

المملكة المغربية



المندوبية السامية للتخطيط

ⵜⴰⵎⴷⴰⵢⵜ ⵜⴰⵎⴰⵍⴷⴰⵢⵜ ⵜⴰⵏⵔⴰⵢⵜ | ⵙⴰⵎⴰⵢⵜ ⵜⴰⵏⵔⴰⵢⵜ

HAUT-COMMISSARIAT AU PLAN

ROYAUME DU MAROC

*_*_*_*_*_*

HAUT COMMISSARIAT AU PLAN

*_*_*_*_*_*

INSTITUT NATIONAL

DE STATISTIQUE ET D'ECONOMIE APPLIQUEE



INSEA

Institut National de Statistique et

d'Economie Appliquée

Projet de Fin d'Etudes

Organisme d'accueil : HIVEPROD



Développement d'un générateur d'application basé sur des web services

Préparé par : **M. BADR KACIMI**

Sous la direction de : **Mme. Rajaa SAIDI (INSEA) (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 : INFORMATIQUE

Devant le jury composé de :

- **Mme. Rajaa SAIDI (INSEA) (INSEA)**
- **M. /Mme Prénom et Nom (INSEA)**
- **M. ZOULAY Abderrahim (Hiveprod)**

Juin 2018 / PFE **N°**

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.

Je t'offre ce modeste travail pour te remercier pour tes sacrifices et pour l'affection dont tu m'as toujours entourée

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



Remerciements

Je tenais 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 tenais à remercier mon encadrante a l'INSEA Mme :Saidi Rajae et Mr : Selmane SENTISSI (CEO) et Abderahim Zoulay (responsable R&D) et Ayoub Bakour à 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 travail et de l'enrichir par leurs propositions.

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

Liste des tableaux

Tableau 1 : Temps réel de la suite Fibonacci	19
Tableau 2 : Description textuelle du cas d'utilisation « Cote client Front-End»	25
Tableau 3 : Description textuelle du cas d'utilisation « Coté serveur Back-End»	25

Liste des figures

Figure 1 : Domaine d'activités	13
Figure 2 : Organigramme de l'organigramme	14
Figure 3 : Pile de modélisation (OMG)	16
Figure 4 : Le processus Scrum	17
Figure 5: Sprints	21
Figure 6 : user story <<User Management>>	21
Figure 7 : Burndown Chart	22
Figure 8 : use case << Générateur d'application web>>	24
Figure 9 : Génération de JSON Schéma- Meta data	28
Figure 10 : Scénario globale de génération de Front-end d'une application web	29
Figure 11 : Extrait de JSON Schéma (schéma descriptif de POJO)	31
Figure 12 : Scénario de génération d'un POJO	31
Figure 13 : Approche du style d'architecture REST	32
Figure 14 : JWT Authentification	33
Figure 15 : Schéma technique du générateur	35
Figure 16 : Schéma technique d'interaction avec l'application web générée	35
Figure 17 : Template Flex	43
Figure 18 : Page d'authentification	43
Figure 19 : JSON Editor – page accueil	44
Figure 20 : Menu gauche(SideNavDropDown)	44
Figure 21 : SubMenu	45
Figure 22 : saisit les entêtes du tableau	45
Figure 23 : Buttons send ()	46
Figure 24 : Composante Forms	46
Figure 25 : Création de POJO	47
Figure 26 : Champs de Schéma POJO	47
Figure 27 : Attribut libelle	48
Figure 28 : Attribut prix	48
Figure 29 : Attribut quantité	49
Figure 30 : Génération POJO ()	49
Figure 31 : REST schéma	50

Figure 32 : Saisie Méthode de REST-1	50
Figure 33 : Saisie Méthodes REST-2	51
Figure 34 : Génération de REST Controller.....	51
Figure 35 : Message de confirmation	52
Figure 36 : client TS –REST Client	52
Figure 37 : Page d'authentification d'application web générée	53
Figure 38 : Gestion produit - menu gauche	54
Figure 39 : Ajouter un produit.....	54
Figure 40 : Formulaire pour ajouter et modifier.....	55

Liste d'abréviations

Abréviation	Désignation
API	Application Programming Interface
EJB	Entreprise JavaBean
HTML	HyperText Markup Language
IDE	Integrated Development Environment
IHM	Interface Homme Machine
JAVA EE	Java Enterprise Edition
DOM	Document Object Model
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
AWS	Amazon Web Services
TS / JS	Typescript / JavaScript
CSS	Cascading Style Sheets
WAR	Web application Archive

Table des matières

Table des matières

Dédicace	3
Remerciements	4
Liste des tableaux	5
Liste des figures	5
Liste des abréviations	6
Résumé	7
Introduction générale	10
Chapitre 1 : Contexte général du projet	12
1. Organisme d'accueil	13
1.1 Présentation	13
1.2 Domaines d'activités	13
1.3 Organigramme et portefeuille de produits	14
2. Définition de projet	15
2.1 Problématique et motivation	13
2.2 Objectif de projet	13
3. Conduite de projet	14
3.1 Présentation de Scrum	14
3.2 Utilisation Scrum dans notre contexte	15
Conclusion	19
Chapitre 2 : Génération back-end et Front-end	23
1. Analyse de l'existence	21
2. Spécification fonctionnel de Projet	21
3. Génération coté client –Front End	22
1.1 Description	23
1.2 Approche classique	24
1.3 Approche adoptée	24
1.3.1 Description	25
1.3.1 Scenario	26
2. Génération coté serveur –Back End	27
2.1 Définition	27
2.2 Approche adoptée	28s
1.3.1 Pattern MVC	28
1.3.1 Sécurité des web services	29
3. Schémas technique globale	31
Conclusion	32
Chapitre 3 : Réalisation	37
Template Flex	33
1. Coté client –Front End	34
1.1 Page d'authentification-JSON Editor	35
1.2 Page accueil-JSON Editor	35
1.3 Création de Menu	36
1.4 Création de sous-Menu	36
1.5 Création d'un tableau	37
1.6 Création d'un tableau	37
1.7 Génération cote client	38
2. Coté client –Back End	39
2.1 Création de POJO	39
2.2 Champs de Schéma POJO	40
2.3 Création de Menu	40
2.4 Attribut libelle	40
2.5 Attribut quantité	43
2.6 Génération POJO	43

2.7 Création de REST Controller	44
2.8 Champs de Schéma POJO	44
2.9 Création de REST Controller	44
2.10 Génération client TS (REST client).....	45
2.11 Page d'authentification de l'application générée.....	46
2.12 Gestion produit.....	46
2.13 Ajout d'un produit	47
2.14 Formulaire pour modification/création de produit	47
Chapitre 4 : Technologie et outils	49
Conclusion et perspectives	53
Bibliographie et webographie	54
Annexe.....	55

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, nous avons suivi la méthodologie SCRUM conjointement. Après avoir mis le projet dans son contexte, nous avons 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.

Nous avons 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, nous avons illustré un exemple complet de génération d'une application web.

Le résultat final de notre travail est une d'une part des couches génériques indépendantes pouvant être utilisées dans toute application web pour éviter la répétition de code, et d'autre part l'analyse de la possibilité d'intégration des solutions basées sur l'intelligence artificielle afin de minimiser de plus l'intervention humaine dans le développement et la génération des applications web.

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 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 problématique et les objectifs du projet. Ainsi la conduite de 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 implémentes dans 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 des détails de générateur.

Chapitre **1** : Contexte général 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 chapitres :

- 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.

Chapitre 1 : Contexte général du projet

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

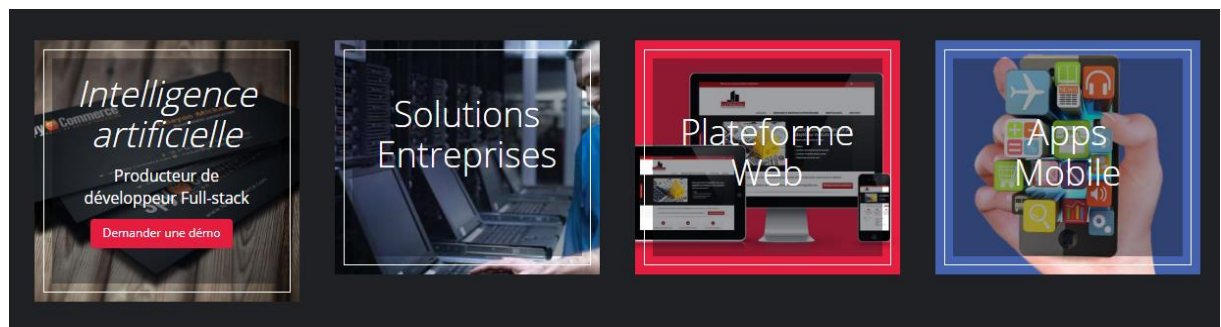


Figure 1 : Domaine d'activités

1.3. Organigramme et portefeuille de produits

HIVEPROD est, administrativement, un corps 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.

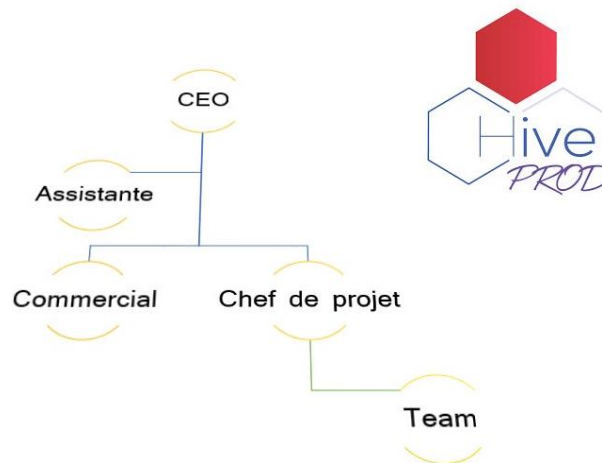


Figure 2 : Organigramme de l'organigramme

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, ECM, 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. Définition 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

La réalisation des applications de gestion aujourd'hui, 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 et offrent le support d'Ajax, une panoplie de Frameworks JavaScript ont vu le jour pour venir offrir des composants riches et gérer le parcours des éléments du DOM ainsi que les traitements asynchrones avec Ajax. 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, Comme dans les autres sciences, on s'est de plus en plus appuyé sur la modélisation pour essayer de maîtriser cette complexité, tant pour produire le logiciel (conception) que pour le valider (test).

Par exemple, un programme informatique peut être "représenté par" un ensemble de diagrammes de flux de données. Un modèle de données pourrait également représenter un autre modèle. Par exemple, une fonction mathématique peut être représentée par une approximation numérique. La seconde relation, appelée "conforme à", indique la dépendance d'un modèle par rapport à un langage de modélisation.

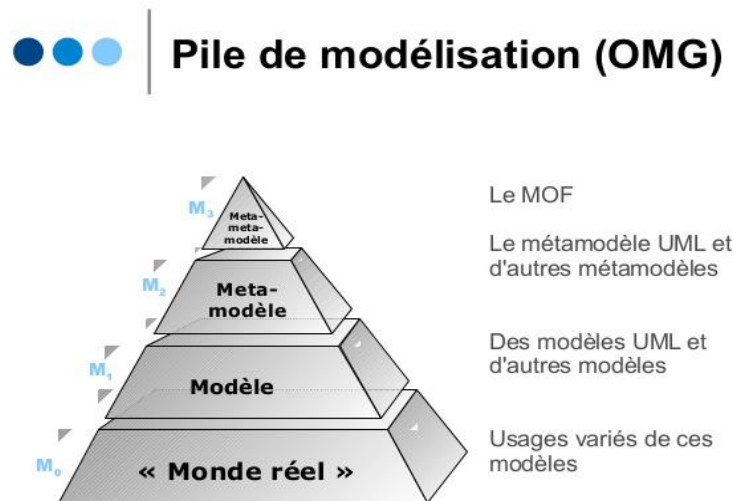


Figure 3 : Pile de modélisation (OMG)

2.2. Objectifs du projet

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

- A partir du point d'entrée (Schéma), 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 à chaque fois.

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.

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.

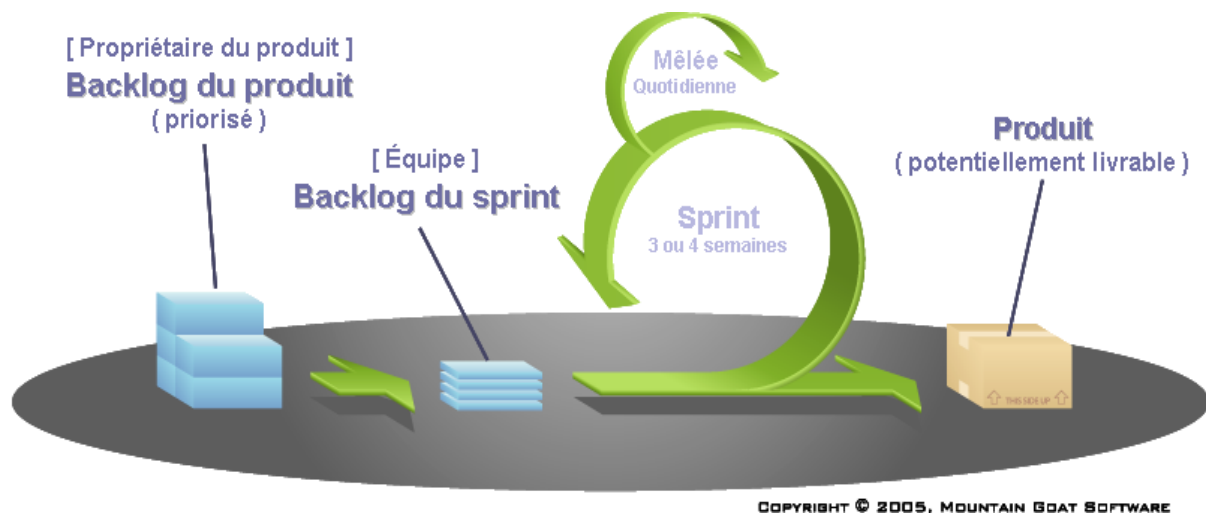


Figure 4 : Le processus Scrum

Pourquoi 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.

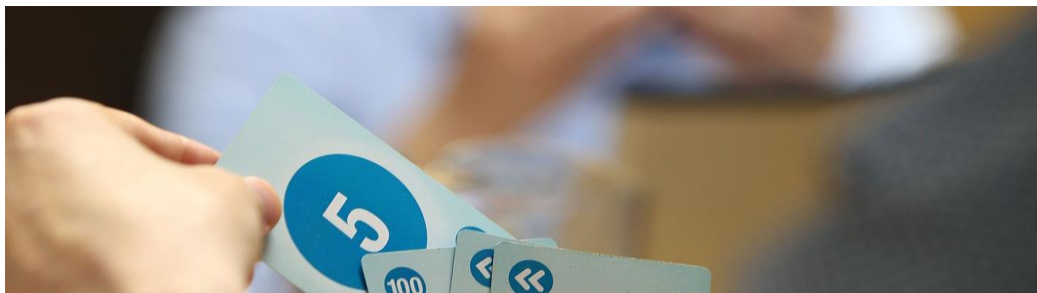
Product Backlog

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 Scrum : il incorpore le framework Scrum

Il répond en effet à plusieurs exigences du framework :

- Un carnet de produit
- Un carnet de sprint
- Un Scrum board qui facilite d'ailleurs les daily sprints
- Une section versions qui liste tous vos livrables
- La possibilité d'estimer vos tâches
- La possibilité de faire du reporting – sprint burndown (graphique d'avancement de sprint) et release burndown (graphique d'avancement de releases).

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.

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.

User management

Edit Comment Assign To Do In Progress Done

Type: Story Status: DONE (View workflow) Assignee: Kacimi Badr

Priority: Medium Resolution: Done Reporter: Salmane Sentissi

Component/s: None

Labels: None

Sprint: HPGEN Sprint 18/19

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 enregistré 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 enregistré 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** 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

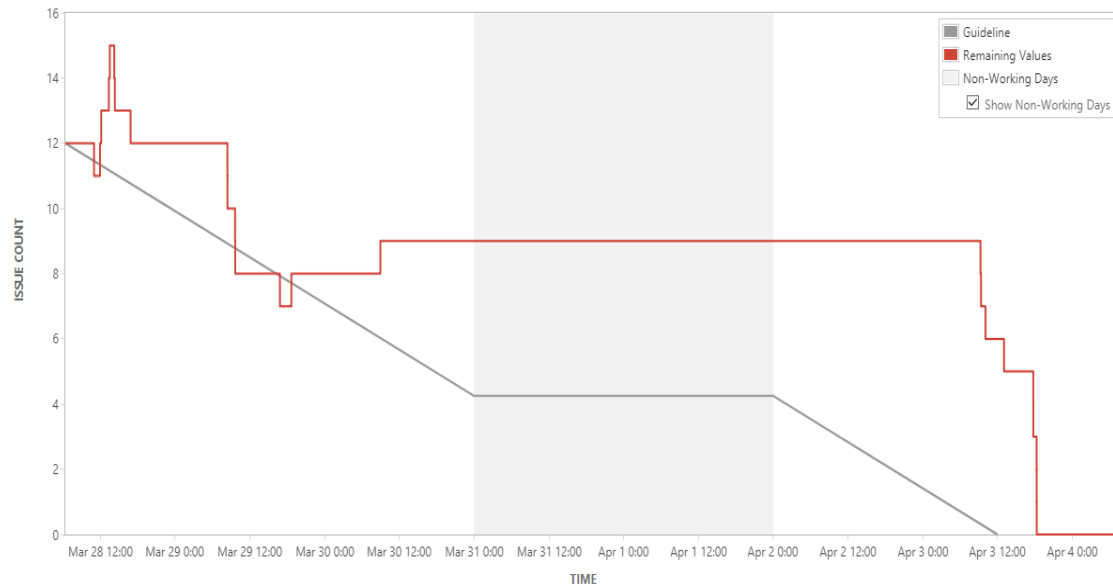


Figure 7 : Burndown Chart

Conclusion

Dans ce premier chapitre, nous avons 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, nous avons 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.

Chapitre 2 : Génération back-end et Front-end

1. Analyse de l'existence

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

Une application 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.

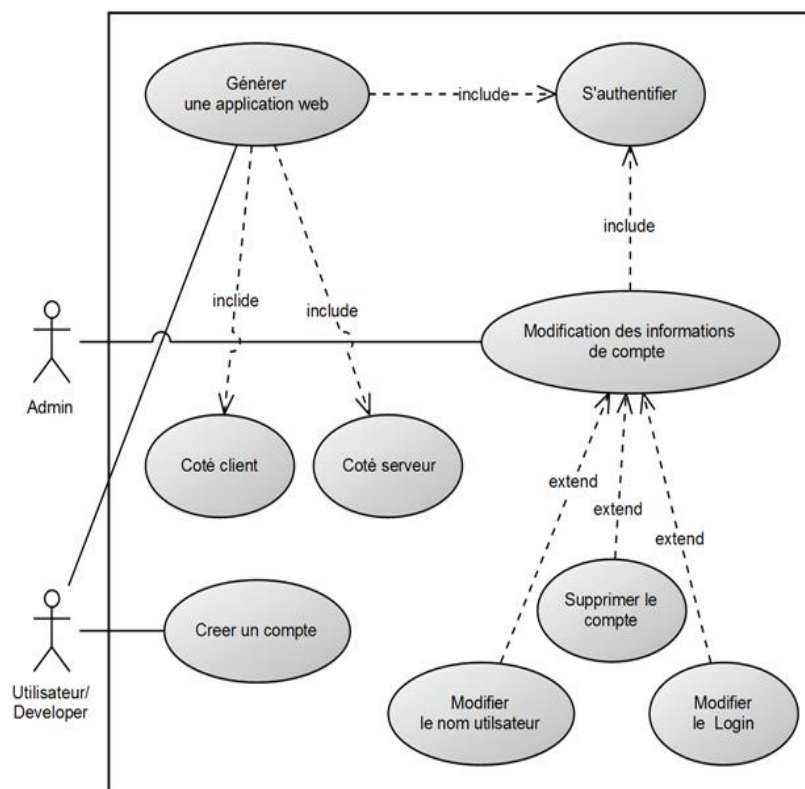


Figure 8 : use case << Générateur d'application web >>

Pour rendre notre diagramme des cas d'utilisation plus lisible et afin de décrire le comportement d'un système, les concepteurs d'UML proposent l'utilisation d'une technique nommée la description textuelle des cas d'utilisation.

Cas d'utilisation	Cote client
Acteurs	Utilisateur
Précondition	Utilisateur authentifié
Post-condition	L'utilisateur choisi une Template
Scénario nominal	1- L'utilisateur choisi les composants voulues de Template 2- L'utilisateur remplit les champs 3- La Génération de cote client
Scénario alternatif	N/A

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

Cas d'utilisation	Cote Serveur
Acteurs	Utilisateur
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 3- L'utilisateur saisie le REST API 4- La Génération de REST API (exposer les ressources)
Scénario alternatif	N/A

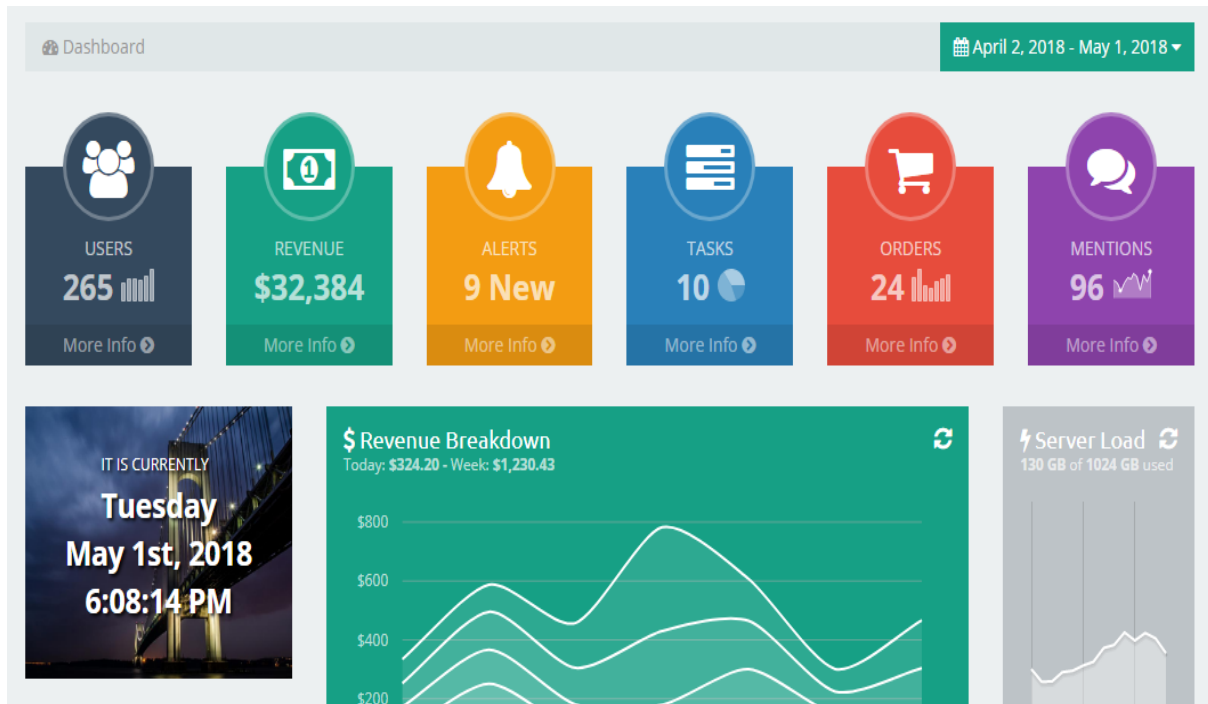
Tableau 3 : Description textuelle du cas d'utilisation « Coté serveur Back-End»

3. Génération coté client Front-End

3.1. Description

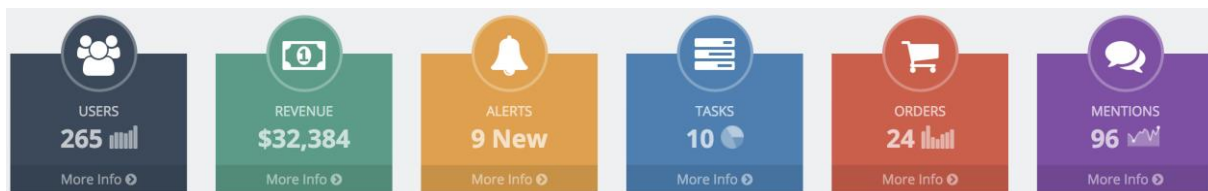
Dans une Template, La partie statique concerne la structure, caractères HTML des composants qui sont destinés au navigateur web et non aux humains !

La partie dynamique concerne les données, Les textes, Les icônes, Les couleurs qui contiennent des informations intéressantes pour les humains 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 :



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 "vrais" valeurs qui intéressent un humain 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 humains !
Comment notre générateur pourra les détecter, les extraire et les transformer dynamiquement en des composants Angular ?

Nous allons faire l'exercice manuellement, étape par étape, ensuite on apprendra à générateur de le faire automatiquement ...

L'HTML c'est de l'XML s'il est bien formater. Et c'est notre ça dans une Template.

Nous allons donc transformer l'HTML en JSON en utilisant XPATH :

Si vous utilisez la même source HTML initiale avec le XPath suivant sur :

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

Vous obtenez :

```
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>'
```

Ce qui est pas mal car ça contient les textes intéressants pour des humains et des développeurs !

C'est élément sont "variables" d'un client humain à un autre, donc l'Output de générateur doit être un composant Angular où il externaliser 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 c'est soit :

- Il liste toutes les combinaisons dans une IHM et laisse l'utilisateur ou le développeur l'aider à faire son choix (qu'on va adopter dans ce chapitre)
- Soit le générateur d'applications se base sur un système ANNs (Artificial neural networks systems) pour connaître la différence entre les différentes parties de HTML !

3.2. Approche classique

Si vous désirez personnaliser, modifier ou encore mettre à jour d'un site réalisé à l'aide d'un Template, vous aurez 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 que J'ai parlé dans la partie I.1.Template et 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 choisit les composantes de son interface et les modifies, l'idée s'était de travailler avec les métadonnées des composantes de Template pour générer des interfaces personnalisées pour chaque client en utilisant une interface conviviale IHM (JSON Editeur), toute en limitant les entrées (input validation) grâce aux mécanismes de JSON Schéma.

Ces métadonnées Comme le XSD pour le XML, le JSON SCHEMA permet de définir la structure et le type de contenu d'un document JSON

Ce langage de description de contenu de données JSON est lui-même défini par un schéma, dont les balises de définition s'auto-définissent (c'est un exemple de définition récursive).

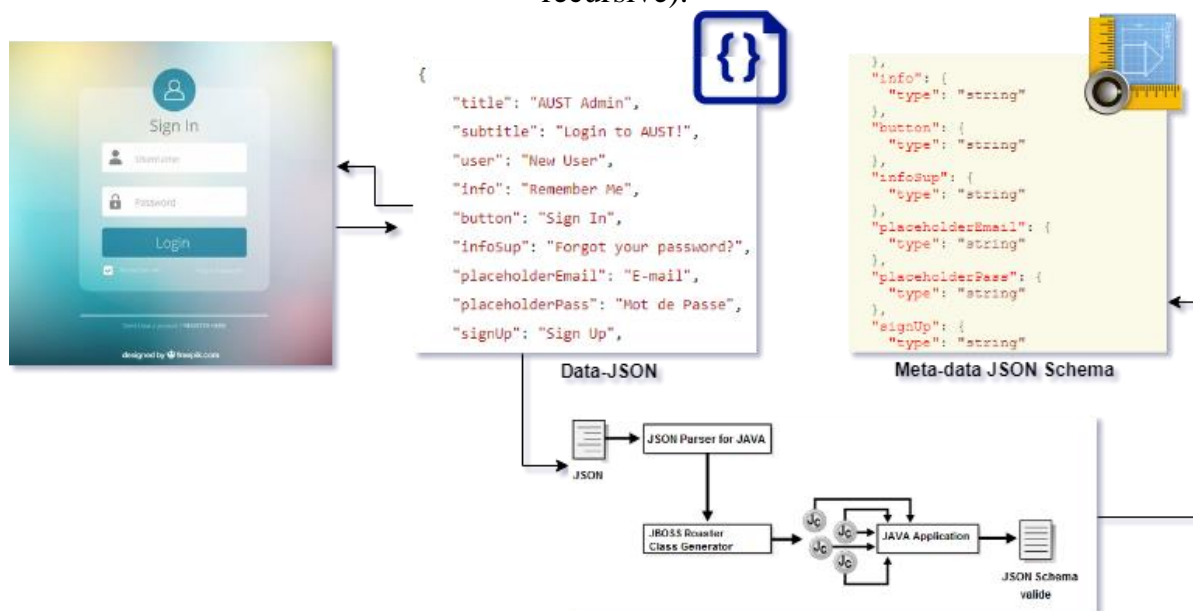


Figure 9 : Génération de JSON Schéma- Meta data

3.3.2. Scénario

- ⇒ Générer de JSON Schéma (métadonnée de JSON) de toutes les composantes d'une Template 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).

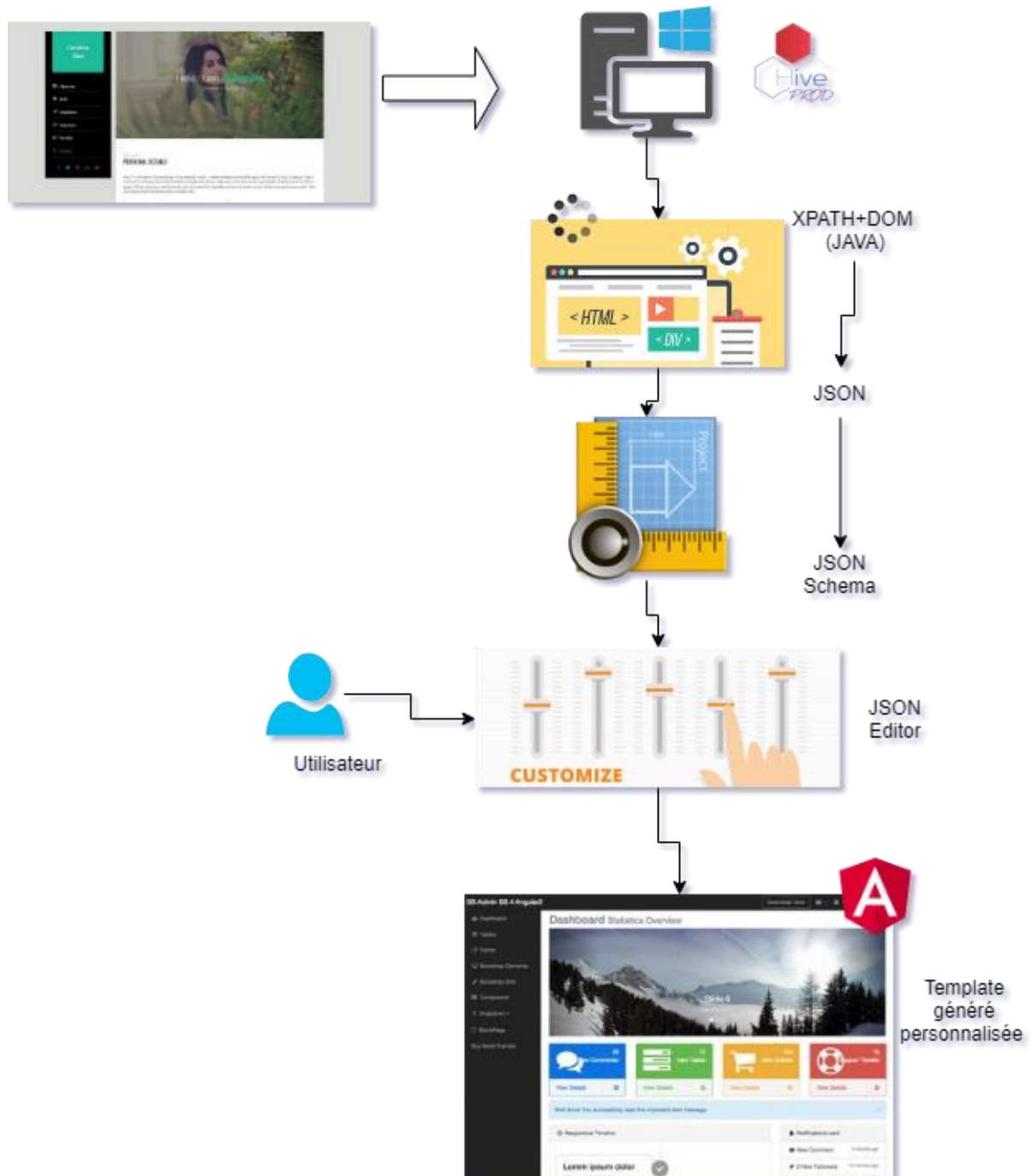


Figure 10 : Scénario globale de génération de Front-end d'une application web

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 (sous format war) dans les serveurs de l'entreprise.

4. Partie Serveur Back-End

4.1. Définition

Une application ou un programme « back-end » sert indirectement aux services frontaux, généralement parce qu'il se trouve plus près de la ressource requise ou qu'il a la capacité de communiquer avec elle.

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. On s'appuyant sur Schéma descriptive d'un POJO pour faire comprendre au générateur comment générer des POJO à travers les règles 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), il reste maintenant le modèle et le contrôleur. La première étape l'écriture d'une JSON Schéma (métadonnée) de POJO de model (les types de variables, sérialisation, annotations JPA valables etc...) qui contient les règles de génération de POJO :

```

{
  "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 11 : Extrait de JSON Schéma (schéma descriptif de POJO)

Le Schema definit Un JSON contenant le nom de classe variable etc... pour l'exploiter dans la generation de POJO en utilisant le package Java Roaster. Des conditions sont mis en place dans le JSON Schema ainsi dans le code de generation afin de valider l'input et respecter les standards JAVA.

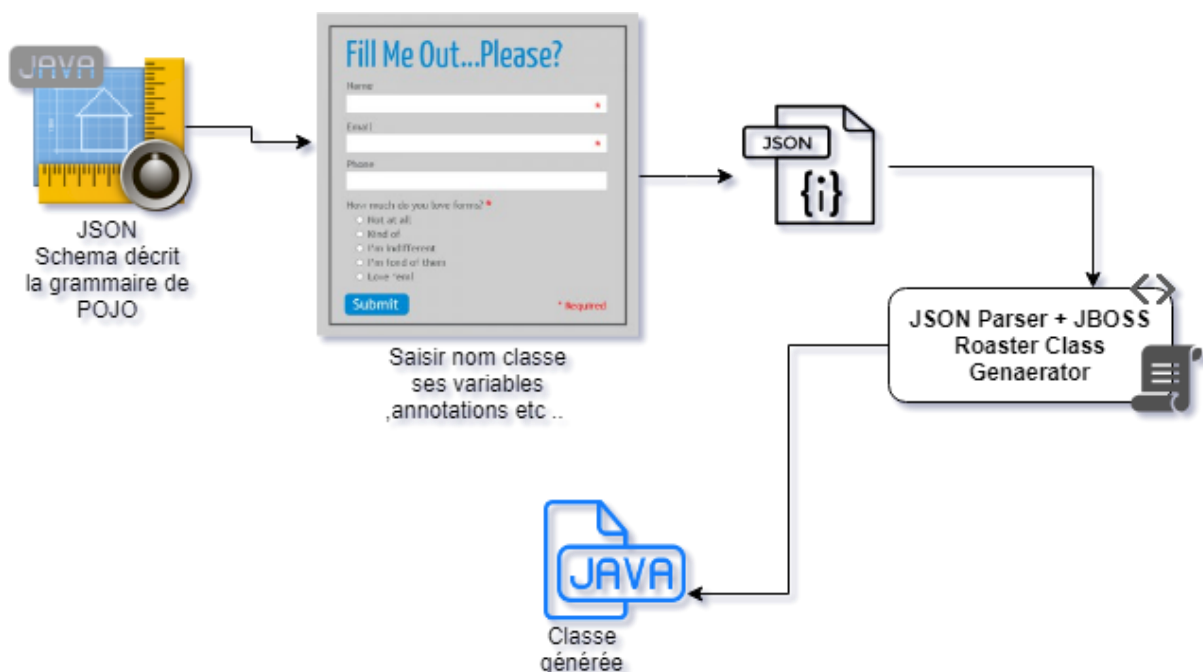


Figure 12 : Scenario de génération d'un POJO

Les Pojo générées contient des annotations du spécification JPA et leurs tables associés se persistent dès la première compilation de projet.

REST, c'est un moyen d'obtenir des ressources externes ou d'exposer une partie des tiennes dans un format "brut" style JSON ou XML.

La génération de Rest Controller 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...).

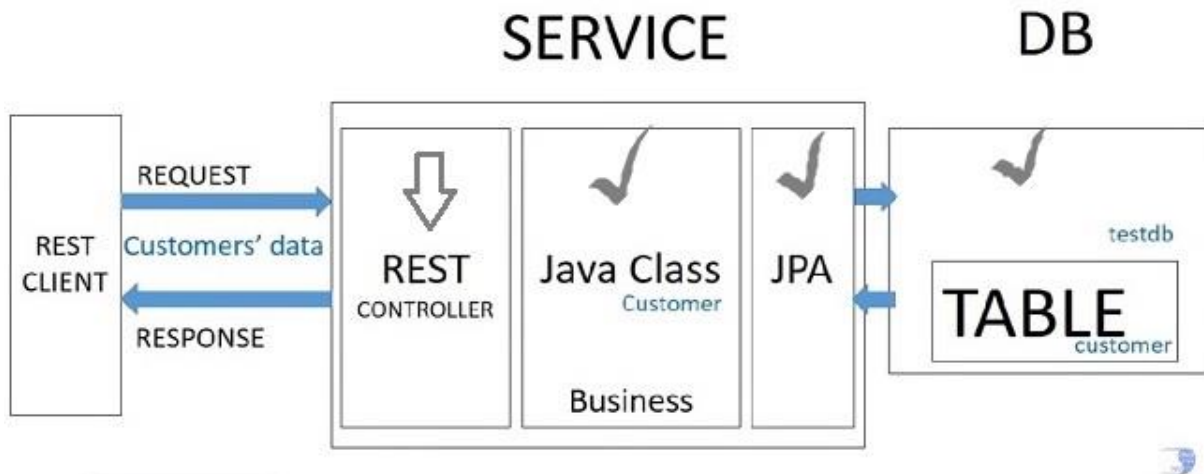


Figure 13 : Approche du style d'architecture REST

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 dans 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 a partie de cette contrat .

2.2.2 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.

Le format d'un jeton JWT est particulièrement bien adapté pour circuler dans une uri et donc pour les applications REST.

JWT est actuellement utilisé par beaucoup de monde et en particulier par Microsoft pour requêter Windows Azure Active Directory via Graph API. Il s'agit également du format standard de OAuth 2 et OpenID Connect.

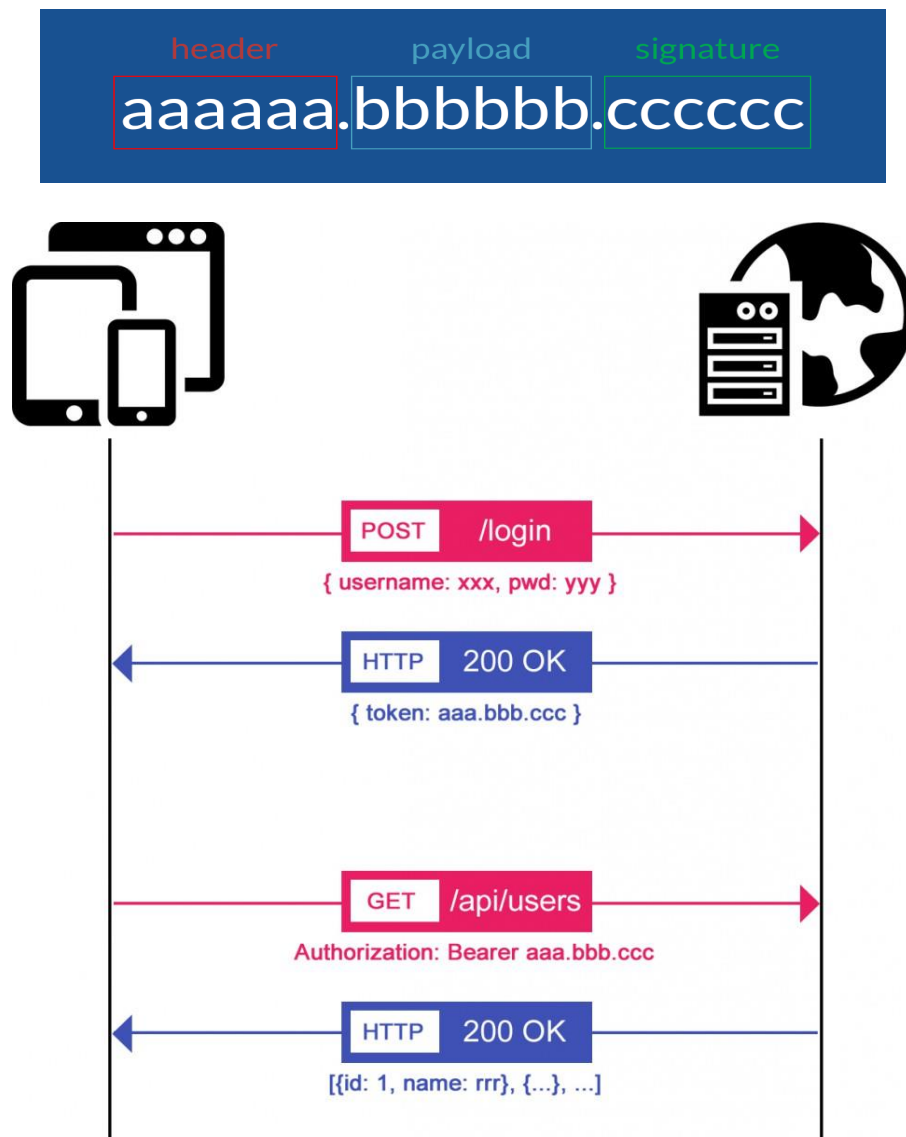


Figure 14 : JWT Authentication

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.

Il y a trois parties séparées par un point, chaque partie est créée différemment. Ces trois parties sont :

➤ 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.

2.2.3 Schéma technique globale

Schéma technique globale du générateur :

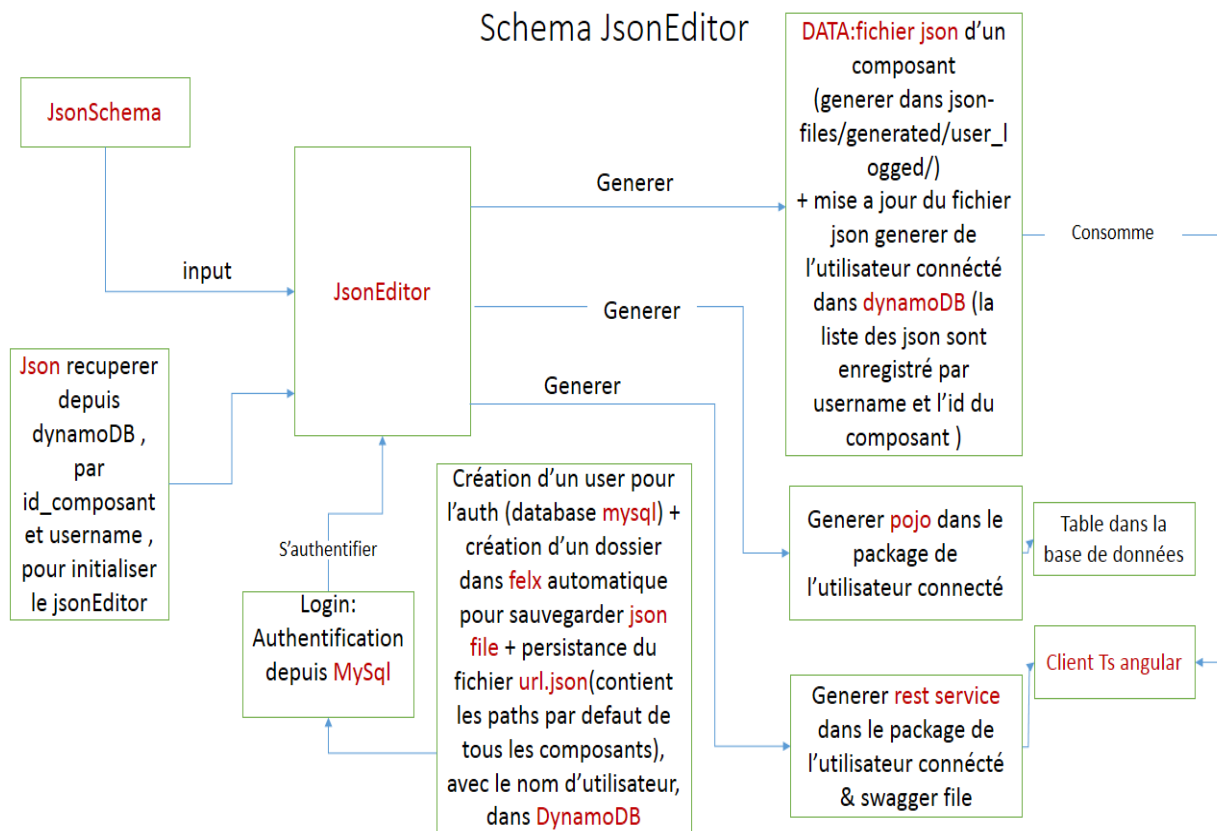


Figure 15 : Schéma technique du générateur

Interaction avec l'application web générée :

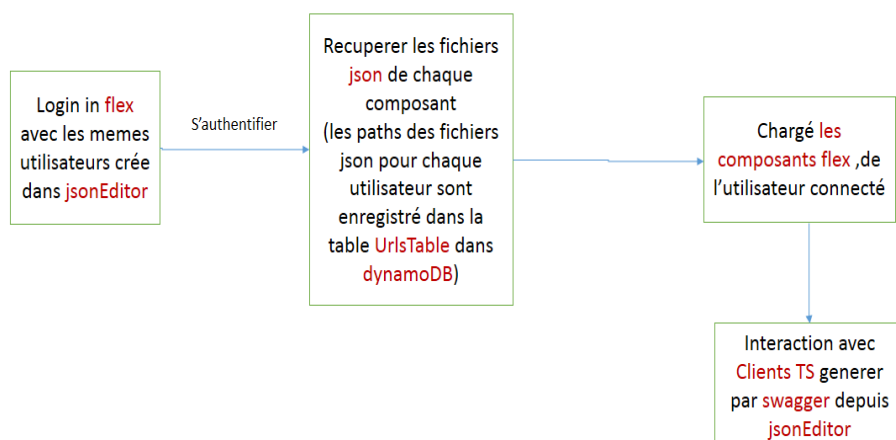


Figure 16 : Schéma technique d'interaction avec l'application web générée

Conclusion

Dans ce deuxième chapitre consacré à l'analyse et spécification, nous avons commencé par une analyse des architectures techniques existantes. Nous avons présenté en un premier lieu les caractéristiques des applications de gestion adopté par HiveProd.

Ensuite, nous avons critiqué l'existant des différentes solutions existantes afin de ressortir les parties répétitives dans la phase de développement. Finalement, nous avons 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 suivies pour la génération d'une application web en coté client et en coté serveur.

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) chargé de la présentation.
- 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 frameworks 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.

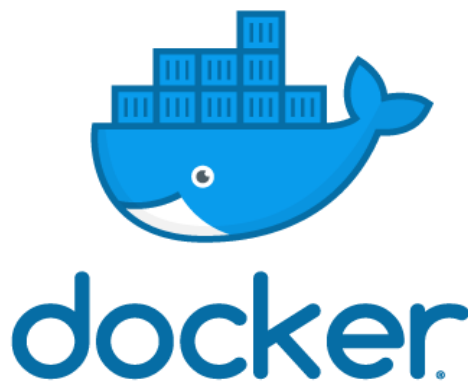
Pourquoi Spring ?

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.



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.



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.

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.





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.

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.

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.





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

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.



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,



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.



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 (cote client)
- Passer à la partie serveur et base de données
- Tester l'application

Chapitre 3 : Réalisation

On a travaillé sur une Template qui s'appelle Flex dans ce projet pour le test de générateur.

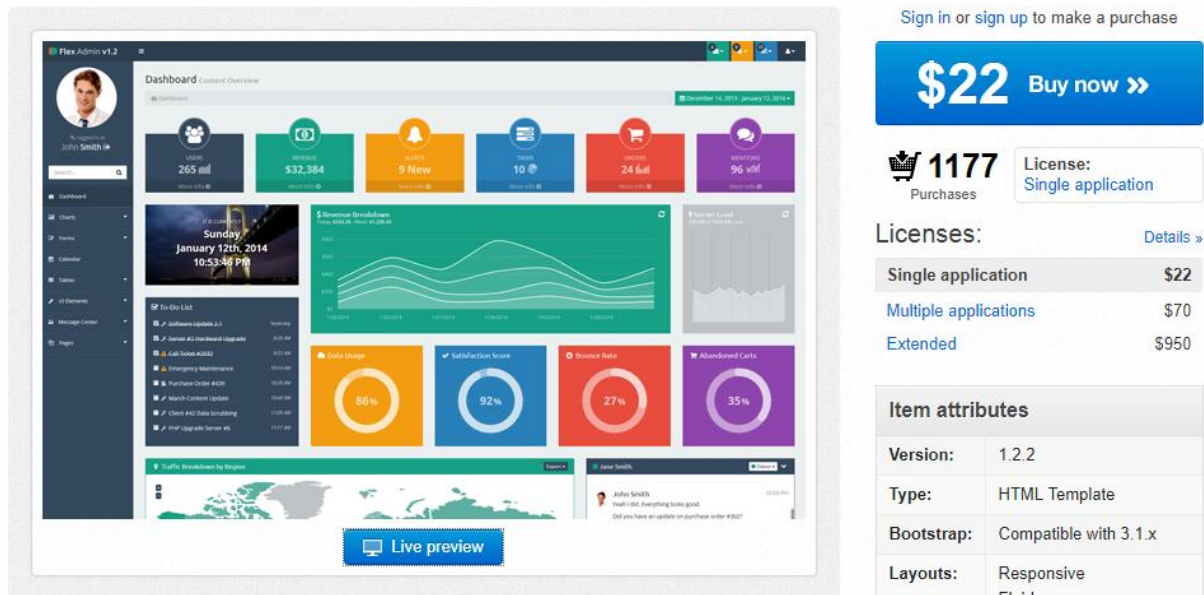


Figure 17 : Template Flex

L'étape generation des JSON schéma des composantes de Template est fait pour qu'on puisse laisser a l'utilisateur la liberté de personnaliser son interface.

1. Coté client (Front-End)

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

Enter Your Login:

Enter Your password:

Hint : administrator/admin

Figure 18 : Page d'authentification

-La bouton Add a new user : Créer un nouveau compte pour un utilisateur

L'utilisateur connecté est l'administrateur

Figure 19 : JSON Editor – page accueil

⇒ La première étape est de créer un Menu « Menu gauche(SideNavDropDown) » à l'aide de JsonEditor

Figure 20 : Menu gauche(SideNavDropDown)

1. Sélectionner Jschema 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 :

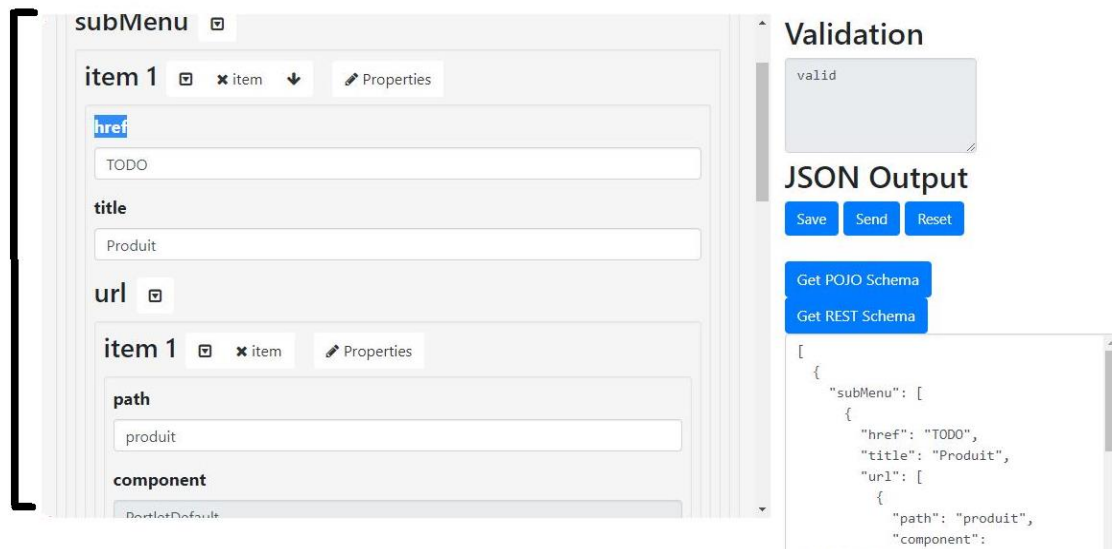


Figure 21 : SubMenu

- ✓ 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 bouton **send**.

La deuxième étape est de générer le tableau et le formulaire

Donc on choisit **new table** pour créer un nouveau tableau

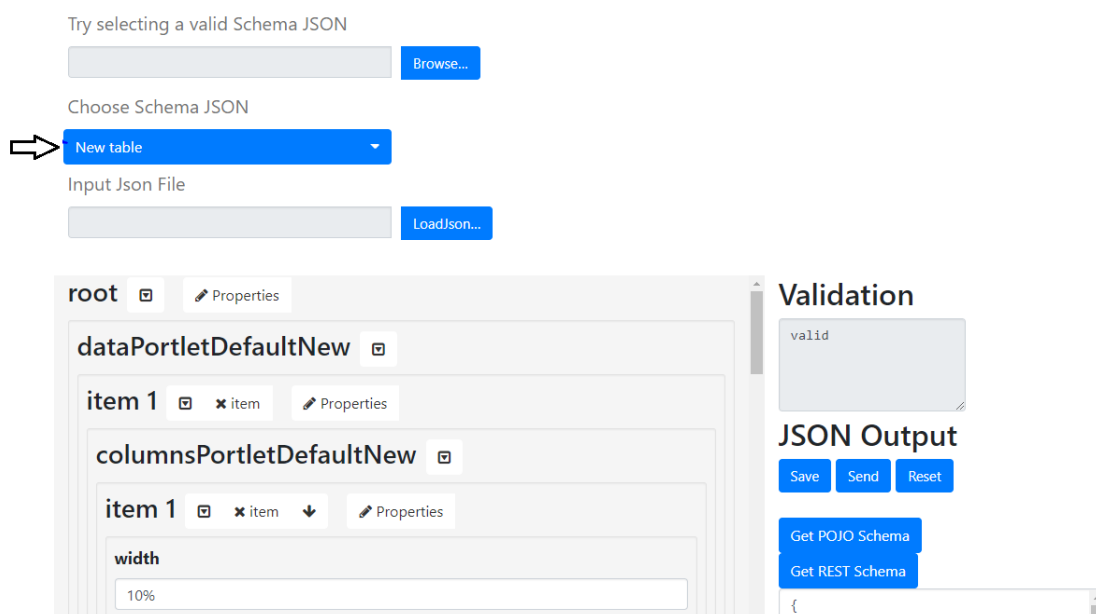


Figure 22 : saisit les entêtes du tableau

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

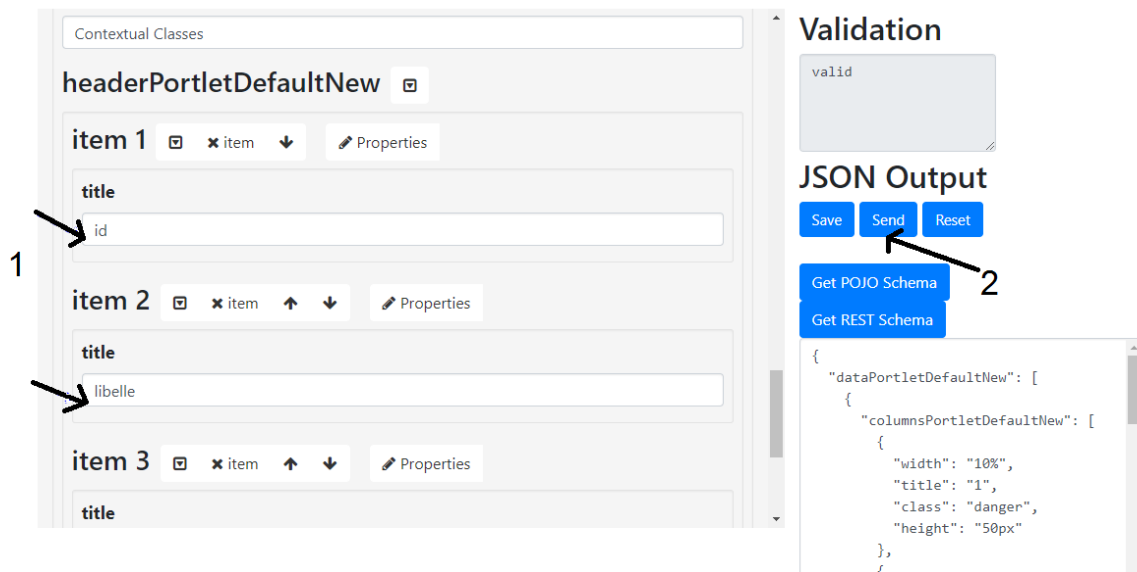


Figure 23 : Buttons send ()

Lors de send le fichier json s'enregistre dans le projet flex et précisément dans l'emplacement de projet Flex (Template) dans les serveurs de l'entreprise. De la même manière on crée le formulaire avec les mêmes données qu'on veut modifier

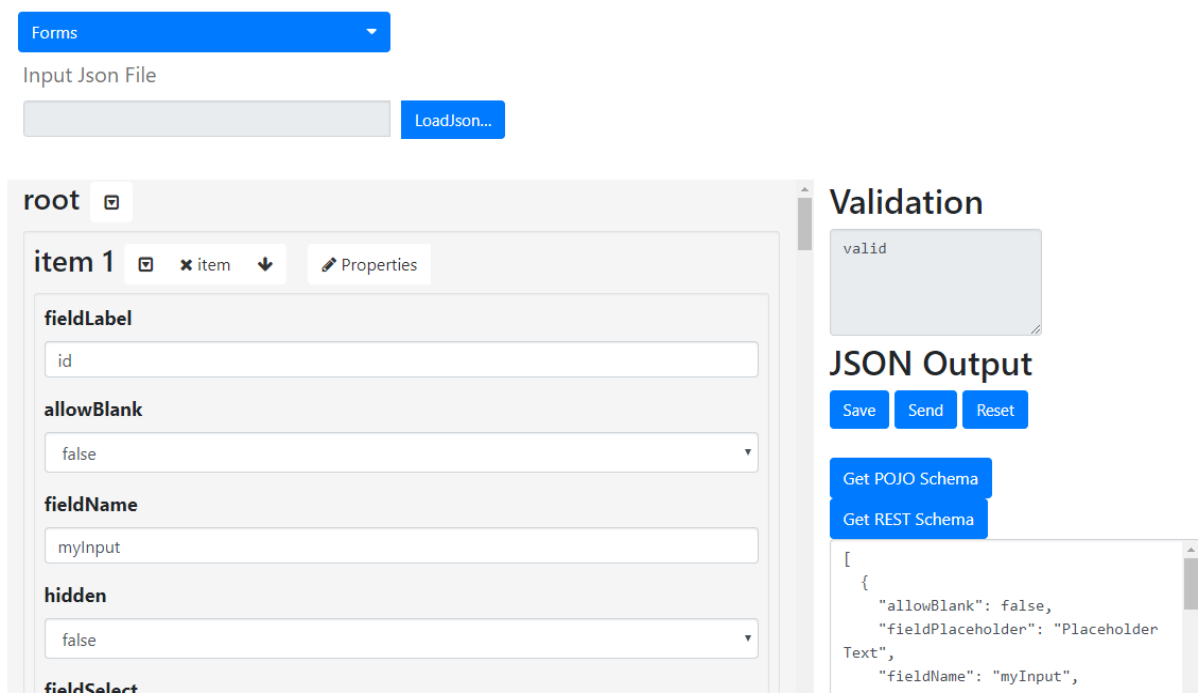


Figure 24 : Composante Forms

2. Coté serveur (Back-End)

-La deuxième étape est de créer le POJO.

Pour cela on clique sur le Botton Get POJO Schema pour importer le schéma.

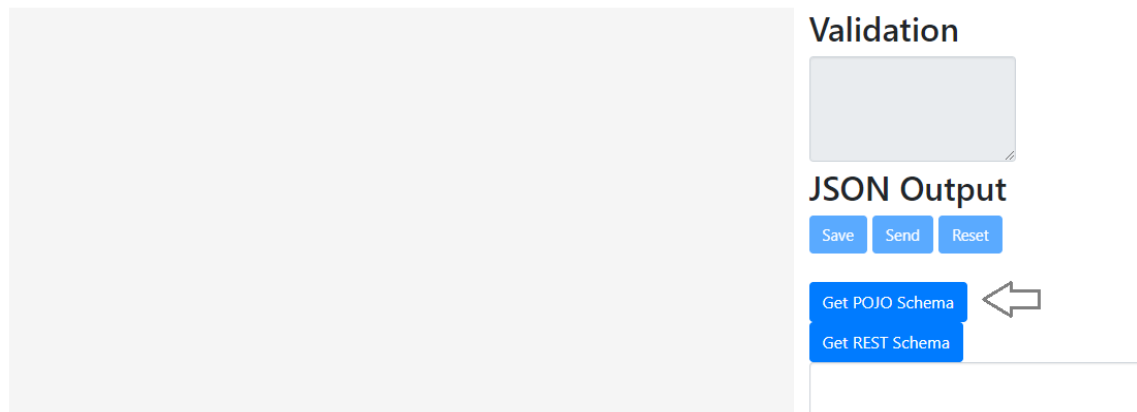


Figure 25 : Création de POJO

-Ensuit on saisit les données de notre POJO

Le nom de POJO et les attributs avec leurs annotations.

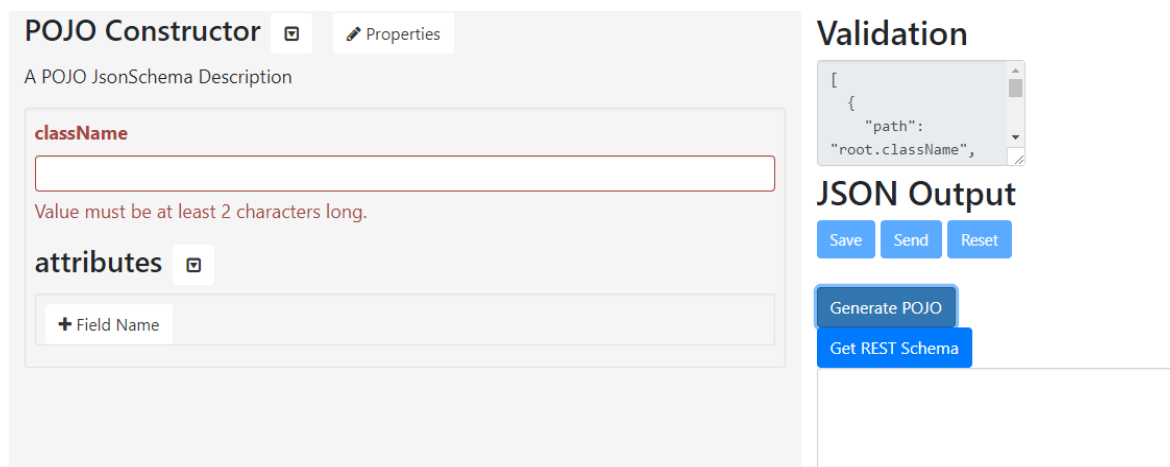
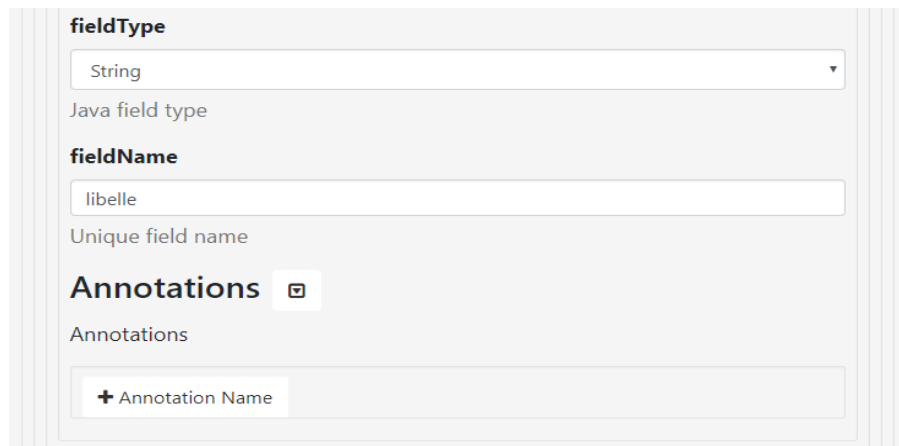


Figure 26 : Champs de Schéma POJO

ClassName : Produit

Id : int avec l'annotation @id et @AutoGenerated

+ Attribut libelle



The screenshot shows a configuration form for a field named 'libelle'. It includes a 'fieldType' dropdown menu set to 'String', a 'fieldName' text input containing 'libelle', and an 'Annotations' section with a '+ Annotation Name' button.

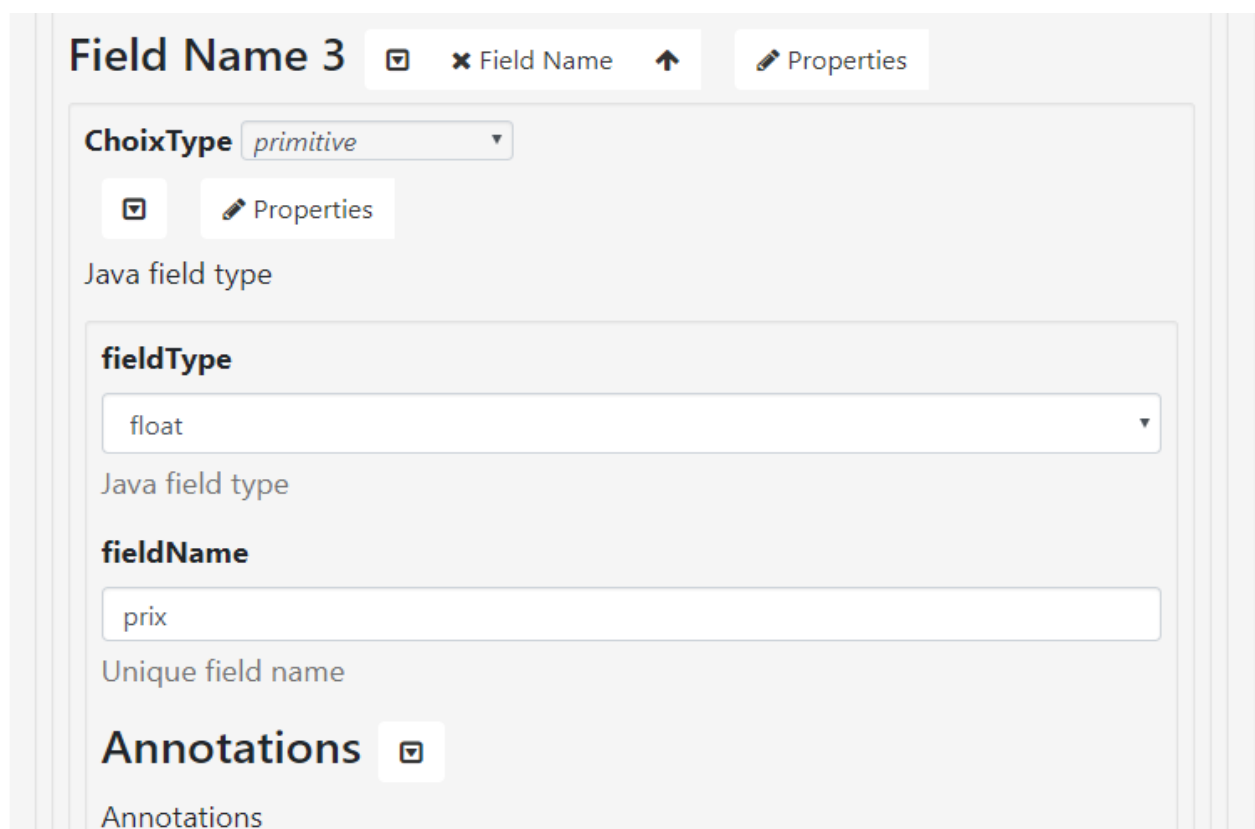
fieldType
String
Java field type

fieldName
libelle
Unique field name

Annotations ☒
Annotations
+ Annotation Name

Figure 27 : Attribut libelle

+ Attribut Prix



The screenshot shows a configuration form for a field named 'prix'. It includes a 'ChoixType' dropdown menu set to 'primitive', a 'fieldType' dropdown menu set to 'float', a 'fieldName' text input containing 'prix', and an 'Annotations' section with a checked checkbox.

Field Name 3 ☒ **Field Name** **Properties**

ChoixType primitive
☒ **Properties**

Java field type

fieldType
float
Java field type

fieldName
prix
Unique field name

Annotations ☒
Annotations

Figure 28 : Attribut prix

+ Attribut quantité

Field Name 4 [icon] x Field Name [icon] Properties

ChoixType primitive

[icon] Properties

Java field type

fieldType

long

Java field type

fieldName

quantite

Unique field name

Annotations [icon]

Annotations

Figure 29 : Attribut quantité

-A la fin on clique sur le Boton : Generate POJO

Field Name 1 [icon] x Field Name [icon] Properties

ChoixType primitive

[icon] Properties

Java field type

fieldType

int

Java field type

fieldName

id

Unique field name

Annotations [icon]

Annotations

Validation

valid

JSON Output

Save Send Reset

Generate POJO Get REST Schema

```
{
  "className": "Produit",
  "attributes": [
    {
      "ChoixType": {
        "fieldType": "int",
        "fieldName": "id",
        "Annotations": []
      }
    }
  ]
}
```

Figure 30 : 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 Bouton : *Get REST Schema*, on aux méthodes par défaut (getAll, getById, upDate, delete, add)

The screenshot shows the 'REST Constructor' interface with the 'Validation' tab selected. The 'JSON Output' section displays a JSON schema for a 'Produit' class. The schema includes a 'path' for 'root.className' and a 'ChoixType' object with 'fieldType' 'long', 'fieldName' 'id', and 'Annotations' including 'javax.persistence.Id'.

```

[
  {
    "path": "root.className",
  }
]

```

The 'JSON Output' section also includes buttons for 'Save', 'Send', and 'Reset'. Below these are buttons for 'Get POJO Schema' and 'Get REST Schema', with an arrow pointing to the 'Get REST Schema' button.

Figure 31 : REST schéma

Ensuit ouvrir les *collapses* pour remplir les champs manquées :

The screenshot shows the 'REST Constructor' interface with the 'Methodes' tab selected. The 'Methodes' section displays a list of methods, with 'Methode Name 1' selected. The 'methodAccess' dropdown is set to 'public', the 'methodeType' dropdown is set to 'GET', and the 'methodeReturnType' dropdown is set to 'ChoixType object'. The 'JSON Output' section displays a JSON schema for a 'Produit' class, similar to the one in Figure 31.

```

[
  {
    "path": "root.className",
  }
]

```

Figure 32 : Saisie Méthode de REST-1

-Saisit les données de notre POJO

The screenshot shows the 'type of return' configuration section on the left and the 'Validation' section on the right.

type of return

- ChoixType**: object
- methodeType**: Collection<Object>
- NameOfObject**: (indicated by an arrow)
 - Unique field name
 - Value must be at least 2 characters long.
- methodeName**: getAll
 - Unique methode name

Validation

- Path: "/root.Methodes.0.me"

JSON Output

- Buttons: Save, Send, Reset
- Buttons: Get POJO Schema, Get REST Schema
- JSON Schema Preview:


```
{
  "className": "Produit",
  "attributes": [
    {
      "ChoixType": {
        "fieldType": "long",
        "fieldName": "id",
        "Annotations": [
          {
            "fieldType": "javax.persistence.Id"
          }
        ]
      }
    }
  ]
}
```

Figure 33 : Saisie Méthodes REST-2

Si les données sont validées cliquer sur le Botton : Generete Rest

Le Service se génère dans le package : org.jsoneditor.web+ {username}

The screenshot shows the 'parameter of annotation 1' configuration section on the left and the 'Validation' section on the right.

parameter of annotation 1

- parameterName**: value
- parameterValue**: supprimer un élément
- Buttons: + parameter of annotation, x Last parameter of annotation
- Buttons: + Annotation Name, x Last Annotation Name
- Buttons: + Methode Name, x Last Methode Name, x All

Validation

- Status: valid (circled)

JSON Output

- Buttons: Save, Send, Reset
- Buttons: Get POJO Schema, Generate Rest (indicated by an arrow)
- JSON Schema Preview:


```
{
  "ChoixType": {
    "parameterType": "org.springframework.ui.Model",
    "parameterName": "model",
    "parameterAnnotation": "",
    "parameterAnnotationValue": ""
  }
}
```

Figure 34 : Génération de REST Controller

Si le Service Rest est générer, le message suivant s'apparu.

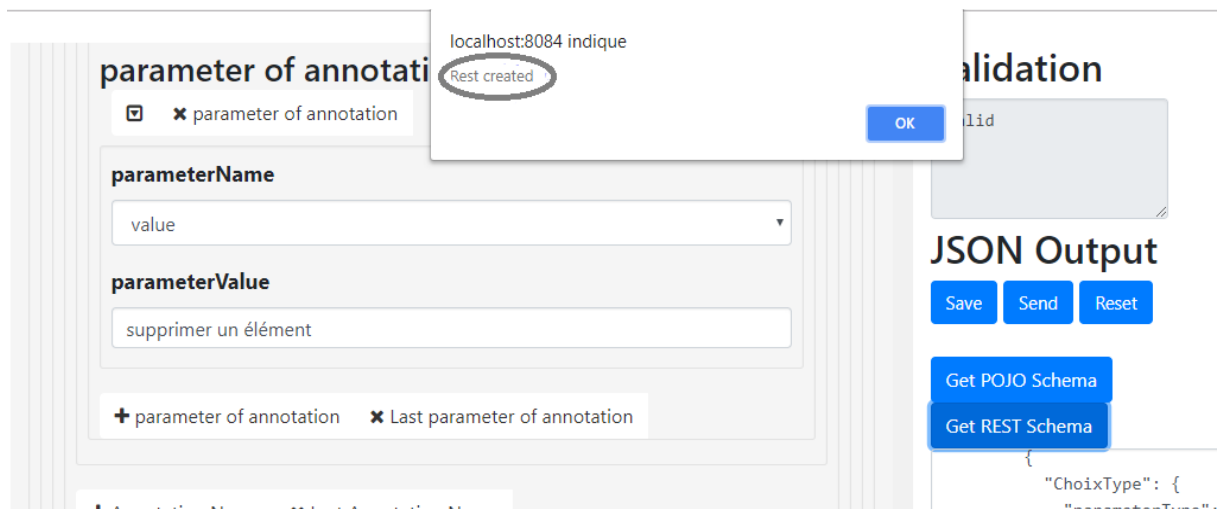


Figure 35 : Message de confirmation

La dernière étape est généré le client TS à l'aide swagger en exécutant deux une commandes et une simple configuration.

⇒ Résultat doit être comme suit :

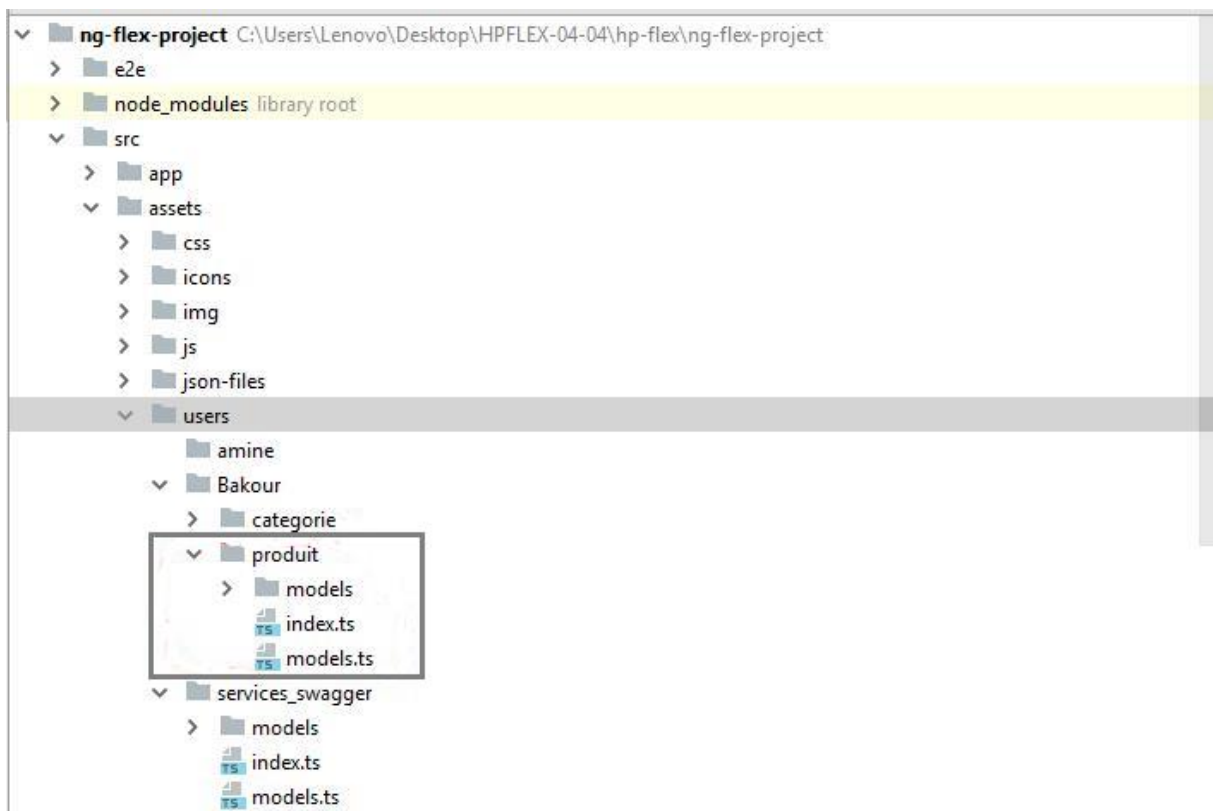


Figure 36 : client TS –REST Client

Application web générée

Gestion de produit

Page authentication :

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.

Les identifiants sont les même que dans JSON Editor.

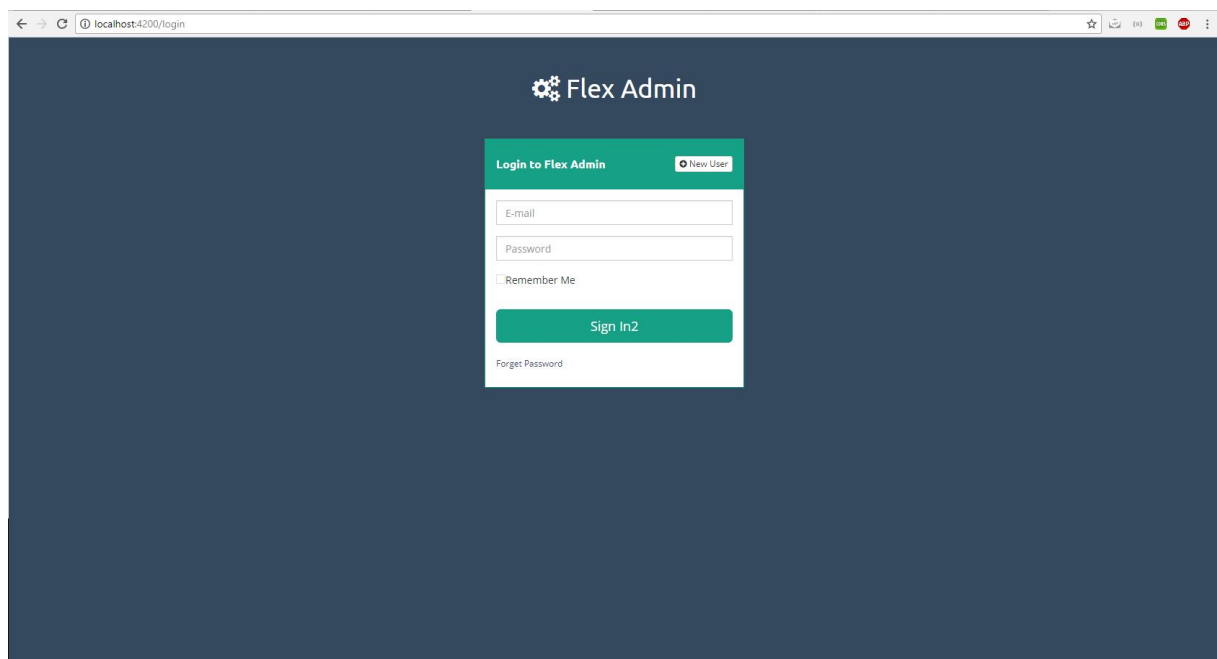


Figure 37 : Page d'authentification d'application web générée

L'utilisateur va trouver tous les composants qui lui concernent

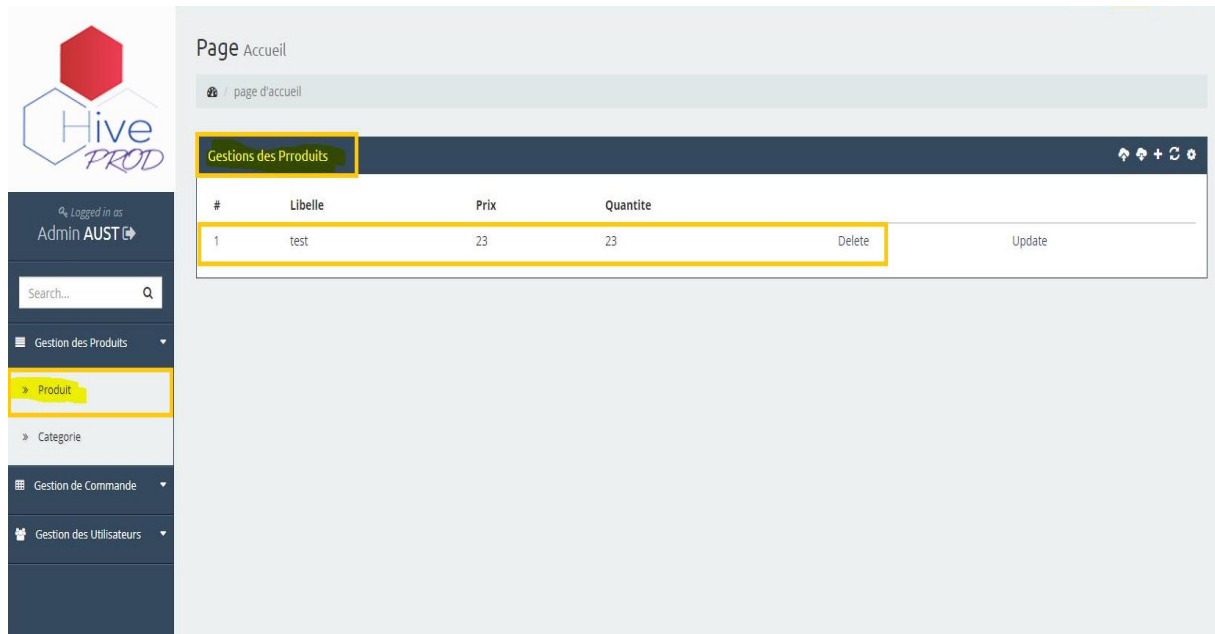


Figure 38 : Gestion produit - menu gauche

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 .
- ✓

⇒ Add Produit :

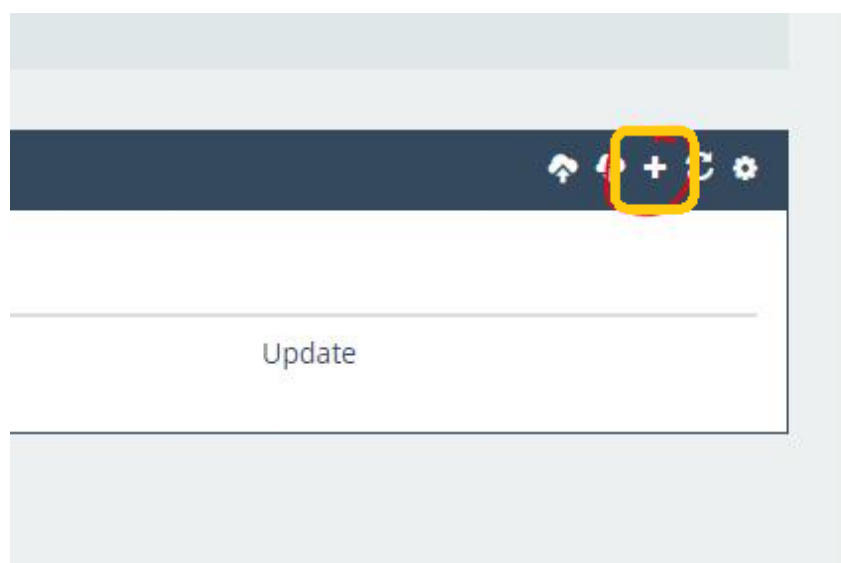


Figure 39 : Ajouter un produit

- ⇒ **Delete** : supprime la ligne depuis la base .
- ⇒ **update** : modifie le produit selectionné
- ⇒ Formulaire pour ajouter et modifier :

The screenshot displays the 'Hive PROD' Admin interface. On the left is a dark blue sidebar with the logo at the top, followed by 'Logged in as Admin AUST' and a search bar. Below 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, it says 'Page Accueil' and 'page d'accueil'. A modal window titled 'Formulaire' is open, containing a 'New Load' button and three input fields labeled 'Libelle', 'Prix', and 'Quantite', each with 'Placeholder Text'. A 'Submit' button is at the bottom of the form.

Figure 40 : 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.

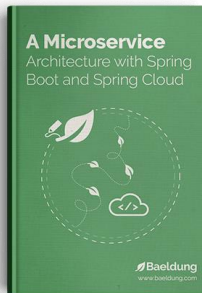
Le stage a été pour moi une occasion de découvrir l'environnement professionnel et de s'habituer aux méthodes de travail dans le monde de l'entreprise, au sérieux ainsi qu'à la dynamique de groupe. Techniquement, j'ai pu avoir de nombreuses formations me permettant de maîtriser les différentes technologies 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 seront ajoutées pour enrichir ce projet, en ajoutant des nouvelles fonctionnalités comme l'intelligence artificielle, on a fait pu faire comprendre au générateur quelle sont les règles à travers des JsonSchema. Maintenant en s'appuyant sur un algorithme génétique et un réseau neurone, le générateur devra trouver il-même son schéma et pour connaitre la différence entre les différentes parties de HTML comme un développeur !

Bibliographie & Webographie

Bibliographie



Spring Boot and Spring Cloud with Baeldung- 2017



RESTful Java Patterns and Best Practices Paperback – September 19, 2014

Webographie :



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



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



<https://dzone.com/java-jdk-development-tutorials-tools>

Annexe

```
src/main/java
├── org.jsoneditor
│   ├── org.jsoneditor.authentication
│   ├── org.jsoneditor.aws.sqs
│   ├── org.jsoneditor.model
│   └── org.jsoneditor.model.badr_user
│       └── produit.java
│   ├── org.jsoneditor.repository
│   └── org.jsoneditor.web
├── src/main/resources
├── src/test/java
├── JRE System Library [JavaSE-1.8]
├── Maven Dependencies
├── bin
├── src
├── target
└── dockerfile
```

Package pour chaque utilisateur qui contient son back-End (ici le model)

Available authorizations

username:
badr@example.com

password:
.....

type:
Request body

client_id:
client

client_secret:
secret

Scopes:

☐ read
read all

☐ write
access all

Authorize

-Sécurité des web services grâce au Oath **JWT** et Swagger

-Login pour accéder aux ressources de l'application.

Scopes are used to grant an application different levels of access to data on behalf of the end user. Each API may declare one or more scopes.

API requires the following scopes. Select which ones you want to grant to Swagger UI.

oauth2 (OAuth2, password)

Authorized

Token URL: `http://localhost:8080/api/oauth/token`

Flow: password

username: `badr@example.com`

password: `*****`

type: request-body

client_id: `*****`

client_secret: `*****`

Logout

Accès aux ressources
après authentification.

Génération des POJO-Roster en phase de développemnt

```

RestGenerator.java
HtmlResource.java
GetMapping.class
Test.java

JavaClassSource source = Roaster.create(JavaClassSource.class);

source.setName("RestController").setPublic().addAnnotation(RestController.class);
source.addAnnotation(Api.class);

source.addMethod().setName("Create").setPrivate().setBody("return null;")
    .addAnnotation(RequestMapping.class);

Console

<terminated> RestGenerator [Java Application] C:\Program Files\Java\jdk1.8.0_144\bin\javaw.exe (Mar 12, 2018, 2:34:28 PM)
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.DeleteMapping;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PutMapping;

@RestController
@Api
public class RestController
{

    @RequestMapping
    private void Create()
    {
        return null;
    }

    @DeleteMapping
    private void Delete()
    {

```