

Rapport de stage

Stage de fin d'études

Master Impairs

01 avril 2024 - 27 septembre 2024

KAABECHE Rayane

tuteur de stage : TRINQUART Mathieu

professeur référent : DELPORTE Carole



sopra  steria

Université Paris Cité

Sopra Steria

Montpellier



Table des matières

1	Remerciements	2
2	Introduction	3
3	Sopra Steria	4
3.1	Le groupe Sopra	4
3.2	Le pôle santé social emploi	4
4	Le Projet NGA	5
4.1	Résumé du projet	5
4.2	Organisation du projet et des équipes	5
4.3	Changement d'organisation	8
4.4	Les technologies utilisées	10
4.4.1	Environnement organisationnel	10
4.4.2	Environnement Technologique	10
4.5	Architecture du projet	12
5	Ma contribution	14
5.1	Exemple de développement d'une US côté back-end	14
5.1.1	Prise d'une US et son analyse	14
5.1.2	Développement de la fonctionnalité	18
5.1.3	Développement des Test	24
5.1.4	Validation d'une US	27
5.2	Participation au différente cérémonie	28
5.2.1	La Daily	28
5.2.2	La Rétrospective	29
5.2.3	La Sprint review	30
6	Conclusion	32

1 Remerciements

Je tiens à remercier toutes les personnes qui m'ont accompagné et encadré durant ce stage :

→ Trinquart Matthieu, mon tuteur dans l'entreprise qui, a été présent tout le long de ce stage et qui n'a jamais hésité à prendre de son temps pour m'aider, ou encore m'expliquer certaine partie du projet

→ Nom de la deuxième personne

2 Introduction

Dans le cadre de ma deuxième année de Master en informatique, à l'Université Paris Cité, j'ai du réaliser un stage de fin d'étude d'une durée de 6 mois. Étant en spécialité IMPAIRS, j'ai été à la recherche d'un stage en adéquation avec les différentes connaissances acquise au cours de mon parcours.

J'ai orienté mes recherches sur des entreprises qui proposaient des projets complexe et intéressant, avec l'utilisation d'outils et de framework moderne et des problématiques algorithmiques intéressante. Mes exploration se sont beaucoup fait sur des projets orienté sur du web, car j'ai une appétence particulier pour ce domaine. C'est ce qui ici m'a amené à rejoindre Sopra Steria Montpellier

J'y ai été affecté au pôle santé sociale emploi, sur un projet appelé la NGA : Nouvelle Gestion Administrative de l'URSSAF.

J'ai donc rejoint le projet le 2 avril 2024, dans un équipe fonctionnant avec la méthode Agile, plus particulièrement SAFe. J'ai pu durant ce stage mettre en pratique les connaissances acquises durant ma formation, qu'elle soit en Algorithmique, en Programmation Orientée Objet ou encore en agilité. le stage m'a aussi apporté de nouvelles compétences, l'apprentissage de certain framework, les différentes cérémonie agile, comment s'organise un projet en entreprise, comment les échanges avec le client sont effectués et comment mettre en pratique leur demande, mais aussi plein d'autre aspect de la vie en entreprise.

Ce rapport présentée une partie du déroulé de mon stage avec une introduction de l'entreprise, le fonctionnement du projet avec la méthodologie appliquée, ainsi que le travail que j'ai pu apporter à l'entreprise.

3 Sopra Steria

3.1 Le groupe Sopra

Sopra Steria est une société française d'édition de logiciel et de service numérique, appelé plus souvent par l'appellation "ESN". C'est une société qui est né en 2015 de la fusion de Sopra et Steria. Elle compte aujourd'hui 20 000 collaborateurs en France, et plus de 55 000 au total dans le monde.

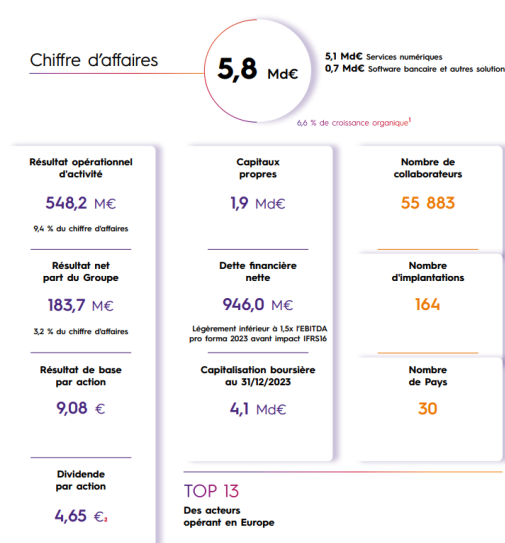


Figure 1 – les chiffres clés

On peut voir ici quelques chiffres qui concernent la société, c'est l'une des plus grosses sociétés de service française, étant dans le top 5, mais aussi européenne.

3.2 Le pôle santé social emploi

Comme dit plus haut, j'ai été affecté au pôle Santé Social Emploi. Cette agence se concentre dans les secteurs public en travaillant avec de nombreuses parties prenantes comme le gouvernement, les institutions et les collectivités. Elle aide les institutions à réinventer leur modèle dans le respect des politiques publiques et de la maîtrise de leur budget. Elle aide à apporter des objectifs précis comme :

- Améliorer en continu la qualité de service pour les usagers dans la collecte, la gestion et la redistribution des ressources.
- Accroître la productivité, dans le respect de contraintes budgétaires toujours plus serrées.
- Garantir l'excellence opérationnelle à tous les niveaux.

4 Le Projet NGA

4.1 Résumé du projet

Le projet consiste en la refonte et la modernisation d'une partie de l'ancien Système de gestion administrative dit GA de l'URSSAF. C'est une application interne de gestion des cotisations de divers populations d'individus. Le but ici est de repenser complètement ce système afin de donner naissance à une Nouvelle Gestion Administrative, ici dite NGA.

Le projet a commencé en 2019, avec une première population, les Artistes Auteurs (AA), qui à l'heure où on parle est déjà en production et est utilisé par certains agents de l'URSSAF. Par la suite une autre catégorie a été ajoutée au projet, les Travailleurs Indépendants (TI), celle sur laquelle j'ai du travailler. Même si pour l'instant l'application est limitée aux deux populations citées plus haut, elle pourrait être amenée dans le futur à être étendue à d'autres populations.

4.2 Organisation du projet et des équipes

Le projet est conduit selon la méthodologie Scrum en appliquant le framework SAFe, l'application étant complexe, il y a donc plusieurs équipes qui doivent travailler dessus en coordination. Ce framework permet d'avoir une meilleure synchronisation dans le cadre d'un aussi gros projet.

Comme dit au dessus le projet se base sur SAFe (Scaled Agile Framework), un cadre Agile qui s'appuie sur la méthodologie Scrum. En Scrum, les périodes de travail sont divisées en ce qu'on appelle des Sprints, qui, dans notre cas, avaient une durée de 3 semaines. Chaque équipe était composée d'au moins un Business Analyst, d'un Scrum Master et de l'équipe de développement. Chacun de ces rôles sera détaillé plus bas dans une section adéquate. De plus, Scrum propose de nombreuses cérémonies, comme le Sprint Planning, la Daily Standup, le Sprint Review, et d'autres, qui seront également détaillées plus bas.

Alors que Scrum est conçu pour des projets à petite et moyenne échelle, SAFe apporte des améliorations spécifiques pour gérer des projets à grande échelle, impliquant plusieurs équipes.

En SAFe, le déroulement du projet se compose d'incrément, eux-mêmes composés de sprints. Chaque incrément est divisé en 4 sprints, avec 3 sprints pour développer les fonctionnalités attendues durant cette période, et un dernier sprint, appelé Sprint Innovation and Planning (IP), d'une durée de trois semaines également. Ce sprint est généralement

dédié à l'innovation technique, comme l'ajout de nouvelles technologies ou la mise à jour de certaines versions. Cependant, dans notre cas, ce sprint avait principalement pour utilité la correction d'anomalies, mais aussi la mise en place du planning du prochaine incrément.

Pour incrément, plusieurs fonctionnalités majeures, appelées "fonctionnalités", sont ajoutées au projet. Ces "fonctionnalités" sont attribuées à plusieurs équipes, puis sont découpées en plusieurs tâches, appelées User Story (US). Chaque US est rédigées par les Business Analyst, et doivent être validé par le client avant le commencement de leur développement.

Plusieurs réunions Scrum et SAFe sont organisé tout au long du projet :

1. le Sprint Planning qui nous permet de valider toutes les User Stories qui devront être faites durant le sprint. Cette réunion nous permet aussi de planifier de manière plus global le prochain sprint.
2. La séance d'estimation, qui consiste à attribuer une valeur de complexité appelée "Story Point" à chaque User Story du sprint. Pour attribuer cette valeur, chaque développeur de l'équipe propose une estimation, avant d'entamer une discussion sur chacun des choix, afin de parvenir à un consensus et déterminer une note finale.
3. La Daily qui se déroule chaque matin et où chaque personne partage ce qu'elle a effectué la veille et ce qu'elle compte faire dans la journée. On y parle aussi de l'avancement global de l'équipe
4. En plus de tout cela chaque vendredi, nous organisons une réunion appelée "V1" pour suivre l'avancement de chaque équipe dans son sprint.
5. La Sprint review qui se déroule en fin de sprint durant laquelle chaque équipe montre aux clients ce qu'elle a produit durant ce sprint, en faisant une démonstration des différentes fonctionnalités développées.
6. La rétrospective qui se passe juste après la sprint review, C'est une réunion où l'on fait le point sur le sprint qui vient de se dérouler. On y donne notre ressentie, en parlant du bien comme du mauvais, on débat sur certain points, afin d'en ressortir des axes d'améliorations à apporter sur les prochains sprint.

7. Le sprint IP où toutes les équipes sont réunies afin d'améliorer le code, la couverture de test, de proposer certaines améliorations ou encore corrigé des bug. La tâche la plus importante dans notre cas, est la correction d'anomalie, travaillant sur un très gros projet, plusieurs comportements non attendus peuvent survenir, et le sprint IP est le moment le plus propice à leurs corrections.
8. Pendant le sprint IP, nous avons un PI Planning, durant lequel toute l'équipe se réunit pendant 2 jours dans l'un des sites pour se retrouver en présentiel afin de prendre connaissance des nouvelles fonctionnalités et de les découper en différents User Stories sur les sprints du prochain incrément, en fonction des capacités de l'équipe pendant les différents sprints.

Puis nous avons d'autres pratiques qui s'ajoutent à ceux de la méthode Scrum dût au framework SAFe que nous utilisons :

- Solution builder (SB) : Le Solution Builder est un développeur dont la tâche consiste à prendre une User Story et à la développer dans l'application tout en assurant la clarté de son code. Il doit réaliser des tests sur son code pour garantir sa qualité. Le Solution Builder est également en mesure de résoudre des anomalies (des bugs découverts dans l'application) ou de revoir le code des autres afin d'approuver la qualité de travail d'autres Solution Builders.
- Business Analyst (BA) : Les Business Analysts ont pour rôle de faciliter la communication entre les développeurs et les clients en dialoguant avec ces derniers. Leur objectif est de comprendre pleinement les besoins du client pour l'application. Ils sont responsables de la rédaction des User Stories et de s'assurer de la cohérence de nos tests Cucumber. Chaque fois qu'un développeur commence à travailler sur une User Story, il doit effectuer un point appelé "kick-off" avec le Business Analyst responsable de cette User Story afin de s'assurer d'avoir bien saisi tous les aspects de la tâche.
- Scrum Master : Le rôle du Scrum Master au sein de l'équipe est d'assurer une mise en œuvre efficace de la méthodologie Agile pour le projet. Il est chargé de coordonner les différentes réunions Agile, de veiller à ce que les membres de l'équipe respectent les procédures et de garantir la synchronisation entre les différentes équipes, vu qu'on applique le framework SAFe.
- Leader Technique : Le Leader Technique est chargé de la supervision technique et coordonne les activités de développement logiciel. Son rôle est d'orienter l'équipe

sur les aspects techniques du projet, tels que le choix de l'architecture, les bonnes pratiques de développement et les décisions techniques. Le Leader Technique se charge principalement des tâches les plus complexes, vérifie attentivement le code des autres Solution Builders pour garantir sa qualité, et apporte fréquemment son soutien pour aider les Solution Builders à surmonter leurs difficultés.

4.3 Changement d'organisation

Un des moments compliqué auquel j'ai dû faire face, a été le changement d'équipe et d'organisation, qui a eu lieu pendant la période de transition entre l'incrément 15 et l'incrément d'après mise en production dit "PostMep". On est passé de trois équipes à deux, avec une équipe pour la mise en production (Mep), avec une livraison prévue pour la fin d'année 2024, et mon équipe, qui est celle de d'après mise en production.

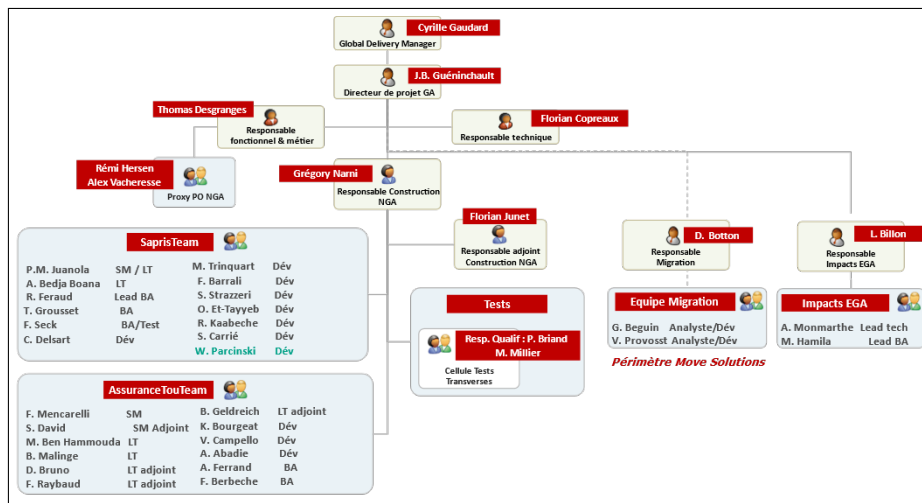


Figure 2 – Exemple de scénario

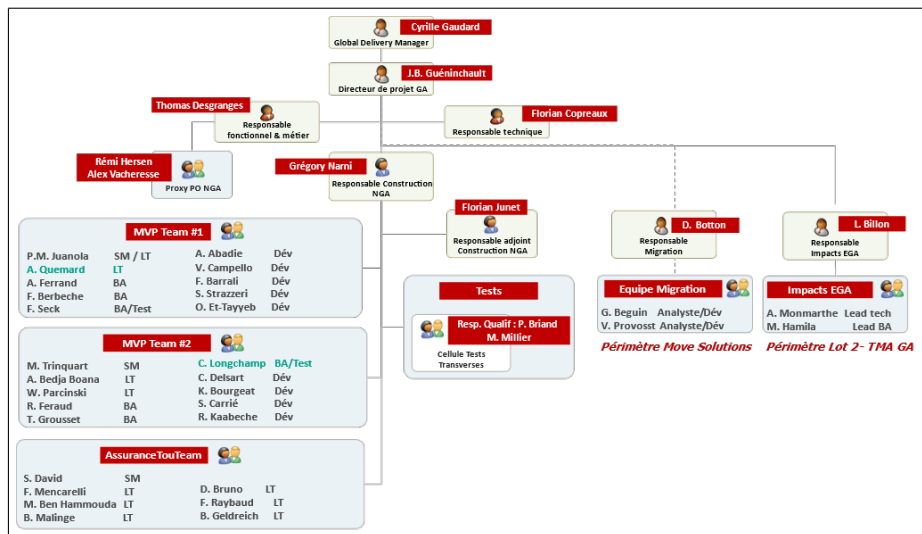


Figure 3 – Organisation de l'équipe après l'incrément 15

L'équipe Mep s'est basé sur la dernière version livrée à ce moment là, c'est-à-dire celle de l'incrément 15 et à partir de cette version elle s'est occupée, de la correction de différentes anomalies, de l'ajout de certaines fonctionnalités, mais aussi de problème de performance et d'optimisation.

Pour ce qui est de l'équipe PostMep, on s'est occupé principalement du développement de nouvelle fonctionnalité, avec aussi quelque Story Point par sprint consacré à l'analyse d'anomalie.

4.4 Les technologies utilisées



Figure 4 – board kanban

4.4.1 Environnement organisationnel

1. Jira : Outils de gestion de ticket côté Sopra Steria.
2. Redmine : Outil de gestion de ticket côté du client.
3. Confluence : Documentation fonctionnel du projet.
4. Environnement Microsoft Teams : Permet la communication écrite/orale au sein de l'équipe, avec une gestion de fichier et dossier partagé.

4.4.2 Environnement Technologique

1. Gestion de l'application
 - a. Jenkins : outil mis en place pour l'automatisation des compilations d'applications, utilisé en partie pour la livraison au client.
 - b. SonarQube : Outil d'analyse statique du code permettant un retour automatisé sur la qualité du code.
 - c. GitLab : Outil de gestion de version de projet et de code, pour l'ensemble de l'application.
2. Structure du code en Java et TypeScript
 - a. Spring : Framework logiciel permettant de définir la structure d'une application.

- b. Spring Boot : Extension de Spring qui facilite la création d'applications autonomes en fournissant une configuration par défaut et des conventions pour accélérer le développement.
 - c. Lombok : Librairie permettant la génération automatique de code récurrent (comme les getters, setters, constructeurs) se basant sur les annotations Java.
 - d. Angular : Framework côté client pour construire des applications web dynamiques en TypeScript.
 - e. Camunda :
3. Base de données
- a. PostgreSQL : Système de gestion de base de données relationnelle open source, réputé pour sa robustesse et ses fonctionnalités avancées telles que la gestion des transactions, les vues matérialisées, et les types de données complexes.
4. Test de l'application
- a. JUnit : Framework de test unitaire pour Java permettant d'écrire et d'exécuter des tests automatisés afin de vérifier le bon fonctionnement du code.
 - b. Mockito : Framework facilitant la mise en place de tests unitaires, en simulant des objets ou des comportements (mocking) pour isoler les composants à tester.
 - c. Cucumber : Framework permettant la mise en place de tests d'intégrations automatisés, qui ont vocation à encadrer le comportement général d'une application en utilisant un langage naturel (Gherkin).
5. Autres outils utilisés :
- a. Kafka : Outil permettant la diffusion de messages en continu, séparées en différents topics, et utilisé pour le traitement des flux de données en temps réel.
 - b. Docker : Plateforme de virtualisation par conteneurs, facilitant le déploiement d'applications en emballant tous les composants nécessaires à leur exécution dans un environnement isolé et portable.

4.5 Architecture du projet

Le projet adopte une architecture dites de micro-services. Cela veut dire que l'application est séparée en plusieurs "composant" indépendant, chacun d'entre eux ayant une utilité propre.

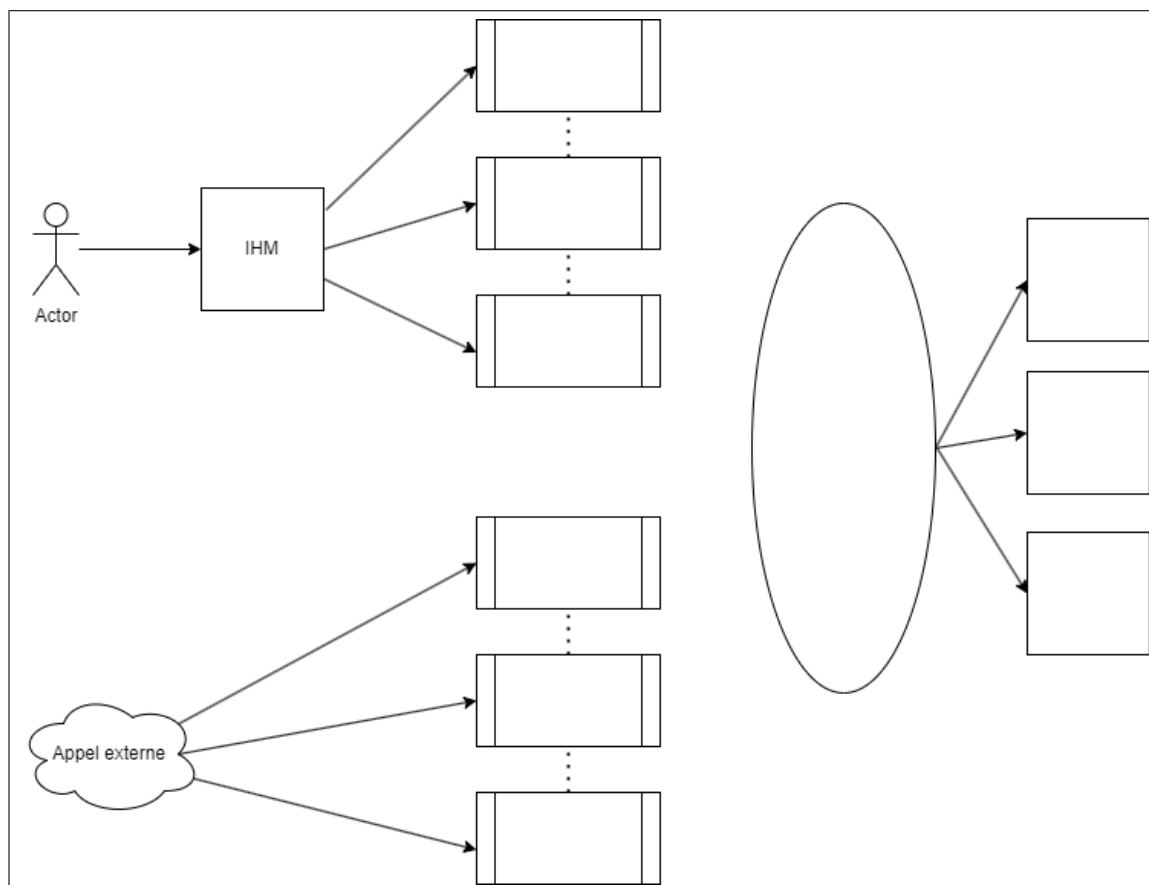


Figure 5 – Schémas de l'architecture du projet

Comme on peut le voir sur le schémas, l'architecture micro service ici est assez particulière, les appel au api rest ne se font pas directement sur les services, mais passe plutôt par des orchestration Camunda, qui représente chacune un concept métier. Elles communiquent avec les différents services. Certaines opérations étant très complexe, cela permet de travailler sur plusieurs services tout en évitant les duplications de code qui peuvent être commune à chaque orchestration. Une Orchestration va recevoir des données, qu'elles vont traiter, passant par plusieurs "Adapter" qui vont chacun s'occuper d'appeler un service spécifique pour traiter certaines données. Ces appels aux orchestration peuvent se faire depuis l'IHM, avec une action d'un agent de l'URSSAF, ou de manière "Automatique", où les informations seront fournies via une liasse, qui est un fichier XML contenant les in-

formations pour une affiliation, une radiation ou une modification, qui sera lu puis fourni à l'application

5 Ma contribution

5.1 Exemple de développement d'une US côté back-end

5.1.1 Prise d'une US et son analyse

On commence par choisir une US sur le board kanban du sprint actuel, ou toutes les US à faire, en cours ou même terminer sont visibles et elles sont rangées par fonctionnalité :

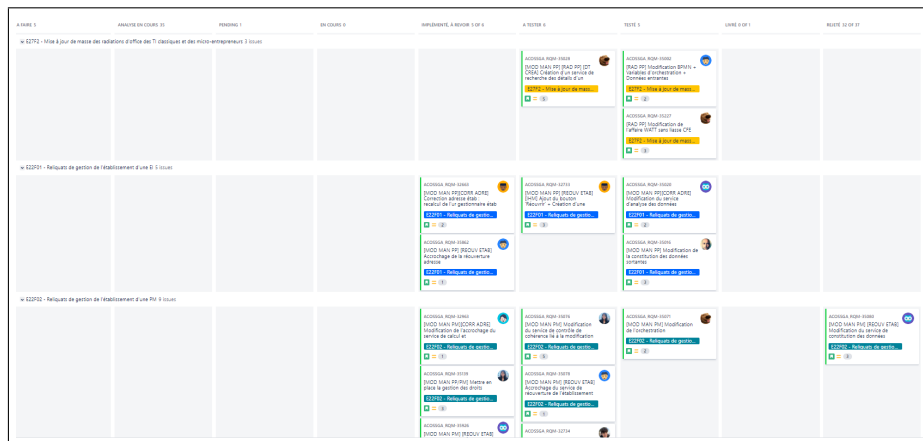


Figure 6 – board kanban

Après la prise d'une us, on change son statut en "Analyse en cours" et on s'assigne dessus, pour notifier de son commencement.

Une US se présente de cette manière :

<p>En tant que Système GA</p> <p>Je veux modifier le service de détermination des opérations du dossier</p> <p>Afin de permettre l'annulation d'une opération d'une activité secondaire et d'un individu CC</p> <p>Fonctionnalité(s) impactée(s) :</p> <p>DF10-SD03-FT400-Détermination des opérations du dossier (Modification)</p> <ul style="list-style-type: none"> • Modif swagger : ajouter donneesEntrantes.idActivite et donneesEntrantes.Conjoint en entrée • Etape A-1 : récupérer que l'activité sur laquelle on veut travailler si donneesEntrantes.idActivite est renseigné • Etape B-1 : Get le dossier individuLie que si activité CC sur l'entreprise de l'activité secondaire du dirigeant si donneesEntrantes.idActivite est renseigné • <u>Note</u> : Pour l'acti secondaire uniquement, annulation d'affi, annulation de rad et annulation de changement : <ul style="list-style-type: none"> ◦ de groupe professionnel ◦ de code profession ◦ de caisse retraite PL <p>Règle(s) impactée(s) :</p> <p>Fichier(s) de référence lié(s) :</p> <p>[Refonte GA] Accrochage_services_orchestration_annulation_derniere_operation_vGA.2.17.1.xlsx (modification)</p> <ul style="list-style-type: none"> • Modification de l'onglet FT400 - Liste opérations
--

Figure 7 – board kanban

On y retrouve un résumé fonctionnel en premier, ensuite la fonctionnalité impacté, ici "DF10-SD03-FT400".

Juste en dessous on peut voir le ou les fichiers de référence, qui sont des fichier excel qui vont servir à savoir par exemple à savoir quelles variables on va devoir alimenter et de quelle manière.

La première étape d'une US, est son analyse, on va passer un certain temps (cela peut être 1 heure comme une journée), à lire les règles impacté, leurs changements et leurs ajouts, voir par moment regarder dans le code ce qui est déjà fait. Généralement je note le tout, et si j'ai des questions je les prépare pour l'étape d'après. En deuxième on procède à un "kick-off". Cela consiste généralement en un appel avec un BA et un Testeur. Durant cet appel on va en premier lieu expliquer ce qu'on a compris de la tache, voir si tout est en accord avec ce qui est demandé d'un point de vu métiers et fonctionnel, poser quelque question si nécessaire, et on termine avec le Testeur, avec qui on s'accorde sur la méthodologie de Test, et sur le rôle de chacun de se côté la.

Après ça on peut changé le statuts de l'us sur jira en "started", pour indiquer au reste de l'équipe qu'on débute son développement à tout le monde.

Cette us va consister en une modification de service, avec l'ajout de nouvelles conditions, et l'ajout de deux nouvelles variables dans l'un des DTO.

Pour faire simple, sur l'application, on a une fonctionnalité d'annulations de dernière opération, qui permet à un agent de l'URSSAF de revenir en arrière sur un dossier. Cette fonctionnalité, au moment du développement de l'us ne permettait l'annulation d'opération que sur un établissement principal. Durant ce sprint, une des fonctionnalité ajoutée, consistait en l'ajout de la possibilité d'effectuer cette opération sur des établissements secondaires ou sur des Conjoint Collaborateur. Pour ajouter cette nouvelle fonctionnalité, on a du effectué plusieurs US, dont la mienne, qui elle va permettre de récupérer les bonne Opération a annulé.

Si on regarde sur l'US, on peut voir que j'ai 2 Étape de la règle FT400 à modifié, la A-1 et la B-1.

On à ici l'ajout de deux nouvelles variables, qui vont permettre de conditionner c'est deux étapes. C'est deux nouvelle variable sont "conjoint" un boolean et "idActivite un Long.

Je vais en premier recherché sur l'orchestration camunda concerné quelle service je vais devoir modifier. Pour faire simple, pour chaque grosse partie de l'application on a une orchestration, comme par exemple l'orchestration de Radiation-PP ou encore d'annulation. Ici celle qui nous concerne est l'orchestration d'annulation de dernière opération.

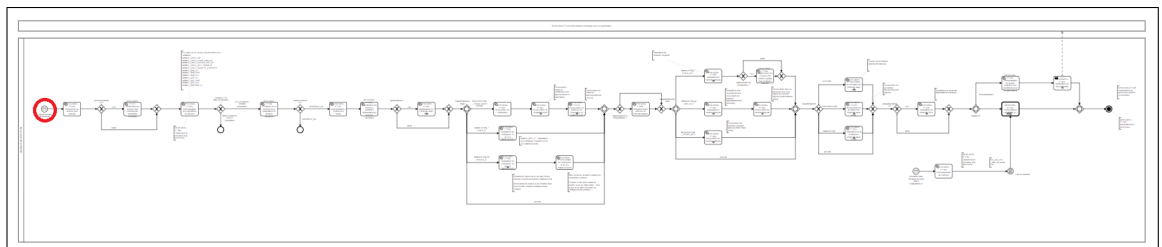


Figure 8 – Bpmn D'annul d'op

On a ici le bpmn d'annulation de dernière opération. Ici chaque boite est lié à une classe java que l'on va appeler un "Adapter". On va donc avoir un appel par une api rest qui va être effectué, qui va lancé le start de l'orchestration (le cercle rouge), et chaque Adapter va appeler un service pour effectué certaine opération, pour qu'à la fin après avoir tout parcouru, dans notre cas l'annulation soit effective. Pour cette US je dois modifier la FT400, si on regarde sur le bpmn on peut voir à quelle class Java cela correspond :

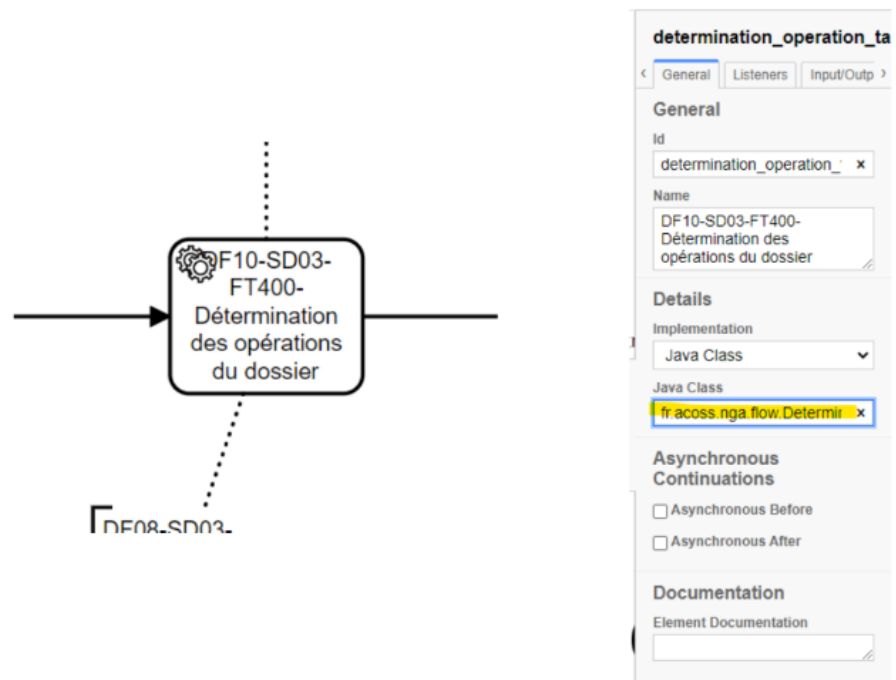


Figure 9 – Exemple de scénario

Ce qu'on peut voir surligné en jaune est la class java de l'adapter. Cette classe va juste prendre certaines données, les envoyer au service pour qu'il détermine la liste des opération annulable, et ensuite les mettront dans ses données.

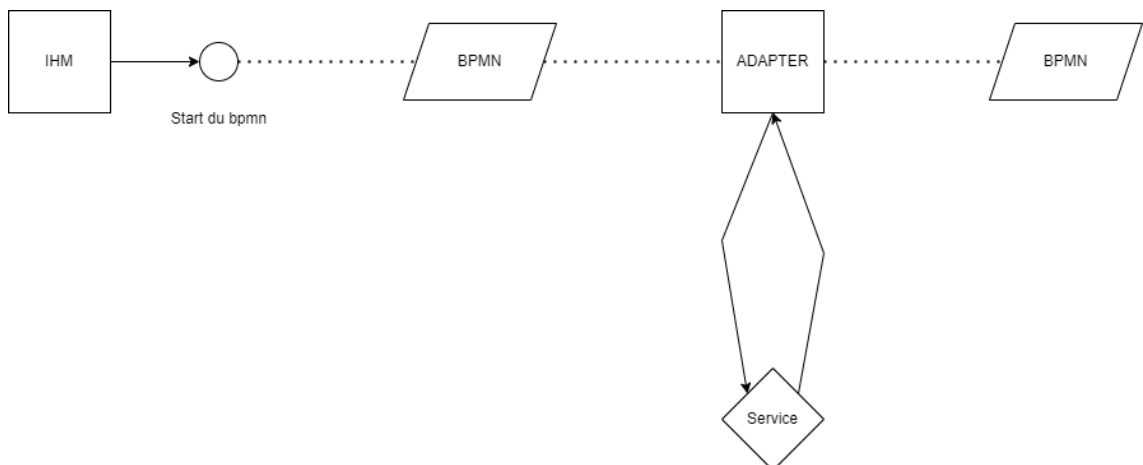


Figure 10 – Exemple de scénario

Si on regarde la class en question on peut voir que l'appel est fait sur un service qui se nomme "calcul". Ici cette fonction va envoyer à l'aide d'un appel reste, un objet Java

représentant une liste d'opération, et va recevoir en retour aussi une liste d'opération, qui sera les opérations que "servCalcul" a déterminé comme annulable (selons certaine règle établie).

```
private ResponseEntity<DeterminationOperationsAnnulablesOutput> callDeterminationOperationsAnnulables(final DeterminationOperationsAnnulablesInput input) throws PrismeSystemException {  
    return backendToBackendCallHelper.callRestBackend(this.urlCalcul + ConstantesEndpoints.ENDPOINT_DETERMINER_OPERATIONS_ANNULABLES,  
        this.urlCalcul, ConstantesEndpoints.ENDPOINT_ARCHIMED_NOT_SECURED_DETERMINER_OPERATIONS_ANNULABLES,  
        calculCallByArchimed,  
        HttpMethod.POST,  
        determinationOperationsAnnulablesInput,  
        DeterminationOperationsAnnulablesOutput.class,  
        this.calculScope);  
}
```

Figure 11 – Exemple de scénario

5.1.2 Développement de la fonctionnalité

Maintenant qu'on a le service à modifier, on peut commencé les ajouts demandé. La toute première étape est l'ajout de "conjoint" et de "idActivite" dans une class DTO appelé "DeterminationOperationsAnnulables". Pour cela on utilise un outil appelé "Swagger", qui dans notre cas va nous servir à partir d'un json D'obtenir plusieurs class, que ce soit en Java, ou Typscript, et c'est ce qui va justement nous permettre de faire le lien entre le Frontend et le Backend, une modification de "swagger" suffit pour "mettre à jour" les deux.

```
"DeterminationOperationsAnnulables": {  
    "type": "object",  
    "properties": {  
        "dossierIndividu": {  
            "$ref": "#/definitions/DossierNationalDto"  
        },  
        "dossierIndividuLie": {  
            "$ref": "#/definitions/DossierNationalDto"  
        },  
        "idActivite": {  
            "type": "integer",  
            "format": "int64",  
            "description": "idActivite",  
            "example": "1"  
        },  
        "conjoint": {  
            "type": "boolean",  
            "description": "conjoint",  
            "example": false  
        }  
    }  
}
```

Sur l'us il était mention de deux étapes de la règles a modifier la A-1 et la B-2.
 Sur l'étape A1 si l'on compare l'ancienne version à la nouvelle on obtient ça :

Etape A. Création de la liste des opérations annulables pour un individu (à partir du *dossierIndividu*)

Etape A-1 : Alimentation de la liste avec les activités individus de type dirigeant

A partir du *dossierIndividu* passé en entrée, alimenter l'objet *listeOperationsAnnulables* de la manière suivante :

- Récupération de l'ensemble de la liste des activités individus *dossierIndividu.individuRgaDto.activiteIndividu[]* dont :
 - *apourQualiteExercees.cdQualiteExercee* égale à 04, 05, 06, 07 ou 18
- La liste des activités individus est alimentée en fonction de la règle **RC_ALIM_492_001**

Figure 12 – Etape A-1 avant

① Cette étape est jouée uniquement dans le cas d'une annulation de dernière opération sur un dossier dirigeant (*Conjoint* = false).

Etape A-1 : Alimentation de la liste avec les activités individus de type dirigeant

A partir du *dossierIndividu* passé en entrée, alimenter l'objet *listeOperationsAnnulables* de la manière suivante :

- Si *idActivite* est alimenté en entrée de la fonctionnalité alors
 - Identification par son identifiant *idActivite* et récupération de l'activité individu concernée dans *dossierIndividu.individuRgaDto.activiteIndividu*.
- Sinon
 - Récupération de l'ensemble de la liste des activités individus *dossierIndividu.individuRgaDto.activiteIndividu[]* dont :
 - *apourQualiteExercees.cdQualiteExercee* égale à 04, 05, 06, 07 ou 18
- La liste des activités individus est alimentée en fonction de la règle **RC_ALIM_492_001**

Figure 13 – Etape A-1 avant Apres

On peut voir que deux nouvelle condition on été ajouté. La première ligne souligné, rajoute une règle, l'étape A n'est maintenant jouée que si Conjoint est faux. La deuxième ligne souligné, elle change les activités annulables récupérée. Si idActivite est null alors la condition reste la même, sinon on va récupérer les activités qui ont pour Id idActivite. Dans le code ca nous donne ca :

```
Long idActivite = determinationOperationsAnnulables.getIdActivite();
boolean isAnnulActiSecondaireOuCC = null != idActivite;

boolean isConjoint = Objects.requireNonNullElse(determinationOperationsAnnulables.isConjoint(), defaultObj: false);
```

Figure 14 – Exemple de scénario

Ici on récupère la variable *idActivite*, et on crée un boolean *isAnnulActiSecondaireOuCC*, qui va nous servir à plusieurs endroit pour savoir dans quel cas nous sommes. Ensuite on récupère *isConjoint*, avec comme valeur par défaut (si null) "false".

On a ensuite l'appel à une fonction qui s'appelle "determinerOperationsAnnulablesDossierIndividu", qui correspond à l'étape A, on va conditionner son appel par rapport à is-Conjoint, comme demandé plus haut et modifier la fonction.

```
DossierNationalDto dossierIndividu = determinationOperationsAnnulables.getDossierIndividu();
if(!isConjoint){
    determinerOperationsAnnulablesDossierIndividu(operationsAnnulables, dossierIndividu, idActivite, isAnnulActiSecondaireOuCC);
}
```

Figure 15 – Exemple de scénario

```
List<ActiviteIndividu> activiteIndividus;

// Etape A-1 : Alimentation de la liste avec les activités individus de type dirigeant
if (isAnnulActiSecondaireOuCC) {
    activiteIndividus = dossierIndividu
        .getIndividuRgaDto() IndividuRgaDto
        .getActiviteIndividu() List<ActiviteIndividu>
        .stream() Stream<ActiviteIndividu> |
        .filter(act -> Objects.equals(act.getId(), idActivite))
        .collect(Collectors.toList());
} else {
    activiteIndividus = OperationAnnulableUtils.
        recupererActiviteIndividusFromQualiteExercee(
            dossierIndividu.getIndividuRgaDto().getActiviteIndividu(),
            QUALITES_EXCERCEES_DIRIGEANT);
}

if (!CollectionUtils.isEmpty(activiteIndividus)) {
    majOperationsAnnulablesFromActiviteIndividus(operationsAnnulables, activiteIndividus);
}
```

Figure 16 – Exemple de scénario

Ce que j'ai ajouté ici, et la condition If-Else. Ce qu'il y a dans le else correspond à l'ancien code, lui inchangé. Pour ce qui est du "if", j'ai fait en sorte de faire ce qui était demandé, c'est à dire d'itérer sur la liste d'activité, et la filtrer par rapport à l'idActivite, ce qui fait ici dans le stream, avant de reconvertir le tout en une Liste.

On a ici terminé la partie concernant l'étape A-1.

Pour ce qui est de l'étape B-1 si l'on compare l'ancienne version à la nouvelle on obtient ça :

Etape B. Création de la liste des opérations annulables pour un individu lié (à partir du *dossierIndividuLie*)

Cette étape est à jouer seulement si un dossier *individuLie* est renseigné en entrée du service, sinon passer directement à l'étape C. Elle permet de venir ajouter dans la *listeOpérationsAnnulables*, la liste des opérations pouvant être annulées sur l'individu lié, (actuellement le conjoint collaborateur)

Etape B-1 : Alimentation de la liste avec les activités individus de type conjoint collaborateur

A partir du *dossierIndividuLie* passé en entrée, alimenter l'objet *listeOpérationsAnnulables* de la manière suivante :

- Récupération de l'ensemble de la liste des activités individus : *dossierIndividuLie.individuRgaDto.activiteIndividu[]* dont
 - *apourQualiteExercees.cdQualiteExercee* égale à 08 ou 09

La liste des activités individus est alimentée en fonction de la règle **RC ALIM_492_001**

Figure 17 – Étape B-1 avant

A partir du *dossierIndividuLie* passé en entrée, alimenter l'objet *listeOpérationsAnnulables* de la manière suivante :

- Si *Conjoint* = true et *idActivite* est alimenté en entrée de la fonctionnalité
 - Identification par son identifiant *idActivite* et récupération de l'activité individu concernée dans *dossierIndividuLie.individuRgaDto.activiteIndividu[]*
- Sinon si *Conjoint* = false et *idActivite* est alimenté en entrée de la fonctionnalité alors
 - Récupération de l'ensemble de la liste des activités individus : *dossierIndividuLie.individuRgaDto.activiteIndividu[]* dont
 - *apourQualiteExercees.cdQualiteExercee* égale à 08 ou 09
 - L'activité est rattachée à la même entreprise *fkEntreprise* que celle du dirigeant (**Etape A-1**)
 - L'activité est comprise dans la période d'activité de l'activité du dirigeant (**Etape A-1**)
 - *dtDebActivite Dirigeant* ≤ *dtDebActivite CC*
 - *dtFinActivite CC* ≤ *dtFinActivite Dirigeant*
- Sinon
 - Récupération de l'ensemble de la liste des activités individus : *dossierIndividuLie.individuRgaDto.activiteIndividu[]* dont
 - *apourQualiteExercees.cdQualiteExercee* égale à 08 ou 09

Figure 18 – Étape B-1 après

Dans cette étape on veut déterminer la liste des opération annulables du conjoint du dirigeant. Ce qu'on a entré c'est le dossier de même conjoint : *dossierIndividuLie*

Ici on peut voir que la modification de l'étape B-1 est un peu plus complexe, on se retrouve avec 3 conditions.

1. La première est si *Conjoint* est true et *idActivite* est alimenté, alors la *listeOpérationsAnnulables* sera alimenté avec toutes les activités dans le *dossierIndividuLie*, qui ont pour id, *idActivite*.
2. Pour la deuxième si *Conjoint* est faux et que *idActivite* est fourni alors on récupère toutes les activités du conjoint qui sont d'un types spécifiques (ici 08 ou 09) et qui sont liées à la même entreprise que celle du dirigeant. Ces activités doivent aussi être comprise dans la même période que l'activité du dirigeant.
3. Et pour la troisième et dernière condition on récupère juste toutes les activités de la personne qui ont pour code de qualité exercée 08 ou 09, sans vérifier d'autres critères spécifique

```

List<ActiveIndividu> activeIndividusLiees = null;

if (isAnnulActiSecondaireOuCC && isConjoint) {
    activeIndividusLiees = dossierIndividuLie.getIndividuRgaDto() IndividuRgaDto
        .getActiveIndividu().stream() Stream<ActiveIndividu>
        .filter(activi -> Objects.equals(activi.getId(), idActive)).toList();
} else if (isAnnulActiSecondaireOuCC) {
    LocalDate dtDebut = activeDir.getDtDebActive();
    LocalDate dtFin = activeDir.getDtFinActive();
    Long fkEntrepriseDir = activeDir.getFkEntreprise();
    activeIndividusLiees = OperationAnnulableUtils.
        recupererActiveIndividusFromQualiteExercee(dossierIndividuLie.getIndividuRgaDto().
            getActiveIndividu(), QUALITES_EXCERCEES_DIRIGEANT_LIE) List<ActiveIndividu>
        .stream() Stream<ActiveIndividu>
        .filter(activeIndividu -> activeIndividu.getFkEntreprise().equals(fkEntrepriseDir))
        .filter(activeIndividu -> activeIndividu.getDtDebActive().isAfter(dtDebut) || activeIndividu.getDtDebActive().isEqual(dtDebut))
        .filter(activeIndividu -> OperationAnnulableUtils.dtFinAfterDtFinCC(dtFin, activeIndividu.getDtFinActive()))
        .collect(Collectors.toList());
} else {
    activeIndividusLiees = OperationAnnulableUtils.
        recupererActiveIndividusFromQualiteExercee(dossierIndividuLie.getIndividuRgaDto().
            getActiveIndividu(), QUALITES_EXCERCEES_DIRIGEANT_LIE);
}

return activeIndividusLiees;

```

Figure 19 – Fonction recupererActivitesAnnulablesIndividu

On a ici la fonction entière de l'étape B-1 que j'ai faite, que l'ont va décrire partie par partie.

```

if (isAnnulActiSecondaireOuCC && isConjoint) {
    activeIndividusLiees = dossierIndividuLie.getIndividuRgaDto() IndividuRgaDto
        .getActiveIndividu().stream() Stream<ActiveIndividu>
        .filter(activi -> Objects.equals(activi.getId(), idActive)).toList();
} else if (isAnnulActiSecondaireOuCC) {

```

Figure 20 – Exemple de scénario

Pour la première condition le code est assez simple, comme pour l'étape A-1 on vient filtrer la liste en fonction de l'Id avec un simple Stream, avant de reconvertir le tout en une liste.

```

} else if (isAnnulActiSecondaireOuCC) {
    LocalDate dtDebut = activiteDir.getDtDebActivite();
    LocalDate dtfin = activiteDir.getDtFinActivite();
    Long fkEntrepriseDir = activiteDir.getFkEntreprise();

```

Figure 21 – Exemple de scénario

Pour ce qui est de la deuxième condition, qui elle s'applique quand idActivité est alimenté mais que Conjoint est false, on doit en tout premier filtrer en fonction de cdQualiteExercee (ici une variable matérialise par un nombre qui détermine quelle qualité est exercé par l'individu), heureusement pour nous une fonction existe déjà pour ça. Par la suite on va devoir filtrer en fonction d'un attribut nommé fkEntreprise qui correspond à un identifiant unique pour une entreprise. On va ici comparé cette variable, avec celle contenue dans chaque Activité. On va par la suite devoir que pour les activités que l'ont va sélectionner, les dates soit cohérentes

On va ici en tout premier préparer les variables dont on a besoin. Que ce soit pour les dates, ou pour l'id "fkEntreprise", on va les récupérer directement sur le dirigeant.

```

    activiteIndividusLiees = OperationAnnulableUtils.recupererActiviteIndividusFromQualiteExercee(dossierIndividuLie.getIndividuRgaDto().getActiviteIndividu()
        .stream() Stream<ActiviteIndividu>
        .filter(activiteIndividu -> activiteIndividu.getFkEntreprise().equals(fkEntrepriseDir))
        .filter(activiteIndividu -> activiteIndividu.getDtDebActivite().isAfter(dtDebut) || activiteIndividu.getDtDebActivite().isEqual(dtDebut))
        .filter(activiteIndividu -> OperationAnnulableUtils.dtFinAfterDtFinCC(dtfin, activiteIndividu.getDtFinActivite()))
        .collect(Collectors.toList());

```

Figure 22 – Exemple de scénario

Ici On va en tout premier filtrer en fonction de "cdQualiteExercee", pour cela on utilise une fonction qui prends nos activités, et une liste de Code qualité Exercée et nous retourne une nouvelle liste :

```

public static List<ActiviteIndividu> recupererActiviteIndividusFromQualiteExercee(final List<ActiviteIndividu> activiteIndividus, final List<String> qualiteExercees) {
    return activiteIndividus.stream()
        .filter(activiteIndividu -> ActiviteUtils.containQualitesExerceesPresent(activiteIndividu, qualiteExercees))
        .collect(Collectors.toList());
}

```

Figure 23 – Exemple de scénario

Ensuite, nous allons créer un stream à partir de cette liste filtrée. Nous allons d'abord utiliser la fonction "filter" des streams pour récupérer les activités avec le bon Id fkEntrprise. Ensuite, nous allons affiner la liste en fonction des dates.

- Premier filtrage par date de début d'activité : Nous appliquons d'abord un filtre pour ne conserver que les activités dont la date de début (dtDebActivite) est postérieure ou égale à la date de début (dtDebut) de l'activité du dirigeant, dans la bonne période.

- Deuxième filtrage par date de fin d'activité : Nous appliquons ensuite un autre filtre qui utilise une fonction appelée `dtFinAfterDtFinCC`. Cette méthode vérifie que la date de fin de l'activité récupérée (`activiteIndividu.getDtFinActivite()`) est avant ou égale à la date de fin spécifiée (`dtfin`). Ici, une méthode a dû être créée, car la condition était un peu plus complexe. Des erreurs du type "null Pointer" pouvaient survenir, la date de fin du dirigeant pouvant être "null", avec une activité toujours en service.

```
/** Permet de savoir si la date dtFin est supérieur ou égale à la date dtFinCC ...*/  
7 usages  rayane kaabeche  
public static boolean dtFinAfterDtFinCC(LocalDate dtFin, LocalDate dtFinCC) {  
    return null == dtFin || (null != dtFinCC && (dtFinCC.isBefore(dtFin) || dtFinCC.isEqual(dtFin)));  
}
```

Figure 24 – Exemple de scénario

```
} else {  
    activiteIndividusLiees = OperationAnnulableUtils.  
        recupererActiviteIndividusFromQualiteExercee(dossierIndividuLie.getIndividuRgaDto().  
            getActiviteIndividu(), QUALITES_EXCERCEES_DIRIGEANT_LIE);  
}  
return activiteIndividusLiees;
```

Figure 25 – Fin de fonction

La toute dernière partie de la fonction et le tout dernier "else" ou a juste a récupérer les activités en fonction de "cdQualiteExercee", avant de return notre list

5.1.3 Développement des Test

En ce qui concerne les tests, nous appliquons une méthode appelée Test-Driven Development (TDD). En TDD, on commence toujours par écrire un test simple, qui, au départ, est censé échouer. Ensuite, on développe une partie de la fonctionnalité pour que ce test passe. On complexifie ensuite le test, on développe la suite de la fonctionnalité, et ainsi de suite. De cette manière, nous nous assurons que notre développement fonctionne en passant les tests, plutôt que d'adapter les tests à notre code.

```

@Test
void dtFinAfterDtFinCCTest() {
    LocalDate date1 = LocalDate.of( year: 2014, month: 2, dayOfMonth: 2);
    LocalDate date2 = LocalDate.of( year: 2014, month: 2, dayOfMonth: 2);
    Assertions.assertEquals(OperationAnnuableUtils.dtFinAfterDtFinCC(date1, date2), actual: true);

    date1 = LocalDate.of( year: 2014, month: 2, dayOfMonth: 2);
    date2 = LocalDate.of( year: 2024, month: 2, dayOfMonth: 2);
    Assertions.assertEquals(OperationAnnuableUtils.dtFinAfterDtFinCC(date1, date2), actual: false);

    date1 = LocalDate.of( year: 2014, month: 2, dayOfMonth: 2);
    date2 = LocalDate.of( year: 2013, month: 2, dayOfMonth: 2);
    Assertions.assertEquals(OperationAnnuableUtils.dtFinAfterDtFinCC(date1, date2), actual: true);

    Assertions.assertEquals(OperationAnnuableUtils.dtFinAfterDtFinCC( dtFin: null, date2), actual: true);

    Assertions.assertEquals(OperationAnnuableUtils.dtFinAfterDtFinCC( dtFin: null, dtFinCC: null), actual: true);

    Assertions.assertEquals(OperationAnnuableUtils.dtFinAfterDtFinCC(date1, dtFinCC: null), actual: false);
}

```

Ici notre première exemple de Test, est pour la fonction `dtFinAfterDtFinCC()`, ou l'on va lui donner deux date, est qu'elle return bien "True" si la date en première argument est la meme ou supérieure à la deuxième, et sinon return "False". Pour ce la on utilise la fonction `assertEquals` de JUnit, qui va prendre deux arguments et vérifier si ils sont égaux.

```

@Test
void determinerOperationsAnnuellesCCTest() throws IOException {
    // Arrange
    DeterminationOperationsAnnuelles input = TestUtils.prepareDonneesFromFile
        ( pathFichier: TEST_FOLDER + DOSSIER_CONJOINT_INPUT, DeterminationOperationsAnnuelles.class);
    ResponseDeterminationOperationAnnuableDto expectedResponse = TestUtils.prepareDonneesFromFile
        ( pathFichier: TEST_FOLDER + OPERATIONS_COMPLETED_CONJOINT, ResponseDeterminationOperationAnnuableDto.class);

    List<OperationAnnuableDto> expectedOperationsAnnuelles = expectedResponse.getOperationsAnnuelles();
    List<OperationAnnuableSimplifieeDto> expectedOperationAnnuableSimplifiees = expectedResponse.getOperationsAnnuellesSimplifiees();

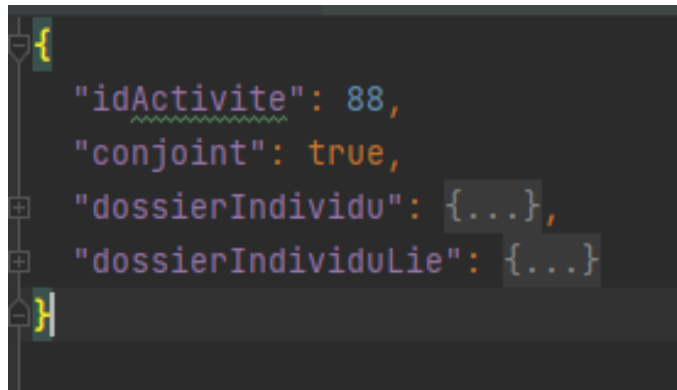
    // Act
    ResponseDeterminationOperationAnnuableDto response = this.service.determinerOperationsAnnuelles(input);
    List<OperationAnnuableDto> operationAnnuellesOutput = response.getOperationsAnnuelles();
    List<OperationAnnuableSimplifieeDto> operationAnnuellesSimplifieesOutput = response.getOperationsAnnuellesSimplifiees();

    // Assert
    assertThat(operationAnnuellesOutput)
        .isNotEmpty()
        .hasSameElementsAs(expectedOperationsAnnuelles)
        .hasSameSizeAs(expectedOperationsAnnuelles);
    assertThat(operationAnnuellesSimplifieesOutput)
        .isNotEmpty()
        .hasSameElementsAs(expectedOperationAnnuableSimplifiees)
        .hasSameSizeAs(expectedOperationAnnuableSimplifiees);
}

```

Figure 26 – Exemple de scénario

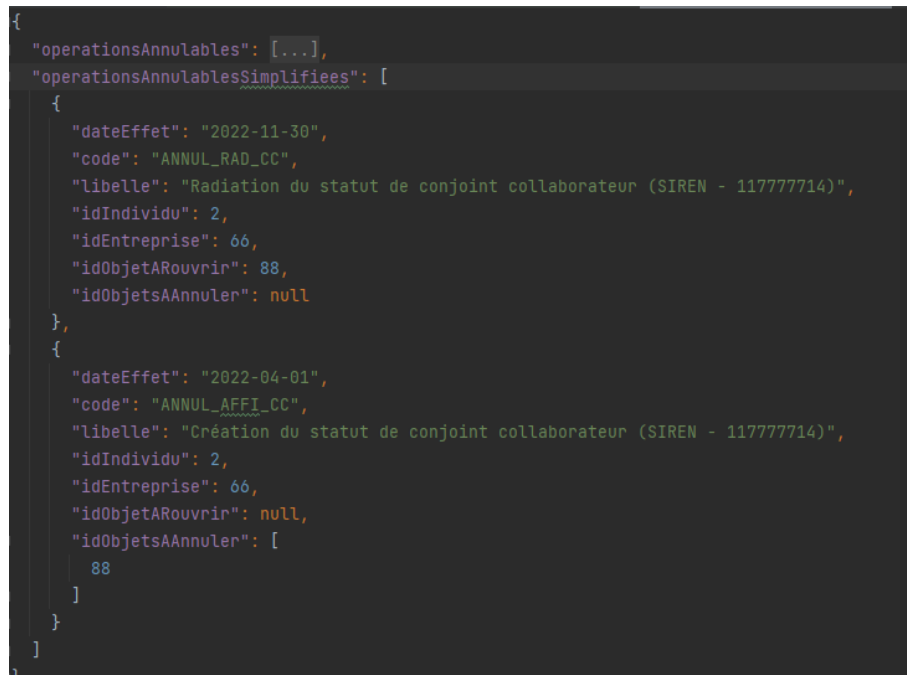
Nous passons, au cas de test un peu plus complexe, ou ici nous allons tester avec un certain scénario de test l'étape A-1 et B-1. On va en tout premier préparé nos jeux de données (JDD), qui vont être sous forme de fichier JSON, nous en avons un en input (données reçu par le service), et un en output (données qui sont censé être renvoyé par le service)



```
{
  "idActivite": 88,
  "conjoint": true,
  "dossierIndividu": {...},
  "dossierIndividuLie": {...}
}
```

Figure 27 – Input

Ici on a l'input, avec un "idActivite" de renseigné, un la variable "conjoint" a "True", donc on rentrera bien dans un des cas de mon US.



```
{
  "operationsAnnulables": [...],
  "operationsAnnulablesSimplifiees": [
    {
      "dateEffet": "2022-11-30",
      "code": "ANNUL_RAD_CC",
      "libelle": "Radiation du statut de conjoint collaborateur (SIREN - 117777714)",
      "idIndividu": 2,
      "idEntreprise": 66,
      "idObjetARouvri": 88,
      "idObjetsAAnnuler": null
    },
    {
      "dateEffet": "2022-04-01",
      "code": "ANNUL_AFFI_CC",
      "libelle": "Création du statut de conjoint collaborateur (SIREN - 117777714)",
      "idIndividu": 2,
      "idEntreprise": 66,
      "idObjetARouvri": null,
      "idObjetsAAnnuler": [
        88
      ]
    }
  ]
}
```

Figure 28 – Output

Et pour ce qui est de l'output, on va avoir une liste "operationsAnnulables" et une liste "operationsAnnulablesSimplifiees".

On va ici dans le test appelé notre service avec notre Input, ici avec l'appelle à "determinerOperationsAnnulables(input)", on va ensuite pouvoir confronter le résultat avec notre

attendue. On va pour cela comparer la taille des deux listes avec la fonction "hasSameSizeAs" mais aussi voir si tout les éléments coïncident avec "hasSameElementsAs".

Après avoir fait passé tout les, il nous reste une dernière étapes : les Cucumbers.

Les Cucumbers sont des tests qui vont contrôler le service par appel rest, c'est ce qui se rapproche le plus d'un cas pratique. Pour chaque partie d'un service, des tests Cucumbers y sont associé, avec un répertoire d'input (ce qu'on envoie au service) et un répertoire d'output (ce qu'on veut recevoir du service). Parfois on a aussi un répertoire nommé "dossier" qui vont etre des informations à insérer en base nécessaire pour les test, mais ca n'est pas le cas ici. Tout comme pour nos TU, chaque répertoire va être composé fichier Json

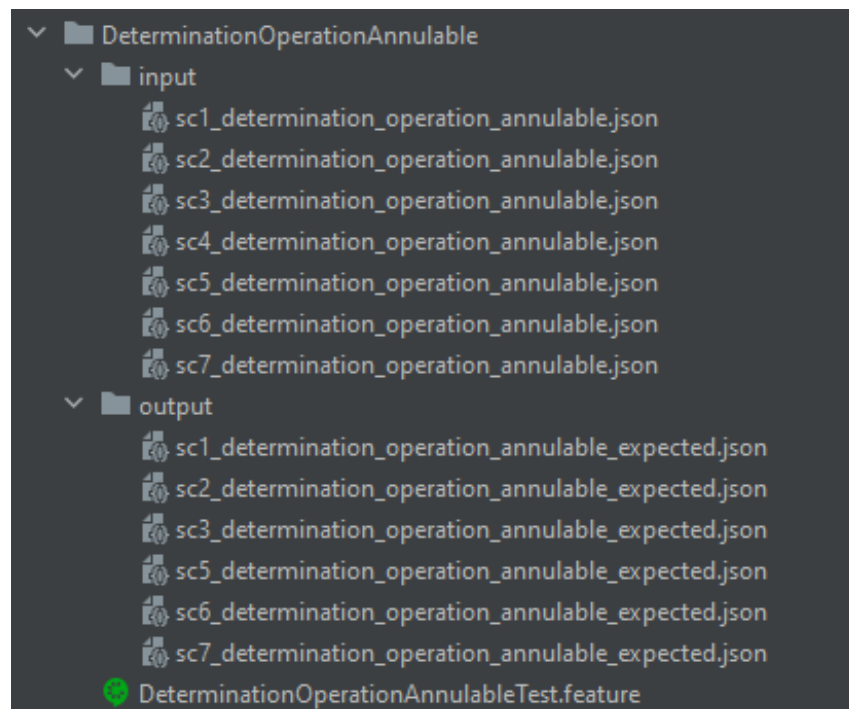


Figure 29 – Enter Caption

Dans notre cas, le jeu de données rédigé pour nos TU étant cohérent fonctionnement, on pu être repris quasiment a l'identique. j'ai juste eu à ajouter ces Test à ceux déjà existent.

5.1.4 Validation d'une US

La dernière étape d'une US, est sa validation, une fois terminé on envoie un message sur le groupe des développeur de notre équipe, avec un lien vers le ou les Merge request qu'on a créé. Une fois cela fait, on peut recevoir des commentaires de la part des autres développeurs, qui vont poser des questions sur certaines partie, ou proposer des changement/optimisation à apporter dans le code.

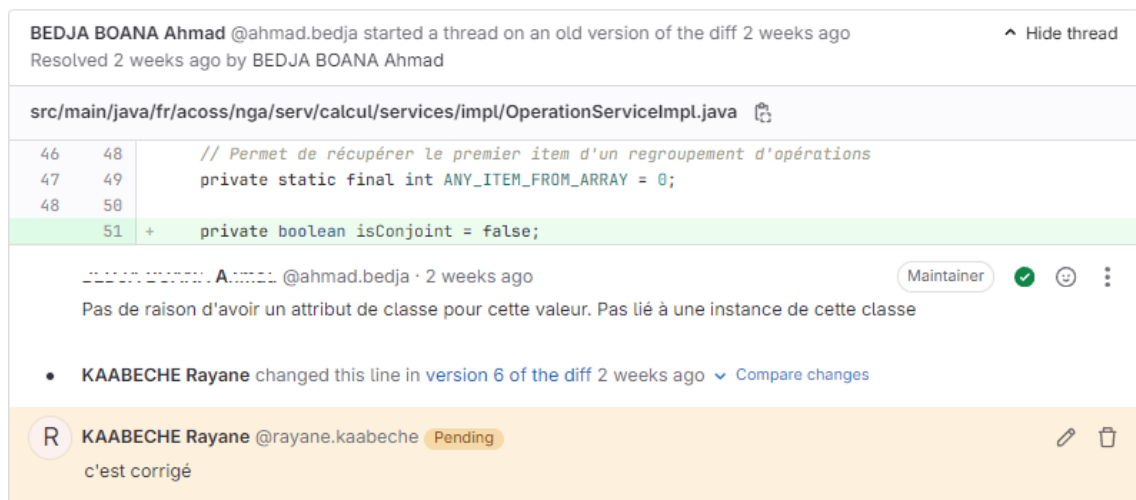


Figure 30 – Exemple de retour

Une fois ces retours pris en comptes, on peut merge notre branch sur celle de l'équipe et la déployer sur nos environnement. Après ça on exécute les pipeline de cucumber, qui vont servir de preuve de bon fonctionnement.

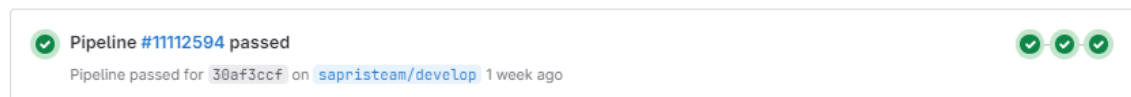


Figure 31 – Enter Caption

Si tout passe sans erreur, alors on peut envoyer ce qu'on appelle les "preuves cucumber" à un de nos testeur, pour que tout soit validé.

5.2 Participation au différente cérémonie

En plus d'avoir codé sur certaines fonctionnalité, j'ai aussi participé activement au cérémonie Scrum sur le projet. Que ce soit en interne comme avec la Daily, mais aussi au près du client avec certaines présentation que j'ai du faire.

5.2.1 La Daily

En plus d'y participer quotidiennement, on a pour coutume le vendredi de demander qui souhaite animer la Daily la semaine suivante. Je me suis proposé de le faire à plusieurs

reprises. C'est quelque chose d'assez simple pour le coup, on y présente au début le burn-down, qui est un graphique représentant l'avancement des US, avant de passer la main à chaque personne de l'équipe

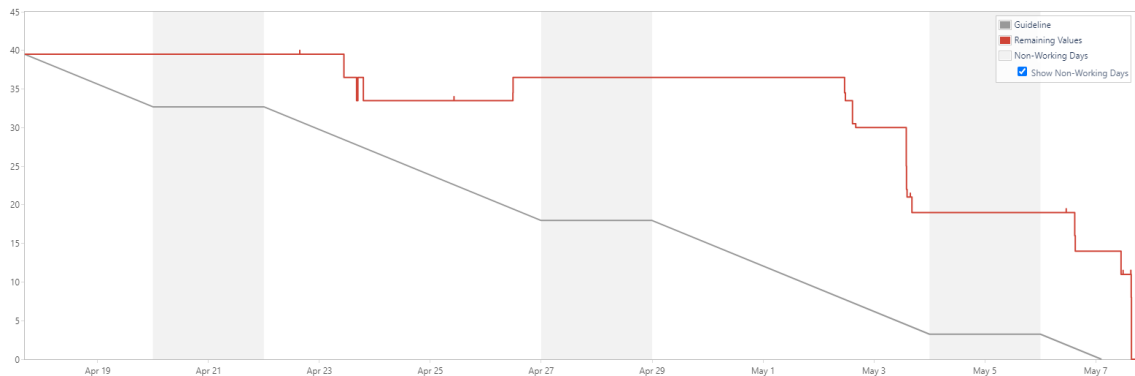


Figure 32 – Exemple de burndown

5.2.2 La Rétrospective

C'est une tradition que chaque personne du groupe présente l'une des Rétrospectives. C'est la dernière réunion d'un sprint, souvent conviviale, où un thème est choisi et chaque partie de la cérémonie est présentée sous forme de jeu. Cela apporte un peu de légèreté et nous permet de débattre de différents points sans tension. J'ai eu l'occasion d'en présenter une, où nous avons discuté des actions proposées précédemment et de leur mise en œuvre, abordé différents sujets liés au sprint, puis proposé de nouvelles actions, chacune étant portée par un individu.

Actions	Porteur	Echeance
V2 : réduire les écarts entre la doc et le code	Thibault	SPIP
Dojo IHM (Outils de base, Angular)	Bastien	SPIP
Continuer de communiquer un ordre de priorité sur les anos	QUALIF	SPIP

Figure 33 – Actions pour les prochains sprint

5.2.3 La Sprint review

J'ai aussi eu l'occasion de rédiger et de présenter devant le client certaines parties de sprint review. J'ai en premier préparé ce qu'on appelle le support de demo, qui va être présenté juste avant la démonstration, et qui va consister en plusieurs slide retraçant les US de ce sprint, chacune focaliser sur une seule fonctionnalité.

E22F02 - Reliquats de gestion de l'établissement d'une PM

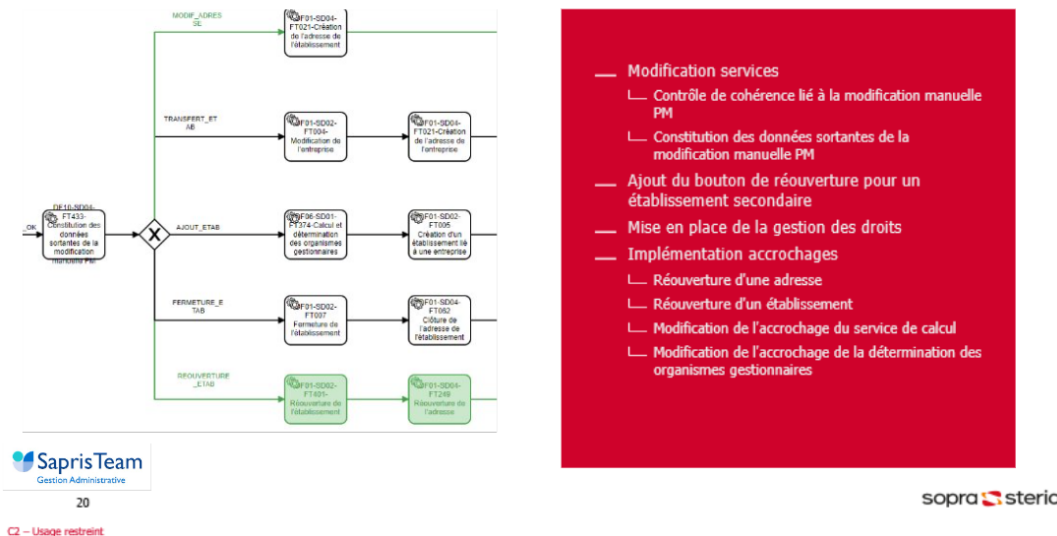


Figure 34 – Exemple de support de demo pour une seul fonctionnalité

J'ai également rédigé et préparé une partie d'une des démonstrations. Pour chaque fonctionnalité terminée, une démonstration peut être effectuée. Je me suis personnellement occupé d'une démonstration sur l'IHM, portant sur la fonctionnalité d'annulation de dernière opération, sur laquelle j'avais travaillé. Cela implique de préparer un jeu de données adéquat, qui sera inséré en base juste avant. Ces données sont sous forme de fichiers XML, appelés liasses. La création d'une liasse est généralement la tâche la plus longue. Ensuite, on rédige le scénario, que l'on intègre ensuite directement dans les slides.

```

1 =====
2 Scénario 1 - Annulation d'affiliation d'un établissement secondaire et d'un CC
3 =====
4 =====
5 Preparation
6 =====
7 - Lancement de la liasse [liasse1_AffTI_PM_couple 3.xml] : affiliation du dirigeant Thomas Lavie sur une PM au 01/02/2024, NIR
  [182100600964559] SIREN[224573451] et d'une CC Sophie Lavie NIR[288120600964538] SIREN[224573451]
8
9 - Lancement de la liasse [liasse2_AffTI_EI_couple 2.xml] : affiliation du dirigeant Thomas Lavie sur un établissement secondaire
  EI au 01/02/2024, NIR[182100600964559] SIREN[655932184] et d'un CC Sophie Lavie NIR[288120600964538] SIREN[655932184]
10
11 =====
12 Execution :
13 =====
14 - Sur ui-dossier ouvrir Informations de l'individu sur le dirigeant et le conjoint
15
16 - Montrer sur Informations de l'individu du dirigeant que le bouton "Annuler dernière opération du statut" n'apparaît que pour
  l'établissement secondaire.
17
18 - Montrer sur Informations de l'individu du CC que le bouton "Annuler dernière opération du statut" apparaît sur les deux
  établissements.
19
20 - Faire l'annulation du Conjoint puis celui de l'établissement secondaire.
21

```

Figure 35 – Exemple de scénario



6 Conclusion

jdsqsdd