|  |
| --- |
| SMILE - EPSI |
| Dossier professionnel RNCP 35584 - Expert en Informatique et en Système d’Information |
| Promotion 2021-2023 – Responsable Pédagogique : Sophie DEMOIS |

|  |
| --- |
| Ryan Murati  08/01/2023 |



# Table des matières

[I- Table des matières 2](#_Toc142863469)

[II- Remerciements 3](#_Toc142863470)

[III- Introduction 4](#_Toc142863471)

[IV- Environnement professionnel 5](#_Toc142863472)

[A) Présentation de l’entreprise SMILE 5](#_Toc142863473)

[i) Organisation de la structure 5](#_Toc142863474)

[ii) Présentation du SI, acteurs, informations et flux 5](#_Toc142863475)

[B) Activités de la structure 6](#_Toc142863476)

[i) Les différentes équipes de SMILE 6](#_Toc142863477)

[ii) Les objectifs des activités 7](#_Toc142863478)

[iii) Les clients réguliers 7](#_Toc142863479)

[C) Mon activité à SMILE 9](#_Toc142863480)

[i) Place dans les équipes 9](#_Toc142863481)

[ii) Missions confiées 9](#_Toc142863482)

[V- Valorisation des compétences 10](#_Toc142863483)

[A) Projet WSO2 10](#_Toc142863484)

[i) Explication en détail du projet 10](#_Toc142863485)

[ii) Quelles compétences validées par le projet ? 20](#_Toc142863486)

[iii) Recul et améliorations envisageables 21](#_Toc142863487)

[B) Projet INRAE 22](#_Toc142863488)

[i) Explication en détail du projet 22](#_Toc142863489)

[ii) Compétences validées par le projet ? 23](#_Toc142863490)

[iii) Recul et améliorations envisageables sur la compétence et bonnes pratique 23](#_Toc142863491)

[C) Projet « mtes déclaration évènement » 24](#_Toc142863492)

[i) Explication en détail du projet 24](#_Toc142863493)

[ii) Quelles compétences validées par le projet ? 37](#_Toc142863494)

[iii) Recul et améliorations envisageables sur la compétence et bonnes pratique 38](#_Toc142863495)

[VI- Liste de compétences non validées par mes projets. 39](#_Toc142863496)

[VII- Conclusion 40](#_Toc142863497)

[A) Résumé des projets 40](#_Toc142863498)

[B) Veille technologique et apprentissage en dehors du cursus scolaire 40](#_Toc142863499)

[C) Ouverture sur l’avenir 41](#_Toc142863500)

[VIII- Annexe 42](#_Toc142863501)

# Remerciements

Je tiens à remercier toutes les personnes qui m’ont accompagné lors de mes deux années en alternance à SMILE, ainsi que toutes celles qui m’ont aidé lors de la rédaction de ce rapport. En premier lieu, j’adresse mes sincères remerciements à **Sophie DEMOIS** (ainsi que toute l’équipe pédagogique de l’EPSI) qui, grâce à son écoute et sa réactivité, nous a offert les meilleures conditions de travail pour deux années d’études. J’aimerai ensuite remercier mes deux manageuses de SMILE, **Cécile WALKENER** et **Thi** **thuy Linh LE**, qui m’ont suivi tout au long de mon alternance, et m’ont présenté les collaborateurs avec lesquels j’allais travailler.

Je remercie également toutes les personnes du pôle GED de SMILE, notamment **Nicolas Vezian**, qui ont su m’intégrer à l’entreprise lors de mon arrivée, et pour leur disponibilité et leur aide précieuse à chacune de mes questions ou difficultés.

Un grand merci à mes anciens équipiers de l’INRAE, particulièrement **Nicolas Brun** avec qui j’ai eu plaisir de travailler pendant 8 mois, et qui m’ont formé avec patience et bienveillance.

# Introduction

SMILE développe et apporte des solutions open source pour divers clients. J'ai donc, tout au long de mon apprentissage, collaboré auprès de clients issus de secteurs différents, pour leur apporter des solutions digitales et développer leur projet de transformation numérique.

Après une présentation de l’entreprise SMILE, de son SI, de ses équipes et des missions qu’elle propose en général, je décrirai les compétences que j’ai pu acquérir lors des projets que j’ai mené dans l’entreprise. Cette présentation contiendra une description des différentes tâches effectuées, des difficultés que j’ai pu rencontrer et comment je les ai surmontées tout en gardant un regard critique sur les possibles axes d’améliorations de mon travail.

Lors de mon arrivée à SMILE, j’ai été placé dans une équipe de deux personnes, afin que je sois formé sur une technologie que je ne connaissais pas, WSO2 API Manager. Passés quelques mois à acquérir les compétences nécessaires sur cette technologie, j’ai été à même de proposer de développer un logiciel de démonstration, permettant de présenter aux clients potentiels les possibilités que la technologie offrait.

Après le succès de ce projet solo, j’ai rejoint l’INRAE (Institut national de la recherche agronomique), en régie, avec l'idée d’une formation sur leurs technologies et leurs projets pendant 6 mois suivi d’une autonomie projet les 6 mois suivants.

Après mes 6 mois de formation, je me suis rendu compte que ce projet ne me stimulait pas. J’ai donc fait une demande a l’INRAE et a SMILE afin de changer de projet plus tôt, ce qu’ils ont accepté après délibération. J’ai donc continué deux mois à l’INRAE avant de rejoindre un projet le 01/03/2023 pour le gouvernement, dans une équipe de 7 personnes en Agilité SCRUM.

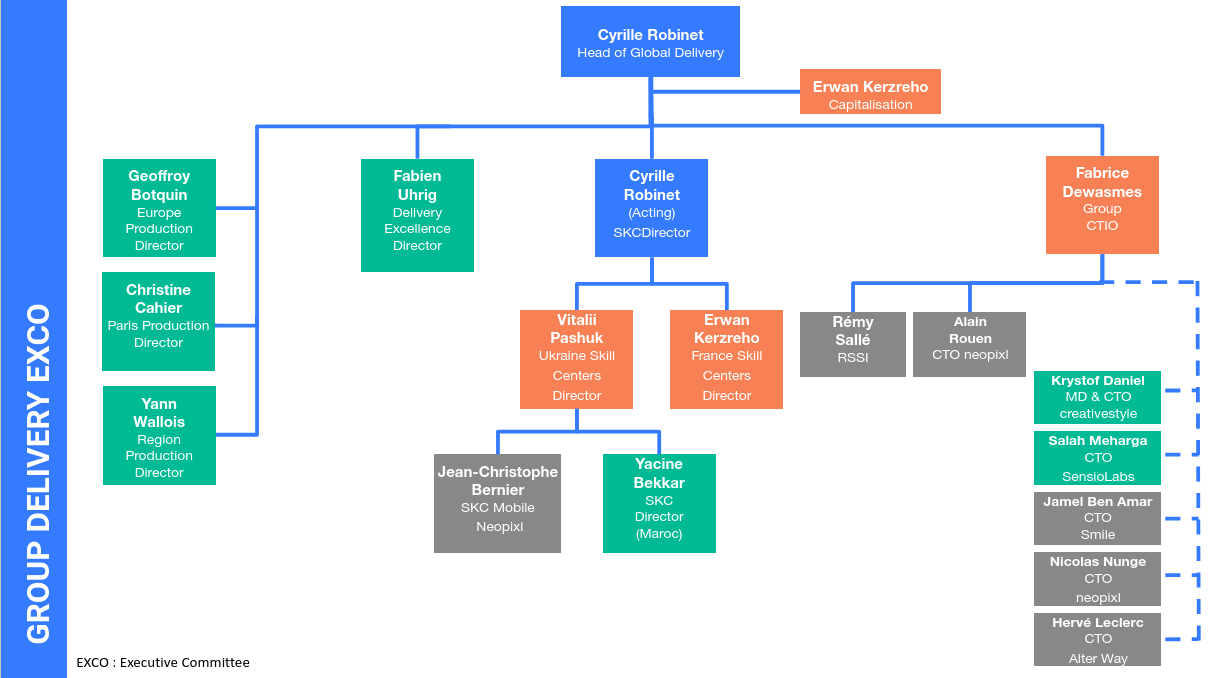
Je parlerais aussi, pour démontrer l’acquisition de certaines compétences, d’anciens projets effectués lors de mon cursus d’apprentissage.

# Environnement professionnel

## Présentation de l’entreprise SMILE

### Organisation de la structure

SMILE est une société spécialisée dans le développement de solutions open source. Crée en 1991 à Paris, elle est maintenant implantée dans 17 agences réparties dans 9 pays différents. Elle offre des solutions à de nombreux grands comptes français et européens, comme la préfecture de Lyon, la SNCF ou encore Total et beIN sport.

La société emploie plus de 2000 collaborateurs, et l’agence de Lyon est composée de 150 collaborateurs, dispersés en différentes équipes que nous retrouverons plus tard dans le dossier.

Voici l’organigramme de la structure de direction lors de mon arrivée :

Le groupe SMILE m’a embauché en octobre 2021 en tant que développeur java apprenti, placé sous Jamel Ben Amar. Lors de mon alternance, le groupe m’a confié différentes missions, que je décrirai tout au long de ce rapport.

### Présentation du SI, acteurs, informations et flux

## Activités de la structure

### Les différentes équipes de SMILE

Comme présenté précédemment, SMILE compte plusieurs équipes qui travaillent sur plusieurs projets différents. Cette partie va détailler les objectifs et les missions des équipes proches de mon travail. Les équipes peuvent-être séparées en trois pôles :

**Data**

Les équipes du pôle data suivent des projets dont le but est de mettre en œuvre un système de gestion de données. Cela se traduit par :

* De la relation client afin de trouver quelles sont les bonnes technologies pour implémenter ce système de gestion.
* Une création de systèmes qui correspondent aux demandes clients, cherchant à convenir au mieux aux besoins, en restant le plus optimisé possible.
* Un accompagnement client lors de la restitution du projet.

Les collaborateurs de ces équipes peuvent être amenés à utiliser diverses technologies comme *Talend*, *Power BI*, *Jedox* ou encore des API Manager tel que ***WSO2 API Manager.***

**Java/Backend**

Le pôle Backend est composé de multiples équipes, travaillant sur de nombreux projets, pour des clients ou en interne. Les collaborateurs de ces équipes fonctionnent en Agilité (SCRUM) dans des équipes d’environ 6 à 8 personnes constituées de développeurs backend, frontend, de testeurs, et d’un ScrumMaster.

Leur travail consiste à développer de nouvelles features sur un projet donné en Java, de participer aux créations des US (User Stories), faire de la relecture de code ainsi que de participer aux temps d’équipe agile.

**Régies**

Il est possible que des clients fassent appel à SMILE pour qu'un collaborateur ou une équipe, soit détaché dans leur propre structure pour aider au développement de leur projet. On appelle cela un projet en régie. C’est une coutume assez fréquente chez SMILE ; c'est ainsi que j'ai rejoint l’INRAE (**I**nstitut **N**ational de **R**echerche pour l’**A**griculture l’alimentation etl’**E**nvironnement).

Les autres pôles de SMILE (pour lesquels je n'ai pas collaboré) sont les suivants :

Javascript,Drupal, PHP, E-commerce, GED/ECM, Fonctionnel et testing, Project Management

### Les objectifs des activités

Les équipes du groupe SMILE suivent 4 « grandes valeurs » lors de leurs activités, afin d’atteindre un objectif concret, celui de pousser l’opensource afin de libérer les innovations dans l’IT et les entreprises.

Les 4 valeurs suivies sont les suivantes : la volonté d’être **ouverts** (Open Source et innovant) ; constituer des équipes d’**experts**, performants et curieux ; assurer des projets **durables**, sécurisés, en gardant une vision écologique ; ainsi que de continuer d’avoir une approche **humaine**, avec de l’écoute et du partage, permettant des projets sans conflits, avec une proximité entre tous les acteurs des projets.

### Les clients réguliers

Grâce à son expertise et à son ancienneté, SMILE a pu obtenir la confiance de nombreuses entreprises et intervient donc chez de nombreux clients différents. Certains font régulièrement appel aux services de SMILE, voici une présentation non exhaustive de ces clients :

* beIN SPORTS

Le défi de beIN SPORTS était de créer un site web flexible capable d’assurer la publication de contenus multilingues et multisites, permettant à 60 journalistes et éditorialistes, répartis dans 4 salles de rédaction, de pouvoir facilement contribuer sur le site.

Après 2 ans de développement SMILE produit une plateforme open source développée dans 4 pays et 7 langues, permettant l'intégration de nombreux prestataires.

Avec ce site SMILE remporte en 2015 en Symphony Awards du « *best B2C website of the year* »

* Suez

Lors de leur affiliation, pendant 2 ans SMILE intervient chez Suez sur les technologies *Symphony* et *eZ publish.*

Dans cette période Suez recherche des experts afin de concevoir une plateforme web optimisée en seulement un mois. Ce projet sera donné à une équipe Smilienne, qui développera la solution en seulement 30 jours.

Un an après ce projet Suez décide d'accorder sa confiance à SMILE en lui confiant la conception et la maintenance de ces 3 sites web.

* EDF

Depuis 2016 SMILE occupe la fonction d'experts référent au sein du département SPENCER – ESA (département centralisé des besoins informatiques du groupe).

Actuellement plus de 60 tickets sont gérés chaque mois sur des technologies diverses et variées.

* Total

SMILE et Total sont en lien depuis de nombreuses années. En effet 4 équipes sont mobilisées en parallèle contenant 20 collaborateurs, 7 projets sont réalisés par année avec Total et plus de 500 tickets sont traités chaque année.

* Vitam

Vitam est un projet demandé par l'État français. Le but est de créer une plateforme d'archivage, permettant un accès et une documentation facile et rapide, dont le code source serait redistribué et réutilisé par tous. Missionnés sur le projet, SMILE se lance dans la conception d'une plateforme hautement sécurisé, réutilisable par le plus grand nombre avec une durée de vie d’au moins 20 ans.

SMILE choisi une architecture modulaire pour le projet, permettant une flexibilité technologique ainsi qu’une optimisation plus simple dans le futur.

Le projet fut un succès et SMILE intervient encore quotidiennement dans la production et la maintenance de la plateforme.

## Mon activité à SMILE

### Place dans les équipes

Chez SMILE, j’ai été placé dans 3 équipes différentes que j’expliquerais plus en détail dans la partie IV. Voici malgré tout une courte introduction à ma place dans ces-dites équipes :

**Equipe WSO2**

Lors de mon projet sur *WSO2 API Manager (APIM ou AM)* j’étais apprenti **« API Constructer »** dans une équipe composée de 2 personnes à mon arrivée.

**Régie INRAE**

Pendant ma mission en régie a l’INRAE, j’ai rejoint une équipe de 7 personnes afin de développer en Agilité une solution de gestion de rôles et habilitations utilisant la solution *OpenIDM* de forgerock.

**Equipe Java**

J’ai rejoint un projet java en tant que second développeur Backend. Ce projet consiste en un développement d’une application web permettant la création de formulaires lors d'accidents entre véhicules contenant des matières dangereuses. Ce projet est destiné au ministre de l’Écologie et a été développé par une équipe de 8 personnes : 2 développeurs backend, 4 développeurs front-end un Scrum Master et notre Product Owner.

### Missions confiées

Création d’API, apprentissage WSO2, Développement solution démo client en java WSO2

INRAE = OpenIDM forgerock, création d’API, gestion de rôles et habilitation, contact client

Java : création de formulaire pour le gvt recensant les accidents de transports contenant des matières chimiques

Puis autre projets (CGI MCO puis dev projet Xénon – Canada – MSPR)

# Valorisation des compétences

## Projet WSO2

### Explication en détail du projet

Lors de mon arrivée chez SMILE, contrairement à ma fiche de poste (développeur java junior), je n’ai pas rejoint un projet java. En effet, j’ai été placé dans une équipe composée de deux personnes (Stéphane et Lihn), dont l’expertise était le logiciel *WSO2 AM*.

Le but de cette équipe « WSO2 » était d’installer, chez des clients demandeurs, la solution WSO2 AM. Puis de former les collaborateurs du client chez qui la solution a été installé afin de leur permettre de comprendre le logiciel. Cela pouvait requérir de modifier les APIs clients en format swagger API, afin qu’elles soient utilisables par WSO2 AM.

Qu’est-ce que *WSO2 API Manager* ?

*WSO2 AM* est un gestionnaire d'API, c’est-à-dire un logiciel permettant aux entreprises de créer, publier et gérer des APIs. Il prend en charge la gestion du cycle de vie des API, le développement d'applications, le contrôle d'accès, la limitation de débit et l'analyse dans un seul système. L’exposition des APIs est gérée via un outil d’édition et de publication ; et la consommation via une plateforme sur laquelle l’utilisateur pourra s’inscrire aux APIs souhaitées. Le transfert d’information est réalisé via une passerelle (Gateway) permettant l’interopérabilité des SI des deux parties (éditeur et souscripteur).

Voici un schéma d’explication :

Une image contenant texte, capture d’écran, Police, Graphique

Description générée automatiquement

Comme dit précédemment, j’ai rejoint ce projet dès mon arrivée à SMILE malgré mon titre de développeur java junior. J’ai donc dû me former sur cette solution avant de pouvoir moi aussi entamer des processus d’installation chez les clients.

Pour cela, j’ai suivi une auto-formation, interne a SMILE, pendant 6 semaines, afin d’apprendre au mieux le fonctionnement de la solution. Pendant cette formation, j’ai appris à créer et utiliser des APIs en swagger, les ajouter a WSO2 AM, créer des « applications » grâce à ces APIs, créer des rôles, des Gateway et à sécuriser les APIs.

**[PARLER DE LA FORMATION, QUOI D’APPRIS, QUOI DE COMPLEXE, OVERCAME SMTH ?]**

J’ai par ailleurs rédigé aux alentours de décembre 2021 un article interne pour SMILE, expliquant comment configurer Auth0 comme « Key Manager ». Vous pouvez le retrouver en Annexe.

Cette présentation m’a permis d’affirmer la fin de mon autoformation et de lancer mon autonomie sur les différents projets qui pourraient nécessiter ma présence.

Malheureusement, après quelques peaufinages de l’article en lien avec le responsable *WSO2* de SMILE, Gregory Eve, les clients demandant une installation *WSO2 APIM* se firent de plus en plus rare, et les projets que mes collègues avaient nécessitaient des compétences dans des technos que je ne maitrisais pas, comme *Talend* ou *Elasticsearch*.

J’ai donc demandé un point avec mon manager, mes coéquipiers ainsi que Gregory, afin de leur proposer le développement d’une application en ***Java***, utilisant l’**api manager** de ***WSO2***, qui permettrait de faire des démonstrations des possibilités d’utilisation de *WSO2 APIM* dans un contexte de projet.

Cette proposition a été très bien reçue par mon manager et Gregory, puis après quelques délibérations nous sommes venus à un objectif plus précis : développer une application en java, permettant de démontrer l’utilisation de 3 APIs intégrés sur une gateway *WSO2 APIM.*

J’ai ainsi pu commencer le développement de mon projet, Interne à SMILE et en solo. Pour commencer j’ai effectué un benchmark des API que je pouvais utiliser, voulant rester dans un contexte professionnel et en lien avec l’essence de SMILE, je me suis dirigé vers des APIs OpenSource, des *Open APIs.*

Lors de ce benchmark, 3 API sont ressorties par leur praticité et le fait qu'elle soit open source.

La première API que j'ai choisie est une API de découpage administratif appelée **API Géo**, provenant du site api.gouv.fr, qui référence des informations sur toutes les villes françaises (comme son code INSEE, sa superficie, son nombre d'habitants, etc.) ; Ensuite, j’ai inclus au projet **CoronavirusTracker**, une API postée sur github, qui référence l'évolution de l'épidémie de COVID-19 dans le monde ou dans un pays demandé ; Enfin, pour ajouter des informations à la ville choisie par l'utilisateur, j'ai intégré **WeatherAPI**, qui permet d'avoir des informations sur la météo sur une ville donnée grâce aux retours de l’API Géo.

Ces 3 API me semblaient être un choix adapté et professionnels pour une démonstration client, et après présentation à mon équipe, ce choix a été validé. J'ai malgré tout dû abandonner une API lors de mes recherches, en effet, la première appli que j'avais choisie pour l'épidémie de COVID-19 ne correspondais pas aux besoins du projet.

Cette étape de maquettage et de benchmark m’a pris environ deux semaines, et, dès l'accomplissement de la maquette, j'ai réuni mes 2 équipiers ainsi que mon manager afin d'avoir un retour avant de commencer le développement.

Voici à quoi ressemblait la maquette acceptée par mon équipe :



J’avais donc prévu que l’application soit constituée de deux écrans :

-Le premier contenant une liste déroulantes des villes de France, permettant à l’utilisateur de choisir sur quelle ville il voudrait un focus d’informations.

-Le second serait constitué de 3 parties distinctes, la première, en vert, contiendrait les informations « basiques » de la ville choisie, c’est-à-dire son nombre d’habitant, sa surface, sa région et ses code postaux (car une ville de France peut avoir plusieurs codes postaux, par exemple Metz en a trois). La deuxième partie, en bleu, contiendrait des informations météorologiques de la ville, les températures maximum et minimum, la vitesse des vents, l’humidité, etc. Et la troisième partie, en rouge, contiendrait les informations relatives aux Covid-19, l’épidémie encore assez présente dans la période de réalisation du projet.

Après l'étape de maquettage de benchmark j'ai dû intégrer les API que j'avais choisies dans une instance locale de WSO2 APIM, afin d'utiliser une option proposée par le logiciel permettant d'exporter les API en différents SDK. Cette fonction permet de générer les APIs en 4 formats différents utilisable au gré du développeur : en *Android*, en *Java*, en *javascript* ainsi qu'en *Jmeter*, comme présenté ci-dessous.

Une image contenant texte, capture d’écran, logiciel, Icône d’ordinateur

Description générée automatiquement

Une fois les ressources des API exportées en classes Java, j'ai pu commencer le développement de mon logiciel en *Spring MVC*, une technologie que ma collègue Lihn m’avait conseillé pour ce genre de projet.

Voici une brève explication du fonctionnement de Spring MVC.

Comme son nom l'indique Spring MVC utilise le design pattern MVC (modèle-vue-contrôleur) afin de créer des applications web en Java Avec le Framework Spring.

Le design pattern MVC est constitué de trois éléments : le modèle qui définit la structure de la donnée, la vue qui est l’élément que nous allons présenter à l'utilisateur (l’UI), et le contrôleur qui contient les fonctions qui modifie les données de la vue et du modèle.

Au centre de *Spring MVC* se trouve la dispatcher servlet, qui gère toutes les requêtes et réponses http, c’est le contrôleur principal de notre projet. Dans les modèles, nous allons retrouver les données qui seront transmises vers les vues. Les vues, dans *Spring MVC*, sont implémentées en Java Server Pages (JSP), conçues pour être implémenté de manière optimisée.

Lors de la création de mon projet, je n'avais jamais utilisé la solution *Spring MVC*, j'ai donc en premier lieu suivi un tutoriel sur le site W3 School afin de me familiariser avec la technologie et le lien entre Spring et le design pattern MVC que j'avais déjà utilisé auparavant.

En premier lieu j'ai créé un projet Spring sur start.spring.io, en choisissant un projet Maven en Java et en ajoutant les dépendances nécessaires à Spring MVC.

J'ai ensuite créé mon premier JSP, un index qui ne contenait qu'un texte en hyperlien qui renvoyait sur la 2nde page que je comptais créer. J'ai ensuite modifié mon web.xml afin qu'il contienne les données nécessaires au bon fonctionnement de mon projet j'ai créé un dispatcher servlet qui se chargeait au lancement de mon application. Avec cela j'ai créé une servlet nommée HelloWorldservlet, qui allait agir comme ma deuxième servlet, et à cela j'ai ajouté un « helloWorld.jsp » qui ne contenait qu'un titre servant d'exemple. En dernier lieu j'ai créé un contrôleur « HelloController » qui faisait la jonction entre le lien hypertexte et la vue « helloWorld » lorsque l’utilisateur clique sur le lien.

Ce tutoriel était très basique mais couvrait l'entièreté des nécessités principales afin de réaliser un projet Spring MVC. J'ai donc pu débuter la réalisation de mon projet de démonstration dès la fin de ce tutoriel.

J'ai donc commencé par créer mon projet Maven sur start.spring.io comme effectué précédemment. J'ai aussi généré 3 JDK (Java Development Kit) à partir de la fenêtre « développeur » de *WSO2 APIM*, que j’ai transformé en classes grâce au build de Maven. Ce build m’a permis d’intégrer les 3 *JDK* dans mon projet, me permettant d’utiliser les fonctions liées à ces kits de développement.

J’ai ensuite créé 3 classes différentes, InfoCommunes.java, InfoCorona.java et InfoMeteo.java, en suivant un tutoriel dans la documentation de *WSO2 APIM*, expliquant comment lier les JDK créés par le logiciel avec les 3 API précédemment implémentés sur mon instance locale de *WSO2 APIM*.

Voici un exemple suivi d'explications de la création d'une des classes afin de pouvoir utiliser l'API avec Maven :



Dans cette classe nous pouvons remarquer 3 blocs distincts, un bloc de déclaration, un bloc de configuration pour l'API Commune ainsi qu'un 3e bloc instanciation d'une liste, suivi d'un appel à une fonction de l'API Commune.

Dans la 1ere partie, j’ai déclaré « statcommune », une instance de l’objet « CommunesAPI » crée dans le jdk de l’API commune ; ainsi que « client », de l’ApiClient. L'objet ApiClient permet à notre objet statcommune d'appeler l’API Communes après la configuration.

La 2e partie concerne justement la configuration de l'objet client qui contient 3 fonctions de configuration pour la réception des données de l'API ainsi que la fonction « setBasePath » qui va contenir l'URL d'appel à l'API. Cette url est stockée dans une classe « Constants.java », créé auparavant, contenant toutes les constantes du projet.

Enfin, la 3e partie de notre fonction concerne la demande d'informations spécifiques à l'API commune. En effet, la classe « Communes API » contient plusieurs fonctions d'appel vers l'API commune, demandant plusieurs informations en fonction de la réponse attendue. Ici j'avais besoin de certaines données spécifiées dans la liste « needInfo ». Je fais ensuite appel à la fonction communeCodeGet, qui demande, en argument, un code insee ainsi qu'une liste de retours demandé, suivi d’un format de retour et de la « geometry », concernant les coordonnées GPS de la ville.

Cette fonction « getNeededInfoCommune » nous permet donc, lors de son appel avec un code Insee, d'appeler l'API Communes et d'avoir, en retour, un objet « commune » contenant le nom, les codes postaux, la surface, le département, la région ainsi que le nombre d'habitants d'une ville de France.

Comme dit précédemment, j'ai créé les classes InfoCorona.java et InfoMeteo.java, qui contiennent similairement la même fonction que InfoCommune.java, c'est-à-dire un appel à l'API en l'attente d'une réponse contenant des données demandées. Vous pourrez retrouver ces fonctions en annexe.

Après avoir créé les trois contrôleurs des APIs, et les avoir testés avec Postman, j’ai créé en parallèle mes modèles, vues et contrôleurs permettant l’utilisation de l’application.

Comme expliqué précédemment, il ne me fallait que deux écrans, le premier permettant à l’utilisateur de choisir la ville dont il voulait les informations ; et le deuxième affichant les informations de la ville choisie.

Ayant choisi de développer mon projet en *springMVC*, j’ai créé mes vues au format JSP, format permettant de contenir des balises HTML ainsi que des variables du modèle lié à la vue. J’ai donc débuté le développement du projet par la création des modèles, permettant d’avoir une vision sur les données nécessaires aux créations des vues.

Le premier modèle créé est nommé « SelectVille.java », et sera utilisé pour les données du 1er écran de sélection des villes, et ne contient que deux attributs, une liste de Communes « listeDonnéeesCommunes », que nous remplissons avec un fichier JSON contenant toutes les villes de France et un attribut « préférences » contenant le nom et le code Insee de la ville.

La fonction permettant de récupérer une liste de Communes provient d’une classe « ParseData.java » dont je parlerais ultérieurement.

Parallèlement, j’ai développé mon deuxième modèle, nommé InfoAPIsFront, contenant les données des 3 APIs. Voici le modèle, suivi d’une explication de sa structure :

Une image contenant texte, capture d’écran, logiciel

Description générée automatiquement

On peut tout d’abord observer l’appel aux trois contrôleurs précédemment développés avec, comme paramètre, le code Insee de la ville choisie par l’utilisateur afin de récupérer les données des APIs que nous souhaitons afficher.

Ensuite nous ajoutons toutes les données que nous souhaitons afficher sur la vue dans le modèle, ces ajouts sont décomposés en 4 blocs, référençant les APIs distinctes.

Les 3 premiers blocs sont assez rudimentaires, ne s’agissant que de simples initialisations de variables, mais le 4e bloc est assez intéressant, car il contient des calculs permettant d’afficher le nombre de cas et décès journaliers de l’épidémie de coronavirus en France, donnée dont j’avais prévu l’affichage dans les maquettes.

L’api des cas de covid-19 ne permettait pas un retour des cas/décès journaliers mais totaux de toute l’épidémie, afin donc d’avoir ces valeurs journalières, j’ai soustrait le nombre de cas totaux du jour précédent la requête aux nombres de cas totaux du jour de la requête.

Ainsi, mes deux modèles étaient complétés, il me fallait maintenant créer mes vues et mes contrôleurs.

J’ai commencé par créer ma vue « SelectVille.jsp », constituée, en plus d’un header et d’un footer classique, d’un formulaire permettant de sélectionner une ville dans une liste déroulante contenant toutes les villes de France, ainsi qu’un bouton de validation, qui envoyait la donnée du formulaire (le nom de la ville que l’utilisateur sélectionne et son code Insee).

La ville choisie était donc envoyée lors du clic sur le bouton de validation vers le contrôleur « UserPreferences.java », contrôleur qui ne contient qu’un string, la ville choisie par l’utilisateur, avec un getter et un setter.

Après la vérification du bon fonctionnement de la vue (vérifications effectuées avec *Postman* et des console.log afin de vérifier le bon transfert des données entre la vue, le contrôleur et le modèle), j’ai débuté le développement du deuxième écran de l’application.

La deuxième vue, « InfoApi.jsp » est lié au modèle précédemment affiché, « InfoApiFront.java ». Cette vue ne contient trois blocs de données, référencées par le modèle, utilisant le système de grilles en bootstrap, permettant un affichage en 3 parties comme celui prévu sur la maquette.

Dans le paragraphe décrivant le modèle « SelectVille.java », j’avais écrit que je reviendrai sur la classe « ParseData.java », car la création du fichier utilisé dans la classe a été une difficulté à surmonter durant le développement du projet. En effet, je voulais que l'on puisse présenter aux futurs clients la possibilité de choisir n'importe quelle ville dans une liste déroulante dans le premier écran.

Afin que cela soit possible, il fallait trouver un moyen permettant la recherche de toutes les villes, j’ai donc cherché un fichier contenant toutes les villes de France ainsi que certaines informations les concernant. Lors de mes recherches, je suis tombé sur un repo github (disponible en annexe), contenant un fichier Json d'environ 300000 lignes, composé de l'entièreté des villes françaises ainsi que des informations relatives à ces villes comme le code INSEE, le code postal, le nom de la commune, etc. (le lien du repo github est en annexe).

Lorsque j'ai trouvé ce fichier il me fallait faire un choix : fallait-il que je crée une base de données à partir du Json, ou était-il possible de le garder tel quel, malgré un fort temps de latence lors de son parsing ?

Afin de choisir la solution la plus adaptée au projet, je suis allé demander conseil à mes collègues pour avoir leur avis sur la question, dans le contexte de mon projet. En prenant en compte que le projet n’était qu’une démonstration que SMILE ferait aux clients pour montrer la solution *WSO2* APIM, nous cherchions une solution réduisant le coup de latence sans être top couteux au niveau humain. Après délibération nous avons convenu que la latence générée par le parsing était trop importante et qu’il fallait donc adapter le fichier, mais que créer une base de données serait peut-être excessive pour un projet cette envergure. La solution que nous avons choisie est donc la suivante : modifier le fichier Json afin de garder seulement les 10000 plus grandes villes de France, réduisant grandement le temps de chargement lors du parsing du fichier, tout en gardant la démarche de choix de ville lors de la démonstration.

Nous avions aussi pensé à une 3e solution, consistant à stocker le fichier Json en cache dans le navigateur, permettant une charge lente lors de la 1ere recherche mais plus rapide lors des suivantes. Malheureusement, cette solution n'était pas envisageable car l'application était vouée à être lancée puis stoppée à chaque présentation de la solution *WSO2 APIM*, obligeant à chaque démarrage de remettre le fichier en cache en effectuant une recherche.

Afin de modifier le fichier, j'ai créé un script dont le but était de supprimer toutes les villes après les dix millièmes plus grandes. À noter que par « grande » j'entends ici contenant le plus grand nombre d'habitants. Le script est retrouvable en annexe.

Le fichier ainsi créé me permet donc de récupérer les informations précise d'une ville choisie par l'utilisateur dans le premier formulaire de choix. Cette récupération s'organise grâce à une fonction de recherche via un contrôleur, « ParseData.java » que vous pouvez retrouver en annexe.

La création de ce fichier et du contrôleur finalisait le développement du projet de démo, environ un mois et demi (donc réellement 3 semaines sans compter l’alternance) après son commencement.

Comme démontré plus haut, pendant ce projet, malgré le fait que je sois seul, mes collègues Lihn et Stéphane m’ont été d'une grande aide et m’ont apporté un regard objectif sur mes idées et mes états d’avancement. Cela m’a permis de garder un recul sur les besoins du projet, mes idées ainsi que ma façon de développer.

### Quelles compétences validées par le projet ?

Ce projet a validé certaines compétences demandées par ce rapport, voici un état des lieux sous forme de tableau :

|  |  |
| --- | --- |
| Compétences validées | Partie du projet qui valide la compétence |
| A3C6 - Animer des réunions à distance afin de maintenir une dynamique de groupe et renforcer l’esprit d’équipe des membres en télétravail et/ou à distance | Réunion présentation AUTH2 + appli |
| A5C1 – Collecter les besoins métiers des utilisateurs en menant des interviews auprès d’eux pour comprendre leurs activités et leurs contraintes métier afin d’étudier les opportunités et la faisabilité technologique d’une solution applicative spécifique ou métier |  |
| A5C3 Développer une application adéquate selon la stratégie applicative de l’environnement en utilisant un langage de programmation approprié dans le respect du cahier des charges établi afin de répondre aux besoins utilisateurs/directions métiers |  |

### Recul et améliorations envisageables

Lors de la rédaction de cette partie, j’ai pris du temps pour réfléchir sur les différentes actions que j’aurai pu entreprendre dans cette équipe. En effet, malgré les idées que j’ai pu apporter et l’application de démonstration que j’ai développée, je n’ai pas assisté a d’installation du produit chez les clients de SMILE. Ces installations auraient pu m’apprendre beaucoup de choses, notamment les questionnements clients, comment amener une réponse concise et claire, et quand continuer a expliquer les différents outils.

En dehors de ça, dans mon projet, j’aurai aussi pu ajouter une plateforme de CI/CD automatisé lors de mon versionning. Cela aurait pu permettre une maintenance de l’application plus simple. La CI/CD est aussi une méthode presque obligatoire pour des gros projets et j’aurai pu me former en amont.

## Projet INRAE

### Explication en détail du projet

Pendant ma mission au sein de l’équipe Data, mon manager, Kevin Berthier, m’a interpellé au sujet d’une nouvelle mission qui pourrait m’intéresser. D’après Kevin, le projet était en Java, dans une équipe de 7 personnes en SCRUM, et qu’ils étaient prêts à intégrer un alternant dans leurs équipes. La proposition m’intéressait, j’ai donc accepté un point avec ces clients afin d’écouter la proposition et de montrer mes compétences et ma volonté de rejoindre si le projet m’intéressait toujours.

Une semaine plus tard, je rencontrais Olivier **[NOM OLIVIER]**, Denis et Chrisophe, respectivement l’un des managers du centre de developpement de l’INRAE, le product owner (PO) du projet et un des plus anciens développeurs du projet. Pendant cette réunion, le projet OpenIDM m’a été présenté, ainsi que les tenants et aboutissants de ce projet.

Olivier à débuté par me presenter l’équipe, les méthodes de management du projet ainsi que la solution OpenIDM, sans entrer dans les détails. Le projet était constitué de 7 développeurs, une ScrumMaster et d’un PO. Les 7 développeurs étaient répartis sur deux projets liés mais différents dans les technologies utilisées. 4 étaient sur Aria **[A VÉRIFIER]** et 3 sur OpenIDM (dont une nouvelle personne). Un de ces 3 développeurs n’était qu’à 50 % sur le projet, et ils avaient besoin de quelqu’un pour assurer sa charge de travail car il allait bientôt passer à seulement 25 % d’activité. C’est pour cette raison qu’ils ont la possibilité d’accepter un apprenti dans leur équipe.

Après cette présentation de l’équipe, j’ai pu en apprendre d’avantage sur la technologie sur laquelle j’allais développer, *OpenIDM*. J’ai appris que c’est une technologie développée par ForgeRock, qui gère des identités et des accès de comptes, avec un système de rôles et d’habilitations. Je reviendrai en détail sur *OpenIDM* dans la suite du rapport. Cette technologie est développée en JavaScript (et non pas en Java, contrairement à ce que Kevin m’avais dit), et est développée depuis longtemps par le groupe.

À la suite de ces présentations, j’ai pu me présenter, ainsi que mon parcours, mes objectifs et mes motivations. Je leur ai fait part de ma surprise concernant le langage de développement, mais je restais malgré tout intéressé, notamment par leur agilité et leur façon d’implémenter *SCRUM*.

Mon profil les a intéressé, et ils ont conclus l’échange par une validation avec Kevin concernant ma régie. Le marché était le suivant : je signais pour un an, 6 mois pour me former en semi-autonomie et 6 autres mois complètement autonome à développer l’application.

Deux semaines après ce point, j’ai rejoint l’équipe PID-IAM afin d’être formé et de développer sur la technologie OpenIDM.

Qu’est-ce qu’OpenIDM?

**OpenIDM** est un système de gestion d’identité. Développé en Java en 2010, et designé pour être flexible, il utilise par défaut le JavaScript comme langage pour les ajouts de règles métiers. Toutes les fonctionnalités de OpenIDM utilisent un modèle d’API RESTful.

La Gestion des Identités et des Accès (GIA) / Identity and Access Management (IAM) consiste à gérer les utilisateurs et leurs habilitations dans des systèmes d’information. OpenIDM permet donc de gérer quels utilisateurs ont accès à quelle information à travers le temps. Cela implique ainsi d’administrer la création, la modification, et les droits d’accès de chaque utilisateur.

OpenIDM intervient sur plusieurs catégories de l’IAM :

* L’initialisation de l'identité de l'utilisateur
* L’authentification de l’utilisateur
* La gestion des rôles de l’utilisateur
* L’habilitation des rôles
* L’accès aux ressources de l’utilisateur

### Compétences validées par le projet ?

### Recul et améliorations envisageables sur la compétence et bonnes pratique

## Projet « mtes déclaration évènement »

### Explication en détail du projet

Après mon projet a l’INRAE, j’ai finalement rejoint un projet en tant que développeur java, dans une équipe de 8 personnes en agilité SCRUM. Cette équipe était constituée de 2 développeurs backend (Claude Prabel et moi), de 4 développeurs front (Oscar Hernando, Matthias Goupil, Valentin Perello et Frédéric Houdemer), d’un techlead (Marc Terasson, succédé par Adrien Baikeche) ainsi que d’une testeuse (Célia Bocher).

Notre client était le ministère de la transition écologique. Il nous demandait de développer, en *java* et en *vue*.js, une app web à destination des conducteurs de transports de matières dangereuses. L'application leur permet de remplir un formulaire recensant les informations du transport (vitesse, routes, quel type de transports, météo, quels types de matières dangereuses, etc.) lors d’un accident.

Ce formulaire est ainsi stocké dans l’application avant d’être validé par une personne du gouvernement, un gestionnaire, chargé de vérifier la validité et le traitement de la déclaration.

Cette application comporte donc deux profils qui ont accès a différentes données, un profil déclarant (la personne qui remplit le formulaire) ainsi qu’un profil gestionnaire. Cela demande un traitement des accès en fonction du profil, que nous devons gérer dans la partie backend du projet.

Pour se connecter à la webapp que nous devions passer par la plateforme de connexion « cerbère », gérée par le ministère de la transition écologique. Lors de la connexion a cerbère, nous obtenions un token d’authentification (un *JSESSIONID)* contenant les informations de l’utilisateur (profil, genre, etc.).

Comme dit précédemment, notre équipe était gérée en méthode Agile SCRUM. Nous suivions donc les rituels SCRUM chaque jour et fin/début de sprint. Tous les matins à 9h45 nous avions 15 minutes de daily, permettant aux collaborateurs de l’équipe de décrire les actions effectuées la veille, les difficultés rencontrées et les demande d’aide si le besoin est nécessaire. Ces dailys permettaient d’avoir une visualisation de l’avancée du sprint et de savoir quelles tachent étaient en cours. Ces dailys duraient (en majorité) moins de 15 minutes, même si certains prenaient plus de temps (notamment lorsque notre PO les suivait).

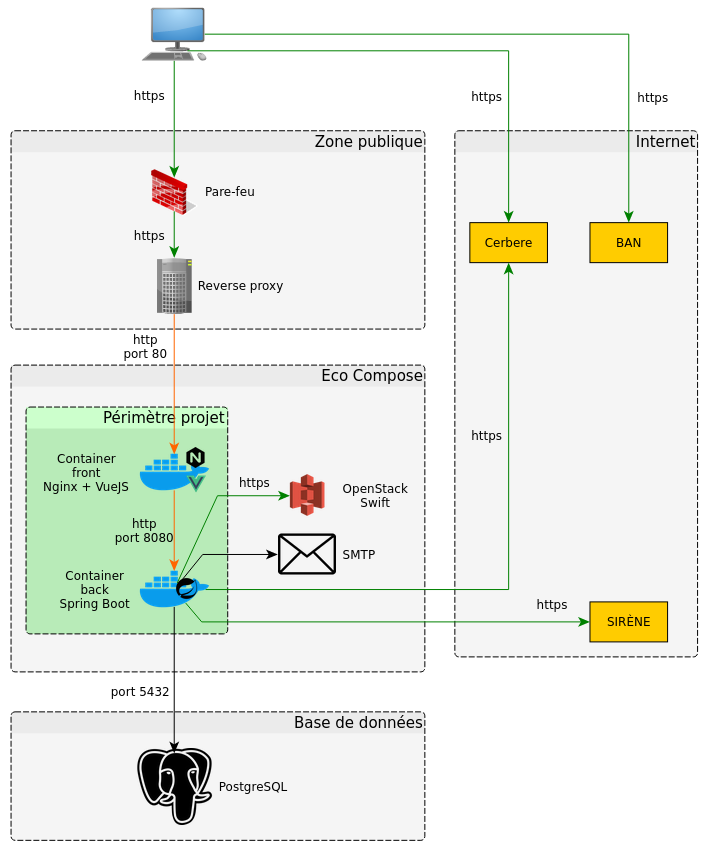
Les demandes clients étaient discutées au cours de sprint planning et transformées en Épopées, contenant plusieurs US (User Stories). Chaque US était ensuite votée via poker planning en fonction de leurs difficultés. Ce vote avait lieu deux fois, une première pour la partie front de l’US et une deuxième pour la partie Backend.

Chaque US sont ensuite sélectionnées par les développeurs, puis développées et testées (TU en Backend) avant d’être merge sur la version du sprint actuel.

Le versionning de notre projet utilisait *git*, et plus particulièrement une version de *gitlab* spéciale propre au ministère de la transition écologique. La branche principale était mise à jour chaque fin de sprint avant d’être utilisée pour créer une branche de travail, appelée sprint-x (x le numéro du sprint).

Nous développions ensuite les US a partir de cette branche. Chaque merge était précédé d’une MR (Merge Request) ainsi que d’une vérification de la validité du CI/CD (Intégration/Développement Continu), par une exécution de notre pipeline gitlab créée par Marc. **[← EST CE QUE CETTE PHRASE EST CORRECTE?]** Si la MR ou le job n’était pas validé, le merge ne passait pas et l’auteur de la MR devait modifier sa branche jusqu’à la validation des deux.

Je suis arrivé dans le projet a la fin du 3e sprint, l’architecture du projet avait déjà été créée, ainsi que les configurations nginx et les liens entre la webapp et la plateforme « cerbère ».



Voici un schéma de l’architecture du projet :

L’architecture du projet est donc décomposée en 4 parties :

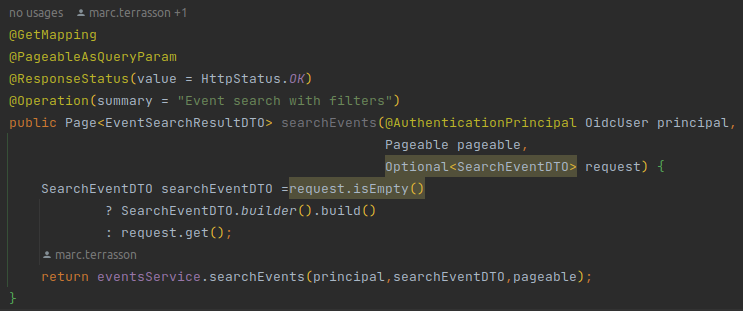
**[EXPLIQUER L’ARCHI]**

J’ai donc rejoint cette équipe à la fin du 3e sprint, et je ne connaissais rien du projet, j’ai donc passé les premières journées à suivre Claude dans ses taches afin de me familiariser avec l’architecture complexe du projet. Pendant ce temps, j’ai appris énormément de nouvelles choses que je n’avais jamais vu en Java. Voici une liste des sujets que j’ai pu apercevoir au cours de ces quelques journées, que j’ai approfondis dans la suite de mon projet et dont je reparlerais au cours de cette partie : les DTO ; les entités, les mappers ; les mappers générés par Spring ; les scripts *flyway ;* l’utilisation de docker pour conteneuriser une base de données et comment y accéder en java ; l’utilisation des JSESSIONID ; les liens entre Postman et des entités externes comme cerbère ; etc.

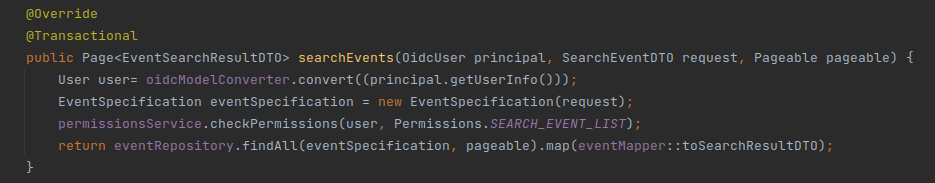
Peu après, j’ai pu commencer à choisir des US et développer les demandes clients rattachées a ces-dites US.

J’ai débuté par l’US 55, qui demandait la possibilité de « filtrer les évènements TMD en tant que gestionnaire ». Il était demandé de pouvoir filtrer en fonction du type de l’évènement, de la date (de début et de fin), du lieu et de pouvoir choisir si l’évènement demandé devait contenait des « reports » ou non. Ces filtres devaient retourner une liste qui s’adaptait en fonction des filtres. Par défaut, tous les filtres devaient être vides. La liste ne devait s’actualiser seulement lors de la pression d’un bouton « rechercher ».

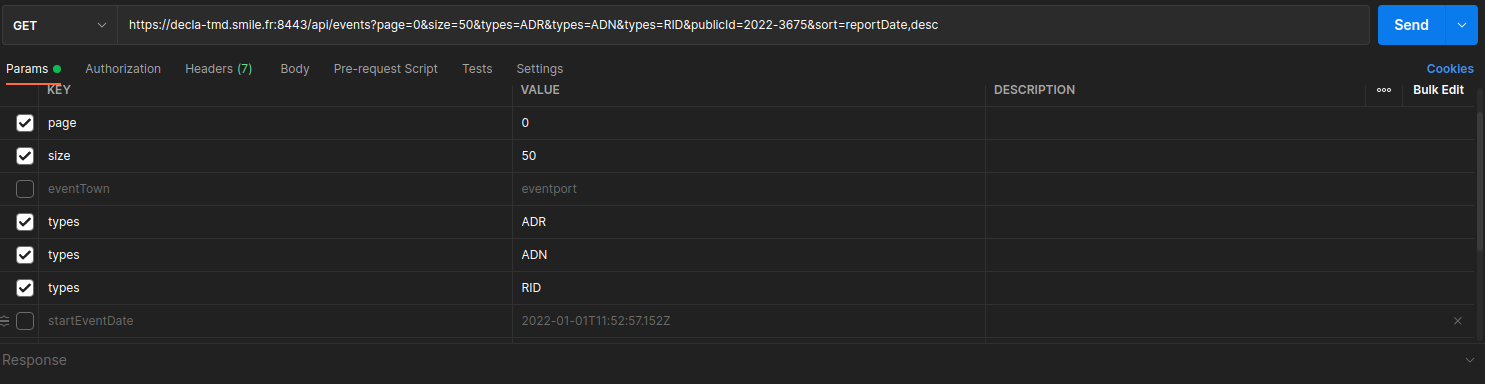
Afin d’implémenter cela, il fallait tout d’abord créer un DTO (Data Transfer Object), un objet qui contiendra certaines données afin de simplifier les transferts de données dans l’application. Ce DTO est appelé « SearchEventsDTO », et est constitué de 7 attributs, qui contiendront les 6 filtres demandés par l’utilisateur lors de la recherche ainsi qu’un ID.

Ensuite, j’ai ajouté ressource de l’API permettant d’effectuer une recherche d’évènement sur la base de données. Pour cela, j’ai ajouté une ressource au contrôleur des évènements « EventsController.java », qui contenait toutes les ressources des évènements vers l’API.

Cette fonction « searchEvents » crée une ressource API du même nom, qui permet de récupérer des évènements avec certains filtres (l’attribut « searchEvendDTO » request). Cette fonction fait un appel à une autre fonction de l’interface « eventsService.java », searchEvents. Développée dans la classe « eventsServiceImpl.java ».

Cette fonction vérifie tout d’abord que l’utilisateur faisant la requête (le « OidcUser » principal) à la permission de faire cette recherche. Si cette permission est validée, la fonction findAll est appelée, retournant une « Page » contenant les différents DTO. Ici on utilise le mapper « eventMapper » afin de transformer les résultats de la requête findAll en DTO, permettant de retourner les attributs attendus dans le contrôleur.

Après le développement de cette fonctionnalité, j’ai effectué une vérification sur Postman afin de confirmer le bon fonctionnement de la ressource implémentée. J’ai ensuite modifié mon code jusqu’à ce que l’appel fonctionne, et voici ce que nous obtenons lorsque l’on recherche un évènement avec des filtres.

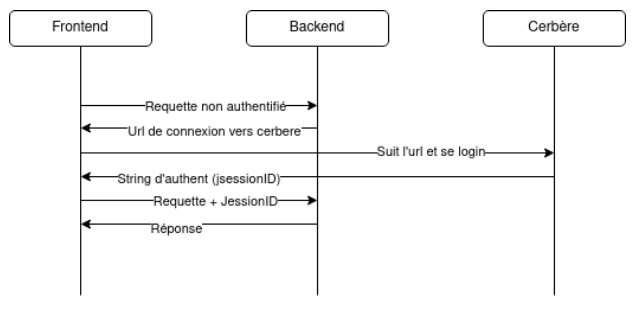
Lorsque l’appel fonctionnait sur Postman, j’ai ensuite écrit des Tests Unitaires (TU) afin de respecter les 80 % de couverture de code du projet. J’ai donc rédigé un test (en m’inspirant de l’existant), « test\_search\_events\_ok », qui simule, avec mockito, une recherche d’évènements avec des filtres.

J’ai ensuite fait là merge request, en demandant à Marc de vérifier la validité de mon code. Après quelques échanges et modifications la MR a été validée par Marc et envoyée sur la branche sprint-4.

La fin de mon développement sur cette US était aussi marquée par la fin du sprint 4, les jours suivants ont été marqués par des points SCRUM, notemment le sprint planning et la review du sprint 4. Cette review s’est organisée autour des ressentis des différents développeurs, nous permettant de parler des points que nous trouvions positifs et négatifs au sprint. Pendant le planning du sprint 5 le client nous a fait part de retours concernant la sécurité de l’application. En effet, l’architecture était faite d’une telle manière que les token d’authentification étaient des JSESSIONID, renvoyés par la plateforme cerbère lors de l’authent. Ce token ouvre des possibles faille de sécurité CSRF (Cross-Site Request Forgery). Pour étayer ces doutes, les clients avaient fait appel à des experts en cybersécurité, les failles CSRF n’étaient qu’une modification attendue parmi d’autres.

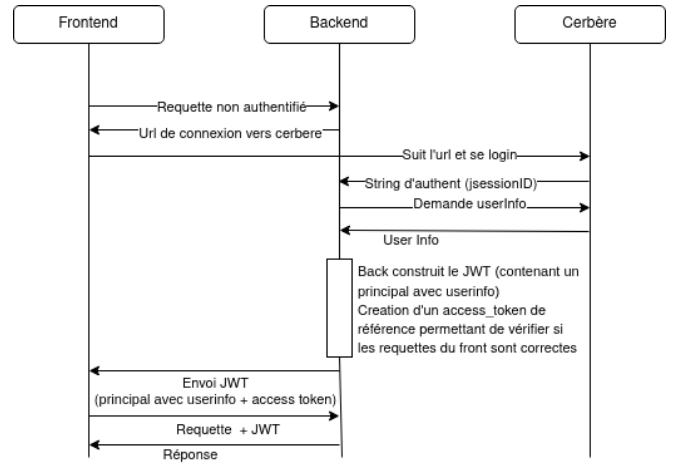
Il fallait donc qu’un développeur modifie l’architecture du projet afin de réduire la possibilité de ses attaques. Marc, notre techlead, s’était porté volontaire de ce changement mais a préféré me demander si la tache m’intéressait, car selon lui, l’expérience qu’elle pouvait m’apporter serait très importante dans mon parcours professionnel, malgré la difficulté de la tâche. J’ai donc accepté de prendre cette tâche, tout en restant supervisé par Marc.

Afin de me diriger vers la bonne solution qu’il envisageait, il m’a conseillé de me renseigner sur les JSESSIONID et les JWT (JSON Web Token) afin de comprendre leurs utilisations. Il m’a aussi demandé de représenter l’architecture des échanges entre notre application et « cerbère » afin de vérifier (et potentiellement corriger) ma compréhension de l’existant. J’ai donc débuté par schématiser notre architecture et les échanges entre les plateformes. Ce schéma à ensuite été révisé par Marc jusqu’à ce qu’il représente correctement notre application.



Cette architecture, comme expliqué précédemment, est sujet à des attaques CSRF. Une CSRF consiste à faire exécuter des actions sur le site Web à un utilisateur légitime, authentifié, et ce à son insu à travers un site malveillant. L’architecture permets à un utilisateur tiers de se placer en « Man in the Middle » (MIM) ou de modifier des évènements via des liens de phishing envoyés aux utilisateurs déjà authentifiés.

Après ma schématisation, Marc m’a expliqué que pour résoudre cette faille, plusieurs solutions existaient mais qu’il était plus intéressant pour moi de me focaliser sur l’utilisation des JWT. Il m’a donc prescrit de me renseigner sur les JWT, comment les utiliser et à quoi servaient ils. J’ai donc passé une partie du sprint à lire les documentations concernant ces tokens, tout en suivant des tutoriels expliquant comment lier Spring Security et les JWT.

À la suite de cet apprentissage, j’ai proposé à l’équipe du projet la solution qui me semblait adéquate, permettant d’utiliser les JWTs dans les échanges de données des formulaires. Cependant cette solution était basée sur la supposition que la plateforme cerbère pouvait renvoyer des JWT a notre plateforme back, j’ai donc demandé au ministère s’il existait une fonctionnalité permettant cela. Après quelques jours d’attente, j’ai reçu une réponse négative. Cela m’a obligé de revoir l’architecture que j’avais imaginé, représenté sur ce schéma :

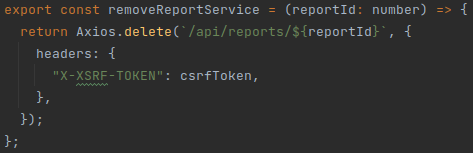
La mise en place des JWT demanderai donc de grandement modifier l’architecture sécurité du projet, demandant de créer 2 nouveaux contrôleurs, de modifier les appels entre back, front, et cerbère. Cela correspondait, selon moi, à une tache ayant 8 points de complexité.

Malheureusement, cette solution pose un problème de temps. En effet, 8 points de complexité équivalent à environ 4 jours de développement pour un développeur initié, et plus pour un junior/alternant. Ce coût en jour/homme étant trop élevé pour le sprint, nous avons pris la décision de chercher une autre solution permettant de contrer les attaques CSRF en prenant moins de ressources humaines.

Afin de notifier les équipiers de ce changement de direction j’ai écrit un court article, expliquant le plus simplement possible (permettant à tout le monde de comprendre) la cause de problème ainsi que les solutions envisagées. Les sujets de cet article furent ensuite discutés lors du daily suivant, afin de discuter de la solution à adopter pour un problème de ce type.

Après discussion, la solution sélectionnée consistait à modifier la configuration de notre serveur Nginx. Cela permettrait de sécuriser les JsessionID en seulement 2 points de complexité maximum. Cette solution permettra de sécuriser notre application aux attaques CSRF, sans avoir a modifier toute la structure architecturale du projet.

J’ai choisi de développer ces modifications sur notre serveur, malgré le fait que je n’avais suivi aucune formation sur Nginx. Les personnes de mon équipe étaient d’accord pour que j’apprenne a utiliser cette technologie avant de réaliser la tâche. J’ai donc passé quelques jours à lire la documentation Nginx tout en suivant des tutoriels et analysant des fichiers de conf déjà crées. Cela m’a permis de comprendre comment notre serveur fonctionnait et quelles modifications il fallait apporter a notre fichier de configuration pour bloquer les attaques CSRF.

J’ai donc modifié le fichier « [nginx.template.conf](https://gitlab-forge.din.developpement-durable.gouv.fr/snum/pnm4/DGPR/decla-event-tmd/-/blob/sprint-8/frontend/nginx.template.conf) » afin d’ajouter les attributs « secure; HttpOnly; SameSite=strict"; » dans notre « proxy\_cookie\_path ». Par ailleurs, j’ai ajouté aux headers de chaque requête ouvrant une faille de sécurité le token XSRF-TOKEN, voici un exemple :

Après un test sur Postman, un test sur ma version locale puis une relecture de mes modifications dans la Merge Request, les changements de notre conf nginx furent poussées sur la branche du sprint n°6.

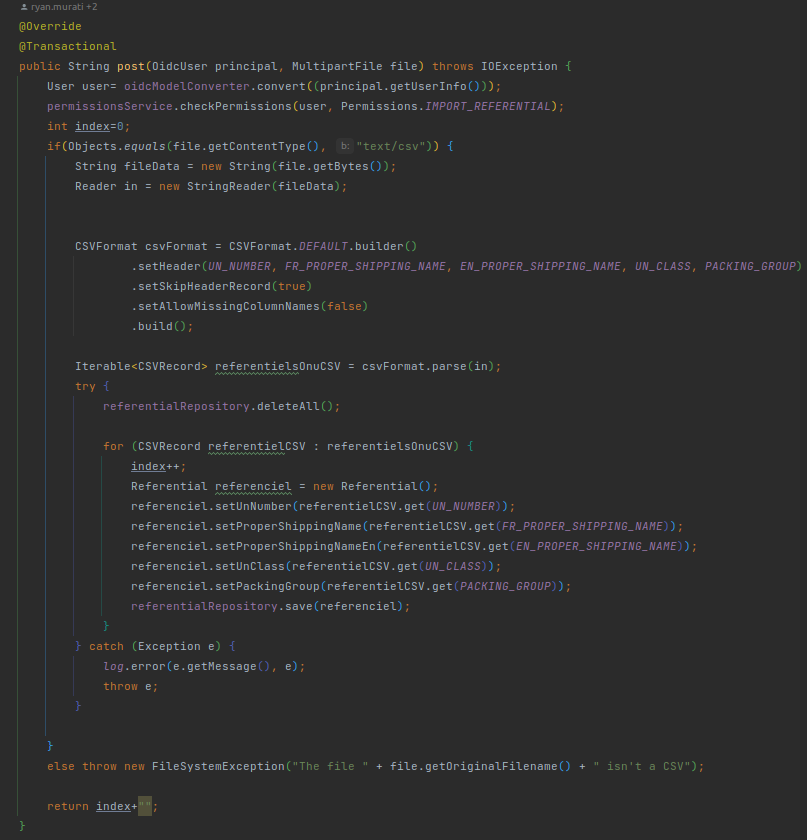
Après avoir terminé cette tâche, j’ai discuté avec Claude (la 2e développeuse backend du projet) afin d’avoir son avis sur les nouvelles US que je pourrai traiter. J’avais comme idée de développer une US de complexité 5 nécessitant de créer des contrôleurs permettant l’import de fichiers csv dans notre base de données. Claude m’a donné sa confirmation, et j’ai commencé à travailler sur l’US 95 : « Création référentiel N° ONU ».

J’ai débuté cette tache en créant un script flyway afin de créer une table « referential\_onu » contenant les différentes données demandées par notre client. En plus de ce script, j’ai par la même occasion crée une classe « ReferentialDTO.java » ainsi qu’une classe entité « Referential.java », contenant les attributs que je traite dans le script flyway. Vous pouvez retrouver le code de ces classes dans l’annexe, ainsi que le script flyway.

Une fois ces classes créées, il me fallait ajouter un « ReferentialController », contenant les ressources **get** et post utilisables par l’API ; ainsi qu’un mapper, permettant de faire le lien entre le DTO et l’entité « Referential ». Le contrôleur fait appel au « ReferentialService.java », une interface qui implémente les fonctions appelées lors du **get** et du **post**.

Ces fonctions sont ensuite « override » dans la classe « ReferentialServiceImpl.java », dans laquelle le traitement des méthodes de l’API est effectué.

Dans cette partie je vais me concentrer uniquement sur la méthode **POST** que j’ai développé, car elle représentait un défi assez intéressant : permettre à l’utilisateur d’importer un fichier csv en données. Pour cela, j’ai utilisé la bibliothèque « MultipartFile » de spring, permettant de récupérer un fichier en attribut d’entrée de la méthode. Voici le code de la méthode Post, que je vais détailler point par point ci-dessous :



Cette méthode peut être décomposée en quatre parties distinctes :

Premièrement, on vérifie que le « OidcUser principal », l’utilisateur qui fait la requête, à bien les permissions permettant d’importer un référentiel. Ces permissions sont gérées dans une classe « Permission.java », contenant (entre autres) une énumération des permissions existantes dans le projet ainsi qu’une méthode assignant les permissions à chaque profil de l’application (Gestionnaire, Déclarant et Consultant).

Après la vérification des permissions, je vérifie que le fichier en entrée est bien un csv. Si la condition n’est pas respectée, une exception est levée. Si elle est validée je crée un « StringReader in », contenant les données du fichier en entrée (fileData étant un String des données du MultipartFIle). Ensuite, je build un « CsvFormat », contenant les headers du fichier csv attendu (nom des headers fixés selon le client). Ce « CsvFormat » est configuré pour passer les headers lors du parsing du fichier par notre « StringReader ». Je crée aussi un « Iterable<CSVRecord> referencielOnuCSV » permettant d’itérer notre csvFormat.

Ensuite, je supprime toutes les valeurs possiblement sauvegardées dans la table référencée par « referentialRepository ». Ce comportement est celui attendu par le client. Je continue le programme en entrant dans une boucle for, qui crée un « Referential referenciel » pour chaque ligne de notre referencielOnuCSV. Après l’initialisation du referenciel, on utilise la fonction « save » du referentialRepository, ce qui ajoute la donnée dans notre table « referential\_onu ». Cette boucle est répétée pour toutes les lignes de l’Iterable et ne s’arrête qu’en cas de fin ou d’erreur. Les erreurs sont gérées par un try/catch.

A la fin du programme, je retourne un string avec le nombre de lignes crées lors de l’import (la variable index est incrémentée à chaque passage dans la boucle for), afin que le front puisse utiliser cette donnée pour montrer à l’utilisateur le bon fonctionnement de la fonction.

Après avoir développé ces fonctionnalités (**Get, GetAll** et **Post**), je l’ai, en premier lieu testé sur Postman, puis j’ai écrit les Tests Unitaires associés aux méthodes. En tout j’ai rédigé 8 tests, permettant de vérifier chaque possibilité des fonctions associées. Puis j’ai fait ma merge request, monitorée par Claude, et j’ai pu passer aux US suivantes.

Après la fin de cette US, mon activité dans le projet arrivait a sa fin, il ne me restait qu’une semaine avant la fin de mon dernier sprint. En effet, le développement de l’application arrivait a sa conclusion et il a été décidé que la maintenance de l’application (en MCO) serait faite par Claude et notre nouveau techlead Adrien.

Pour la dernière semaine j’ai donc décidé de finaliser toutes les US « simples » du backlog, afin que Claude puisse faire le rapatriement des données clients sur notre base. Les US traitées étaient majoritairement des ajouts ou suppression d’informations dans certaines entités (et donc tables).

J’ai quitté le projet le 18/07, après 6 mois d’activité. Le client, mes équipiers et moi-même étions satisfaits de mon travail et j’ai beaucoup apprécié travailler avec ces personnes. Le client était très à l’écoute et les échanges avec lui agréables. Nous n’hésitions pas à le contacter afin d’avoir des précisions sur les taches à réaliser ses réponses étaient toujours construites et il n’hésitais pas a échanger avec nous en vocal pour supprimer tout les points flous.

Ce projet a validé certaines compétences demandées par ce rapport, voici un état des lieux sous forme de tableau dans la prochaine page :

### Quelles compétences validées par le projet ?

|  |  |
| --- | --- |
| Compétences validées | Partie du projet qui valide la compétence |
| A1C2 – Collecter les besoins des directions métiers de l’entreprise afin de bâtir le projet de développement du SI dans le respect des besoins et attentes des directions métiers et de vérifier son alignement stratégique |  |
| A2C3 – Gérer un projet agile en utilisant les méthodes et outils adaptés à ce mode de fonctionnement pour tester, modifier et procéder par itération afin de réduire les délais de remise des projets de développement S.I. |  |
| A2C4 - Etablir des tableaux de bord de suivi de performance (qualitative et quantitative) de l’ensemble des ressources allouées à chaque étape-projet pour anticiper, visualiser et corriger les écarts en temps réel afin de limiter les contraintes de ressources et les  retards dans la réalisation du projet |  |
| A3C2 – Conduire une équipe projet en diffusant les fondamentaux de l’agilité : adaptation, flexibilité et  amélioration continue au sein de l’équipe afin d’être en mesure d’absorber les changements de priorité qui peuvent intervenir dans un contexte de forte contrainte de temps et d’incertitudes |  |

### Recul et améliorations envisageables sur la compétence et bonnes pratique

# Liste de compétences non validées par mes projets.

# Conclusion

## Résumé des projets

En un an et demi chez SMILE, j’ai pu découvrir de nouvelles façons de développer, des méthodes de travail que je ne connaissais pas, des nouvelles technologies. J’ai aussi eu l’occasion de m’intégrer dans des équipes déjà construites et j’ai pu affiner mes méthodes de communications avec les clients.

Mon passage dans l’équipe WSO2 à développé mon indépendance et ma prise de recul face aux retours construits que l’on pouvait me faire. J’ai aussi pu mettre un pied dans le monde de la Data, ce qui m’intéresse de plus en plus (surtout le côté BigData et DataVisualisation, que j’ai pu voir en cours).

Suivi par mes 8 mois à l’INRAE, la mission PID-IAM m’a permis d’évoluer mes compétences en *JavaScript*, une technologie que je n’avais vu qu’en partie lors de ma formation à l’IUT. Cette mission m’a aussi appris à affirmer mes choix envers mes managers tout en ayant du recul sur mes demandes. Cette mission a aussi conforté mon choix de parcours professionnel, en me faisant me rendre compte que la partie frontend et conf ne m’intéressent pas assez.

Le projet DECLA-MTES [bla bla decla mtes]

## Veille technologique et apprentissage en dehors du cursus scolaire

Lors de mon cursus universitaire, mes professeurs et intervenants mettaient l’accent sur le fait de faire de la veille technologique et de suivre les actualités concernant les métiers lié à l’informatique.

Malgré leurs insistances, je n’avais jamais pris le temps de créer des habitudes de veille technologique. J’avais bien suivi quelques flux RSS ou certaines actualités très médiatisées, mais rien de plus. Jusqu’à ce qu’une de mes connaissances me fasse découvrir le site « *Hackernews* », plus particulièrement le client *Hacki*. Hackernews est défini comme «un site de partage social de liens, avec pour objectif de créer une communauté proche de celle de [Reddit](https://fr.wikipedia.org/wiki/Reddit) Il est centré autour de l'actualité des technologies informatiques, du [hacking](https://fr.wikipedia.org/wiki/Hacking) et des [startups](https://fr.wikipedia.org/wiki/Startup), et promeut tout contenu susceptible de ‘’gratifier la curiosité intellectuelle’’ des lecteurs » - Wikipédia. Plus simplement, on peut définir Hackernews comme un forum de partage d’informations concernant l’informatique et ses alentours.

J’ai vite développé l’habitude de passer entre trente minutes et une heure de mes journées à lire des articles qui m’intéressaient tout en développant les connaissances qu’ils m’apportaient lors de mon temps libre. Certains articles m’ont fait découvrir de nouvelles technologies, comme LazyDocker, défini comme « A simple terminal UI for both docker and docker-compose, written in Go with the [gocui](https://github.com/jroimartin/gocui) library. » - LazyDocker Readme.md, que j’ai pu installer et expérimenter. Ou encore Quarkus, un framework java « fonctionnant nativement avec [Kubernetes](https://fr.wikipedia.org/wiki/Kubernetes" \t "Kubernetes), [OpenJDK HotSpot](https://fr.wikipedia.org/wiki/HotSpot" \t "HotSpot) et [GraalVM](https://fr.wikipedia.org/wiki/GraalVM" \t "GraalVM) » - Wikipédia. Ce framework cherche à réduire l’utilisation mémoire et le temps de démarrage des applications java, souvent long en Spring.

Ces deux technologies ne sont que deux exemples parmi des centaines d’autres articles que j’ai pu lire, vous retrouverez une liste en annexe.

## Ouverture sur l’avenir

Avant mes deux années chez SMILE, j’avais comme volonté de devenir un développeur Backend, spécialisé en Java. Cette volonté c’est développé et affiné pendant pendant mes projets, en ajoutant un côté plus humain à ce projet professionnel. A l’heure actuelle, mon projet professionnel est le suivant : devenir un développeur Backend (java de préférence, car c’est un langage très utilisé et toujours actif, tout en restant solide) dans des projets proche des utilisateurs finaux, afin d’avoir des retours sur les fonctionnalités que je pourrai développer. Les échanges clients

# Annexe

Lien de l’article Auth0 : <https://docs.google.com/document/d/1ix73uuw6ou-B6POphNXcAZHEMKA5nOYnNBFuzg0x0R8/edit#heading=h.rozdwj5vcws4>

Lien du repo github du JSON sur le projet wso2 APIM : <https://github.com/high54/Communes-France-JSON>

InfoCorona.java :

Une image contenant texte, capture d’écran

Description générée automatiquement

InfoMeteo.java :

Une image contenant texte, capture d’écran, violet, violette

Description générée automatiquement

ParseData.java :

Une image contenant texte, capture d’écran

Description générée automatiquement