuration interface for 'Field Name 3'. At the top, there are tabs for 'Field Name' and 'Properties'. The 'Field Name' tab is active. Below the title, there is a 'ChoixType' dropdown menu set to 'primitive'. Below this, there is a 'fieldType' dropdown menu set to 'float'. Below that, there is a 'fieldName' text input field containing the text 'prix'. At the bottom, there is an 'Annotations' section with a checkbox icon.

Figure 31 : Attribut prix

+ Attribut quantite

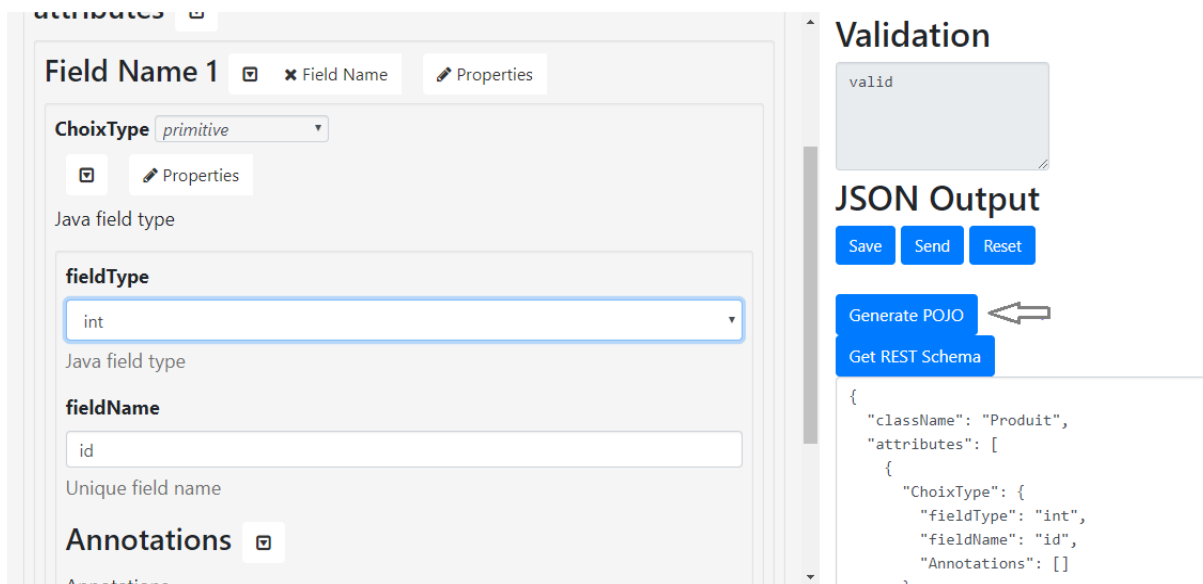


The screenshot shows the configuration interface for 'Field Name 4'. At the top, there are tabs for 'Field Name' and 'Properties'. The 'Field Name' tab is active. Below the title, there is a 'ChoixType' dropdown menu set to 'primitive'. Below this, there is a 'fieldType' dropdown menu set to 'long'. Below that, there is a 'fieldName' text input field containing the text 'quantite'. At the bottom, there is an 'Annotations' section with a checkbox icon.

Figure 32 : Attribut quantité

2.3. Génération de POJO

-A la fin on clique sur le button : **Generate POJO** (cf. Figure 33).



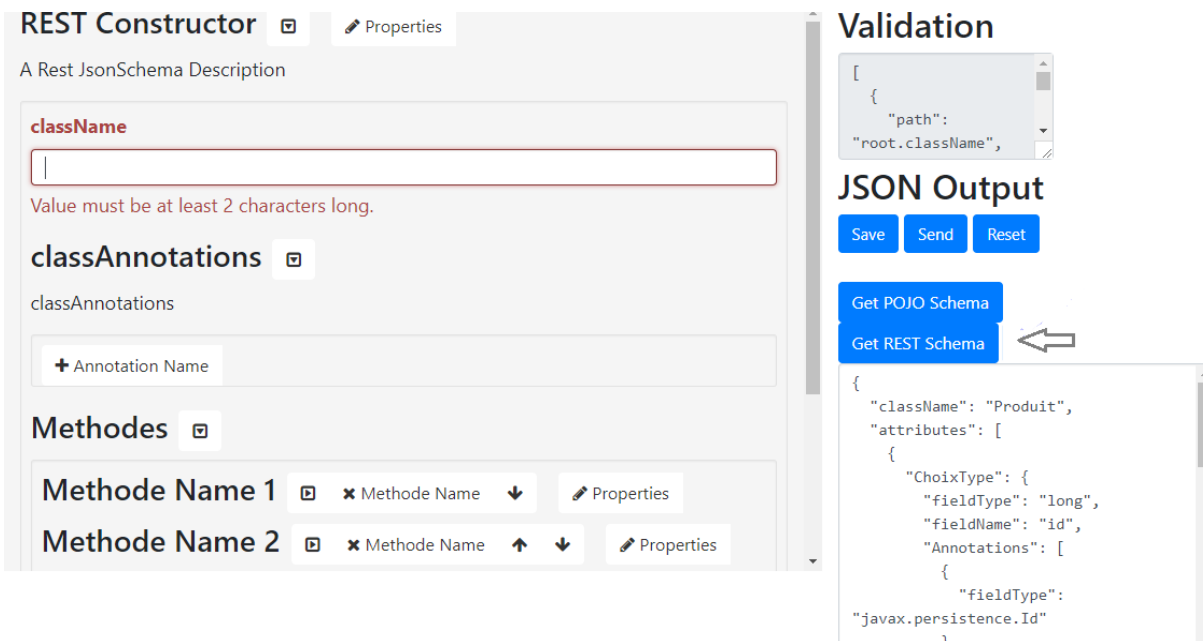
The screenshot shows the 'Field Name 1' configuration panel on the left and the 'JSON Output' panel on the right. In the 'Field Name 1' panel, the 'ChoixType' is set to 'primitive', the 'fieldType' is 'int', and the 'fieldName' is 'id'. The 'JSON Output' panel shows a JSON schema for a class named 'Produit' with an attribute 'id' of type 'int'. A blue arrow points from the 'Generate POJO' button to the 'JSON Output' panel.

Figure 33 : Génération POJO ()

Le POJO s'enregistre dans le package : `org.jsoneditor.model+ {username}`

-La troisième étape est de créer le service Rest

Pour cela on clique sur le button : *Get REST Schema*, des méthodes par défaut (getAll, getByld, upDate, delete, add) sont déjà définies.



The screenshot shows the 'REST Constructor' configuration panel on the left and the 'JSON Output' panel on the right. In the 'REST Constructor' panel, the 'className' is empty, 'classAnnotations' is empty, and 'Methodes' are listed as 'Methode Name 1' and 'Methode Name 2'. The 'JSON Output' panel shows a JSON schema for a class named 'Produit' with an attribute 'id' of type 'long'. A blue arrow points from the 'Get REST Schema' button to the 'JSON Output' panel.

Figure 34 : REST schéma

Ensuite, ouvrir les *collapseds* pour remplir les champs manquées (cf. Figure 33).

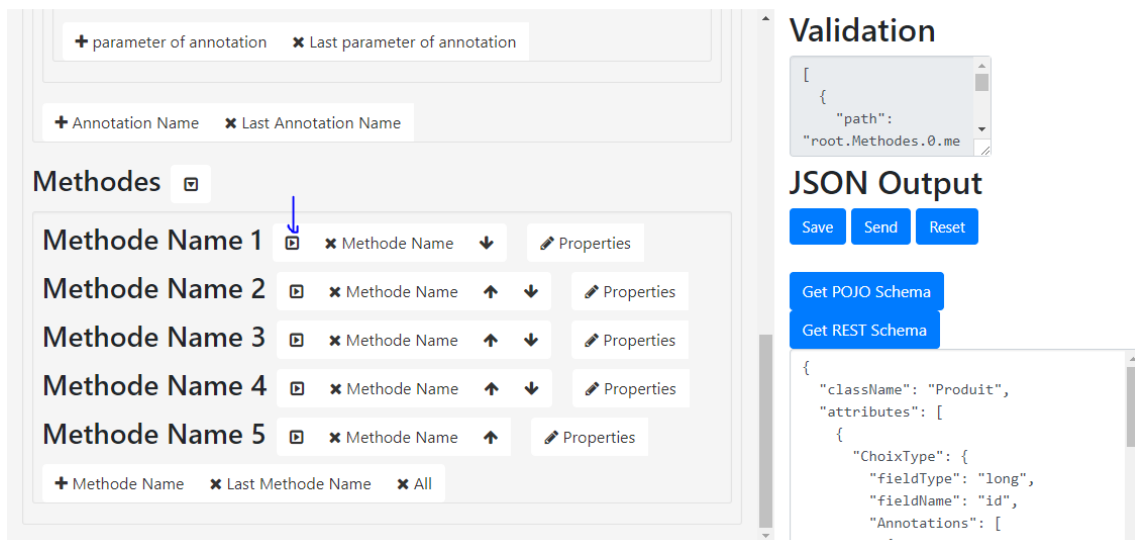


Figure 35 : Saisie Méthode de REST-1

-Saisir les données des méthodes (cf. Figure 34) :

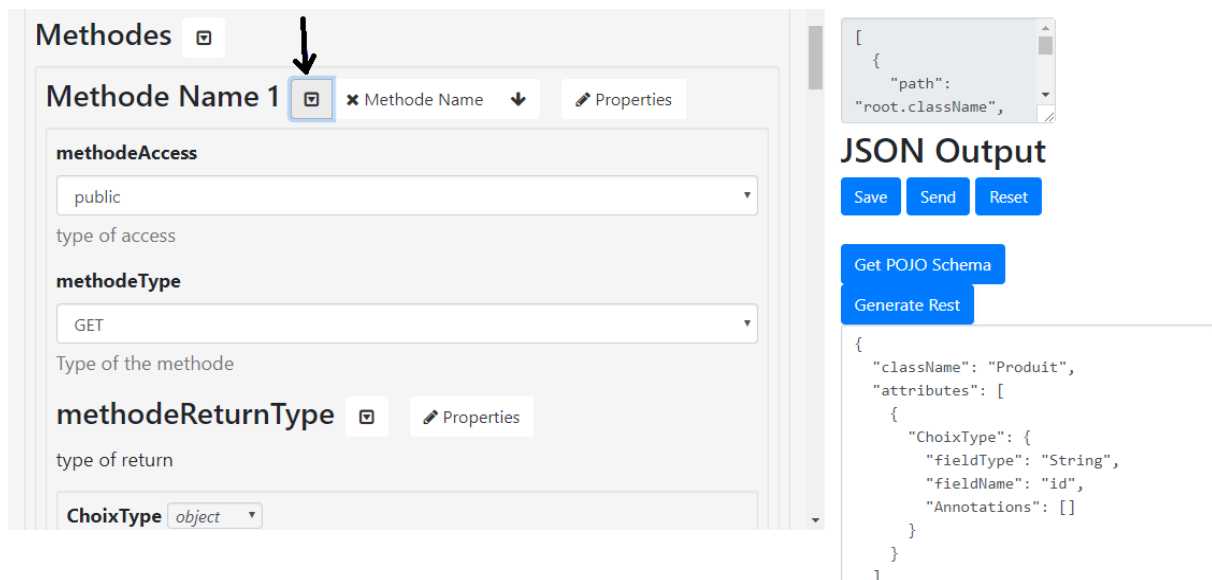


Figure 36 : Saisie Méthodes REST-2

Si les données sont validées cliquer sur le button : **Generete Rest** (cf.Figure.35).

Le Service se génère dans le package : `org.jsoneditor.web+ {username}` et se déploie dans les serveurs de l'entreprise.

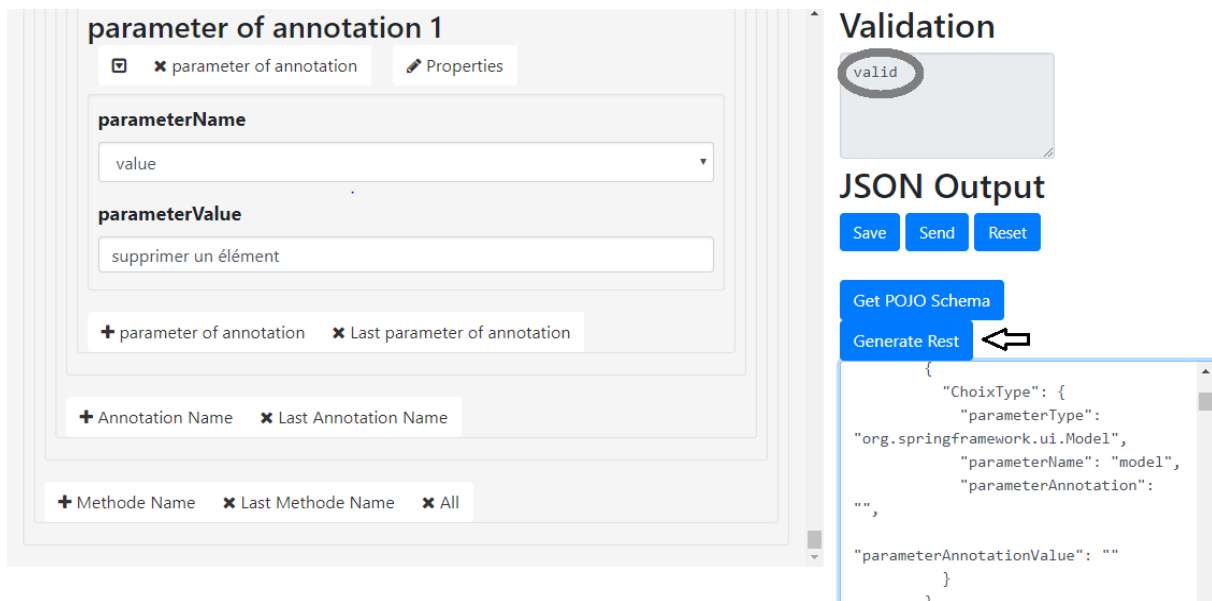


Figure 37 : Génération de REST Controller

Si le Service Rest est généré, le message suivant s'apparu (cf. Figure 36) :

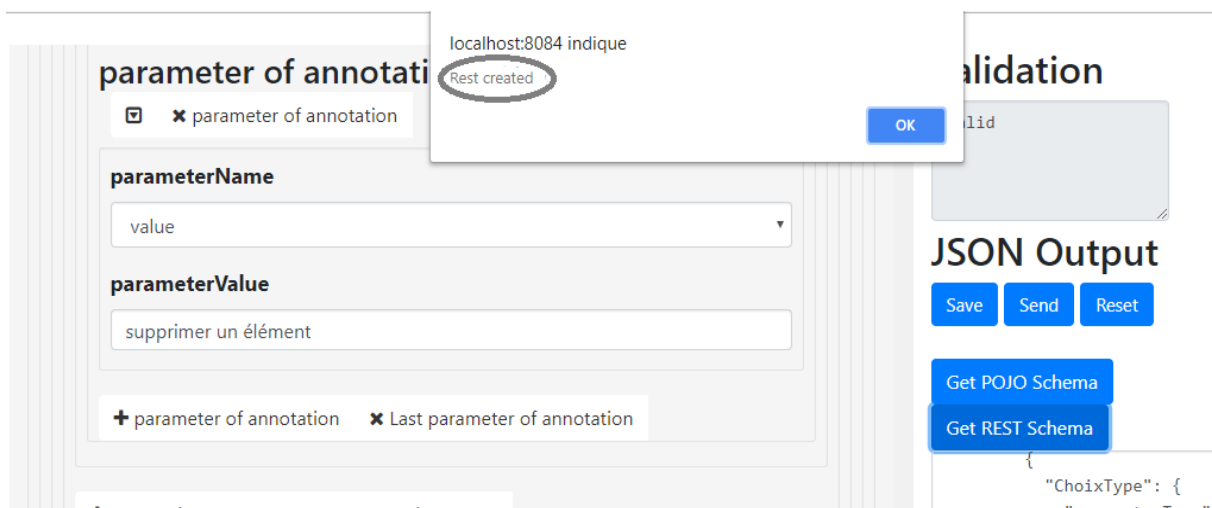


Figure 38 : Message de confirmation

La dernière étape est générée le client TS à l'aide de swagger en exécutant une commande avec une simple configuration.

(Swagger est un standard de description d'APIs REST, afin de rendre accessible et compréhensible de tous (machines et humains) les services mis à disposition et la façon de les utiliser sans avoir accès au code source).

⇒ Le résultat doit être comme suit (cf. Figure 37) :

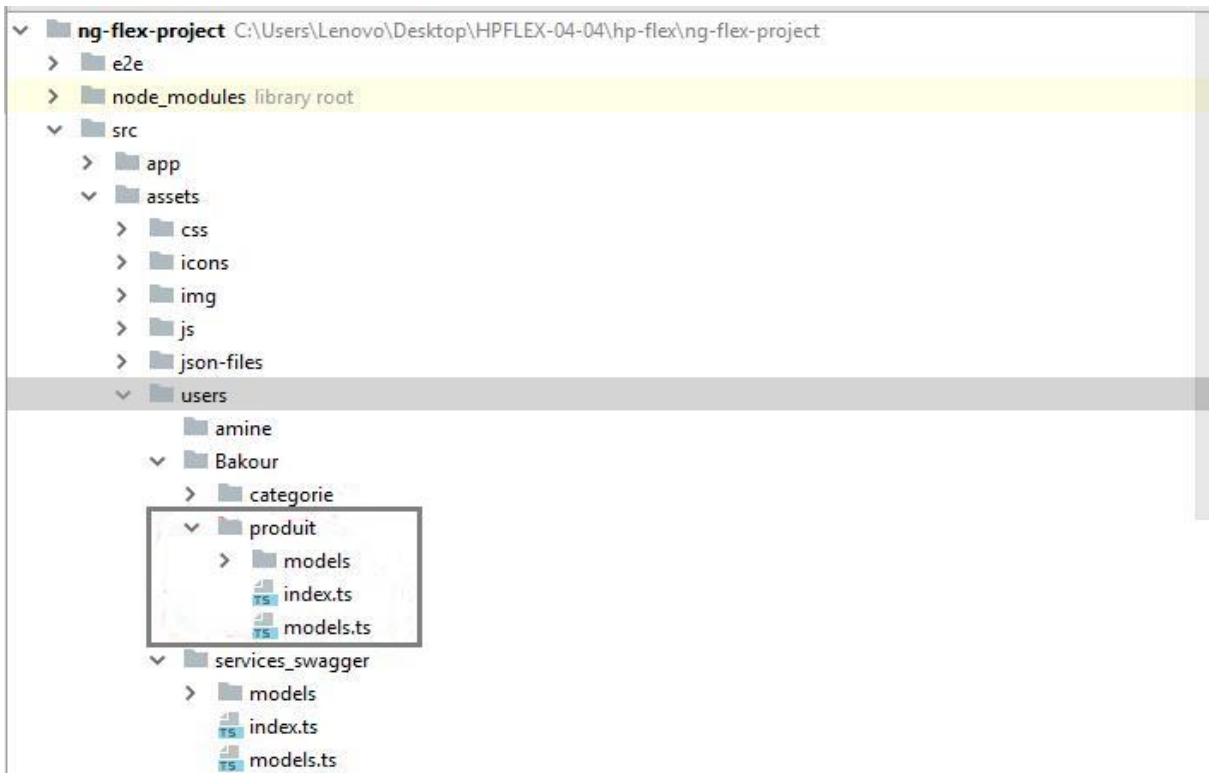


Figure 39 : Client TS –REST Client

3. Application web générée

Gestion de produit

3.1. Page authentification :

La page d'authentification est créée et personnalisée grâce au JSON Editor aussi, ici j'ai donné le résultat directement dans l'application web générée.

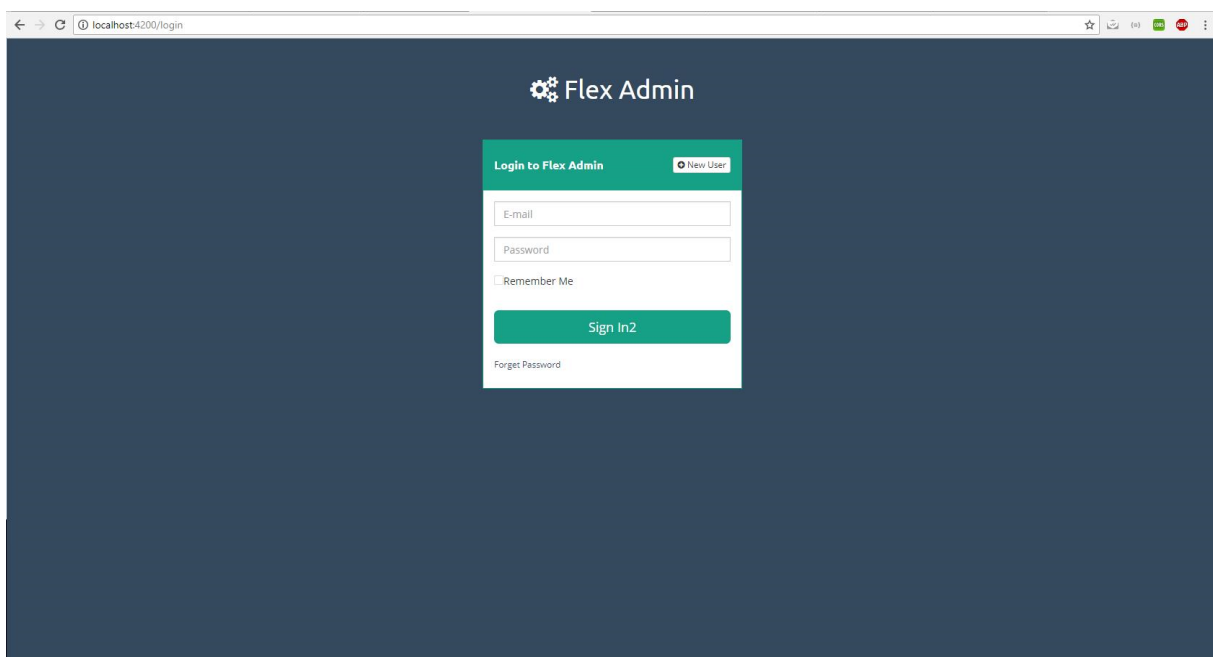


Figure 40 : Page d'authentification d'application web générée

Les identifiants sont les même que dans JSON Editor.

3.2. Page d'accueil

L'utilisateur va trouver tous les composants qui lui concernent

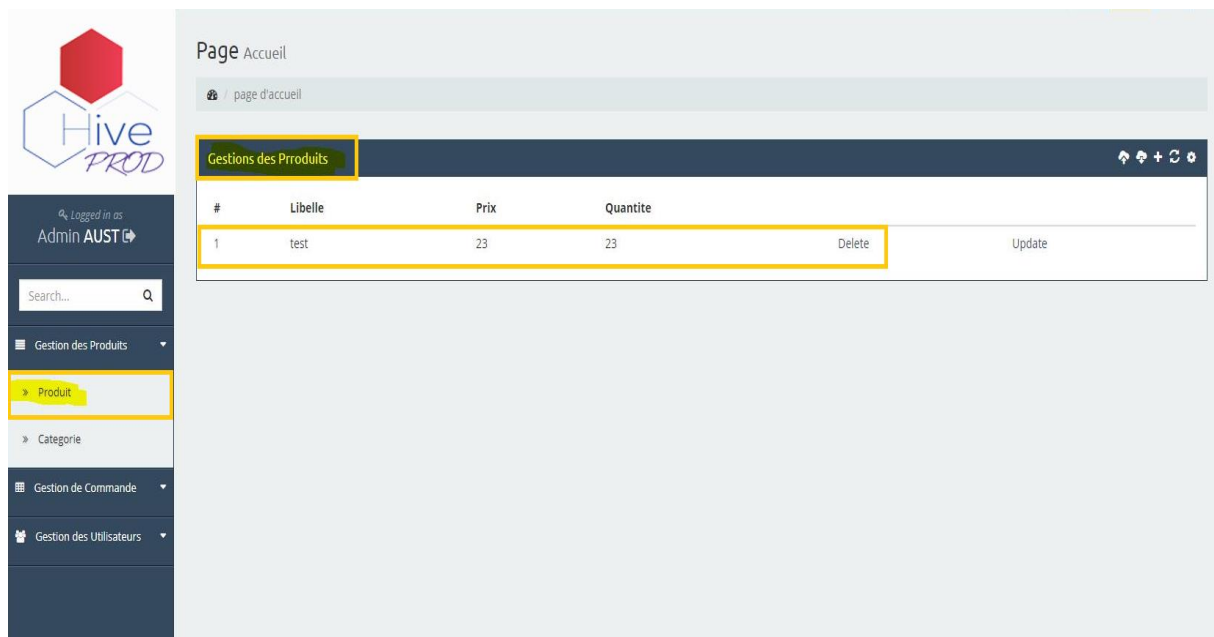


Figure 41 : Gestion produit - menu gauche

On clique sur l'item Gestion des Produits dans le menu gauche :

- ✓ On va avoir deux subMenu « Categorie et Produit », et chaque subMenu va routé le composant concerné .
- ✓ Le Tableaux affiché contient la liste des produits rcuperer depuis le pojo produit on utilisant le client swagger generer .

3.3. Ajout/ suppression un Produit :

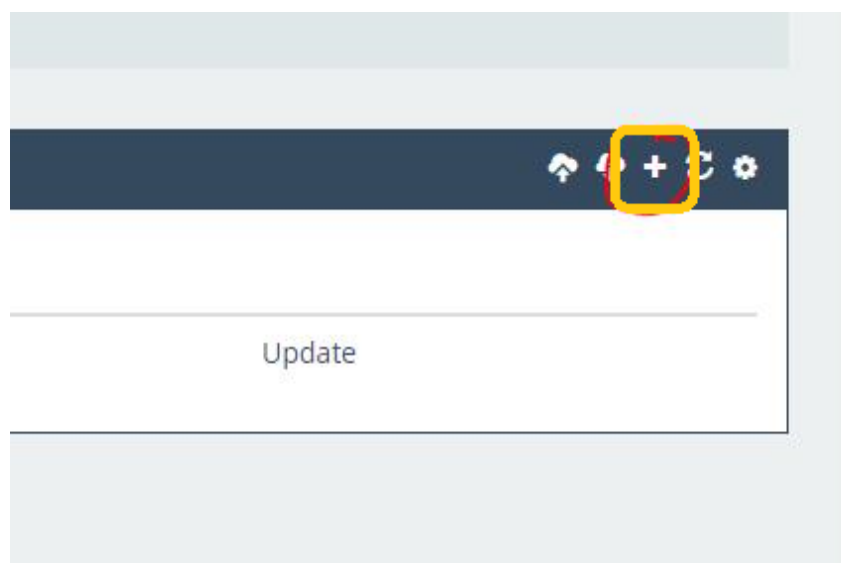
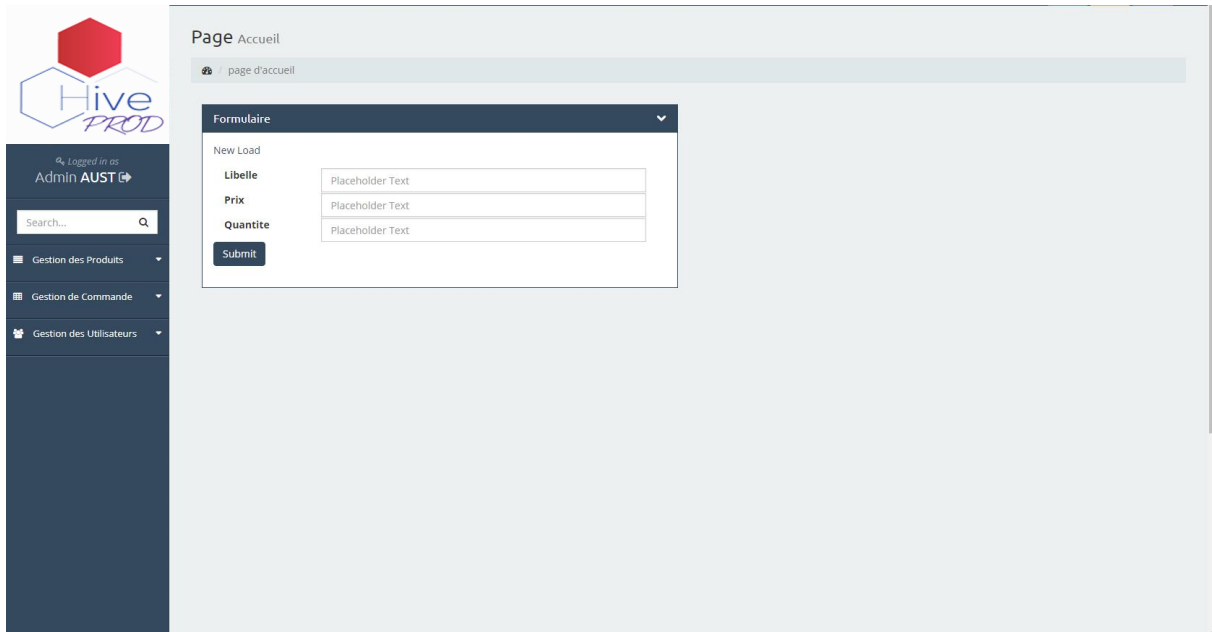


Figure 42 : Ajouter un produit

- ⇒ Add : ajouter un produit
- ⇒ Delete : supprime la ligne depuis la base .
- ⇒ update : modifie le produit sélectionné
- ⇒ Formulaire pour ajouter et modifier :



The screenshot displays the 'Hive PROD' application interface. On the left is a dark blue sidebar with the 'Hive PROD' logo at the top, followed by the user 'Admin AUST' and a search bar. Below the search bar are three menu items: 'Gestion des Produits', 'Gestion de Commande', and 'Gestion des Utilisateurs'. The main content area has a light gray background. At the top of this area, it says 'Page Accueil' and 'page d'accueil'. A modal window titled 'Formulaire' is open, containing a 'New Load' section with three input fields labeled 'Libelle', 'Prix', and 'Quantite', each with a 'Placeholder Text'. A 'Submit' button is located at the bottom of the form.

Figure 43 : Formulaire pour ajouter et modifier

Conclusion et perspectives

Ce projet consiste en la réalisation d'un générateur d'applications web permettant d'améliorer la productivité et réduire le temps de développement en transformant chaque couche d'une application en une architecture générique.

On a pu tirer profit de la force de la généricité et la réflexivité, deux concepts qui nous ont permis d'offrir des modules indépendants et réutilisables dans chaque projet web réalisé en java.

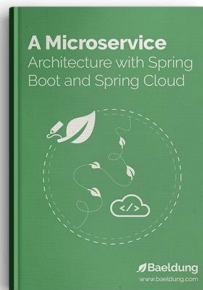
Tout en misant sur l'agilité, et avec la méthodologie Scrum, nous avons pu réaliser les différentes fonctionnalités du backlog les plus prioritaires pour offrir un produit fonctionnel.

Le stage a été pour moi une occasion de découvrir l'environnement professionnel et de s'habituer aux méthodes de travail dans le monde de l'entreprise, au sérieux ainsi qu'à la dynamique de groupe. Techniquement, j'ai pu avoir de nombreuses formations me permettant de maîtriser les différentes technologies et les utiliser à bon escient pour mener à bien ce projet.

En outre, ce travail effectué jusque-là ne représente que 75% du travail demandé. D'autres fonctionnalités comme l'intelligence artificielle seront ajoutées pour enrichir ce projet.

Bibliographie & Webographie

Bibliographie



[1]. Spring Boot and Spring Cloud with Baeldung- 2017.



[2]. RESTful Java Patterns and Best Practices 2014.

Webographie :



<https://www.json-schema.org>



<https://www.youtube.com/channel/UCB12jjYsYv-eipCvBDcMbXw>



<https://dzone.com>

Annexe 1

Génération d'un Template html avec les variables Angular dynamiques depuis un input qui contient seulement du html

Chapitre 2

- On génère un Template html avec les variables angular dynamiques depuis un input qui contient seulement du html :

Méthode on travaille avec les Node(Element) XPATH :

- Au premier lieu, faire un copier-coller du contenu html qu'on veut régénérer dans fichier "InputFile.html". on va ouvrir le fichier inputFile on mode lecture, puis on a créé une liste des éléments qui va contenir tous les éléments de puis la racine body

```
NodeList nList = newDoc.getElementsByTagName ("body");
```

- Deuxièmement, Pour la récupération des attributs et leurs valeurs on a opté pour fonction qui accepte en paramètre une liste des éléments

```
VisitChildNodes (nList) ;
```

Le traitement ce fait sur le type d'élément, si il a des attributs il les ajoutes a une liste des attributs, si l'élément a des élément comme fils un refait le même traitement sinon il ajoute à la liste le Texte , donc vers la fin on a une liste qui contient tous les attribut et leurs valeurs et texte qui sont modifiable. Avec cette liste, on a créé le fichier JSON "output.json" qui va nous servir pour la récupération des données.

- Finalement, pour l'affichage on a utilisé la même liste des attributs et la liste des éléments

Exemple :

```
<div class="col-lg-2 col-sm-6" >
  <div class="circle-tile" >
    <a href="#" >
      <div class="circle-tile-heading green" >
        <i class="fa fa-money fa-fw fa-3x" ></i>
      </div>
    </a>
    <div class="circle-tile-content green" >
      <div class="circle-tile-description text-faded" >
        Revenue
      </div>
      <div class="circle-tile-number text-faded" >
        $32,384
      </div>
      <a class="circle-tile-footer" href="#" >
        <i class="fa fa-chevron-circle-right" ></i>
      </a>
    </div>
  </div>
</div>
```

Figure 44 : Input HTML

⇒ HTML généré :

```

template.html
1 <body class="{{list.class0}}" >
2   <div class="{{list.class1}}" >
3     <div class="{{list.class2}}" >
4       <h4>
5         <strong>{{list.strong3}}</strong>
6       </h4>
7     </div>
8     <div class="{{list.class4}}" >
9       <button class="{{list.class5}}" >{{list.button6}}
10      <i class="{{list.class7}}" ></i>
11    </button>
12  </div>
13  <div class="{{list.class8}}" ></div>
14 </div></body>

```

Figure 45 : HTML généré

⇒ JSON généré :

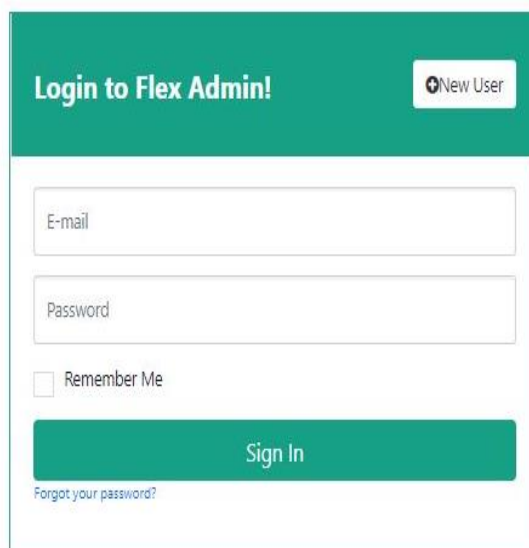
```

{} *output.json
1 {
2   "href12": "#",
3   "div10": "$32,384",
4   "a14": "More Info",
5   "href3": "#",
6   "div8": "Revenue",
7   "class9": "circle-tile-number text-faded",
8   "class7": "circle-tile-description text-faded",
9   "class6": "circle-tile-content green",
10  "class5": "fa fa-money fa-fw fa-3x",
11  "class4": "circle-tile-heading green",
12  "class2": "circle-tile",
13  "class1": "col-lg-2 col-sm-6",
14  "class0": "",
15  "class11": "circle-tile-footer",
16  "class13": "fa fa-chevron-circle-right"
17 }

```

Figure 46 : JSON généré

⇒ Composant généré :



The image shows a login form titled "Login to Flex Admin!". It features a green header bar with the title and a "New User" button. Below the header, there are two input fields for "E-mail" and "Password". A "Remember Me" checkbox is located below the password field. A large green "Sign In" button is at the bottom, with a link "Forgot your password?" underneath it.

Figure 47 : Composant généré

Annexe 2

Génération des classes JAVA à partir de JSON Editor

Chapitre 2

J'ai utilisé le plugin JAVA "org.json.simple" pour récupérer les données de fichier JSON générer à partir de JSON editor pour générer le POJO avec ses attributs.

Exemple de méthodes utilisées de ce plugin pour lire et extraire les valeurs à partir de fichiers JSON.

```
private void getting_POJO_conf_from_JSON(final JavaClassSource source, JSONObject js
    String name = (String) jsonObject.get("className");
    name = name.substring(0, 1).toUpperCase() + name.substring(1).toLowerCase();
    System.out.println(name);

    String key = ""; // Type Java
    String value = ""; // Instance
    String Objecte = null; // Type class
    boolean foundId=false;

    JSONArray attributes = (JSONArray) jsonObject.get("attributes");
    for (Object attr : attributes) {
        System.out.println(attr + "");
    }
```

Figure 48 : Controller JSON Editor GenPOJO

Exemple :

⇒ JSON généré à partir de JSON Editor :

```
{
  "className": "Personne",
  "attributes": [
    {
      "fieldType": "String",
      "fieldName": "nom",
      "Annotations": []
    },
    {
      "fieldType": "String",
      "fieldName": "prenom",
      "Annotations": []
    },
    {
      "fieldType": "int",
      "fieldName": "age",
      "Annotations": []
    },
    {
      "fieldType": "String",
      "fieldName": "address",
      "Annotations": []
    }
  ]
}
```

Figure 49 : Valeurs JSON d'un POJO

⇒ POJO généré :

```
public class Personne
{
    private String nom;
    private String prenom;
    private int age;
    private String address;

    public String getNom()
    {
        return nom;
    }

    public void setNom(String nom)
    {
        this.nom = nom;
    }

    public String getPrenom()
    {
        return prenom;
    }

    public void setPrenom(String prenom)
    {
        this.prenom = prenom;
    }
}
```

Figure 50 : POJO généré

Sécurité des services web avec Swagger et JWT

CHAPITRE 2

Configuration de Swagger et Spring security :

```
final AuthorizationScope[] authorizationScopes = new AuthorizationScope[3];
authorizationScopes[0] = new AuthorizationScope("read", "read all");
authorizationScopes[1] = new AuthorizationScope("trust", "trust all");
authorizationScopes[2] = new AuthorizationScope("write", "write all");

return Collections.singletonList(new SecurityReference("oauth2", authorizationScopes));
}

@Bean
public SecurityConfiguration security() {
    return new SecurityConfiguration
        ("client", "secret", "", "", "Bearer access token", ApiKeyVehicle.HEADER, HttpHeaders.AUTHOR
}

private ApiInfo apiInfo() {
    return new ApiInfoBuilder().title("Authentication API").description("build 18-Apr-2018")
        .contact(new Contact("Badr Kacimi", "www.hiveprod.com", ""))
        .license("hiveprod")
        .licenseUrl("\\https://www.apache.org/licenses/LICENSE-2.0")
}
```

Figure 51: Swagger configuration de sécurité

```

@Override
public void configure(ClientDetailsServiceConfigurer clients) throws Exception {
    clients
        .inMemory()
        .withClient("client")
        .authorizedGrantTypes("client_credentials", "password", "refresh_token", "authorization_code")
        .scopes("read", "write")
        .resourceIds("oauth2-resource")
        .accessTokenValiditySeconds(expiration)
        .refreshTokenValiditySeconds(expiration)
        .secret("secret");
}

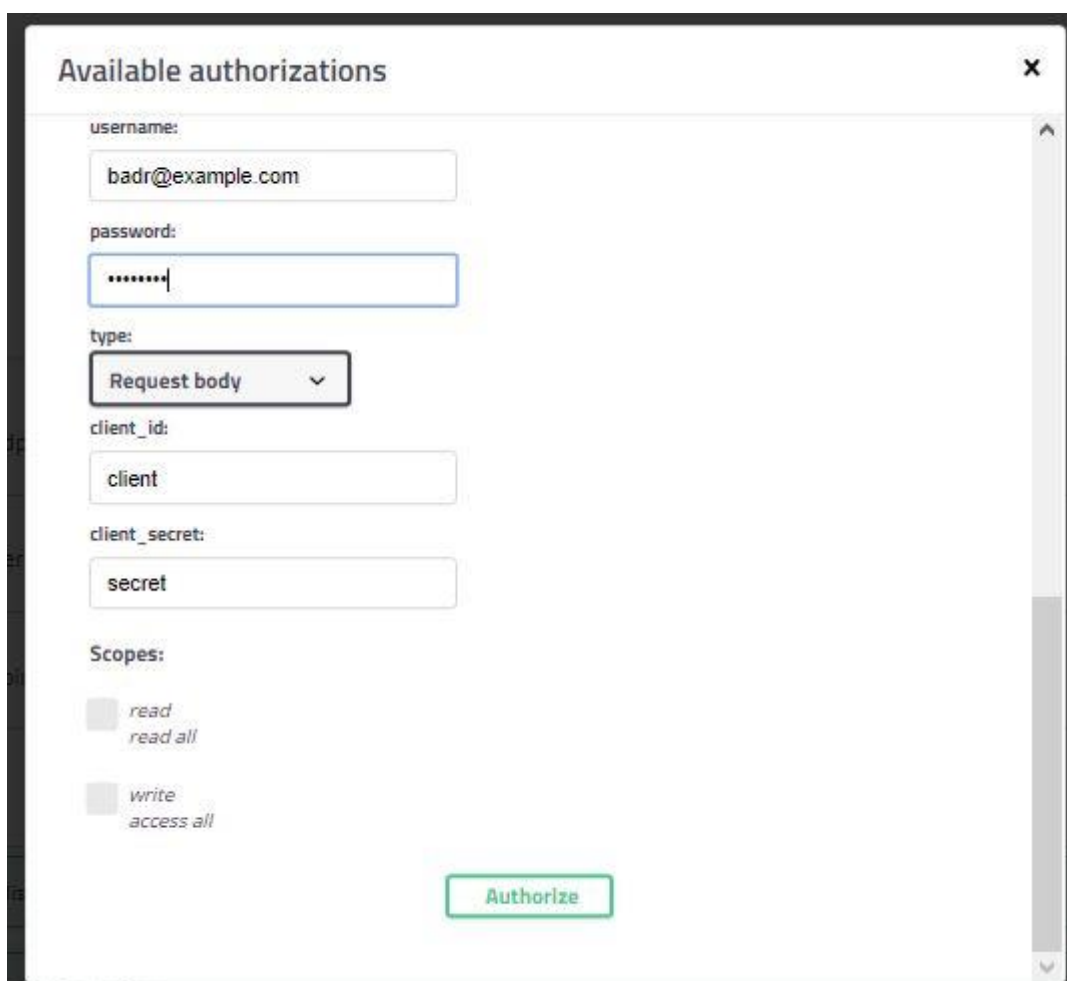
@Bean
public JwtAccessTokenConverter accessTokenConverter() {

    LOGGER.info("Initializing JWT with public key: " + publicKey);
}

```

Figure 52 : Authorization server

⇒ Pour accéder aux ressources de JSON Editor, il faut s'authentifier



The image shows a Swagger authentication dialog box titled "Available authorizations". It contains the following fields and options:

- username:** A text input field containing "badr@example.com".
- password:** A password input field with masked characters "*****".
- type:** A dropdown menu currently set to "Request body".
- client_id:** A text input field containing "client".
- client_secret:** A text input field containing "secret".
- Scopes:** A section with two radio button options:
 - ☐ read read all
 - ☐ write access all
- Authorize:** A green button at the bottom right.

Figure 53 : Swagger authentication

