Bedminster Célian **Lycée Paul-Louis Courier à TOURS**

BTS Services informatiques aux organisations
Option (Solutions logicielles et applications métiers)

Stage de développement et conception informatique au sein des systèmes d'informations d'harmonie mutuelles

Stage effectué du 08/01/2024 au 23/02/2024

Maître de stage : Mme. LOUIS Faustine
Suivi de stage : Mme. CARAMIGEAS Béatrice

SOMMAIRE

RAPPORT D' ACTIVITÉS HEBDOMADAIRE

- Zoom sur l'entreprise
  - Présentation de l'entreprise
  - Présentation du SIHM
  - Présentation de l'environnement matériel et logiciel
  - Présentation du contexte de situation professionnelle
  - Présentation des missions confiées
  - La gestion de projet au sein de mon équipe
- Mon activités en tant que stagiaire chaque semaine
  - Les tâches réalisées
  - Les problèmes rencontrées
  - Les actions menées et les recherches effectuées pour leur résolution
- Conclusion de mon stage
  - Ce qu'il m'a apporté au niveau compétence technique
  - Ce qu'il m'a apporté au niveau relationnel
  - Conclusion

Remerciement

Je remercie d’abord Mme. LOUIS Faustine pour m'avoir donné la possibilité de rejoindre l'équipe de le SIHM ainsi que de m’avoir partagé ses connaissances et de m’avoir offert la possibilité de travailler sur ce projet. Je remercie M. Cordier Benoit, M. Collin Mathieu, et M. Chausson Dimitri ainsi que M. Lecompte Guillaume pour l’aide et les connaissances qu’ils ont pu m’apporter au cours de ces 7 semaines.

Je remercie toute l'équipe Soprano pour leur accueil et leur soutien tout au long de mon stage. L'équipe étant amicale et soudée, cela a créé un bon environnement de travail où j'ai pu améliorer mes compétences.

Ce stage a été une expérience enrichissante pour moi et elle aura un impact positif sur mon avenir. Je suis reconnaissant pour tout ce que j'ai pu apprendre et les opportunités qui m'ont été donné pendant cette période de stage.

Introduction

Lors de ma formation en BTS Services Informatiques aux Organisations (SIO), j'ai effectué un stage de développement informatique qui m'a permis de mettre en pratique mes connaissances acquises au cours de ma formation et de découvrir l'environnement professionnel. Ce rapport de stage a pour but de présenter les missions qui m'ont été confiées, les compétences que j'ai développées, et les enseignements que j’ai pu tirer de cette expérience.

Au cours de ce stage, j'ai eu la possibilité de travailler sur un projet concret et de participer au développement d’une application web. J'ai pu me familiariser avec différents langages de programmation et framework tels que le TypeScript, HTML, le SCSS, SpringBoot et Angular. Cette expérience m'a également permis de développer mes compétences relationnelles.

Dans cette introduction, je présenterai tout d'abord l’entreprise dans laquelle j'ai effectué mon stage. Ensuite, je détaillerai les objectifs que j'ai fixés pour ce stage, la mission qui m'a été confiée et les problématiques que j’ai connues. Je parlerai des compétences que j'ai pu acquérir et développer au cours de cette expérience, et les apprentissages que j'ai tirés de ce stage en développement informatique.

Ce rapport de stage me donne l'occasion de réfléchir à mon expérience et de voir comment j'ai progressé. C'est aussi une possibilité pour moi de partager mes réflexions et mes idées sur le développement informatique et comment il affecte les entreprises.

I : RAPPORT D’ ACTIVITÉ HEBDOMADAIRE

1. Zoom sur l’entreprise

A - Présentation de l’entreprise

Date de création : 2012
Fondateur : Alliance de plusieurs mutuelle (Prévadiès, Harmonie Mutualité, Mutuelle Existence, SPHERIA, Val de France et Santévie)
Statut de l’entreprise : Entreprise Mutualiste à Mission
Effectif : 4 881 personnes

Une mutuelle, également appelée complémentaire santé, est une forme d'assurance qui complète les remboursements de l'assurance maladie obligatoire. L'objectif principal d'une mutuelle est d'aider les assurés à faire face aux dépenses de santé qui ne sont pas couvertes intégralement par l'assurance maladie obligatoire, contribuant ainsi à réduire le reste à charge pour les bénéficiaires. Harmonie mutuelle n’est pas une mutuelle seule, elle la société fille du groupe VYV qui réunit plusieurs autres mutuelles comme MGEN (schéma ci-dessous)

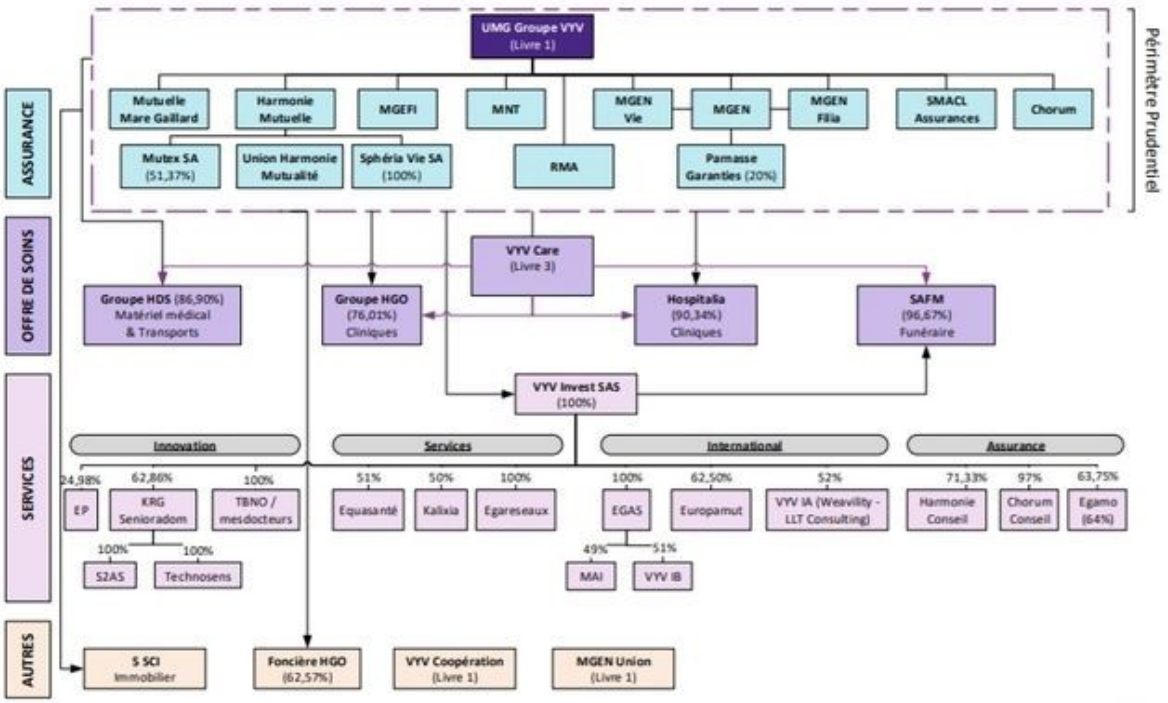


Figure 1 - Organigramme présentant l'organisation du périmètre de combinaison du Groupe VYV au 31 décembre 2019

B - Présentation du SIHM et présentation de l’équipe SCRUM (systèmes d’informations d’harmonies mutuelles)

La direction des systèmes d'information est la direction responsable du système d'information d'une entreprise. Elle est en charge de définir l’architecture du Sl, concevoir, installer, déployer et exploiter le SI.

Pour mon stage j’ai été intégré à une équipe SCRUM. Il y a différents rôles dans ce type d’équipe

On a tout d’abord **le P.O.(Product Owner)** : c’est un chef de projet en mode agile. Il est en charge de satisfaire les besoins des clients en menant à bien la livraison d'un produit de qualité. Il sert d'interface entre l'équipe technique, l'équipe fonctionnelle et les clients. Son rôle est aussi de développer un produit qui réponde au plus près des attentes clients, en accord avec les contraintes de budget et de délais imposées.

Le TecLead : Un TechLead, c’est la personne qui prend le rôle de responsable de la qualité logicielle sur un projet, il s'assure du fonctionnement des technologies des projets.

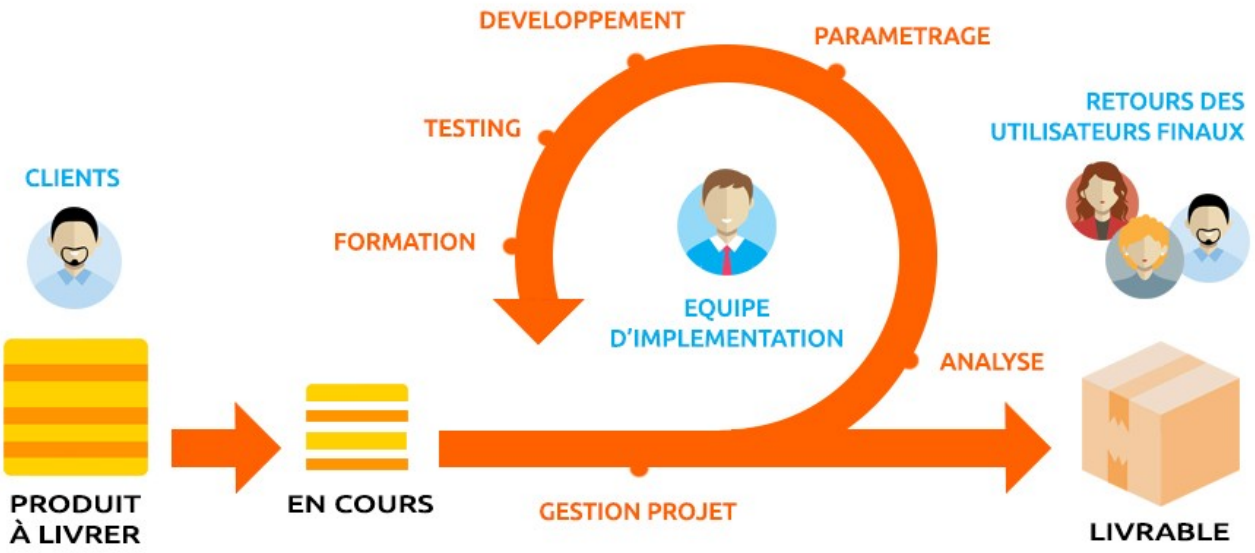
Les Business Analyst : Le Business Analyst Agile se place en tant qu’expert digital, il interagit avec l’ensemble des directions en recueillant les attentes et les contraintes des utilisateurs, en analysant les processus, les stratégies et en gérant l’organisation du travail. Il est en mesure d’évaluer l’efficacité et d’apporter les optimisations nécessaires durant toute la durée de vie du projet.

Les Développeurs : Ils développent l’application sous l’autorité du TechLead et des B.A.

La méthode SCRUM agile divise les différents livrables en « sprint ». Chez le SIHM, chaque sprint dure 3 semaines pour tous les projets, car certains projets dépendent d’autres projets. Il est donc important de ne pas perdre de temps et que tous les projets soient synchronisés.

Tous les jours, nous avons une réunion, un « DM », hebdomadaire qui permet à chacun d’énoncer ses points de blocage, ce qu’il a fait le jour précédent et ce qu’il fera le jour présent.

<u>Présentation du fonctionnemenr de la méthode Agile Scrum:</u>



C - Présentation de l’environnement matériel et logiciel

Au SIHM, on utilise le COPE (Corporate Owned, Personnaly Enabled). Les ordinateurs sont stockés au rez-de-chaussée. Chaque personne doit se tenir à installer son environnement logiciel. En effet, uniquement le VPN (FortiClient pour l’accès en télétravail) ainsi que les logiciels de gestion de postes y sont installés.

Le VPN permet d’accéder à Internet chez soi et d’accéder directement aux réseaux du SIHM.
Tous les postes sont sous Windows 10, Windows 11 n’étant toujours pas sécurisé.

Le SIHM dispose d’une application interne ‘HM Store’ qui permet d’installer les logiciels dont l’on pourrait avoir besoin.

Chaque personne dispose de deux écrans plus ou moins qualitatifs avec une souris, un clavier et un ordinateur portable qui peut varier selon les besoins et la puissance nécessaire. Un développeur n’aura pas le même ordinateur portable qu’un commercial par exemple.

On a aussi à disposition une adresse mail Outlook. Chaque personne possède une adresse mail du type « prenom.nom@harmonie-mutuelle.fr ». Contrairement à mon stage précédent harmonie mutuelle ne fait pas de différence entre les salariés et les personnes venant de l’extérieur.

Personne n’a un propre bureau à son nom, en fait il y a beaucoup de rotation de bureau depuis le Covid. Ce sont des bureaux avec 4 à 5 places pour les plus grands. En général, les rôles de l’entreprise sont réunis ensemble ,c’est à dire les développeurs avec les développeurs, les Business Analystes avec les Business Analystes etc.

Ensuite concernant la gestion de projets, ils utilisent différents logiciels comme Confluence (pour les docs des API) , Jira pour la gestion des Tickets et Git et GitHub pour les dépots de leurs projets.

On a aussi à disposition le logiciel Teams qui permet de communiquer avec ses collègues de n’importe où, créer des réunions, des canaux de discussions, des appels vidéos.

Je suis actuellement dans plusieurs groupes Teams :

- « DM TabGar » : cette réunion qui ne me concerne pas est en rapport avec le projet TabGar (Tableau de garantie). En effet dû à une trop vieille application, celle-ci est refaite. Une réunion a donc été créée pour savoir où chaque personne en est dans les différents Sprints.

- « DM Soprano » :Cette réunion qui a lieu tous les jours entre 9h45 et 10h fait référence au projet sur lequel je travaille actuellement. Ici, c’est encore une fois un groupe qui permet de faire des réunions

- « Team Soprano » : Ici toutes les informations du projet sont données. Cela peut-être une livraison réalisée pour les clients. Les différentes KO(problème(s) majeur(s) détecté(s) sur la fonctionnalité recettée) sur les tickets, les régressions, les recettes finalisées ou en cours, etc., ’est aussi ici que l’on demande lorsqu’on est bloqué sur la conception ou le développement de sa fonctionnalité . La règle c’est qu’au bout de 2h, si on est bloqué sur un problème, on arrête de chercher tout seul et on demande aux autres, pour que cela fasse avancer le projet et ne le freine pas.

D - Présentation du contexte de ma situation professionnelle

Le projet est une application permettant de vendre des produits et des tarifs au sein de contrats collectifs à destination des entreprises. Pour se faire les commerciaux se basent sur un référentiel qui leur permet de créer sur les contrats des produits et tarifs « ébauches». Ces ébauches peuvent être manipulées par les utilisateurs sans impacter leur référentiel.

Pour ajouter leurs produits tarifs sur les contrats, il a été mis en place un algorithme de transformation des données référentielles en ébauche.

Une particularité de l'application soprano, c'est qu'elle n'est pas directement accessible via un lien comme un site-web. En fait, une application mère réalise tout d'abord ses traitements puis lors d'une étape intermédiaire, elle fait appel à l'application Soprano qui va prendre le relais sur certaines étapes puis redonne la main à l'application mère par la suite.

E- Présentation des missions confiées

Ma mission principale est donc de créer un nouveau service REST, avec le Framework Java SpringBoot pour le back-end et Angular pour le front-end, pour palier à ce besoin. Ce service permettra donc de consulter les produits et tarifs utilisés sur d'autres contrats et/ou d'autres propositions ce qui sera très utile aux commerciaux. Une fois cela réalisé, je vais devoir réaliser le « front-end » de ce service. Lorsque le bouton sera appuyé, une fenêtre (sidenav) apparaîtra et mon service sera appelé puis les différentes informations y seront affichées.

J'ai à ma disposition pour réaliser cela différents logiciels comme :

- IntelliJ avec sa licence, c'est un IDE crée par JetBrains très ergonomique et pratique mais payant.

- PostMan qui est un client lourd pour créer et interroger des services web (comme Rested mais en client lourd). L'avantage avec PostMan c'est qu'il est très ergonomique et facile d'utilisation.

- Dbeaver :
  DBeaver est un logiciel permettant l'administration et le requêtage de base de données. Pour les bases de données relationnelles, il utilise un driver JDBC. Pour les autres bases de données, il utilise des pilotes de base de données propriétaire. Je l'utilise pour interroger une base de données Oracle.

- Maven pour le côté Back-end :
  Maven (crée par Apache) est un outil de gestion et d'automatisation de production des projets logiciels Java en général et Java EE en particulier. Il est utilisé pour automatiser l'intégration continue lors d'un développement de logiciel.

- Node.Js est aussi utilisé pour le côte front-end. Node.js est une plateforme de développement Javascript. Ce n'est pas un serveur, ce n'est pas un framework, c'est juste le langage Javascript avec des bibliothèques permettant de réaliser des actions comme écrire sur la sortie standard, ouvrir/fermer des connections réseau ou encore créer un fichier.

- NVM (Node Version Manager) qui comme son nom l'indique permet de gérer les différentes versions de NodeJs, c'est un gestionnaire de version.

Le service REST que je vais réaliser, sera divisé en plusieurs tickets et plusieurs sous tâche . Je devrai réaliser mes tests et fournir un jeu de tests défini au préalable par les Business Analyst. Pour chaque tâches je recetterai chaque ticket.

F -Mon activités en tant que stagiaire chaque semaine

Semaine 1 :

Cette semaine a été une semaine de découverte. En effet, j'ai découvert mon environnement de travail, et ce, sur quoi je travaillerai tout au long de ces 7 semaines.
J'ai donc dû réaliser l'installation de mon poste comme tout développeur d'Harmonie Mutuelle. Un tutoriel est mis à disposition et à suivre à la lettre pour pouvoir installer son poste. Une arborescence de dossier est donnée et elle est à appliquer :

c:/dev

├── tools => répertoire des outils de développement

│ ├── intellij => répertoire avec les versions d'intellij

│ ├── java => répertoire avec les différentes version de JVM

│ ├── maven => répertoire des différentes versions de maven

│ │ └── repository => repository maven local : répertoire avec les jars (et pom) maven téléchargés

│ ├── nvm => répertoire d'installation de l'outil nvm

│ └── node => répertoire d'installation des versions de node (sans utiliser nvm)

└── wks => répertoire des sources de vos projets

Tâches réalisées :
Installation de git et d'intelliJ :

J'ai donc dû commencer par installer Git ce qui m'a permis après d'importer les projets qui sont stockés sur GitHub.

Ensuite j'ai installé IntelliJ.
IntelliJ est un environnement de développement intégré destiné au développement de logiciels informatiques reposant sur la technologie Java. Dans le tutoriel, il est conseillé d'installer la version « .zip » à cause des différents droits d'administrateur demandés lorsque l'on veut installer le « .exe »
Dans le tutoriel, il nous montre une arborescence à suivre pour réunir tous les outils nécessaires.

Il a fallu que j'installe des Plugins :

- SonarLint : SonarLint est un plugin permettant de réaliser les analyse de code SonarQube au fil des développements, directement dans l'IDE. Il a fallu ensuite que je me connecte à une instance SonarQube via un lien fourni.
- Jfrog Xray : JFrog Xray fonctionne avec JFrog Artifactory pour analyser le code et les packages et rechercher les failles de sécurité.

Problème rencontré et résolution :

Après son installation il fallait donc le lancer via l'invite de commande, car il s'agit d'un « .zip » et non un «.exe ». Ici, IntelliJ nous demande de nous connecter à notre compte pour récupérer une licence. Après plusieurs essais de connexion à mon compte harmonie mutuelle sur IntelliJ , rien ne se passait, un chargement infini. J'ai vite compris qu'il y avait un problème avec le serveur proxy.

Il fallait configurer le proxy d'harmonie Mutuelle dans IntelliJ pour que celui-ci ne bloque pas l'accès aux comptes d'Harmonies-Mutuel. Heureusement, le tuto avait tout prévu, car toutes les informations pour configurer le proxy étaient données. Il ne manquait plus qu'à entrer les différentes informations puis la connexion à mon compte ET la récupération de ma licence ONT pu se faire sans soucis.

Tâches réalisées :

Installation de Maven

Ensuite, j'ai installé Maven. Cela permet de gérer son projet et surtout de le compiler. En effet, le Back-end étant en Spring-boot (FrameWork Java) on a besoin de le compiler. Encore une fois, le Maven était en « .zip » j'ai pu facilement l'installer et le mettre dans son dossier respectif. Une fois l'accès à mon compte administrateur j'ai pu accéder aux variables d'environnement.

En 2 étapes j'ai pu réaliser l'ajout de Maven à celle-ci :

- 1) D'abord créer la variable d'environnement M2_HOME avec le chemin c:/dev/tools/maven/<version> .

  exemple : M2_HOME C:\dev\tools\maven\apache-maven-3.9.2
- 2) Ajouter dans le path l'entrée %M2_HOME%\bin

Problème rencontré et résolution :

Comme c'est un « .zip » il fallait l'ajouter dans les variables d'environnement, le seul problème, c'est que pour accéder aux variables d'environnements, il faut un compte administrateur, or, il m'est impossible avec mon compte utilisateur d'accéder à ceci.

J'ai dû faire une demande de création d'un compte administrateur restreint à mon nom. Quand je dis restreint, je veux dire qu'il ne sert qu'à télécharger ou accéder à des logiciels dont l'accès administrateur est nécessaire.

Si l'on va sur ce compte administrateur, on ne peut rien faire en soi. Par exemple, le proxy nous bloque l'accès à Internet, on ne peut pas modifier les paramètres Windows, etc. La demande est à faire auprès du « Service Ami » qui gère les comptes Windows du SIHM. Le compte Administrateur à la forme de : « al.nom-premièreLettreDuPrénom »

Tâches réalisées :

Installation de Java, Node et npm

J'ai ensuite installé Java car le Back-end est écrit en SpringBoot. Ils utilisent encore le JDK8 (ce qui date beaucoup).

Il m'a fallu installer Node.js car on utilise Angular pour la partie Front-End (Angular étant un framework basé sur TypeScript).

Npm est un gestionnaire de version pour Node assez pratique car on peut switcher entre les différentes versions de celui-ci.

Problème rencontré et résolution :
Il fallait que j'ajoute Maven et Java et Node aux variables d'environnement pour pouvoir les utiliser, or , il faut un compte administrateur, j'ai donc attendu que celui-ci soit créé pour les ajouter

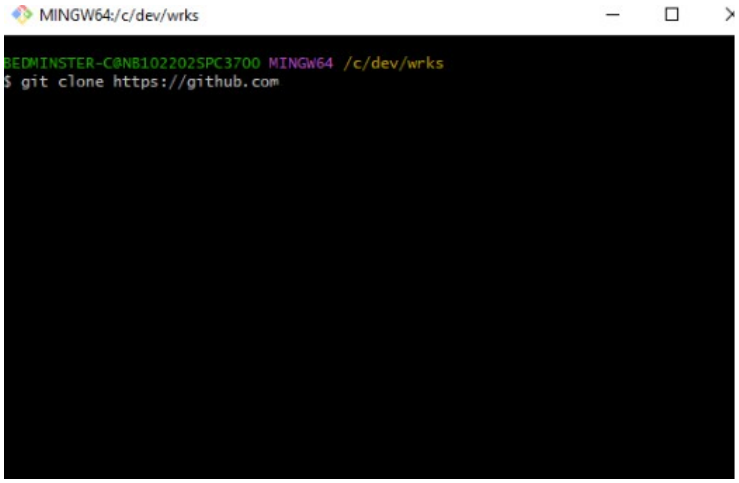
Tâches réalisées :

Importation des projets back-end et front-end

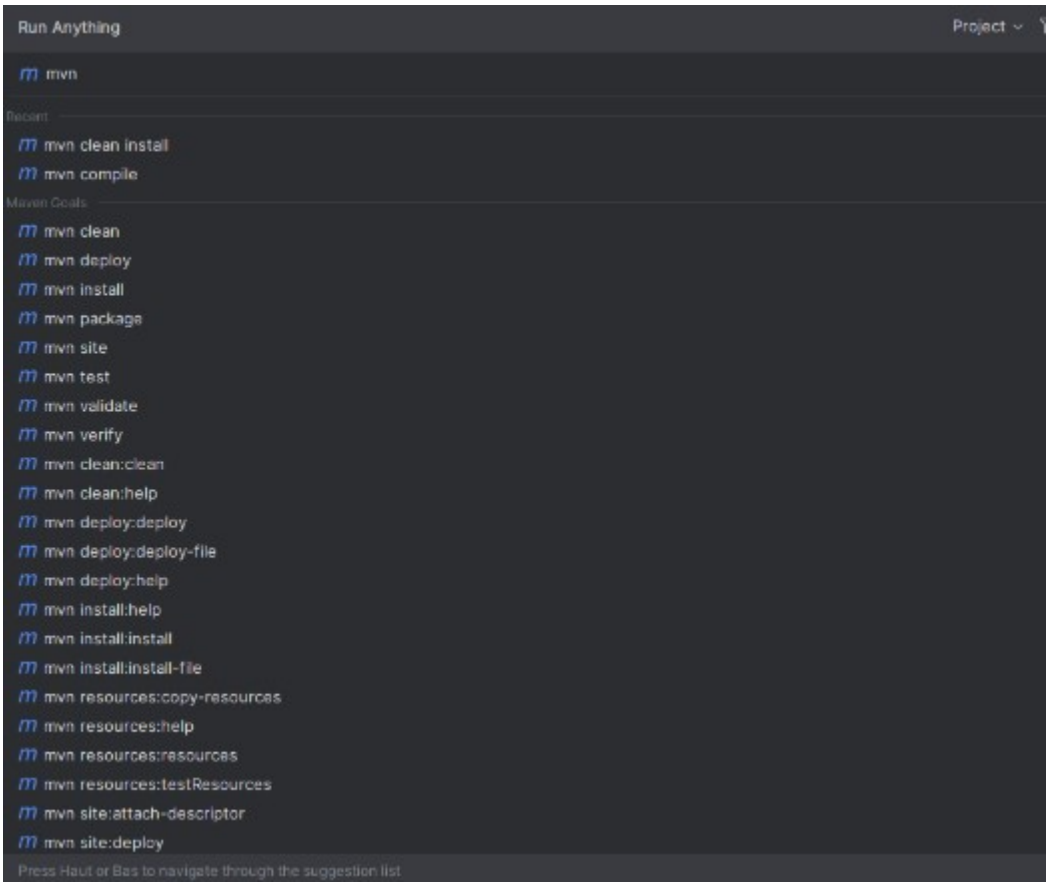
Pour faire cela je me suis placé dans le répertoire c:/dev/wrks/.

Ensuite j'ai fait un clic droit puis « Open Git Bash Here »

puis dans la console j'ai réalisé un « git clone <repository link> :



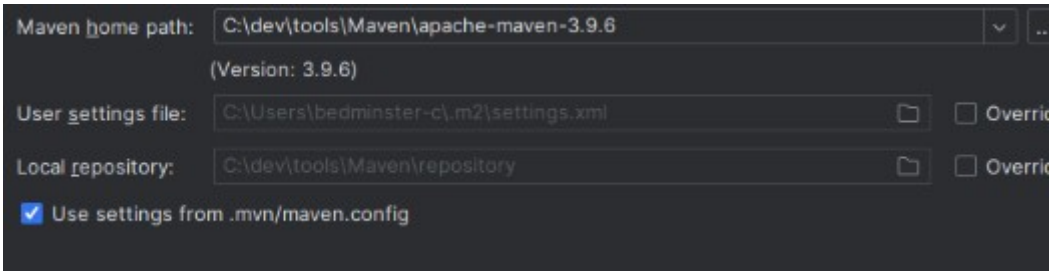
J'ai ouvert les projets avec IntelliJ. Il fallait d'abord compiler le projet back-end avec maven.
Dans Intelli, il y a la possibilité d'ouvrir une console Maven :



À partir d'ici, on peut exécuter différentes commandes et compiler le projet. J'ai donc compilé le projet avec la commande 'mvn clean install' qui compile le projet en effaçant et réinstallant tout au propre.
Ensuite, il a fallu compiler le projet Front (en Angular) ;

Problème rencontré et résolution :

Après compilation du projet, celui-ci me retournait des erreurs de dépendances. Il fallait bien spécifier à IntelliJ quelle version et la localité de Maven (celui installé au préalable) :



Lors de la compilation du projet front ? celui-ci n'était pas reconnu comme une application Angular, car j'avais installé la version Community d'intelliJ , or angular n'est disponible

que sur la version Ultimate. J’ai donc réinstallé la bonne version et exécuté la commande ‘npm install’ (dans la console powershell intégrée à IntelliJ) qui permet d'installer toutes les librairies, frameworks et outils dont a besoin un projet JavaScript. C’est l’équivalent de Maven pour Java.

Semaine 2 :

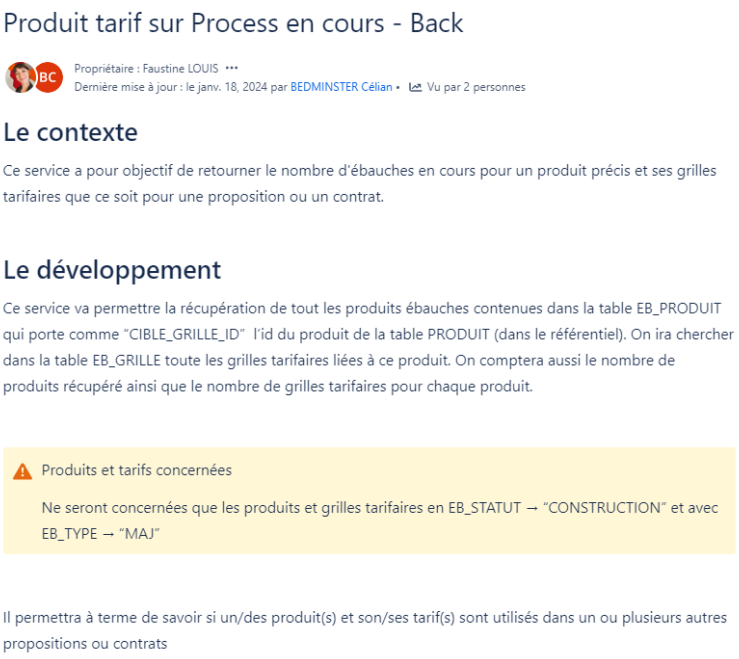
Cette semaine a été une partie conception. En effet j’ai réalisé la conception de mon service ainsi que les maquettes front-end et la création des différents tickets pour intégrer l’équipe SCRUM.

Tâches réalisées :

Écriture des spécifications :

Les spécifications permettent aux développeurs de savoir ce qu’il doivent concrètement faire. Elle permettent d’expliquer aux développeurs le besoin client, les différents paramètres d’entrées, les retours attendus. Parfois pour mieux comprendre ce qu’ils ont à réaliser, un schéma de conception leur est fourni avec les différents cas d’appel du service (schéma ci-dessous) pour mieux comprendre comment coder.

J’ai donc d’abord réalisé les spécifications de la première partie de mon service REST via confluence (crée par Atlassian). J’ai donc d’abord rappelé le contexte de mon service et son utilité. Ensuite j’ai pu décrire les paramètres d’entrée puis le retour d’appel de ce service et enfin j’ai réalisé le retour d’appel de mon service (c’est à dire ce qu’il renvoie).



Je note l’URL d’appel avec un exemple de retour

1. **On remonte les grilles-ébauches liées au produit**

URL d'appel :

```
/v3/ebauche-produits/{idEbaucheProduit}
```

Retour d'appel :

exemple avec le produit-ébauche 1222

v3 : version de l'API
ebauche-produit : pour correspondre au contexte il est important de mettre un nom cohérent
idEbaucheProduit : pour retourner les différentes informations du produits

Exemple de retour :


```
{
    "id": 1222,
    "compteDeCreation": "larose-c",
    "dateDeCreation": "2023-12-04T16:19:15Z",
    "compteDeMiseAJour": "larose-c",
    "dateDeMiseAJour": "2023-12-04T16:19:16Z",
    "codeProduitSource": "S04052",
    "codeProduitCible": "S04052",
    "type": {
        "id": 2,
        "actif": true,
        "code": "MAJ",
        "libelle": "Mise à jour"
    },
```


Lors de la création de mon service je vais devoir faire appel à une autre API (API expérience) qui est réalisée par une autre équipe de développeurs. Ils réalisent une autre application nommée 'Contralto'. Elle a donc des informations supplémentaires dont la base de données de l'application Soprano n'a pas besoin.
Je récupère l'URL à appeler pour récupérer les informations :

2. **On crée une méthode qui va aller chercher les produits en cours via l'id cible et l'id grille cible :**

3. **On appel l'API expérience :**

Paramètres API expérience

Appel de l'API contralto pour récupérer notamment le numéro de proposition grâce à la partie ContraltoService implémenté dans l'API Soprano grâce la méthode getProduitTarif() disponible via l'end-point ci-dessous.

dProduitEbauche

codeProduitEbauche

dTarifEbauche

codeTarifEbauche

ibelleProduitEbauche

codeGrille

```
contralto/v2/ebauches-grilles/experience/produits-tarifs
```

Ensuite je montre un exemple de retour ainsi que les différentes variables retournées

Retour d'appel

IdPm

dateEffet

numJuridiqueContrat

idAvenant

idProposition

libelleCategoriePersonnel

libelleMontage

libelleProduit

idBase

idBaseEbauche

libellePopulationProduitList

libellePopulationTarifList

raisonSociale

optionList

regimeAgricole

Exemple de retour :

Exemple pour la propo 331974 :


```
{
  "idPm":280879,
  "dateEffet":"2024-02-01",
  "numJuridiqueContrat":null,
  "idAvenant":null,
  "idProposition":331974,
  "libelleCategoriePersonnel":"Ensemble du Personnel",
  "libelleMontage":"Unique",
  "libelleProduit":"Produit Unique",
  "idBase":null,
  "idBaseEbauche":null,
  "libellePopulationProduitList":["Actif"],
  "libellePopulationTarifList":[],
  "raisonSociale":"AFH",
  "optionList":null,
  "regimeAgricole":false
}
```

Problème rencontré et résolution :

Je n'avais jamais réalisé ce genre de spécifications il m'a donc été difficile de comprendre ce qu'on attendait de moi. J'ai pris des exemples sur d'autres spécifications réalisées auparavant et posé des questions. J'ai finalement réussi à réaliser ces spécifications.

Tâches réalisées :

Division des tâches de développement et écriture des tickets :

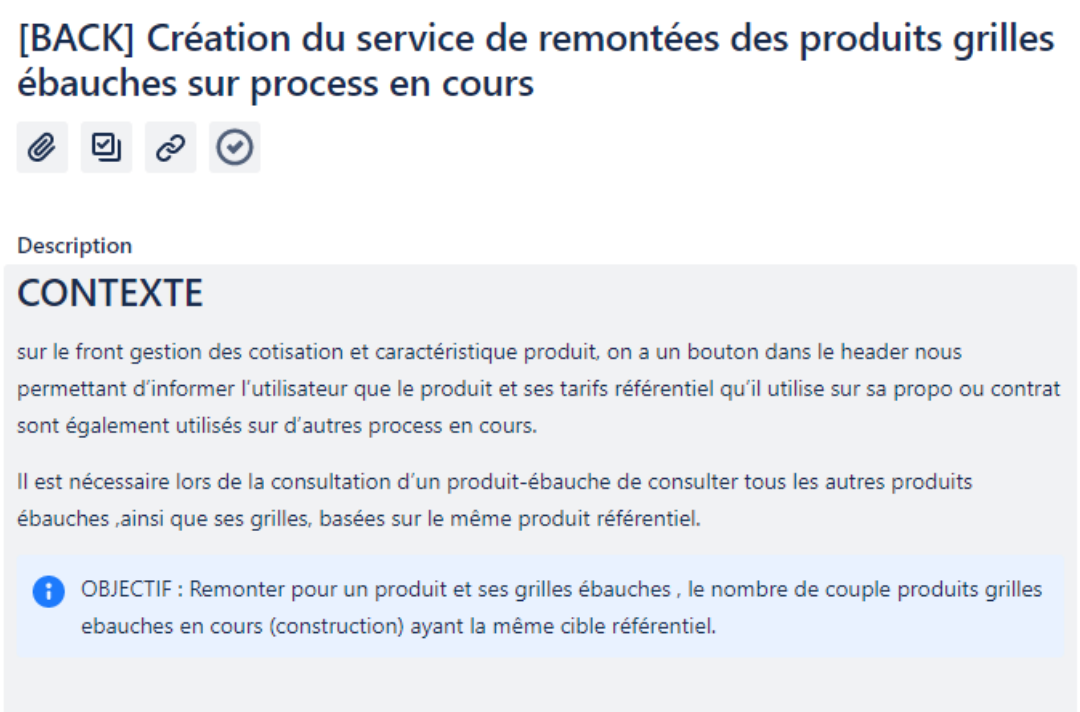
Dans une équipe SCRUM agile, les différentes fonctionnalités sont divisées en plusieurs tickets. Qui eux mêmes peuvent être divisés en sous-tâches. Ici, les tickets ont un nom puis un numéro selon le projet et le nombre de fonctionnalités. Par exemple, un de mes tickets a le nom « REFO-835 ». REFO pour référentiel-offre et 835 car il s'agit du 835ème ticket

J'ai donc décidé de diviser mon service en 3 tickets. En effet, celui- ci ne contient pas que du back-end, j'ai aussi une partie front à réaliser.

<u>Les différents tickets :</u>

Le premier ticket 'REFO-835'

Pour chaque ticket j'ai donné le contexte du ticket :



Puis la stratégie de développement :

Stratégie de développement

Créer un service avec pour donnée en entrée

idebaucheproduit

et sortie attendue à minima

Liste de produits ebauches et les grilles associées et une liste de grilles ebauches associées au produit ebauche

{

id ebauche produit

codeEbaucheProduit

libelleproduit ebauche ,

[

idgrille ebauche

codebgrille ebauche

]

},

{

id ebauche produit

codeEbaucheProduit

libelleproduit ebauche ,

[

idgrille ebauche

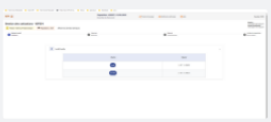
codebgrille ebauche

]

}

enfin les tests d'acceptance que chaque développeur doit remplir.
L'URL permet d'accéder directement au cas courant

Tests d'acceptances

créer une propo avec le produit tarif	requête qui remonte les résultats	service
un produit et tarif MAJ V00732-XIF001 https://soprano-produit.sdw-rec3.hm.dm.ad/gestion-cotisations/produits/3170/grilles/2147?context=eyJjb250ZXh0IjoxLCJ1cmxDYWxsYmFjayI6Imh0dHBzOi8vc3ItcGhvbmllLnNkdy1yZWMzLmhtLmRtLmFkL2NvbnRyYWx0by1jb2xsL3Byb3Bvc2l0aW9uLzQyMDUzNyIsImV0YXBlIjoxLCJpZFByb3Bvc2l0aW9uIjoiNDIwNTM3In0%3D 		
Nouveau produit nouveau tarif		aucun process pour un tarif en creation
produit MAJ et tarif création		on remonte que les infos sur le produit
produit maj un nouveau tarif (création) et un tarif maj		infos remonté que pour les MAJ
produit maj et 2 tarifs maj		liste avec les 2 tarifs
produit maj et 1 tarif en termine abandonné		que pour le produit

Le deuxième ticket 'REFO-836' :

Le contexte :

[FRONT][APIEXPERIENCE] Récupérer les informations contrat propo pour chaque couple produits tarifs sur process en cours

Joindre | Créer une sous-tâche | Associer un ticket | Ajouter Tempo pour la planification et le suivi du temps

Description

CONTEXTE

sur le front gestion des cotisation et caractéristique produit, on a un bouton dans le header nous permettant d'informer l'utilisateur que le produit et ses tarifs référentiel qu'il utilise sur sa propo ou contrat sont également utilisés sur d'autres process en cours.

OBJECTIF : récupérer les informations depuis contralto a afficher dans le front pour les couples produits et tarifs ebauches en cours de process lié à la propo ou contrat sur laquelle on est .

Puis la stratégie de développement :

Stratégie de développement

Appel à l'API expérience

utilisation d'un service existant

et sortie attendue

Informations pour le produit passé en paramètre

```
{
  idPm,
  dateEffet,
  numJuridiqueContrat,
  idAvenant,
  idProposition,
  libelleCategoriePersonnel,
 libelleMontage,
 libelleProduit,
 idBase,
 idBaseEbauche,
 libellePopulationProduitList,
 libellePopulationTarifList,
 raisonSociale,
 optionList,
 regimeAgricole
}
```

Enfin le dernier ticket ‘REFO-837’

Le contexte et la description:

Ici il s’agit d’un ticket ‘Front-end’ la description sera en conséquence différente.

[FRONT] Affichage des produits et tarifs ebauches sur process en cours pour la propo contrat

Joindre | Créer une sous-tâche | Associer un ticket | Ajouter Tempo pour la planification et le suivi du temps

Description

CONTEXTE

sur le front gestion des cotisation et caractéristique produit, on a un bouton dans le header nous permettant d'informer l'utilisateur que le produit et ses tarifs référentiel qu'il utilise sur sa propo ou contrat sont également utilisés sur d'autres process en cours.

OBJECTIF : au clic sur le bouton Produit tarif sur process en cours

Description

Bouton

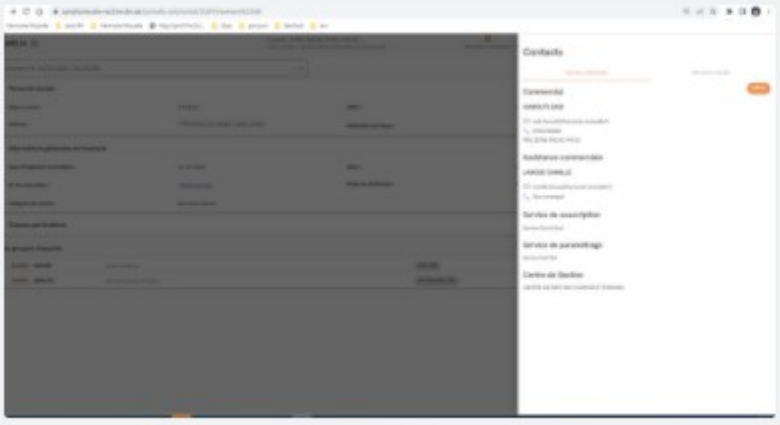
renommer le bouton : Produits tarifs sur Process en cours

accessible à tous les profils

Au clic sur le bouton

une toggle navbar s'ouvre sur la droite .

Exemple sur Contralto



Puis on gère l’affichage des données

Affichage des données

1- produit tarifs en MAJ des données sont existantes

Suite à la remontées des informations nécessaire à l'affichage REFO-835: [BACK] Création du service de remontées des produits grilles ébauches sur process en c ours BACKLOG et REFO-836: [FRONT][APIEXPERIENCE] Récupérer les informations contrat propo pour chaque couple produits tarifs sur process en cours BACKLOG

il faut afficher les données de la façon suivante :

titre : Produits tarifs en cours

afficher ceux de la propo sur laquelle on est

sous titre : les process en cours impactant ces produits tarifs

- Libellé PM - Propo [numéro]/Contrat [numéro]
  - Produit [code produit référentiel] | Ebauche ([code produit ébauche]).
    Tarifs impactés :
    - Tarif [code tarif référentiel=ébauche]
    - Tarif [code tarif référentiel=ébauche]
    - ...
- Libellé PM - Propo [numéro]/Contrat [numéro]
  - Produit [code produit référentiel] | Ebauche ([code produit ébauche]).
    Tarifs impactés :
    - Tarif [code tarif référentiel=ébauche]
    - Tarif [code tarif référentiel=ébauche]
    - ...
    -

2 - Produit en maj et grille en création

est ce que l'on remontes les propo contrat qui porte le produit cible ? @Clément THOMAS je ne vois pas l'intérêt

ou affichage message

service delta Tabgar à appeler

3- pas d’autre process en cours pour un couple produit tarif

" pas de couple produit tarif ebauche partagé en cours de process"

Problème rencontré et résolution :

Comme pour la rédaction des spécifications, il m'a été difficile de comprendre concrètement ce que je devais réaliser. Finalement avec de l'aide j'ai pu terminer tout cela.

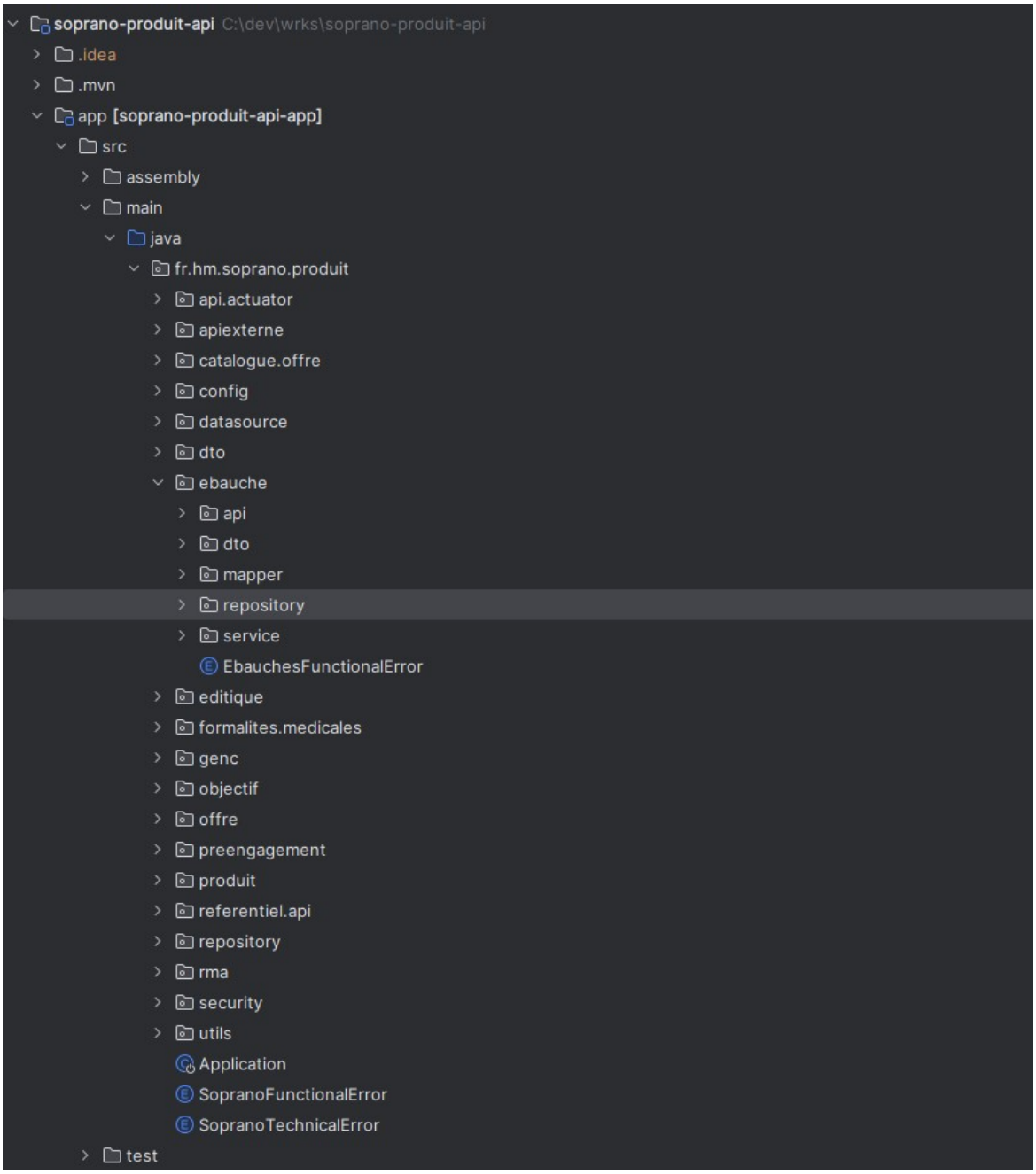
Semaine 3 :

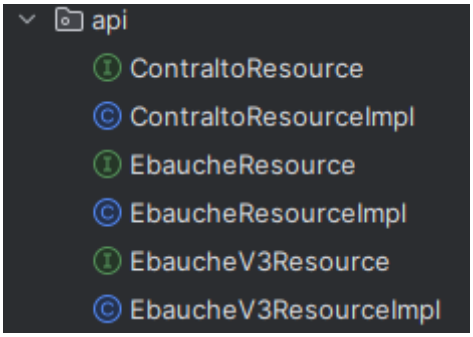
Cette semaine j'ai pu commencer le développement de mon premier ticket le 'REFO-835' qui est un ticket de type 'back-end'.

Le projet a l'arborescence ci-dessous :

Mon service est développé dans le dossier 'ebauche' car je travaille sur les ébauches de produit
Dans le dossier ébauche il y a différents sous-dossiers.

On a d'abord le sous-dossier 'api' :





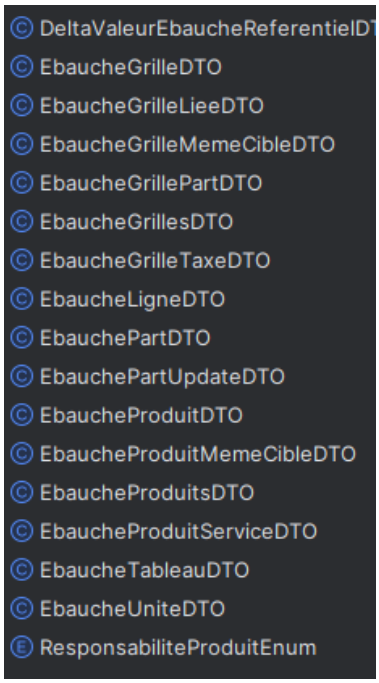
C’est ici que les différentes versions de l’API sont stockées ainsi que les différents codes http de retour :
exemples d'end-point situé dans le fichier ‘EbaucheV3RessourceImpl :

```
egret-j +1
@Override
@Transactional(rollbackFor = Exception.class)
@PostMapping(path = "/{idEbaucheProduit}/grilles", produces = {ALL_VALUE, APPLICATION_JSON_VALUE})
public ResponseEntity<EbaucheGrilleDTO> createEbaucheGrille(@PathVariable(name = "idEbaucheProduit") Long idEbaucheProduit,
                                                            HttpServletRequest request, @RequestHeader HttpHeaders headers,
                                                            @RequestHeader(value = USERNAME_HEADER, required = false) String usernameHeader,
                                                            @RequestParam @DateTimeFormat(pattern = JSON_DATE_PATTERN) LocalDate dateEffetProposition,
                                                            @RequestParam(value = "isBase", defaultValue = "true", required = false) boolean isBase,
                                                            @RequestParam(value = "idGrilleBase", required = false) Long idGrilleBase,
                                                            @RequestParam(value = "isRegimeAgricole", required = false) boolean isRegimeAgricole,
                                                            UriComponentsBuilder uriComponentsBuilder) throws SymphonieFunctionalException {
    EbaucheGrilleDTO ebaucheGrilleCreated = ebaucheGrilleService.saveNouveauTarif(idEbaucheProduit, dateEffetProposition, isBase, idGrilleBase, isRegimeAgricole);
    return responseEntityCreate(headers, uriComponentsBuilder, ressourcePath: V3_EBAUCHE_PRODUITS + idEbaucheProduit + SUBPATH_GRILLES, ebaucheGrilleCreated, ebaucheGrilleCreated.getId().toString());
}

egret-j +1
@Override
@Transactional(rollbackFor = Exception.class)
@PutMapping("/{idEbaucheProduit}/grilles/{idEbaucheGrille}")
public EbaucheGrilleDTO updateEbaucheGrille(@PathVariable(name = "idEbaucheProduit") Long idEbaucheProduit,
                                            @PathVariable(name = "idEbaucheGrille") Long idEbaucheGrille,
                                            @RequestHeader(value = USERNAME_HEADER, required = false) String usernameHeader,
                                            @RequestBody @Valid EbaucheGrilleDTO ebaucheGrille) throws SymphonieFunctionalException {
    if (!idEbaucheProduit.equals(ebaucheGrille.getIdEbaucheProduit()) || !idEbaucheGrille.equals(ebaucheGrille.getId())) {
        throw ERROR_SOP_EBAUCHE_GRILLE_UPDATE_ID_AND_ID_PRODUIT_NOT_MATCHES.toException();
    }
    return ebaucheGrilleService.update(ebaucheGrille, idEbaucheProduit);
}
```

Puis on a leurs codes de retour dans le dossier ‘ EbaucheV3Ressource’ pour swagger :

```
no usages  1 implementation  egret-j +1
@ApiResponses(value = {
    @ApiResponse(responseCode = "201", description = "L'ébauche de grille a été créée"),
    @ApiResponse(responseCode = "500", description = "Problème interne", content = @Content(schema = @Schema(implementation = SymphonieExceptionResponse.class)))})
ResponseEntity<EbaucheGrilleDTO> createEbaucheGrille(@Parameter(description = "Identifiant de l'ébauche de produit", example = "43") @PathVariable Long idEbaucheProduit,
                                                     HttpServletRequest request,
                                                     @RequestHeader HttpHeaders headers,
                                                     @RequestHeader(value = USERNAME_HEADER, required = false) String usernameHeader,
                                                     @RequestParam @DateTimeFormat(pattern = JSON_DATE_PATTERN) LocalDate debutEffet,
                                                     @RequestParam boolean isBase,
                                                     @RequestParam Long idGrillebase,
                                                     @RequestParam boolean isRegimeAgricole,
                                                     UriComponentsBuilder uriComponentsBuilder) throws SymphonieFunctionalException;

2 usages  1 implementation  egret-j +1
@ApiResponses(value = {
    @ApiResponse(responseCode = "200", description = "L'ébauche de grille a été modifiée"),
    @ApiResponse(responseCode = "500", description = "Problème interne", content = @Content(schema = @Schema(implementation = SymphonieExceptionResponse.class)))})
EbaucheGrilleDTO updateEbaucheGrille(@Parameter(description = "Identifiant de l'ébauche de produit", example = "60") @PathVariable Long idEbaucheProduit,
                                     @Parameter(description = "Identifiant de l'ébauche de grille") @PathVariable Long idEbaucheGrille,
                                     @RequestHeader(value = USERNAME_HEADER, required = false) String usernameHeader,
                                     @RequestBody EbaucheGrilleDTO ebaucheGrille) throws SymphonieFunctionalException;
```

Ensuite il y a les DTO :



Les DTO (Data Transfer Object) permettent de récupérer les informations souhaitées une fois que la requête sur la base de données est réalisée.

Exemples :

```java
package fr.hm.soprano.produit.ebauche.dto;

import ...

egret-j +1
@Data
@Builder
@AllArgsConstructor
@NoArgsConstructor
@JsonInclude(JsonInclude.Include.NON_NULL)
public class EbauchePartDTO {

    private Long id;
    @Size(max = 20)
    private String compteDeCreation;
    @JsonFormat(shape = JsonFormat.Shape.STRING, pattern = JSON_TIMESTAMP_PATTERN)
    private LocalDateTime dateDeCreation;
    @Size(max = 20)
    private String compteDeMiseAJour;
    @JsonFormat(shape = JsonFormat.Shape.STRING, pattern = JSON_TIMESTAMP_PATTERN)
    private LocalDateTime dateDeMiseAJour;
    @NotNull
    private EbaucheGrillePartDTO ebaucheGrillePart;
    private ReferentielSousGrilleValeurDTO sousGrilleValeur;
    private BigDecimal valeurPart;

}
```

Ici, on aura dans le retour JSON toutes les propriétés entrées dans la DTO donc par exemple l’ID de l’ébauche, le compte de création, la date de création de l’ébauche, etc.

On a ensuite les mapper. Ils permettent de faire correspondre les données d'une base de données avec les objets créés dans le programme lorsque que l'on requête sur la base de données.

Exemples :

```java
package fr.hm.soprano.produit.ebauche.mapper;

import ...

11 usages  1 implementation  egret-j +2
@Component
@Mapper(componentModel = "spring", uses = {
    TechniqueEbaucheTypeMapper.class,
    TechniqueEbaucheStatutMapper.class,
    ReferentielGammeMapper.class,
    ReferentielNoemieMapper.class,
    ReferentielRegimeMapper.class,
    ReferentielResponsabiliteMapper.class,
    EbaucheGrilleMapper.class,
    EbaucheProduitServiceMapper.class})
public interface EbaucheProduitMapper extends MapperModel<EbaucheProduitEntity, EbaucheProduitDTO> {

    1 implementation  egret-j
    @Mapping(target = "codeProduitSource", source = "produitSource.codeContractuel")
    @Mapping(target = "codeEbaucheProduitSource", source = "ebaucheProduitSource.code")
    @Mapping(target = "libelleEbaucheProduitSource", source = "ebaucheProduitSource.libelle")
    @Mapping(target = "idEbaucheProduitSource", source = "ebaucheProduitSource.id")
    @Mapping(target = "codeProduitCible", source = "produitCible.codeContractuel")
    @Mapping(target = "statut", source = "ebaucheStatut")
    @Mapping(target = "type", source = "ebaucheType")
    @Mapping(target = "changementRegime", ignore = true)
    EbaucheProduitDTO fromEntityToDto(EbaucheProduitEntity entity);

    1 implementation  egret-j
    @Mapping(target = "produitSource.codeContractuel", source = "codeProduitSource")
    @Mapping(target = "ebaucheProduitSource.code", source = "codeEbaucheProduitSource")
    @Mapping(target = "ebaucheProduitSource.libelle", source = "libelleEbaucheProduitSource")
    @Mapping(target = "ebaucheProduitSource.id", source = "idEbaucheProduitSource")
    @Mapping(target = "produitCible.codeContractuel", source = "codeProduitCible")
    @Mapping(target = "ebaucheStatut", source = "statut")
    @Mapping(target = "ebaucheType", source = "type")
    EbaucheProduitEntity fromDtoToEntity(EbaucheProduitDTO dto);
```

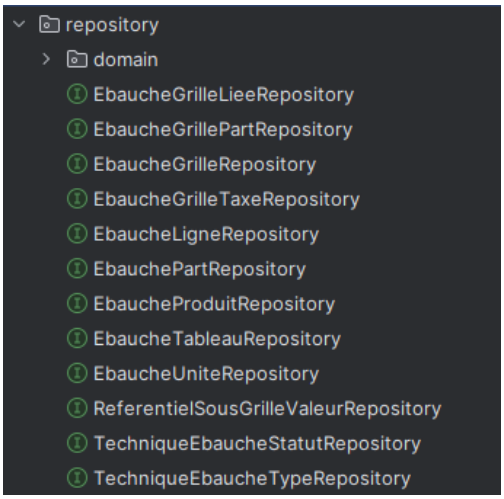
Ici, on a deux méthodes qui permettent de convertir des objets en données et à l'inverse convertir des données en objet. Si la donnée n'a pas le même nom que l'objet, on va alors créer via l'annotation '@mapping' la correspondance. En revanche, si l'objet et la donnée ont le même libellé, Hibernate saura alors faire la correspondance automatiquement.

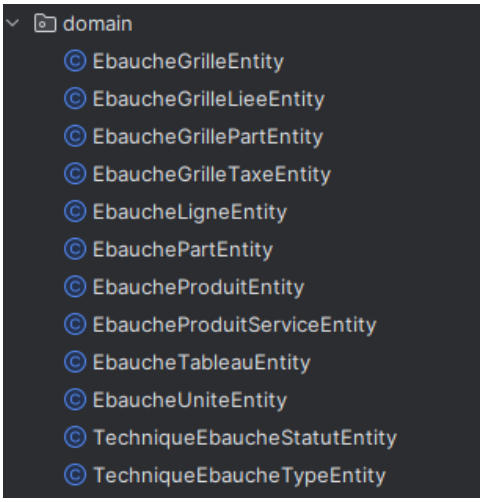
Hibernate a été intégré au projet pour pouvoir directement créer les tables via des annotations. Une dépendance a donc été ajoutée dans le 'pom.xml'

```xml
<dependency>
    <groupId>org.hibernate</groupId>
    <artifactId>hibernate-jpamodelgen</artifactId>
    <version>5.3.7.Final</version>
    <scope>provided</scope>
</dependency>
```

Pour ma part, j'ai utilisé les entity déjà existante.

On a aussi un dossier ‘repository’, c’est ici que toutes les requêtes sont réalisées vers la base de données :





Puis dans le dossier Domain , on a les différentes Entitys qui permettent de créer les tables des ‘ebauches-produit’ dans la base de données.

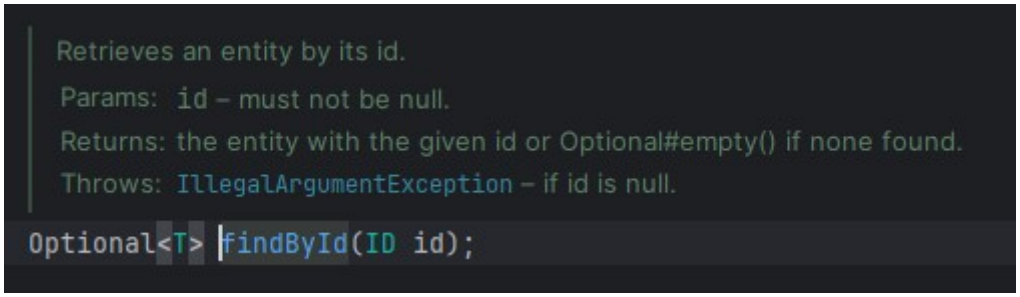
Exemple de requête :

```
@Query(value = "SELECT COUNT(ep) FROM EbaucheProduitEntity ep WHERE ep.code like UPPER(CONCAT('%-', :code)) or ep.code like :code")
int countEbaucheProduitEntitiesByCodeLike(@Param("code") String code);
```

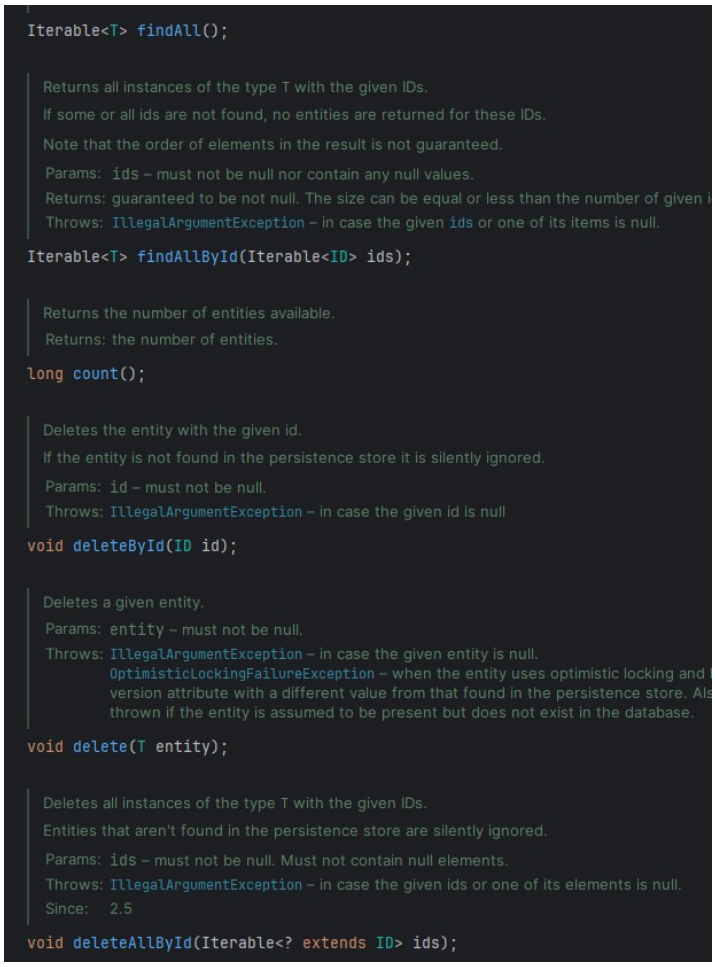
Exemple d’entity :

```
package fr.hm.soprano.produit.ebauche.repository.domain;

import ...

@Entity
@Table(name = "EB_GRILLE_PART")
@SequenceGenerator(name = "seq_eb_grille_part_gen", sequenceName = "SEQ_EB_GRILLE_PART", allocationSize = 1)
@Getter
@Setter
@SuperBuilder
@AllArgsConstructor
@NoArgsConstructor
public class EbaucheGrillePartEntity extends AbstractAuditEntity {

    @Id
    @Column(name = "ID")
    @GeneratedValue(strategy = GenerationType.SEQUENCE, generator = "seq_eb_grille_part_gen")
    private Long id;

    @ManyToOne(fetch = LAZY)
    @JoinColumn(name = "EB_GRILLE_ID", nullable = false)
    private EbaucheGrilleEntity ebaucheGrille;

    @ManyToOne
    @JoinColumn(name = "PARTPAYEUR_ID", nullable = false)
    private PartPayeur partPayeur;

    @Column(name = "ORDRE", nullable = false)
    private Integer ordre;

}
```

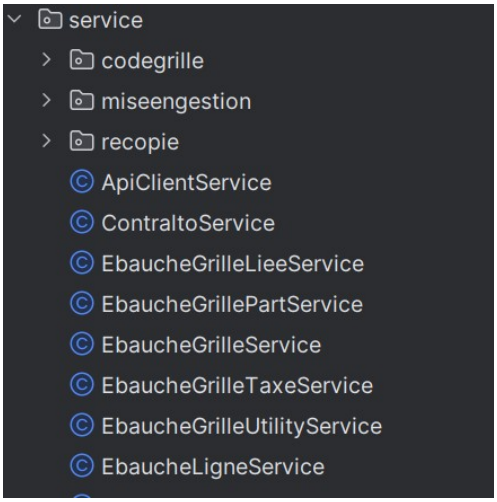
Spring permet aussi de faire des requêtes automatiques avec des méthodes déjà crée. Par exemple la fonction ‘findById’ qui va permettre d’aller chercher une données en base de données via son id :



Il y a d’autre type de fonction comme ‘deleteById’ ou ‘findAll()’ :



Enfin, on a un dernier sous-dossier nommé ‘service’. C’est ici que tout est ‘réuni’ : mapper , requête, gestion des erreurs. C’est le cœur du service. :



Exemple :

```
private Regime switchRegime(Regime currentRegime) throws SymphonieFunctionalException {
    return RegimeEnum.RG.name().equals(currentRegime.getCode()) ?
        regimeRepository.findByCode(RegimeEnum.AM.name()).orElseThrow(() -> ERROR_SOP_REGIME_NOT_FOUND.toException(RegimeEnum.AM.name()))
        : regimeRepository.findByCode(RegimeEnum.RG.name()).orElseThrow(() -> ERROR_SOP_REGIME_NOT_FOUND.toException(RegimeEnum.RG.name()));
}
```

Tâches réalisées :

Création de l’end-point ainsi que la méthode qui est appelée lors de l’appel de mon service :

```
@Override
@GetMapping("/{idEbaucheProduit}/autre-produit")
public ResponseEntity<Set<EbaucheProduitMemeCibleDTO>> consultEbaucheProduitMemeCible(@PathVariable(name = "idEbaucheProduit") Long idEbaucheProduit) throws SymphonieFunctionalException {
    return ResponseEntity.ok(ebaucheProduitService.consultProduitLieeByEbaucheWithProduitCible(idEbaucheProduit));
}
```

Mon service retourne un ‘Set’, c’est une sorte de liste un peu différente.
Comme le type ‘List’ justifie normalement d’une petite taille, j’ai dû trouver une façon différente de retourner une liste.

Voici la différence entre le type ‘Set’ et le type ‘List’ :

	List	**Set**
Duplications	Oui	Non
Ordennement	Ordonné	Dépend de l’implémentation
Accès aux données via index	Oui	Non

Création du cœur du service :

```
1 usage   bedminster-c *
public Set<EbaucheProduitMemeCibleDTO> consultProduitLieeByEbaucheWithProduitCible(Long idProduitEbauche) throws SymphonieFunctionalException {

    Long idProduitCible = null;
    Set<EbaucheProduitEntity> ebaucheProduitEntities;

    EbaucheProduitEntity entityEbaucheProduit = ebaucheProduitRepository.findById(idProduitEbauche)
        .orElseThrow(() -> ERROR_SOP_EBAUCHE_PRODUIT_BY_ID_NOT_FOUND.toException(idProduitEbauche));

    if (entityEbaucheProduit.getProduitCible() == null) {
        throw ERROR_SOP_EBAUCHE_PRODUIT_PRODUIT_CIBLE_IS_GONE.toException(idProduitEbauche);
    }

    idProduitCible = entityEbaucheProduit.getProduitCible().getId();
    ebaucheProduitEntities = ebaucheProduitRepository.findEbaucheProduitEntitiesByProduitCible(idProduitCible, idProduitEbauche);

    if (ebaucheProduitEntities.isEmpty()) {
        throw ERROR_SOP_EBAUCHE_PRODUIT_OTHER_BY_PRODUIT_CIBLE_NOT_FOUND.toException(idProduitCible);
    }
    return ebaucheProduitMemeCibleMapper.fromEntitySetToDTOSet(ebaucheProduitEntities);
}
```

C’est ici que j’ai réalisé mes requêtes et géré les erreurs.

Je teste d’abord si mon idEbauche existe bien (normalement, il est forcément obligé d’exister.). S'il n’existe pas, je lève une exception et je retourne le code http 404. (not found)

404 Not Found

Headers >


```
{
  "code": "404-SOP_EB_F043",
  "libelle": "L'ébauche de produit d'id '49 191' n'a pas été trouvée",
  "infos": null
}
```


Je teste ensuite si mon produit - ébauche a bien un produit cible. Si ce n’est pas le cas je retourne le code http 410 (gone) :

Response (0.427s) - http://localhost:8086/soprano-produit-api/v3/ebauche-produits/167/autre-produit

410 Gone

Headers >


```
{
  "code": "410-SOP_EB_F090",
  "libelle": "Aucun produit référentiel lié au produit ébauche '167'",
  "infos": null
}
```

Dans le dernier cas je vérifie que ma liste de produits-ébauche qui ont le même ‘produitCible’ est vide.
Si c’est le cas je retourne le code 206 (partial content)

Response (0.077s) - http://localhost:8086/soprano-produit-api/v3/ebauche-produits/1187/autre-produit

206 Partial Content

Headers >


```
{
  "code": "206-SOP_EB_F091",
  "libelle": "Aucun autre produit ébauche lié au produit cible '64 073'",
  "infos": null
}
```


Ensuite, je fais appel à la méthode de mon mapper qui va permettre de convertir mes données reçues de la base de données en objets. Il va convertir ma liste d’entitées ‘ebaucheProduit’ en ‘DTO ebaucheProduit’ :

```
package fr.hm.soprano.produit.ebauche.mapper;

import ...

;

3 usages  1 implementation  bedminster-c *
@Component
@Mapper(componentModel = "spring")
public interface EbaucheProduitMemeCibleMapper extends MapperModel<EbaucheProduitEntity, EbaucheProduitMemeCibleDTO> {


    1 usage  1 implementation  new *
    Set<EbaucheProduitMemeCibleDTO> fromEntitySetToDTOSet(Set<EbaucheProduitEntity> entity);


}
```

Via la DTO, il va savoir quelles informations il faut me renvoyer :
Il va me renvoyer toutes ces informations ainsi qu’une liste de grilles (qui correspondent au produit)
J’ai donc aussi créé une DTO pour les grilles.

```
package fr.hm.soprano.produit.ebauche.dto;

import ...

17 usages  bedminster-c
@Data
@Builder
@AllArgsConstructor
@NoArgsConstructor
public class EbaucheProduitMemeCibleDTO {


    private Long id;
    private Long idTableauGarantieRef;
    @Size(max = 50)
    private String code;
    @Size(max = 50)
    private String libelle;
    private Set<EbaucheGrilleMemeCibleDTO> ebaucheGrilles;


}
```

```
package fr.hm.soprano.produit.ebauche.dto;

import ...

11 usages  bedminster-c
@Data
@Builder
@AllArgsConstructor
@NoArgsConstructor
public class EbaucheGrilleMemeCibleDTO {

    private Long id;
    @Size(max = 50)
    private String code;

}
```

1. Une fois l’appel réalisé, mon service me retourne ceci, si tout s’est bien passé :

200 OK

Headers >


```
[
  {
    "id": 582,
    "idTableauGarantieRef": 388317,
    "code": "01-REFO-759_TEST",
    "libelle": "A00246",
    "ebaucheGrilles": []
  },
  {
    "id": 376,
    "idTableauGarantieRef": 388108,
    "code": "01-185845",
    "libelle": "A00246",
    "ebaucheGrilles": [
      {
        "id": 336,
        "code": "XT3003-XT3013"
      },
      {
        "id": 337,
        "code": "XFA001"
      }
    ]
  },
  {
    "id": 403,
    "idTableauGarantieRef": 388135,
    "code": "01-185848",
    "libelle": "A00246",
    "ebaucheGrilles": [
      {
        "id": 373,
        "code": "X00001"
      },
      {
        "id": 363,
        "code": "XT3001-XT3011"
      }
    ]
  }
]
```


On a bien l’id du produit-ébauche, son code, son libellé ainsi que ses grilles, leur code et leur libellé.

Problèmes rencontrés :

Lorsqu’un produit sans produit cible était retourné, j’avais une erreur car celui-ci était ‘null’. Je n’avais pas directement compris qu’il fallait que je gère cette exception.

L’erreur n’étant pas très explicite j’ai du chercher d’où cela venait vraiment :

```
Caused by: java.lang.NullPointerException Create breakpoint : null
    at fr.hm.soprano.produit.ebauche.service.EbaucheProduitService.consultProduitLieeByEbaucheWithProduitCible(EbaucheProduitService.java:110)
```

Une fois que j’avais compris l’erreur j’ai créé l’exception avec l’erreur voulue.

Pour cela, il y a un fichier qui recense toutes les erreurs. On peut donc créer ses propres erreurs. J’ai créé l’erreur suivante :

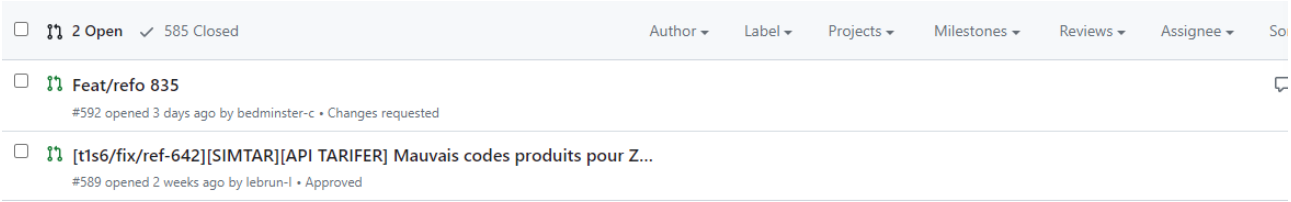
```
ERROR_SOP_EBAUCHE_PRODUIT_PRODUIT_CIBLE_IS_GONE( code: "SOP_EB_F090",  message: "Aucun produit référentiel lié au produit ébauche ''{0}''",  description: "",  GONE)
```

Puis via une condition j'ai vérifié que le produit est nul. Si c'est le cas alors je lève l'exception. :

```
if (entityEbaucheProduit.getProduitCible() == null) {
    throw ERROR_SOP_EBAUCHE_PRODUIT_PRODUIT_CIBLE_IS_GONE.toException(idProduitEbauche);
}
```

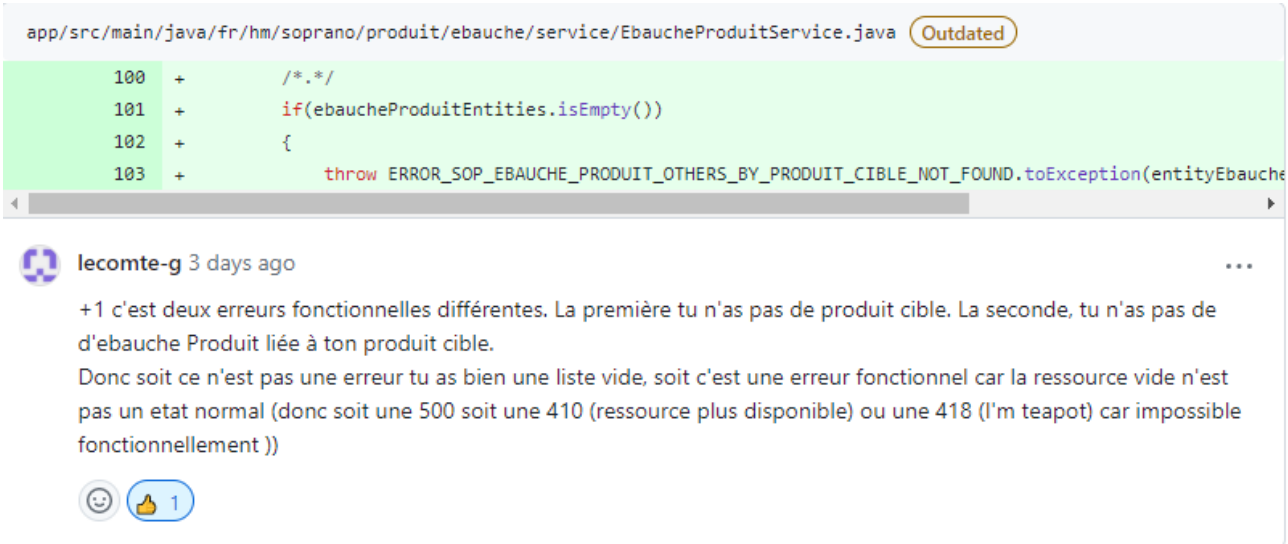
Tâche réalisée :

Une fois qu'un ticket est terminé, il faut faire une 'PR' (pull request), c'est-à-dire qu'on push notre branche sur le git puis l'on doit créer un' pullRequest' pour que notre branche sois merge:



Ici on y voit 2 branches en PR.

Une fois la PR réalisée, il y a une revue de code. Un développeur vient vérifier chaque nouveau changement dans le code puis met des commentaires quand il y a quelque chose à modifier :



Une fois les erreurs corrigées, une autre revue de code est réalisée et si tout est bon alors la branche est mergée sur sa branche-mère.

Ici, comme je travaille sur les ébauches, il y a une branche 'ebauche/lots2' . À partir de cette branche, les développeurs créent leurs branches pour réaliser leurs fonctionnalités.

Semaine 4 :

Tâche réalisée :
Cette semaine, j'ai continué d'améliorer le service que je dois réaliser. Après une revue de code, le techlead m'a demandé de mettre en place la pagination sur mon service.

Les services REST doivent respecter de nombreuses règles et bonne pratiques dont la pagination fait partie. On appelle cela la loi de Postel.

La pagination d'un service permet de récupérer les données petit à petit plutôt qu'en bloc, et la mise en place d'un système de cache, qui permet d'économiser du temps de traitement si plusieurs requêtes identiques sont faites à la suite.

La pagination doit en cas d'appel GET avec une liste en retour, fournir les informations de pagination.
Les retours de listes doivent être paginés avec des valeurs définies par défaut (page 0 et taille de page, entre 5 et 20 éléments par page). Celles-ci doivent pouvoir être pilotées par l'appelant.
Le mécanisme de pagination doit être cohérent sur l'ensemble des URI exposés.
La pagination a pour but de protéger les services d'une surcharge et de favoriser les itérations et les processus batch côté client. Ceci est vrai pour toutes les listes potentiellement supérieures à 50 éléments.

Il existe deux politiques de pagination, chacune possédant ses avantages et inconvénients :
- Par page, avec n° de page et définition de taille de page ;
- Par liens (HATEOAS) ;
- Par clé et taille de page.

La pagination s'appuie sur des champs clairement identifiés et standards.

Pour ma part, j'ai utilisé la pagination standard :

Elle doit au minimum utiliser la pagination standard pour le rendu de liste de ressources.
Cette méthode de pagination s'appuie sur 2 bases :

- la déclaration du souhait de navigation de l'appelant (optionnel) et l'inclusion dans le header du retour des informations de pagination (ou dans le corps du retour).

- La demande de navigation (paramètre de requête) doit être normalisée et globale , c'est à dire identique pour chaque route.

L'appel demande la page 2 avec une taille de 5 éléments par page :

curl GET '[HOST]/v1/personnes-physiques?size=5&page=2'

Le retour doit porter au moins au niveau des headers les champs suivants :
- x-current-page : n° de page courante ;
- x-current-size : nombre d'éléments par page ;
- x-total-count : nombre d'éléments total dans la liste ;
- x-total-page : nombre de pages total.

Si une meta structure est utilisée pour encapsuler les retours il est nécessaire, en plus des headers, d'ajouter la structure suivante :

```
  "page" : {
    "current" : 2,
    "size" : 5,
    "totalPages" : 200,
    "totalElements" : 1000
  },
  ...
}
```

Le service doit exposer les éléments de pagination au niveau des headers même si ceux-ci sont embarqués dans une meta structure.
Après avoir pris connaissance de ces informations, j'ai réalisé les différentes modifications sur mon service.

Je me suis inspiré de code déjà écrit dans le projet.
L'objet 'Page' de Spring était utilisé dans beaucoup de service :

Voici l'objet dans la JavaDoc avec ses méthodes :

Modifier and Type	Method	Description
static <T> Page<T>	empty()	Creates a new empty Page.
static <T> Page<T>	empty(Pageable pageable)	Creates a new empty Page for the given Pageable.
long	getTotalElements()	Returns the total amount of elements.
int	getTotalPages()	Returns the number of total pages.
<U> Page<U>	map(Function <? super T,? extends U> converter)	Returns a new Page with the content of the current one mapped by the given Function .

Lorsque que je rentre ce end-point j'ai un retour 200 :

http://localhost:8086/soprano-produit-api/v3/ebauche-produits/1477/autre-produit?page=0&size=30

Le retour est 200 (OK) car, tous mes éléments ont pu être récupérés et affichés (j'ai donc 30 éléments).

Si par ailleurs, je lui demande de me retourner uniquement 29 éléments le code de retour sera alors 206 (partial content) car il me manque un élément.

On a en fait une méthode qui permet de retourner des codes http différents selon le nombre d'éléments retournés.
Pour définir le nombre d'éléments retournés le constructeur de la classe 'Page' prend en paramètre un objet de type 'Pageable'. Celui-ci lorsqu'il est appelé est converti en deux sous-variables 'page' et 'size'.

'page' correspond à la page que l'on souhaite retourner ;
Exemple :
je veux retourner la page 0 avec 2 éléments :
url : http://localhost:8086/soprano-produit-api/v3/ebauche-produits/1477/autre-produit?page=0&size=2

206 Partial Content

Headers >


```
[
  {
    "id": 26,
    "idTableauGarantieRef": 387765,
    "code": "01-183740",
    "libelle": "B03018",
    "ebaucheGrilles": [
      {
        "id": 13,
        "code": "XIF001"
      }
    ]
  },
  {
    "id": 735,
    "idTableauGarantieRef": 388465,
    "code": "47-REFO-756",
    "libelle": "B03018",
    "ebaucheGrilles": [
      {
        "id": 777,
        "code": "XIF001"
      }
    ]
  }
]
```


url : http://localhost:8086/soprano-produit-api/v3/ebauche-produits/1477/autre-produit?page=1&size=2

Maintenant je souhaite retourner la page 1 avec 2 éléments :

206 Partial Content

Headers >


```
[
  {
    "id": 334,
    "idTableauGarantieRef": 388063,
    "code": "02-185826",
    "libelle": "B03018",
    "ebaucheGrilles": []
  },
  {
    "id": 546,
    "idTableauGarantieRef": 388281,
    "code": "03-185929",
    "libelle": "B03018",
    "ebaucheGrilles": [
      {
        "id": 494,
        "code": "XIF002"
      }
    ]
  }
]
```


Ici tout comme un livre, l'objet 'Page' me donne les éléments suivants de ma liste
Donc la page 1 sera différente de la page 2 etc..
'size' quant à lui correspond aux nombres d'éléments que l'on souhaite retourner :

Exemple :
Je souhaite retourner 3 éléments :
url : http://localhost:8086/soprano-produit-api/v3/ebauche-produits/1477/autre-produit?page=0&size=3

206 Partial Content

Headers >


```
[
  {
    "id": 1008,
    "idTableauGarantieRef": 388737,
    "code": "01-188448",
    "libelle": "B03018",
    "ebaucheGrilles": [
      {
        "id": 1230,
        "code": "XIF001"
      }
    ]
  },
  {
    "id": 1044,
    "idTableauGarantieRef": 388773,
    "code": "03-188473",
    "libelle": "B03018",
    "ebaucheGrilles": [
      {
        "id": 1293,
        "code": "XIF001"
      }
    ]
  },
  {
    "id": 822,
    "idTableauGarantieRef": 388552,
    "code": "01-188350",
    "libelle": "B03018",
    "ebaucheGrilles": [
      {
        "id": 943,
        "code": "XIF001"
      }
    ]
  }
]
```


Maintenant 1 élément :
url : http://localhost:8086/soprano-produit-api/v3/ebauche-produits/1477/autre-produit?page=0&size=1

206 Partial Content

Headers >


```
[
  {
    "id": 308,
    "idTableauGarantieRef": 388037,
    "code": "02-185818",
    "libelle": "B03018",
    "ebaucheGrilles": []
  }
]
```

Voici la mise en place côté technique :

Première chose à changer, le type de retour de la requête sur la base de données :

```
@Query(value = "SELECT ep FROM EbaucheProduitEntity ep  WHERE ep.produitCible.id = :produitCible AND ep.ebaucheType.code = 'MAJ' AND ep.ebaucheStatut.code = 'CONSTRUCTION' AND ep.id <> :idEbauche ")
Page<EbaucheProduitEntity> findEbaucheProduitEntitiesByProduitCiblePagination(@Param("produitCible") Long produitCible, @Param("idEbauche") Long idEbauche, Pageable pageReference);
```

Ici, le type de retour est une ‘Page’, puisque l’on veut pouvoir faire de la pagination.
Puis on change le type de retour du service qui devient aussi ‘Page’ :

```
public Page<EbaucheProduitMemeCibleDTO> consultProduitLieeByEbaucheWithProduitCible(Long idProduitEbauche, Pageable pageable) throws SymphonieFunctionalException {

    Long idProduitCible;
    Page<EbaucheProduitEntity> ebaucheProduitEntitiesPagination;

    EbaucheProduitEntity entityEbaucheProduit = ebaucheProduitRepository.findById(idProduitEbauche)
        .orElseThrow(() -> ERROR_SOP_EBAUCHE_PRODUIT_BY_ID_NOT_FOUND.toException(idProduitEbauche));

    if (entityEbaucheProduit.getProduitCible() == null) {
        throw ERROR_SOP_EBAUCHE_PRODUIT_PRODUIT_CIBLE_IS_GONE.toException(idProduitEbauche);
    }

    idProduitCible = entityEbaucheProduit.getProduitCible().getId();
    ebaucheProduitEntitiesPagination = ebaucheProduitRepository.findEbaucheProduitEntitiesByProduitCiblePagination(idProduitCible, idProduitEbauche, pageable);

    if (ebaucheProduitEntitiesPagination.isEmpty()) {
        throw ERROR_SOP_EBAUCHE_PRODUIT_OTHER_BY_PRODUIT_CIBLE_NOT_FOUND.toException(idProduitCible);
    }

    return new PageImpl<>(
        ebaucheProduitMemeCibleMapper.fromEntityListToDTOList(ebaucheProduitEntitiesPagination.getContent()),
        pageable,
        ebaucheProduitEntitiesPagination.getTotalElements()
    );
```

Enfin au niveau du end-point on retourne une liste :

```
bedminster-c *
@Override
@GetMapping("/{idEbaucheProduit}/autre-produit")
public ResponseEntity<List<EbaucheProduitMemeCibleDTO>> consultEbaucheProduitMemeCible(@PathVariable(name = "idEbaucheProduit") Long idEbaucheProduit,
                                                                                       @PageableDefault(size = 8, page = 0) Pageable pageable) throws SymphonieFunctionalException,
                                                                                                                                                      SymphonieTechnicalException {
    return responseEntityPaginate(ebaucheProduitService.consultProduitLieeByEbaucheWithProduitCible(idEbaucheProduit, pageable));
}
```

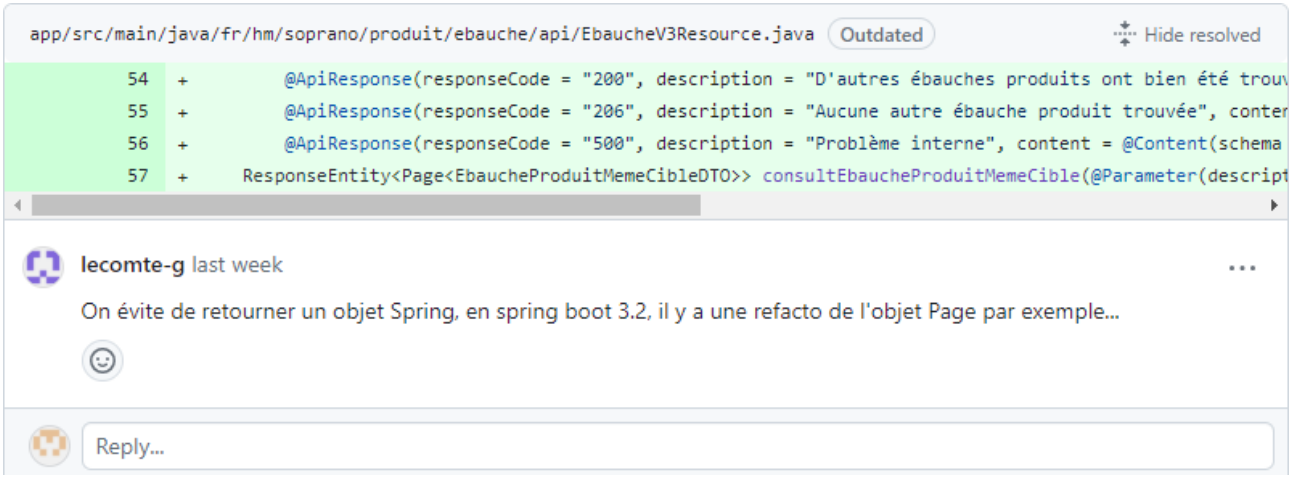
Grâce à la méthode ‘responseEntityPaginate’ ma page peut-être convertie en ‘List’ et gère les codes http de retour comme on a pu le voir un plus haut.

Cette fonction permet aussi d’ajouter les différentes informations de paginations au header :

```
Expires: 0
Cache-Control: no-cache, no-store, max-age=0, must-revalidate
X-XSS-Protection: 1; mode=block
Pragma: no-cache
Date: Mon, 05 Feb 2024 09:13:54 GMT
x-total-count: 30
x-current-size: 1
Connection: keep-alive
x-current-page: 4
x-total-page: 30
X-Content-Type-Options: nosniff
Transfer-Encoding: chunked
Content-Type: application/json
```

Problèmes rencontrées :

Au début, j'avais décidé de retourner un objet 'Page' mais suite à une revue de code, il m'a été demandé de retourner une liste, tout simplement, car il ne faut pas être dépendant d'une technologie. Par exemple cet objet 'Page' dans Spring 3.2.0 a eu une re-factorisation et certaines personnes ont donc eu des problèmes. Cela poserait des problèmes de compatibilité et de régression. Un temps considérable serait perdu :



Semaine 5 :

Cette semaine, j’ai pu débuter mes 2 tickets front-end, j’ai dû m’adapter à un nouveau framework et un nouveau langage de programmation : le TypeScript.
Le TypeScript est en fait basé sur le JavaScript, mais avec les variables qui doivent être typées. Cela améliore donc la sécurité du code.

Angular est un framework assez complexe mais très puissant une fois que l’on sait l’utiliser, son arborescence est aussi assez complexe mais permet de créer des applications web très optimisées.

Angular fonctionne avec des composants :
Un composant est un élément fondamental d’une application Angular. C’est un module qui contient du code HTML, CSS et JavaScript pour afficher une partie de l’interface utilisateur d'une application.

Il est défini par une classe qui contient des propriétés et des méthodes pour manipuler les données et l’interface utilisateur de la partie de l'application gérée par le composant. Un composant peut également inclure des modèles et des directives pour définir l'apparence et le comportement de l'interface utilisateur.

Les composants sont organisés en hiérarchie dans une application Angular, avec un composant principal (appelé composant racine) qui contient les autres composants de l’application. Les composants peuvent être imbriqués les uns dans les autres pour construire une application complexe en combinant des parties plus simples.

Ils sont importants pour la modularité et la réutilisabilité du code dans une application. Ils permettent de diviser l'application en partie plus petite et gérable, ce qui facilite la maintenance et l’évolution de l'application.

Chaque projet Angular dispose de ces différents fichiers :

1. /src

- **index.html** est le template HTML de premier niveau.
- **style.css** est la feuille de style de premier niveau.
- **main.ts** est le fichier de démarrage de l'application.
- **favicon.ico** est le fichier de favicon comme dans tout site web.

2. /app

- **app.component.ts** est le fichier source qui décrit le composant racine. C'est le composant de plus haut niveau de l'application. Un composant est le bloc de construction de base d'une application Angular. La description du composant inclut le code du composant, le template HTML, les styles qui peuvent être décrits dans ce fichier ou dans des fichiers séparés

  Les nouveaux composants seront ajoutés dans ce répertoire.

3. /assets

contient les images utilisées par l'application

4. .angular

regroupe les fichiers requis pour construire l'application

5. .e2e

Contient les fichiers utilisés pour tester l'application

6. .node_modules

Contient tous les packages utilisés par l'application

7. angular.json

Décrit l'application pour les outils de construction. On retrouvera notamment :

- le répertoire source ("src" en général)
- le préfixe ("app" en général)
- l'index ("src/index.html")
- le main ("src/main.ts")
- polyfills dans le cas où on aurait besoin d'ajouter du code pour rendre l'application compatible avec plusieurs navigateurs.
- le chemin vers tsConfig ("tsconfig.app.json")
- les chemins vers les assets (images, fonts, favicon...)
- les chemins vers les feuilles de style ("src/styles.css")
- des informations de configuration (production, local, dev...)

8. package.json

C'est dans ce fichier que sont définies les dépendances de l’application.

On retrouvera notamment :

- les packages @angular
- rxjs, une bibliothèque pour la programmation réactive
- tslib pour le typescript
- CLI pour les outils de développement en ligne de commande
- zone.js pour détecter les changements

9. tsconfig

Ce sont les fichiers qui décrivent la configuration pour le compilateur TypeScrip

Une dernière chose qui peut se trouver dans beaucoup de projets Angular, c'est le composant 'router-outlet' :

```
<router-outlet></router-outlet>
```

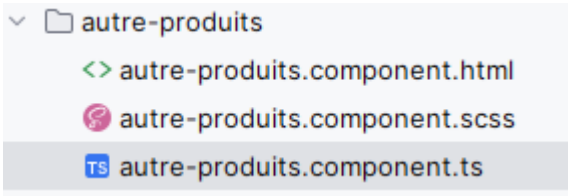
C'est en quelque sorte le 'servlet dispatcher' principale de l'application. C'est lui qui va permettre de rediriger l'utilisateur sur l'onglet voulu

Exemple d'un composant :

```
<soprano-autre-produits *ngIf="isInformations()"></soprano-autre-produits>
```

Ce composant peut-être appelé partout dans le code, il suffit, tout comme les balises HTML, d'ouvrir des chevrons puis les fermer pour appeler ce composant.

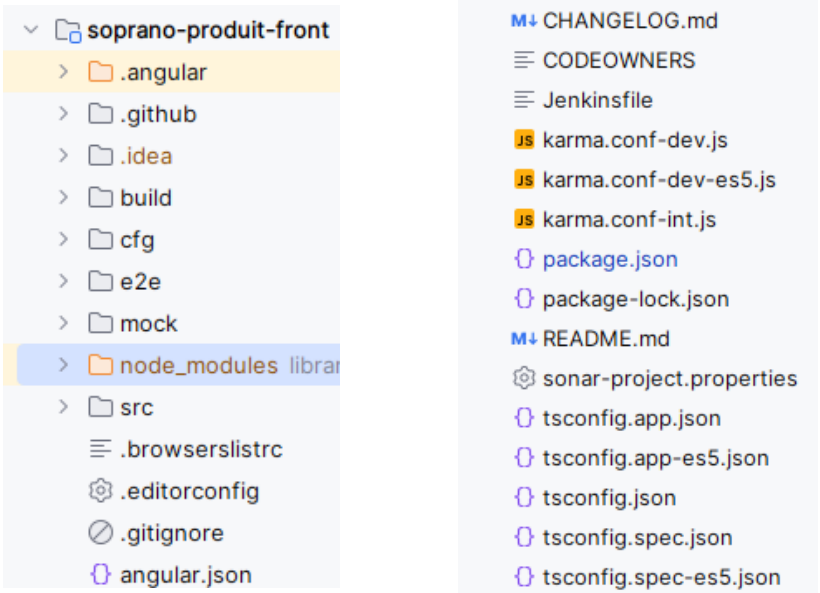
Ce composant est donc composé des 3 fichiers qui le représentent :



Un fichier HTML, un fichier CSS et un fichier TypeScript.

- Le fichier HTML permettra de réaliser 'le corps' du composant
- Le fichier CSS permettra de réaliser le style du composant
- Le fichier TypeScript sera le 'back-end' du composant. Il permettra de gérer des événements faire des appels etc..

Exemple d'arborescence de projet :



Tâche réalisée:

J’ai donc réalisé cette semaine la partie front-end de mon service.

L’attendu est une SideNav qui permettra d’afficher tout les produit- ébauche lié au produit référentiel de base. J’ai donc d’abord cherché comment faire une SideNav avec Angular.

Angular propose des composants déjà tout fait que l’on appelle ‘Angular material’ et qui permettent d’ajouter des éléments à nos pages (un peu comme le framework Bootstrap). Ces composants commencent en général par le préfixe ‘mat’ pour ‘material’.

Il y a notamment 2 composants qui permettent de créer des SideNav. C’est deux composants doivent faire partie respectivement de leur composant père ‘mat-sidenav-container’ et ‘mat-side-nav-drawer’ pour être utilisé.

Leur différence est assez simple :

- l’un permet de créer des SideNav uniquement sur un composant souhaité
- L’autre permet de faire des SideNav sur toute une page (ils recouvrent donc tous les autres composants)

Le premier composant se divise en deux sous composants :

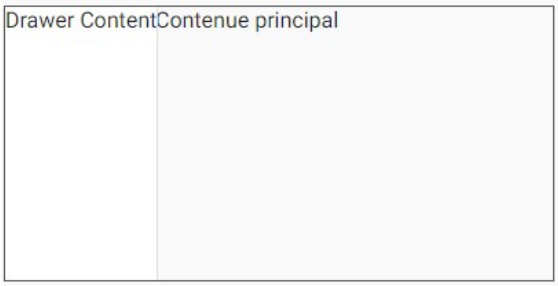
‘mat-sidenav’ et ‘mat-sidenav-content’

On mettra dans le premier sous- composant le composant qui permet d’afficher la SideNav.

Puis dans le second composant on mettra le ‘Main’ c’est-à-dire le reste de la page.

Illustration :

```
<mat-sidenav-container>
  <mat-sidenav>Start</mat-sidenav>
  <mat-sidenav-content>Main</mat-sidenav-content>
</mat-sidenav-container>
```



On a tout d’abord notre ‘mat-sidenav-container’ qui contient nos deux sous composants : ‘mat-sidenav’ et ‘mat-sidenav-content’.

Le premier sous composant contient les différents composants(SideNav) qui contiendront donc les informations souhaitées.

Le second sous composant lui contient le ‘Main’ donc le reste des pages de l’application web.

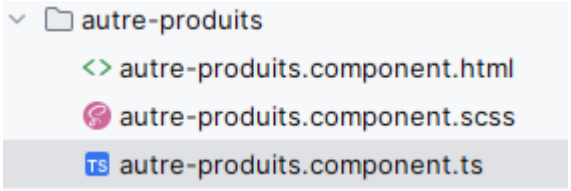
Le deuxième composant, quant à lui, permet d’afficher une SideNav mais sur un composant voulu.

Lui aussi fonctionne avec deux sous composants, mais appelé différemment :

‘Mat-drawer’ et ‘mat-drawer-content’

```
<mat-drawer-container>
  <mat-drawer>Start</mat-drawer>
  <mat-drawer-content>Main</mat-drawer-content>
</mat-drawer-container>
```

Une fois que j’ai pris connaissance de ces différentes informations, j’ai dû commencer à créer mon composant SideNav :



Puis dans le HTML, j’ai pu créer le corps de ma SideNav :

```
<div class="flex-col gap-10">
    <h1>Processus en cours impactant le produit : {{idProduitEbauche}}</h1>
    <div>
        <div class="flex-row gap-10">
            <div class="flex-10">
                <div class="center height-110">
                    <div class="vertical-line margin-top-vertical-line center">
                        <div class="disk"></div>
                    </div>
                </div>
            </div>
            <mat-card class="flex-75">
                <mat-card-content>
                  <div>
                      <p>Raison Sociale :  </p>
                      <ul>
                          <p>Tarif(s) impacté(s) :</p>
                          <li>Tarif :</li>
                          <li>Tarif :</li>
                          <li>Tarif :</li>
                      </ul>
                  </div>
                  <div>
                      <p>Libéllé PM : </p>
                      <ul>
                          <p>Tarif(s) impacté(s) : </p>
                        Pas de tarif
                      </ul>
                  </div>
                </mat-card-content>
            </mat-card>
        </div>
    </div>
```

Je n’ai pas encore réalisé le style car cela fera l’objet de mon dernier ticket.

Dans le fichier TypeScrtipt j’ai réalisé plusieurs choses :

- J’ai tout d’abord donné un nom à mon composant pour qu’il puisse être appelé partout dans le code :

  Il se divise en 3 sous url :

  - le selector donc la balise qui permettra d’appeler ce composant

- le templateUrl ; pour représenter le ‘corps’ du composant
- styleUrls , il est représenté par un tableau car plusieurs URL de style peuvent être liés à un composant

```
@Component({
    selector: 'soprano-autre-produits',
    templateUrl: './autre-produits.component.html',
    styleUrls: ['./autre-produits.component.scss']
})
```

- j’ai appelé ma méthode ‘ getAutreProduits()’ avec en paramètre l’ID du produit-ébauche actuellement consulté :

```
1+ usages   bedminster-c
public getAutreProduits(idEbaucheProduit: number): void {
    this.autreProduitService.getAutreProduitsByIdEbauche(idEbaucheProduit).subscribe(
        next: data : AutreProduitModel[]  => {
            this.autre_produit = data
                .map(data : AutreProduitModel  => new AutreProduitModel(data));
        },

        error: exception => this.errorService.displayMessageError( message: "Récupération impossible", exception)
    );

}
```

Cette méthode appelle ensuite la méthode ‘getAutreProduitsByIdEbauche()’ de la classe AutreProduitService. Elle retourne un Observable de Liste. :

```
public getAutreProduitsByIdEbauche(idEbauche :number): Observable<Array<AutreProduitModel>> {
    return this.ebaucheProduitsService.getAutreProduits(idEbauche);
}
```

Puis cette méthode appelle elle-même une autre méthode qui elle appelle via l’url REST que j’ai créé dans l’application back ma méthode ‘consultEbaucheProduitMemeCible()’ :

```
1+ usages   bedminster-c
public getAutreProduits(idProduitEbauche: number): Observable<Array<AutreProduitModel>> {
    const url : string  = `/soprano-produit-api/v3/ebauche-produits/${idProduitEbauche}/autre-produit`;
    return this.http.get<any>(url).pipe(catchError(this.errorHandler))
}
```

L’appel de toutes ces méthodes se déclenche au clic d’un bouton dans le header :

```
<button class="nav_item" mat-button routerLinkActive="active" (click)="openSideNavMenu( sideNavMenuToOpen: 0)">
    <img src="../../../assets/images/share.png" alt="Documents">
    Produit tarif partagé
</button>
```

En Angular, il y a des méthodes pré-créés qui permettent de détecter des événements. L'élément tracé ici est le clic de l'utilisateur. Dans le bouton on peut y voir donc la méthode '(click)' qui est égale à la méthode 'openSideNav()' avec le paramètre 0.

```
1+ usages   bedminster-c
openSideNavMenu(sideNavMenuToOpen: HeaderButtonEnum) : void  {
    switch (sideNavMenuToOpen) {
        case HeaderButtonEnum.informations:
            this.openInformations();
            break;
        case HeaderButtonEnum.modification:
            this.openModification();
            break;
    }
}
```

Cette méthode permet selon le paramètre passé appeler une méthode différente.

Pour détecter quel est la SideNav que l'utilisateur veut ouvrir, j'ai créé des Enums :

Les énumérations permettent de définir un ensemble fini de constantes, chacune d'entre-elles est séparée des autres par une virgule. Comme ces champs sont constants, leur nom est en majuscule par convention.

Pourquoi ne pas créer directement des constantes ?

- le principal inconvénient de cette technique est qu'il n'y a pas de contrôle sur la valeur affectée à une donnée surtout si les constantes ne sont pas utilisées : il est possible d'utiliser n'importe quelle valeur permise par le type de la variable en plus des constantes définies. Le compilateur ne peut faire aucun contrôle sur les valeurs utilisées
- il n'est pas possible de faire une itération sur chacune des valeurs
- il n'est pas possible d'associer des traitements à une constante
- les modifications faites dans ces constantes notamment les changements de valeurs ne sont pas automatiquement reportées dans les autres classes qui doivent être explicitement recompilées

Voici donc la Class Enulm que j'ai créée :

```
1+ usages   bedminster-c
export enum HeaderButtonEnum {
    informations = 0,
    modification = 1,
}
```

Lorsque que j' appelle 'HeaderButtonEnum.informations' la valeur sera donc égale à 0 etc..

Mon Switch case sera donc capable de reconnaître cela et rentrera donc dans mon premier cas et appelera la méthode openInformation() :

```
1+ usages   bedminster-c
openInformations() : void  {
    this.openSideNav.emit(HeaderButtonEnum.informations);
}
```

Auparavant une annotation @Output a été créée :

```
@Output() openSideNav : EventEmitter<any>  = new EventEmitter<any>();
```

Cela permet de créer un nouvel événement.
Dans un second temps une méthode est appelée et va émètre un éventement :

Cet événement est émis . Le composant auxquelles on passera l'événement en paramètre agira en cons équence.

Ici l'événement déclenche une méthode (openSideNav($event)) qui récupère l'événement envoyé (il s'agira donc d'un enum) puis exécute la méthode ci-dessous :

```
<soprano-header-gestion-cotisation (openSideNav)="openSideNav($event)"></soprano-header-gestion-cotisation>
```

```
openSideNav($event) : void  {
    this.navSelected = $event;
    this.sidenav.open();
}

private navSelected: HeaderButtonEnum;
```

J'assigne cet événement à la variable 'navSelected' que j'ai créée auparavant pour la suite

Puis j'applique la méthode open() à la variable SideNav
Il va alors aller lire la SideNav à ouvrir grâce à l'annotation ViewChild :

```
@ViewChild('sidenav') sidenav: MatSidenavContainer
```

Ici, grâce à un identifiant il va entrer dans le 'mat-sidenav'

```
new *
<mat-sidenav-container #sidenav class="example-container _flex _fill">
    <mat-sidenav position="end"  style="...">
        <soprano-autre-produits *ngIf="isInformations()"></soprano-autre-produits>
        <soprano-modification *ngIf="isModification()"></soprano-modification>
```

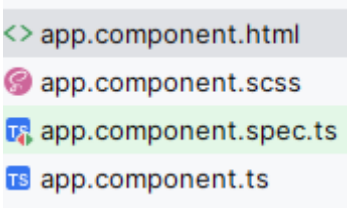
Ce qui se passe ici c'est qu'Angular va venir lire les deux méthodes liées aux composants. Ce sont des booléens qui vont permettre de savoir quel SideNav a été séléctinnée.

```
1+ usages  bedminster-c
isInformations() : boolean  {
    return this.navSelected === HeaderButtonEnum.informations;
}
1+ usages  bedminster-c
isModification() : boolean  {
    return this.navSelected === HeaderButtonEnum.modification;
}
```

S'ils retournent True alors la SideNav va s'ouvrir sinon le composant sera ignoré et donc ne sera pas ouvert.

Une fois tout cela réalisé, je devais donc trouver la façon de faire pour que la SideNav s'affiche au-dessus de tous les composants. J'ai dû remonter à la racine du projet puis appelé les différents composants Angular et celui que j'ai précédemment créé:

Comme vu au-dessus, la racine du projet est sous cette forme :




```
<soprano-http-loader></soprano-http-loader>
  bedminster-c
  <mat-sidenav-container #drawer class="example-container _flex _fill">
      <mat-sidenav position="end"  style="...">
          <soprano-autre-produits *ngIf="isInformations()"></soprano-autre-produits>
          <soprano-modification *ngIf="isModification()"></soprano-modification>
      </mat-sidenav>
      <mat-sidenav-content class="_max-flex-98 ">

          <header-app *ngIf="!isGestionCotisationContext"></header-app>
          <soprano-header-gestion-cotisation [titre]="titre" *ngIf="isGestionCotisationContext" (openSideNav)="openSideNav($event)"></soprano-header-gestion-cotisation>
          <router-outlet></router-outlet>
          <contralto-snackbar></contralto-snackbar>

      </mat-sidenav-content>
  </mat-sidenav-container>
```

On a ici tous les composants sources de l'application qui sont appelées.

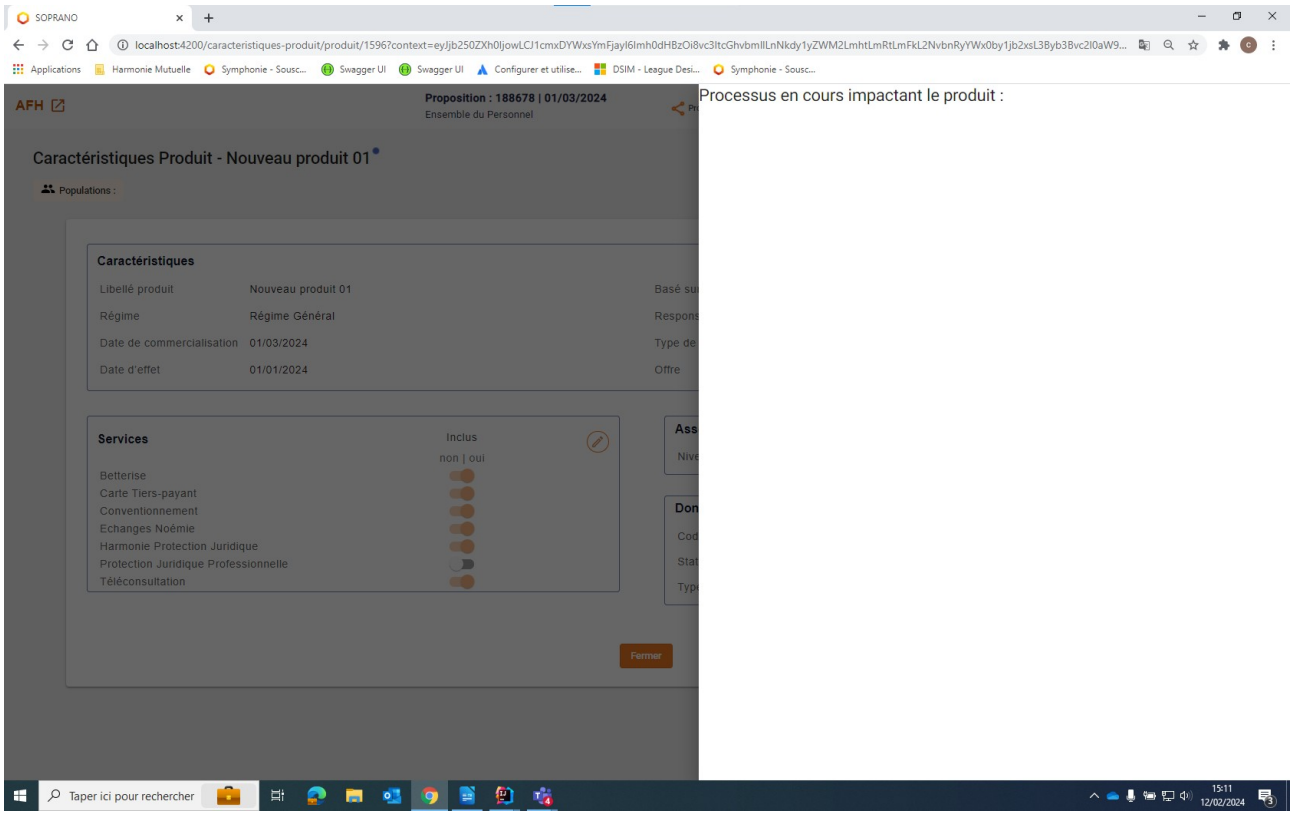
J’ai donc mis à l’intérieur du composant ‘mat-sidenav-container’ tous les autres composants ainsi que ma SideNav pour qu’elle s’affiche au-dessus de ceux-ci.

Puis dans ma balise ‘mat-sidenav’ j’ai mis mon composant qui représente la SideNav ‘soprano-autre-produit’ (comme on l’a vu plus haut le composant est divisé en 2 sous-composants). J’ai ensuite mis le reste des composants dans la balise ‘mat-sidenav-content’ qui contient le ‘Main’

J’ai réalisé un test, et voilà ce que me retourne mon service au clic du bouton dans le header :

Il me retourne le résultat voulu qui va remonter jusqu’à mon composant. Voici le résultat dans la console de Google une fois l’appel réalisé

Puis au niveau du front, j’ai la SideNav voulue qui s’affiche :

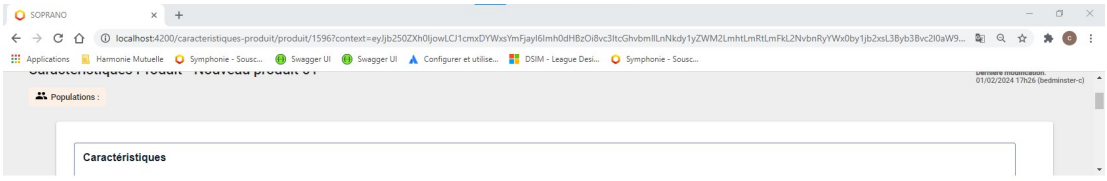


Le front sera l’objet de mon prochain ticket, c’est pour cela qu’il n’y a rien de complété.

Problèmes rencontrés :

Il m’a été difficile de trouver où mettre ma SideNav dans le code j’ai dû demander à d’autres développeurs pour bien comprendre quel composant encapsulé dans le composant de la SideNav.

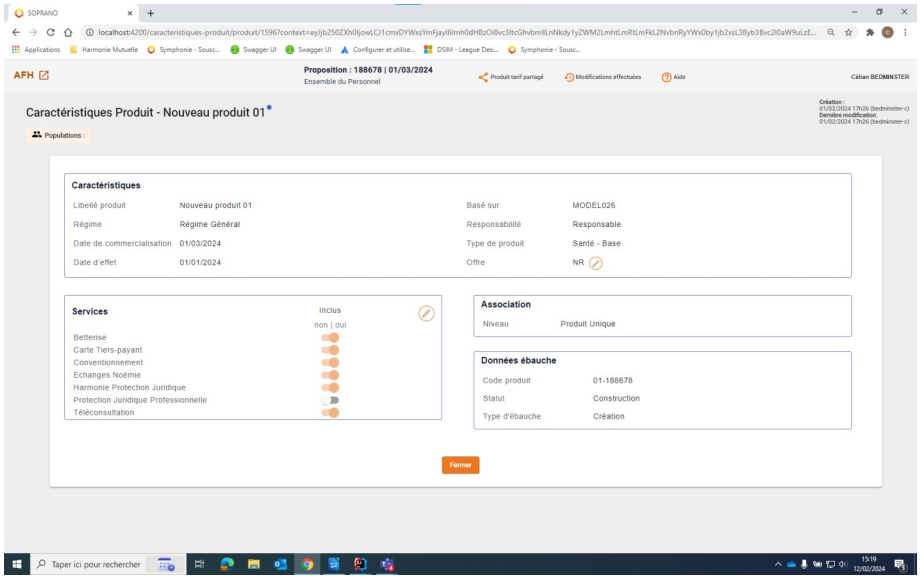
Un second problème que j’ai rencontré a été la SideNav qui coupait la moitié de la page principale :



J’ai dû ajouter une propriété CSS pour que celle-ci prenne bien toute la page :

Je lui ai spécifié une hauteur de 100vh et tout était corrigé :

```
height: 100vh;
```



Semaine 6 et 7:

Lors de ma 6ème et 7ème semaine de stage j'ai terminé mes tickets front-end que j'ai pu les mettre en 'pull-Request draft'. Une P.R draft permet de réaliser une 'pré-pull request' pour que les autres développeurs la valident sans pour autant merge la branche .

Tâche réalisées :

Mon premier ticket étant terminé j'ai pu commencer mon second ticket Front-End. Celui-ci consiste à afficher les informations récupérées tout d'abord en les 'Mockant'.

Un Mock sert à désigner un objet virtuel, de quelque nature que ce soit, qui a toutes les attitudes, toutes les apparences et tous les mouvements de son double réels.

Ici le but était de boucler sur le tableau de produit que je recevais et entrer des informations en dure.

Grâce à la directive *ngFor de Angular on peut réaliser des boucles sur des variables récupérées du côté de TypeScript

Exemple :

```
<div class="processus" *ngFor="let values of infosContralto">
    <h3>Raison sociale : PLC - Proposition :8455436</h3>
    <ul>
        <li>
            <div class="prdt">
                <h4>Produit-Ebauche : <b>789</b></h4>
            </div>
            <div class="tarifs-impactes">
                <h4>Tarif(s) impacté(s) :</h4>
                <div *ngFor="let tarifs of values.produit.ebaucheGrilles">
                    <ul>
                        <li>
                            Tarif : <b>48</b>
                        </li>
                    </ul>
                </div>
            </div>
        </li>
    </ul>
</div>
```

Ici la variable parcourue est 'infosContralto' que j'alimente ici :

```
this.tab.push(this.ebaucheProduitsService.getInfoAPIExperience(params).pipe(
```

J'expliquerai plus en détail après l'utilité de créer un dictionnaire.

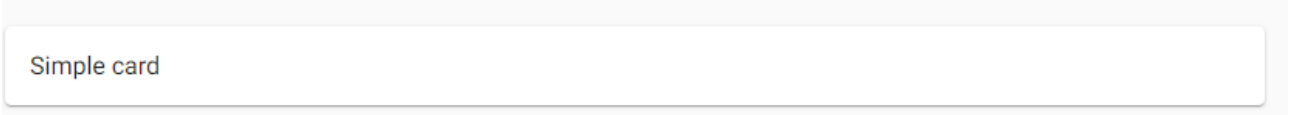
Ici on peut voir qu'au niveau :

- De la raison sociale
- De la Proposition
- Du Produit-Ebauche

- Des tarifs

les informations sont saisies en dur.

Pour afficher les informations j'ai utilisé les composants 'mat-card' et 'mat-card-content' qui permettent de créer des card avec un effet 3d grâce à des ombres :



Le Mock va permettre de vérifier que selon le nombre de retours de l'appel, Angular créé bien le nombre de mock correspondant. Cela va permettre de vérifier qu'il parcourt bien ma liste de produits et n'affiche pas d'erreurs.

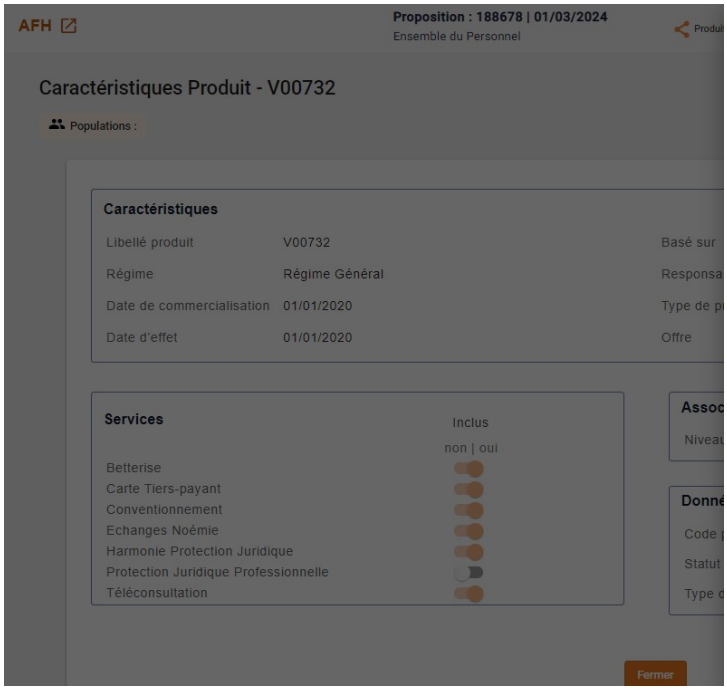
Par exemple, si j'ai un retour de 7 produits il fera alors 7 mock (il créera 7 div) avec pour l'instant les mêmes informations pour chaque mock .

Une fois cela réalisé j'ai pu afficher les différentes informations que je récupère dans l'appel :

```
<div *ngIf="isAutreProduitEmpty(); else listeVide">
    <h1>Processus en cours pour le produit : <b>{{this.idProduitEbauche}}</b></h1>
    <div class="processus" *ngFor="let values of infosContralto">
        <h3>Raison sociale :  - Proposition : </h3>
        <ul>
            <li>
                <div class="prdt">
                    <h4>Produit-Ebauche : <b>{{values.produit.id}}</b></h4>
                </div>
                <div class="tarifs-impactes">
                    <h4>Tarif(s) impacté(s) :</h4>
                    <div *ngFor="let tarifs of values.produit.ebaucheGrilles">
                        <ul>
                            <li>
                                Tarif : <b>{{tarifs.code}}</b>
                            </li>
                        </ul>
                    </div>
                </div>
            </li>
        </ul>
    </div>
</div>
```

On peut voir encore que 2 informations sont restées en dure car elles ne sont pas disponibles dans le premier appel que je réalise. Pour les obtenir il faut que je réalise un second appel à une API interne à Harmonie-Mutelle.

Pour l'instant voici le résultat après le premier appel :



On voit qu'ici tous les produits et leurs(s) tarifs ont bien été affichés mais il reste encore la raison sociale et la proposition à compléter.

Pour cela il faut faire appel à une seconde api interne.

J'ai dû trouver une façon de faire correspondre les informations reçues du premier appel puis du second appel . Comme il s'agit d'appels asynchrones on ne sait pas quand on va recevoir une réponse (l'information d'un produit peut arriver à un moment différent après les 2 appels)
J'ai donc utilisé une librairie que l'on appel rxjs. Elle permet entre autres de faciliter la gestion des événements asynchrones. :

```
52      public getAutreProduits(): void {
53
54          this.autreProduitService.getAutreProduitsByIdEbauche(+this.getIdEbaucheProduit(),this.page,this.size).subscribe(
55              data => {
56                  this.autreProduit = data
57                      .map(data => new AutreProduitModel(data));
58                  this.getInfosApiExperienceAutreProduit();
59              },
60              exception => this.errorService.displayMessageError("Récupération de l'ébauche impossible", exception)
61          );
62      }
63
64      public getInfosApiExperienceAutreProduit(): void {
65
66          let params: any = {};
67
68          for (let produitEbauche of this.autreProduit) {
69
70              params = {
71                  idProduitEbauche: produitEbauche.id
72              }
73              this.tab.push(this.ebaucheProduitsService.getInfoAPIExperience(params).pipe(
74
75                  map((res: ProduitTarifExperience) => {
76                      this.infosContralto.push({ produit: produitEbauche, exp: new ProduitTarifExperience(res) });
77                      return res;
78                  }),
79                  catchError(() => of(null))
80              ));
81          }
82          zip(...this.tab)
83              .subscribe(data => {
84                  // recupération des données dans le pipe.
85              },
86                  exception => this.errorService.displayMessageError("Récupération API expérience impossible", exception)
87              );
88
89      }
```

Explication :

Tout d'abord j'appelle ma méthode 'getAutreProduit()' qui envoie une requête http pour récupérer tous mes ébauches de produit lié à un même produit référentiel.

Une fois que l'appel est terminé j' appelle ma seconde méthode (ligne 58) qui elle va, appeler pour chaque produit ébauche l'API Expérience, et va récupérer les informations complémentaires.

Pour faire cela je boucle sur ma variable 'autreProduit' ce qui va permettre de changer 'idProduitEbauche' dans la variable params (ligne 70)

Une fois que j'ai réalisé cela je prépare mes appels en les mettant dans un tableau (ligne 73). La méthode qui sera appelée existe déjà dans le projet, j'ai donc juste du l'appeler :

```
public getInfoAPIExperience(params: Record<string, any>): Observable<ProduitTarifExperience> {
    const url = `/soprano-produit-api/v2/contralto/ebauches-grilles/experience/produits-tarifs`;
    Object.keys(params).forEach(key => params[key] == null && delete params[key]);
    const queryParams = new HttpParams({fromObject: params});
    return this.http.get<any>(url, {params: queryParams}).pipe(catchError(this.errorHandler));
}
```

Puis je prépare aussi le traitement qui sera fait pour chaque appel. Ici il devra me mettre dans un tableau, un objet Produit Ebauche ainsi qu'un objet PorduitTarifExpérience (qui à ce moment n'est pas créé d'où le 'new').

Pourquoi je fais ça ?

Tout simplement parce que j'ai besoin de faire correspondre chaque information des produit-ébauches des 2 appels. Je mets en place de quoi lever une exception grâce au 'catchError()'

Une fois que cela est réalisé je mets mon tableau d'appel préparé dans la méthode 'zip'(ligne 82). C'est ici que rxjs intervient. La méthode zip permet de combiner des appels en les réalisant dans l'ordre. Tant qu'un appel n'est pas terminé le suivant ne sera pas lancé. Cela permet de récupérer toutes les informations des produit-ébauches du second appel dans le même ordre que celles du premier appel.
Le zip retourne un tableau d'observable que l'on peut exploiter comme l'on veut ensuite (dans mon cas j'ai créé un dictionnaire). Pour envoyer mes requêtes j'utilise la méthode 'subscribe' sur la méthode 'zip()'.

Du côte du front j'ai juste besoin de lui spécifier à quel objet je veux accéder dans mon dictionnaire créée auparavant :

'exp' quand j'ai besoin d'accéder au retour du second appel :

```
3>Raison sociale : {{values.exp.raisonSociale}} - Proposition : {{values.exp.idProposition}}
```

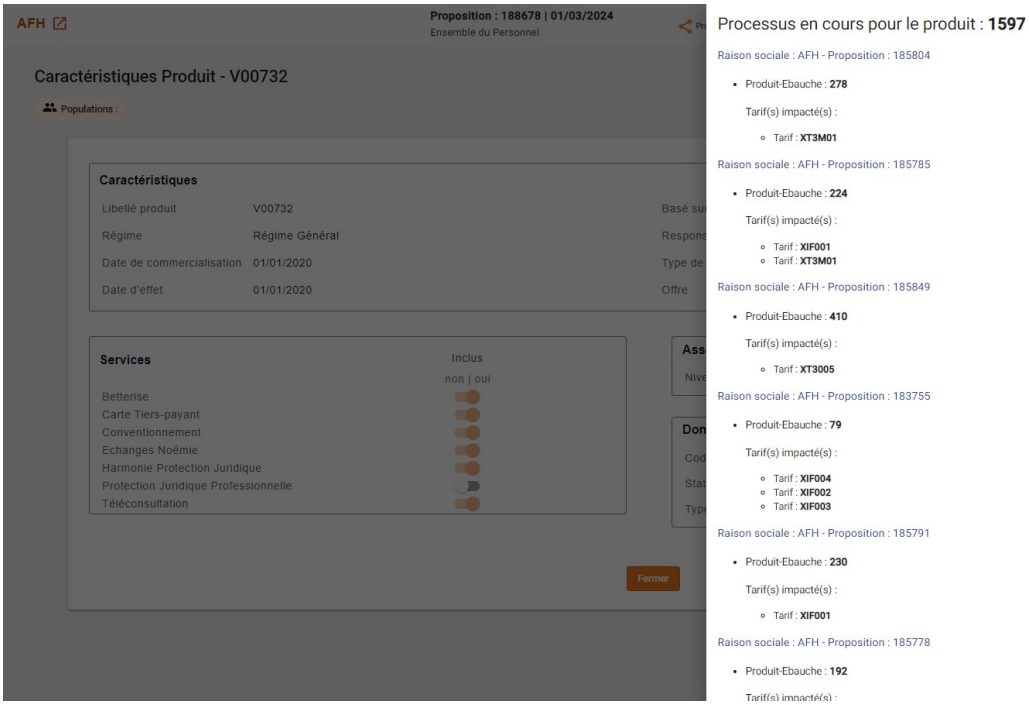
'produit' quand j'ai besoin d'accéder au premier appel :

```
<b>{{values.produit.id}}
```

Ici le 'values' est la variable qui me permet de parcourir mon dictionnaire:

```
*ngFor="let values of infosContralto">
```

Résultat final:

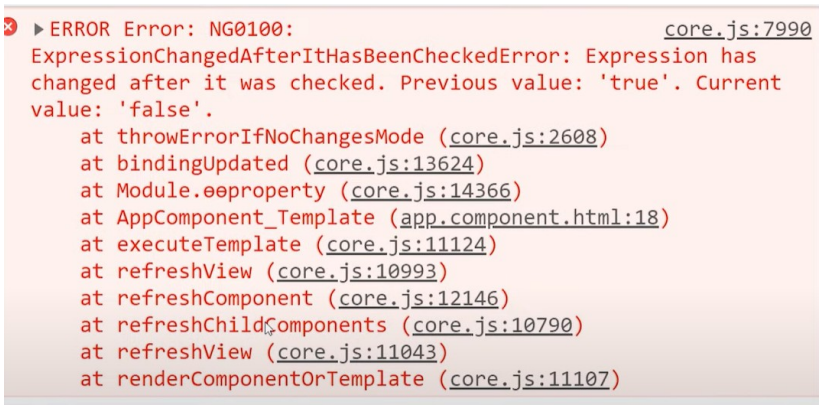


On a bien pour chaque produit, ses grilles ainsi que son numéro de proposition avec sa raison sociale.

L'esthétique de la page sera sûrement changée d'ici là.

Problème rencontré :

J'ai tout d'abord voulu créer une collection de type 'Map' pour stocker les retours d'appel de mes différents objets. Mais malheureusement une erreur est apparue :



Cette erreur m'était renvoyée car tout simplement, pendant qu'Angular essayer accéder à ma collection pour afficher les différentes informations dans le Front, de mon côté, je continuais à réaliser les appels.

Je modifiais donc tous les index dans ma collection puisque au fur et à mesure de mes appels, ma collection se remplissait, mais Angular pendant ce temps accédait lui aussi à ma collection pour afficher les informations.

J'ai corrigé cette erreur en créant un autre type de collection.

V - Conclusion de mon stage

A – Ce qu’il m’a apporté au niveau compétence technique

Ce stage m'a permis d’acquérir de nombreuses compétences en développement informatique. Tout d’abord d’un point de vue programmation. En effet grâce à ce stage, j’ai pu développer de nouvelles compétences dans les Frameworks tels que Angular et SpringBoot. Ces deux Frameworks très puissant sont utilisés dans beaucoup d’entreprises ce qui est très bon avantage. J’ai découvert un nouveau langage appelé le TypeScript qui est aussi un langage très puissant et qui permet de faire beaucoup de choses.

Dans un second temps d’un point de vue organisation, en effet j’ai pu découvrir qu’est-ce qu’un gestionnaire de source et comment m’en servir. J’ai pu découvrir des nouveaux outils de gestions de projets très puissants et des méthodes de débogages intéressantes.

B – Ce qu’il m’a apporté au niveau relationnel

En ce qui concerne le volet relationnel, ce stage m'a offert l'occasion de découvrir le milieu professionnel et découvrir les différents métiers de l’informatique que je ne connaissais pas. J'ai pu partager mes idées, résoudre des problèmes et bénéficier de l'expertise de mon maître de stage. Cette expérience a renforcé ma capacité à communiquer, et à m'adapter à différents méthodes de travail.

C – Conclusion

En conclusion, ce stage en développement informatique avec ces différents langages de programmation a été une opportunité pour améliorer mes compétences techniques, renforcer mes aptitudes relationnelles et approfondir ma vision du domaine de la programmation. Je suis reconnaissant d'avoir eu cette opportunité et je suis convaincu que les connaissances et l'expérience acquises pendant ce stage me seront bénéfiques tout au long de ma carrière dans le domaine de l'informatique.