



Certification pour le titre de concepteur- développeur d'applications

Projet CEPP

Amon Amadis

Tuteur : David Marinello

25/01/21

Remerciements

J'aimerais remercier les personnes qui m'ont accompagné dans cette aventure et leur consacrer quelques modestes lignes. Sans ces personnes, la concrétisation de mon objectif n'aurait pas été possible et pour cela je leur suis extrêmement reconnaissant.

Tout d'abord, dans le cadre de l'Afpa, j'aimerais remercier Patricia Gallais qui a été la première personne à me donner cette chance de reconversion vers l'informatique et qui n'a pas cessé de nous aider à atteindre nos objectifs en gardant le groupe de l'Afpa soudé. Ensuite, j'exprime ma gratitude envers tous nos formateurs qui ont donnés d'eux-mêmes pour nous enseigner les bases et les subtilités du développement. Ce ne sont autres que Philippe Viguier, Bruno Collin, Djamila Causse et Pascal Dangu, le catcheur mystérieux ! Enfin, j'ai une pensée émue pour mes compagnons d'aventure, avec qui j'ai partagé de beaux moments que ce soit dans le travail ou dans le divertissement. Des prises de tête terribles sur quelques lignes de code mais aussi de grands moments de complicité et d'amusement ont scellé notre amitié et je ne saurais trop les remercier. Merci à Charlène, Joan, Kevin, Alexandre, Théo, Guillaume, Clémentine, Ahmed, Frédérik, Vincent et Alix.

Dans la sphère Atos, je suis reconnaissant envers Lise Romans qui m'a permis d'intégrer Atos et qui a cru en moi en gardant un œil bienveillant tout au long de ce parcours. De même pour Nathalie Gres qui nous a notamment accompagné pendant nos réunions du Culture-book et apporté des conseils utiles. Ensuite j'aimerais remercier Yann Collin, qui m'a accueilli dans son agence et apporté un soutien pour le démarrage dans l'entreprise. Merci aussi à David Marinello, mon tuteur d'alternance, qui m'a intégré dans l'entreprise avec la plus grande sympathie. Je remercie également mes co-équipiers et compagnons de bureau, Lamiae, Nabil, Yan, Amine et Mehdi qui m'ont apporté leur savoir mais aussi leur amitié. Et enfin, je clôture ce témoignage par la personne qui a eu les nerfs de répondre à toute mes interrogations quelles qu'elles soient et qui a toujours pris du temps pour m'aider à avancer dans mon travail et à comprendre de la plus simple à la plus complexe des tâches, Aymeric. Je te remercie sincèrement pour toute la pédagogie dont tu as fait preuve et avec quelle simplicité et humanité tu as su m'intégrer à l'équipe.

Merci à tous !

Table des matières

Introduction.....	4
Compétences du référentiel qui sont couvertes par le projet	5
Abstract	6
Les acteurs du système	7
Présentation du système	8
Gestion de projet	10
Spécifications techniques.....	12
Spécifications fonctionnelles.....	16
Tâche n°1	16
Tâche n°2	17
Tâche n°3	18
Tâche n°4	18
Tâche n°5	19
Tâche n°6	19
Réalisations	21
Activité-type 1 : Développer des composants d’interface	21
Exemple n°1 : Modification d’un obligé	21
Exemple n°2 : Inspecteur – Export obligés – Ajout contacts	27
Activité-type 2 : Développer la persistance des données	29
Exemple n°1 : Affichage des messages.....	29
Activité-type 3 : Développer une application multicouche	30
Exemple n°1 : Modification d’une fiche-action	30
Exemple n°2 : Statistiques – Inspections à contrôler	33
Exemple n°3 : Durée d’affichage pour les inspections	37
Description d’une situation de travail avec recherche	40
Bilan personnel.....	42
Tables des illustrations.....	43

Introduction

J'ai eu la chance d'effectuer mon contrat de professionnalisation chez Atos, une des 10 plus grandes ESN (Entreprise de Services du Numérique) sur le plan mondial. Ce dossier technique est le fruit du travail fourni pendant près de 5 mois. J'ai pu apprendre le métier de concepteur-développeur d'applications grâce à un projet de grande envergure et à une équipe formidable. Le projet en question est appelé CEPP, acronyme de Certificats d'Economie de Produits Phytosanitaires. Le dossier a pour but principal d'exposer les tâches que j'ai pu effectuées au sein de l'entreprise pour valider les différentes compétences nécessaires à l'obtention du titre.

J'expose donc en premier lieu les différentes compétences visées dans ce dossier. Un court résumé en anglais suit ensuite pour expliciter brièvement le contexte du projet et de l'application. C'est également ici que je parle des différentes technologies utilisées au cours de cette alternance. Ensuite, je plonge plus profondément dans le contexte de l'application pour décrire précisément le système et ses acteurs. S'ensuit une description de la gestion de projet, soit le descriptif du fonctionnement de l'équipe à laquelle j'appartenais pour ce projet. Puis viennent les spécifications du projet autant techniques que fonctionnelles, essentielles pour comprendre l'objectif des tâches décrites à posteriori. Justement, c'est le contenu de la partie suivante : les réalisations que j'ai jugé importantes et non redondantes dans la validation des compétences nécessaires à la certification. Dans cette partie, j'ai divisé en trois grands axes les réalisations : développer des composants d'interface, développer la persistance des données et développer une application multicouche. A tout cela s'ajoute la description d'une situation de travail qui m'a donné du fil à retordre. Sont inclus des termes de recherche en anglais sur internet. Pour terminer, ce dossier est clôturé par un bilan personnel où j'expose entre autres ce que m'a apporté cette expérience.

Compétences du référentiel qui sont couvertes par le projet

Développer des composants d'interface utilisateur

Développer une interface utilisateur de type desktop

Développer des composants d'accès aux données

Développer la partie front-end d'une interface utilisateur

Développer la partie back-end d'une interface utilisateur

Développer la persistance des données

Développer des composants dans le langage d'une base de données

Développer une application multicouche

Développer des composants métier

Construire une application organisée en couches

Préparer et exécuter les plans de tests d'une application

Figure 1 : compétences traitées dans ce dossier

Abstract

Atos is a French IT service and consulting company which offers expertise in Cloud, Big Data, Cybersecurity and Business Applications. It employs 120 000 people in 73 different countries with a 12 billion euros revenue.

The client I worked for is the equivalent of Department of Agriculture (*Ministère de l'Agriculture, de l'Agroalimentaire et de la Forêt*) which is the administration in charge of regulation and policy for agriculture, food and forestry.

I took part in the CEPP (*Certificat d'économie de produits phytopharmaceutiques*) project which helps to spread good economical uses of phytopharmaceutical products for farmers while ensuring to maintain economic performance. It aims to strengthen the role of phytopharmaceutical products distributors without additional taxes for farmers. The main goal is to reduce the use of phytopharmaceutical products when an alternative solution exists to ultimately lower their impact on human health, environment and biodiversity.

This project takes form in a web application with Angular as main front-end framework and Java for back-end on which is added the Spring framework (Spring Boot, Spring Data, Spring Security). The database used is Postgres.

The web application allows the management of certificates which are proofs that a distributor sold a replacement for a phytopharmaceutical product. The Department of Agriculture assigns to each of these distributors a quantitative value to match at the end of the year. These certificates can also be sold or traded.

Les acteurs du système

Dans cette partie, je décris les différents acteurs du système qui entrent en jeu pour la compréhension des tâches que j'ai pu réaliser.

Les **obligés** sont les distributeurs de produits phytopharmaceutiques à des professionnels (agriculteurs) et les agriculteurs redevables de la redevance pour pollution diffuses lorsqu'ils achètent les produits à l'étranger.

Les **collaborateurs** sont les personnes physiques, les utilisateurs qui vont représenter les obligés. Dans l'application, ils peuvent avoir un seul mandat, donc agir au nom d'un seul obligé ou être multi-mandats. Ils pourront donc changer d'obligé en fonction de leur besoin.

Les **Teneurs de registre** sont les agents du MAAF (Ministère de l'Agriculture, de l'Agroalimentaire et de la Forêt) qui effectuent le paramétrage des actions via les fiches-actions et qui peuvent être amenés à réaliser des contrôles d'éligibilité des actions présentées et sont garants de la délivrance des certificats.

Les **inspecteur SRAL** sont des agents du MAAF qui effectuent des contrôles sur place (dans l'entreprise concernée). Ils utilisent l'application pour récupérer les informations sur les actions, les Obligés.

Les **Administrateurs** sont des Teneurs de registre avec des droits en plus, notamment la gestion des utilisateurs, mais aussi des Obligés. C'est lui qui crée, modifie ou supprime les différents acteurs.

Présentation du système

L'application développée par Atos a pour objectif de gérer des Certificats d'Economies de Produits Phytopharmaceutiques, plus tard référés en tant que « CEPP ».

Ces CEPP ont pour objectif d'obliger les distributeurs à inciter les agriculteurs, ou les agriculteurs eux même s'ils achètent ces produits à l'étranger, à mettre en œuvre des actions permettant de réduire l'utilisation de produits phytopharmaceutiques.

Les produits de traitement des semences, les produits de biocontrôle et les usages non agricoles sont exclus.

L'application

- doit être un outil simple, fiable, sécurisé permettant la déclaration des actions pour l'obtention de CEPP par les obligés.
- doit permettre la gestion, le contrôle, l'administration du système, la fabrication des fiches-actions par les teneurs de registre et les inspecteurs.

Le **NODU** est le nombre de doses unités de produits phytopharmaceutiques permettant le calcul des CEPP. Ils sont générés via la déclaration d'action.

L'unité de compte des actions est le CEPP.

Les **obligations** pour les obligés sont exprimées en CEPP. Cette obligation par obligé est calculée à partir de la moyenne des ventes exprimées en NODU des 5 dernières années (≠ calculs si pas 5 années). L'obligation portera sur 20% de cette moyenne.

Les obligés mettent en œuvre des **actions** permettant de réduire les produits phytopharmaceutiques. Ces actions doivent être conformes à des **actions standardisées**. L'action standardisée ou **fiche-action** définit la nature de l'action, les pièces justificatives de la réalisation de l'action, la valeur annuelle des actions correspondant exprimées en NODU et CEPP ainsi que le nombre d'années d'impact de l'action.

Pour chaque fiche-action, une nomenclature permet le calcul en NODU. Le triplet fiche action/produit ou matériel ou conseil/quantité permet le calcul.

Un CEPP est délivré, cédé, annulé ou redimensionné.

DOSSIER TECHNIQUE

Fonctionnalités de l'application :

- Les teneurs de registre définissent des actions standardisées ou fiche-actions. Une fiche-action permet d'obtenir un nombre de NODU.
- Les obligés choisissent parmi ces fiches-actions celles qui lui sont applicables. En en sélectionnant une, l'obligé remplit le formulaire correspondant et peut simuler ou calculer le nombre de NODU qu'il pourra obtenir. Il peut enregistrer la déclaration si elle lui convient puis la déclarer (ou publier ou valider) s'il veut la mettre en œuvre.
- Cette déclaration est ensuite validée automatiquement ou par le teneur de registre.
- Dès validation l'obligé obtient le CEPP.

Pour gérer le système :

- L'application doit gérer les utilisateurs par le biais d'administrateurs propres à chaque organisme
- L'application doit permettre le contrôle des déclarations lors de la validation et aussi en cours d'expérimentation par les Teneurs de registre ou les Inspecteurs délégués.

D'autres fonctionnalités permettent :

- D'imprimer les certificats
- De gérer les actualités
- La communication entre les Obligés et les Teneurs de registre et/ou les administrateurs
- D'éditer des états de situation et des statistiques
- L'échange de CEPP entre obligé/obligé
- De réaliser des inspections

Gestion de projet

L'organisation du projet a initialement été proposée en suivant la méthode Agile. Cependant, celle-ci demandant une plus forte implication du client dans le processus de développement, la gestion de projet s'est finalement déroulée en cycle en V. Voici une illustration de celui-ci :

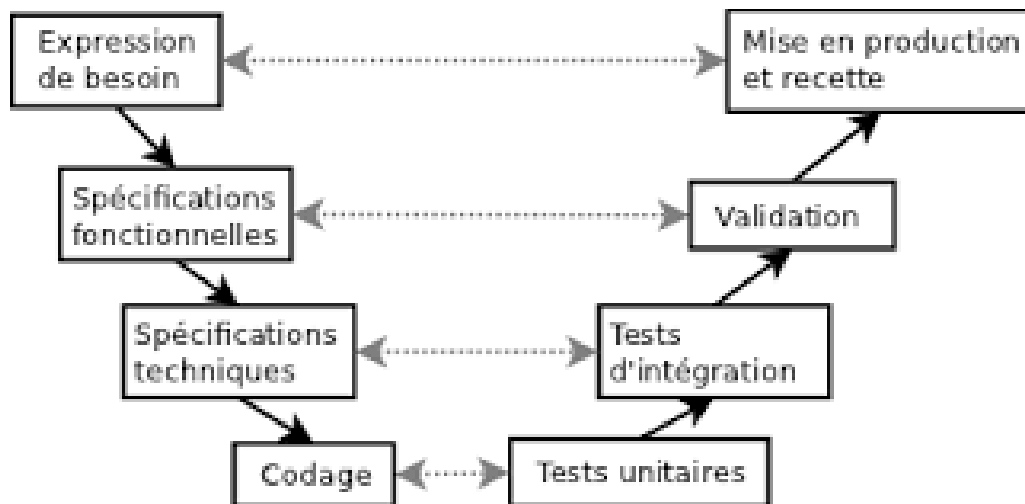


Figure 2 : schéma du cycle en V

Le flux descendant permet de bien détailler le produit jusqu'à sa réalisation. Cela comprend l'expression des besoins, les spécifications fonctionnelles et les spécifications techniques. A l'issue de ce flux, on retrouve la programmation qui sera ensuite suivie du flux montant. C'est dans cette partie que tous les tests et la partie validation du client va rentrer en jeu.

En ce qui concerne l'environnement humain, l'équipe est composée de 3 à 4 personnes, soit un chef de projet et 2 ou 3 développeurs, sur la durée de mon alternance.

Le suivi et le planning sont menés à bien grâce aux outils suivants :

- Jira/Confluence pour le suivi de projet avec le client, cela lui permet d'interagir directement avec le chef de projet ou même les développeurs sur les différentes tâches à accomplir.
- Microsoft Planner pour le suivi en interne de l'équipe, principal outil d'organisation entre développeurs et chef de projets (ou avec d'autres développeurs).

Les horaires de travail étaient des horaires classiques de type « bureau ». Quand au lieu de travail, ce fut dans un premier temps dans les locaux d'Atos puis en télétravail dans le cadre des restrictions gouvernementales appliquées pour la crise du Covid-19.

DOSSIER TECHNIQUE

Des normes de codages ont également été appliquées pour les besoins du Ministère de l'Agriculture et de l'Alimentation notamment pour le nommage des entités, méthodes ou encore fichiers.

Spécifications techniques

Les conditions sanitaires particulières de cette année demandent un environnement technique tout aussi particulier. Notamment, le besoin d'un VPN Atos pour accéder à l'environnement de développement de l'application. C'est un besoin spécifique au télétravail car sur site, le réseau Atos est disponible sur tous les postes configurés à cet effet.

Une fois la connexion au VPN Atos effectuée, il est nécessaire de se connecter à une machine virtuelle pour accéder à l'environnement et aux applications du Ministère de l'Agriculture et de l'Alimentation sont mis en place. C'est dû aux normes de sécurité imposés par le Ministère, en effet, chaque développeur doit être habilité pour pouvoir accéder aux données et apporter des modifications à l'application.

Dans la partie gestion de projet, j'ai parlé de Jira et de Confluence qui servaient d'intermédiaire entre le client et Atos, les Jiras en plus de servir de communication entre les deux partis, sont utilisés aussi en tant que sources de spécifications.

Pour le système de contrôle de versions, j'ai utilisé Subversion pour la mise en commun des différents développements. Un système assez simple dans la mesure où les membres de l'équipe de développement travaillaient tous sur la même branche. Bien sûr, avant de commit un changement sur le code, il est nécessaire de faire une mise à jour du code pour éventuellement gérer les conflits en cas de modification sur les mêmes fichiers. Ce système sera bientôt modifié sur toutes les applications du Ministère. En effet, il est prévu de faire une migration vers Git. Les principales raisons de ce changement sont d'une part de permettre une meilleure gestion des fichiers disponibles aux développeurs et ceux qui doivent rester à usage personnel des agents du Ministère. D'autre part, la mise en place de Git permettrait de faire l'usage de repos distants avec des sites comme GitHub/GitLab qui ont des fonctionnalités pour mieux gérer le projet. Par exemple, en cas de nouvelle arrivée d'une personne dans le projet, il pourrait tout avoir à disposition tout de suite avec une interface plus moderne. Cela permettrait également de déployer de manière automatique, contrairement à ce qui est en place aujourd'hui où tous les déploiements doivent se faire en ligne de commandes.

Une autre particularité de ce projet est qu'il doit être utilisé avec Apys. La boîte à Outils Apys est un cadre de développement permettant de construire des applications de type Rest. Cette boîte à outils repose sur le framework Spring.

L'objectif du produit Apys est de répondre aux besoins suivants :

- Faciliter le travail du développeur sur une stack Java connue

DOSSIER TECHNIQUE

- Faciliter l'intégration des spécificités du MAA dans les développements logiciels
- Cadrer les développement Java en imposant une architecture logicielle
- Faciliter le déploiement des applications au centre de production
- Mutualiser des patterns/comportements techniques dans des bibliothèques partagées aux différents projets

Pour atteindre ces objectifs, la stratégie du produit est :

- d'utiliser un framework Java reconnu par les développeurs : Spring
- de faire des connecteurs/surcouches/librairies légères permettant de contextualiser
- de fournir un cadre de développement
- de forcer le livrable et la manière de déployer

Voici une représentation graphique ce que peut apporter Apys :

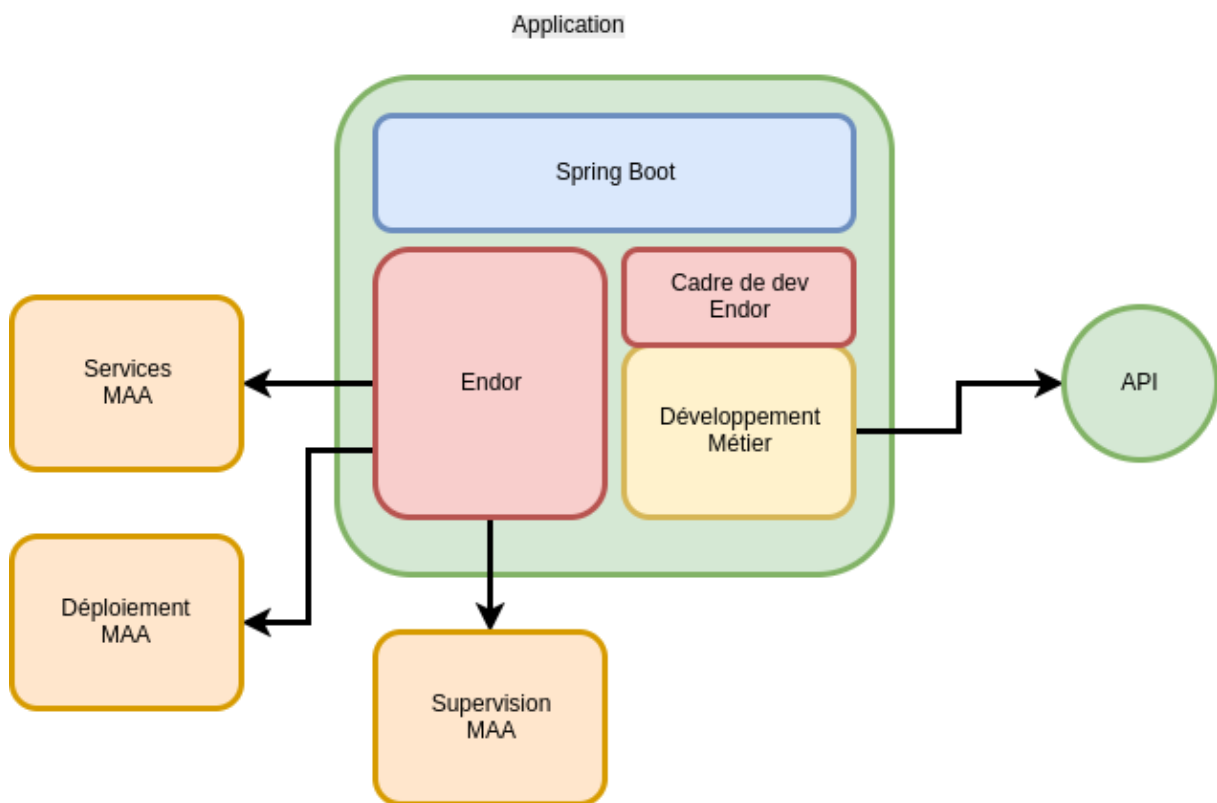
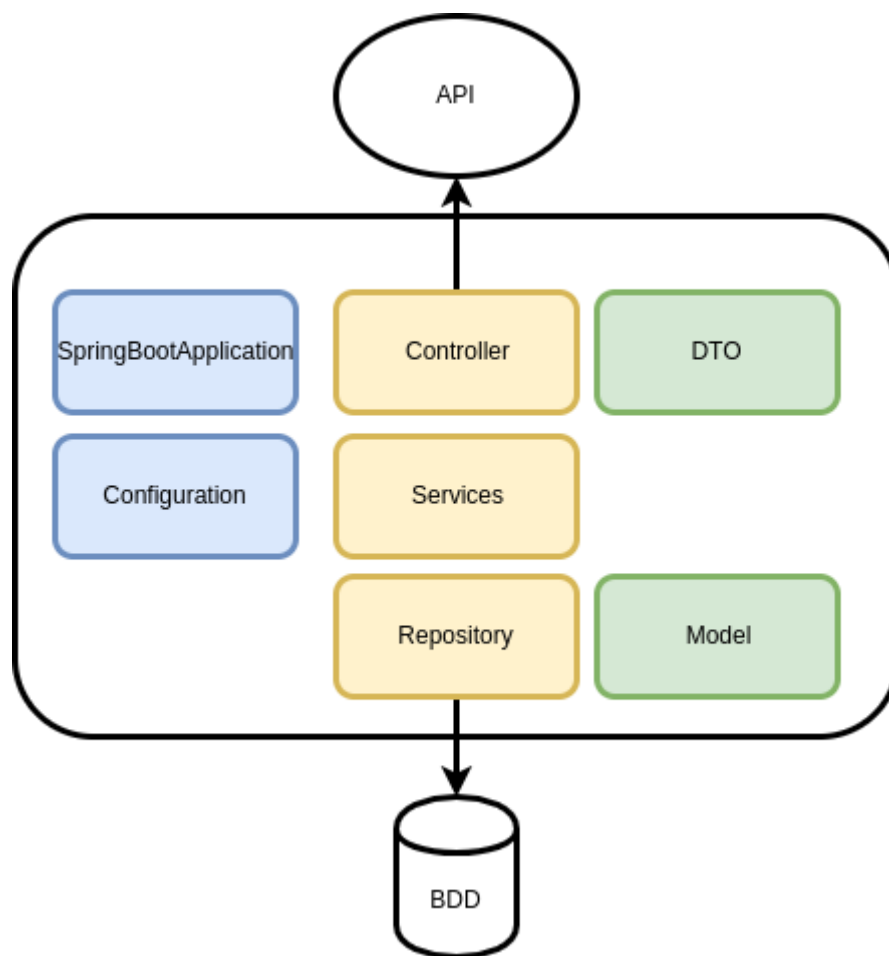


Figure 3 : schématisation de ce qu'apporte l'environnement Apys

Ici le schéma se réfère à Endor : c'est l'ancien nom d'Apys, mais le principe reste le même.

L'architecture dans les applications qui utilisent Apys est organisée en couche comme le schéma ci-après :

DOSSIER TECHNIQUE



De manière assez classique, dans la Restapp, on retrouve les couches :

- des Controller qui fait le lien entre l'API et le reste de l'application en back-end. Il reçoit les requêtes http et retourne des DTO (Data Transfer Object).
- des Services font la communication entre les Controller et les Repository.
- des Repository qui permettent d'interroger la base de données avec des requêtes SQL et qui utilisent des Model en pour transposer les entités de la base en objets Java qu'on pourra manipuler par la suite

Il est intéressant également de noter la particularité du système d'authentification des applications du Ministère. En effet celles-ci fonctionnent toutes de la même manière dans le cadre de l'utilisation d'Apys. L'authentification se déroule en deux temps :

- vérification d'un Ticket CAS (aussi appelé EAP pour le Ministère)
- vérification des droits d'accès à l'application

Ci-après la cinématique d'authentification :

DOSSIER TECHNIQUE

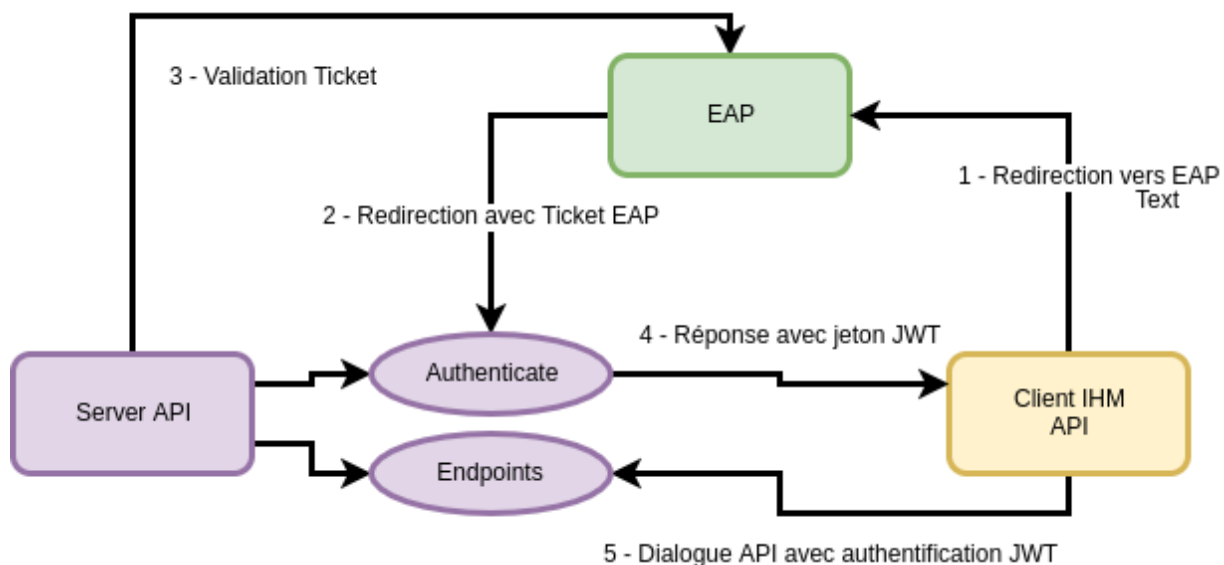


Figure 4 : schéma de la cinématique d'authentification sur l'environnement Apys

- 1 - Redirection du client vers le serveur CAS (EAP)
- 2 - Redirection vers l'endpoint d'authentification : /auth/token
- 3 - Validation du Ticket CAS par le serveur via un client CAS
- 4 - Réponse au format json de l'API retournant un token JWT d'accès access_token
- 5 - Dialogue de l'API avec le jeton JWT d'accès access_token pour valider chaque appel aux endpoints de l'API et identifier le client tant que celui-ci n'est pas expiré.

Spécifications fonctionnelles

Dans cette partie, je décris les besoins clients des différentes tâches que j'ai utilisées en tant qu'exemple dans la suite de ce dossier.

Tâche n°1

Connecté en tant qu'administrateur, j'accède au détail d'un obligé.

L'évolution consiste à ajouter une fonctionnalité dans la partie haute de l'écran : Modifier obligé.

Ce nouveau bouton n'est accessible qu'à l'administrateur et lui permet de modifier les informations suivantes :

- Raison sociale
- Numéro SIREN
- Numéro SIRET
- Valeurs des obligations (pour les années correspondant à la campagne en cours ou postérieures)

Lorsque l'utilisateur clique sur le bouton 'Modifier obligé', une pop-up s'ouvre afin de présenter les données en modification.

Les 3 premiers champs sont pré-remplis avec les valeurs lues dans la table T_CEPP_OBLIGE. Les valeurs sont modifiables :

- Raison sociale : Saisie libre limitée à 100 caractères
- N° SIREN : Numérique sur 9 caractères
- N° SIRET : Numérique sur 14 caractères

La modification des obligations de l'obligé se présente sous la forme d'un tableau constitué de 3 colonnes. Les campagnes affichées sont la campagne en cours et postérieures présentes dans la table T_CEPP_LIGNE_DE_COMPTE pour cet obligé et ce type d'opération. Le champ campagne est non modifiable alors que le champ 'Nombre d'obligations' qui reprend T_CEPP_LIGNE_DE_COMPTE.valeur est modifiable (seuls les caractères numériques sont autorisés).

DOSSIER TECHNIQUE

Dans l'entête de la dernière colonne se trouve l'action '+ Ajouter des obligations sur une campagne'. Cette action a pour effet de créer une nouvelle ligne dans le tableau avec la combo-box (si plusieurs valeurs) des campagnes réactualisée (Les valeurs possibles sont la campagne en cours + les 2 années suivantes mais seules les campagnes non encore affichées sont présentées.).

Dans le cas où la campagne en cours et les 2 suivantes ont déjà des obligations définies, le bouton '+ Ajouter des obligations sur une campagne' n'est pas actif.

La colonne contient ensuite l'action de suppression propre à chaque ligne. Cette action met la valeur de l'obligation à 0 pour l'obligé et la campagne sélectionnée.

Ce petit tableau a déjà été développé dans le cadre de la création d'un obligé.

Le fait de cliquer sur le bouton 'Valider' enregistre les informations en base de données :

- Raison sociale : T_CEPP_OBLIGE.raison_sociale
- Numéro SIREN : T_CEPP_OBLIGE.numero_siren
- Numéro SIRET : T_CEPP_OBLIGE.numero_siret
- Valeurs des obligations : T_CEPP_LIGNE_DE_COMPTE.valeur pour T_CEPP_LIGNE_DE_COMPTE.type_operation_id = 2 et T_CEPP_LIGNE_DE_COMPTE.oblige_source_id = T_CEPP_OBLIGE.oblige_id et T_CEPP_LIGNE_DE_COMPTE.campagne_id= campagne concernée.

Tâche n°2

Connecté en inspecteur, la page d'accueil affiche l'écran de recherche des obligés.

Sur la partie basse de l'écran, il est possible d'exporter les données après les avoir sélectionnées.

L'évolution consiste à ajouter les colonnes suivantes dans l'export (entre la colonne 'Raison sociale' et la colonne 'Commune') :

- 6 colonnes pour les informations concernant les collaborateurs de l'obligé sous la forme suivante :
 - o Nom/prénom : T_CEPP_CREDENTIALS.last_name+first_name
 - o Telephone : T_CEPP_CREDENTIALE.phone_number
 - o Nom/prénom : T_CEPP_CREDENTIALS.last_name+first_name
 - o Telephone : T_CEPP_CREDENTIALE.phone_number
 - o Nom/prénom : T_CEPP_CREDENTIALS.last_name+first_name
 - o Telephone : T_CEPP_CREDENTIALE.phone_number
- Code postal : T_CEPP_ADRESSE.code_postal avec T_CEPP_OBLIGE.adresse_id=T_CEPP_ADRESSE.adresse_id

DOSSIER TECHNIQUE

Les données des collaborateurs sont récupérées en tenant compte des filtres suivants :

- Collaborateurs de l'obligé : T_CEPP_MANDAT.oblige_id=obligé sélectionné et T_CEPP_MANDAT.user_id=T_CEPP_CREDENTIALS.user_id
- Collaborateurs ayant un mandat actif : T_CEPP_MANDAT.expiry_date is null et T_CEPP_MANDAT.user_id=T_CEPP_CREDENTIALS.user_id
- Collaborateurs actifs : T_CEPP_CREDENTIALS.expiry_date is null

Les collaborateurs seront triés selon leur type de mandat (collaborateurs principaux avant les collaborateurs secondaires) :

- T_CEPP_MANDAT.type_mandat_id = 1 avant T_CEPP_MANDAT.type_mandat_id = 2

A l'intérieur de chacune des sous-listes (principal et secondaire), les collaborateurs seront triés selon leur date de création de leur mandat (du plus récent au plus ancien) (T_CEPP_MANDAT.beginning_date).

Ce sont les coordonnées des 3 premiers collaborateurs qui sont affichés dans le tableau.

Tâche n°3

Connecté en tant qu'obligé multi-mandats, des messages d'information s'affichent à la connexion. Les modifications suivantes sont à apporter :

- La durée d'affichage est de 30 secondes : Passer à 20 secondes.
- Le déclenchement de l'affichage des messages (conversation+acquisition) ne doit se faire qu'au moment de la connexion ou au changement de mandat (et non à chaque chargement du tableau de bord). Cela doit permettre de résoudre le fait qu'un message déjà affiché, ne s'affiche à nouveau.
- Quand on change de mandat, fermer les messages du mandat précédent (sinon ils se cumulent ancien mandat + nouveau mandat). L'affichage des messages doit correspondre uniquement au mandat en cours.

Tâche n°4

Connecté en Teneur de registre ou Administrateur, je clique sur le menu Statistiques.

Cette évolution va permettre de créer une nouvelle entrée dans l'écran statistiques positionnée en 5ème position.

Caractéristiques de la statistique :

- Titre : "Liste des actions non conformes par obligé"
- Texte : "Campagne, Raison sociale, SIREN, Référence, Numéro de l'action NC, Nombre d'années, Numéro de la fiche-action, Nombre de CEPP total de l'action NC pour la

DOSSIER TECHNIQUE

campagne en cours, Nombre de CEPP injustifiés de l'action NC pour la campagne en cours"

- Export : les données sont issues de la vue V_CEPP_STATISTIQUES_INSPECTION_REFERENCES. Le nom du fichier généré est de la forme 'stats-actions-NC-oblige_AAAA-MM-JJ.CSV' avec AAAA-MM-JJ correspondant à la date du jour.

Tâche n°5

Connecté en Teneur de registre ou Administrateur, je clique sur le menu Statistiques.

Cette évolution va permettre de créer une nouvelle entrée dans l'écran statistiques positionnée en 10ème position.

Caractéristiques de la statistique :

- Titre : "Liste des actions à contrôler"
- Texte : "Date de déclaration, Campagne, Numéro de l'action, Raison sociale, SIREN, Nombre d'obligations, Nombre de CEPP de l'action, Nombre de CEPP, Numéro de la fiche-action, Nom, Unité, Quantité, Numéro AMM, Commentaire interne MAA, Observations, Proposition cellule, Décision"
- Export : les données sont issues de la vue V_CEPP_STATISTIQUES_ACTIONS. Seules les actions dans l'état 'A contrôler' sont exportées (statut_action_id=3). Les 2 dernières colonnes 'Proposition cellule' et 'Décision' ne sont pas présentes dans la vue. Ce sont des colonnes qui doivent être créées au moment de l'export. Seul le titre est renseigné. Ces colonnes seront ensuite utilisées pour écrire des informations de façon 'manuelle'. Le nom du fichier généré est de la forme 'stats-actions-a-controler_AAAA-MM-JJ.CSV' avec AAAA-MM-JJ correspondant à la date du jour.

Tâche n°6

Connecté en Teneur de registre ou Administrateur, je peux visualiser la liste des inspections.

Cette liste contient les inspections 'récentes' (Le tableau affiche les inspections qui datent de moins d'une période indiquée dans le paramètre P_DureeAffichageInspection).

La valeur initiale de ce paramètre était de 1 mois et partait de la date de l'inspection.

L'objet de cette évolution est de modifier, pour les inspections à l'état 'Validé' :

- La durée : Passer de 1 à 2 mois
- La date de début : Le délai doit courir à compter de la date de validation de l'inspection (T_CEPP_INSPECTION.date_validation) et non la date de l'inspection (T_CEPP_INSPECTION.date_inspection).

DOSSIER TECHNIQUE

Pour les inspections à l'état 'Enregistré', celles-ci doivent être toujours visibles, sans tenir compte d'un quelconque délai (elles n'ont pas vocation à rester dans cet état).

Réalisations

Activité-type 1 : Développer des composants d'interface

Exemple n°1 : Modification d'un obligé

Compétences couvertes dans cet exemple

Développer une interface utilisateur de type desktop

Développer la partie front-end d'une interface utilisateur web

Tâches ou opérations effectuées

Utilisation d'une librairie pour gérer une modale :

La modale affichée ressemble à la capture d'écran suivante :

The screenshot shows a web modal titled "Modifier un obligé" with a close button (X) in the top right corner. The modal contains three input fields for identification: "Raison sociale" (containing "EMC2"), "Numéro SIREN" (containing "775616626"), and "Numéro SIRET" (containing "77561662600187"). Below these fields is a section titled "Obligations" which contains a table. The table has two columns: "Campagne" and "Nombre d'obligations". The first row shows "2020" in the "Campagne" column and an empty input field in the "Nombre d'obligations" column. To the right of the table is a button labeled "+ Ajouter des obligations sur une campagne". Below the table, a red error message states "Le nombre d'obligations doit être renseigné". At the bottom right of the modal are two buttons: "Annuler" and "Valider".

Campagne	Nombre d'obligations
2020	<input type="text"/>

Figure 5 : capture d'écran de la concrétisation de la tâche "Modification d'un obligé"

Elle est appelée dans le fichier html du composant parent au composant qu'on cherche à créer :

DOSSIER TECHNIQUE

```
<ngx-smart-modal
  #edit identifier="edit"
  [customClass]=" 'custom-modal' "
  [dismissable]="false"
  [closable]="false">
  <app-oblige-edit
    (modif)="editOblige($event)"
    (closed)="onCloseEdit()"
    [oblige]="oblige"
    [obligationsByOblige]="obligationsByOblige">
  </app-oblige-edit>
</ngx-smart-modal>
```

Figure 6 : exemple d'utilisation de la librairie "ngx-smart-modal" pour la création d'une modale dans un fichier html

La librairie « ngx-smart-modal » possède un moyen de l'identifier et d'y faire référence, ici l'attribut « identifier ». La librairie permet d'injecter notre propre composant, ici « app-oblige-edit » étant son selector. Je passe ensuite des objets grâce aux crochets, ici par exemple, « [oblige] » est l'attribut du composant nouvellement créé tandis que «oblige » est l'attribut du composant mère.

Création d'un composant Angular :

Pour créer un nouveau composant Angular, il suffit de taper la commande : `ng g c dossier-quelconque/nom-du-composant`. Ici `g` est l'alias de « generate » et `c` est l'alias de « component ».

Attention, une fois le composant créé, il faut aussi l'affilier à un module, sans quoi il ne sera pas fonctionnel au sein de l'application. Dans notre cas, l'ajout de ce composant a fait l'objet d'une réorganisation des modules de l'application (`ng g module nom-module`).

Dans le module, il faut déclarer le composant dans « declarations » et l'exporter dans « exports ».

Si le nouveau module doit être utilisé dans un autre module, il faut l'ajouter dans les « imports » de cet autre module.

Création d'un bouton, attachement d'un évènement et conditionnement de l'accès :

DOSSIER TECHNIQUE

```
<button
  *ngIf="authService.hasAdministrationCepp()"
  class="btn btn-sm btn-custom2"
  (click)="onEdit()"
  [translate]=" 'adminPro.editOblige.title' ">
</button>
```

Figure 7 : exemple de création de bouton dans un fichier html

La balise « button » marque la création d'un nouveau bouton.

L'attribut « class » permet de donner une classe au bouton et de pouvoir lui appliquer un style uniforme avec le reste de l'application. Je n'ai pas touché au style du bouton, mais juste réutilisé la classe déjà stylisée via un css global sur l'application.

L'attribut « *ngIf » indique une directive Angular, cela permet d'afficher le bouton en fonction d'une variable. Ici, « authService » est un autre composant (un service) dans lequel on va appeler la fonction « hasAdministrationCepp() ». Cette fonction retourne un booléen qui va décider dynamiquement de l'affichage du bouton.

L'attribut « (click) » permet au click de l'utilisateur d'appeler la fonction « onEdit() » qui se trouve dans le fichier .ts (Typescript) associé. En l'occurrence la fonction onEdit() permet le déclenchement de l'affichage de la modale qui nous intéresse.

L'attribut « [translate] » est utilisable grâce à la librairie « @ngx-translate » importée dans le fichier .ts associé. Il permet d'aller chercher dans un fichier de traduction (internationalisation/i18n) en fonction de la langue indiquée par l'utilisateur (par défaut le français) une chaîne de caractère. En l'occurrence, l'entrée « adminPro.editOblige.title » correspond à « Modifier un obligé ».

Modification dynamique du tableau par l'utilisateur :

Ci-dessous, le code html du bouton qui permet à l'utilisateur d'ajouter une ligne correspondant à une obligation pour la campagne choisie :

DOSSIER TECHNIQUE

```
<button type="button"
  class="btn btn-link"
  [disabled]="fullCampaign"
  (click)="ajoutObligationLigne()">
  </span> {{'transfert.addObligationBtn' | translate}}
</button>
```

Figure 8 : exemple de bouton déclencheur d'une action visible par l'utilisateur dans un fichier html

L'attribut « [disabled] » permet de désactiver le bouton en fonction d'un booléen contenu dans le fichier .ts (TypeScript) associé. Si la valeur de « fullCampaign » est true, le bouton sera actif. Si elle est false, le bouton sera désactivé et l'utilisateur ne pourra pas cliquer dessus.

Au clic de ce bouton, la fonction « ajoutObligationLigne() » est appelée et l'utilisateur pourra ajouter une ligne de compte à l'écran :

Figure 9 : visuel d'une nouvelle ligne ajoutée suite à l'action de l'utilisateur

Cette ligne contient une liste déroulante de numéros de campagnes, un champ qui accepte seulement des nombres et un bouton « » est affiché pour supprimer la ligne. La fonction est codée de la façon suivante :

```
ajoutObligationLigne() {
  // récupère les campagnes déjà sélectionnées
  const campagneAlreadySelected = this.obligationsArray.controls.map(control => control.get('campagne').value);
  // récupère les campagnes possibles mais non affichées
  const firstElementNotSelected = this.listCampagneId.filter(camp => !campagneAlreadySelected.includes(camp));
  // Creation des controls
  const campagne = new FormControl((firstElementNotSelected && firstElementNotSelected.length > 0) ?
    firstElementNotSelected[0] : null, [Validators.required]);
  const nbrObligation = new FormControl(null, [Validators.required, Validators.pattern("^[0-9]*([.]{0,1}[0-9]+)?$"));

  // Creation de FormGroup avec les FormControl
  this.obligationFormGroupLigne = new FormGroup({ campagne, nbrObligation });

  // Cas où on ajoute un obligé via l'edit : bouton désactivé / cas via la création d'obligé : affichage d'une erreur
  if ( this.initFromEdit && !this.fullCampaign ) {
    this.obligationsArray.push(this.obligationFormGroupLigne);
    if ( this.obligationsArray.length === 3 ) {
      this.fullCampaign = true;
    }
  }
}
```

Figure 10 : code de la méthode qui ajoute une ligne d'obligations au tableau dans le fichier .ts correspondant

Deux variables (« campagneAlreadySelected » et « firstElementNotSelected ») pour récupérer respectivement les campagnes déjà sélectionnées et les campagnes possibles (selon les spécifications, la campagne en cours et les deux prochaines campagnes). Ensuite je crée un « FormControl » qui font partie des possibilités comprises dans Angular. Ils permettent notamment de contrôler des champs de saisie par l'utilisateur plus simplement et de façon

DOSSIER TECHNIQUE

automatique. Ils sont généralement compris dans un « FormGroup » qui permet de contrôler les valeurs et la validité de tout un formulaire. On voit ici que sur les FormControl, on peut ajouter des « Validators » par défaut ou custom. Dans cet exemple, on a ajouté la nécessité pour le champ d'être rempli avec « Validators.required » et la nécessité pour la saisie de correspondre à une regex avec « Validators.pattern ». Ici on ajoute au « FormArray » déjà existant tous nos contrôles. Un « FormArray » représente un tableau de « FormControl », « FormGroup » ou d'autres « FormArray ». La ligne est ainsi créée et l'utilisateur peut rentrer des informations ou la supprimer.

Envoi de l'information vers la back-end :

Au clic sur le bouton « Valider » les informations sont envoyées vers un service qui va se charger de communiquer avec le point d'entrée adéquat de façon à assurer la persistance des données. Mais avant d'arriver au service, l'information doit être transportée entre les différents composants imbriqués entre eux. Un système important permettant d'échanger des données entre composants est le système des @Input et @Output dans Angular, il permet de mettre en place des Observer/Observable de façon simplifiée. @Input est utilisé pour passer de l'information d'un composant parent vers un composant enfant et @Output permet de faire l'inverse. Je vais décrire ce fonctionnement ici dans le cadre de la tâche à effectuer. Ci-dessous, une capture écran de code qui se déclenche à la validation de la modale dont on a parlé précédemment :

```
envoyer() {  
  this.updateOblige();  
  let obligeEditToSend: ObligeEdit = new ObligeEdit();  
  obligeEditToSend.oblige = this.oblige;  
  obligeEditToSend.obligations = this.obligations ? this.obligations : null;  
  this.modif.emit(obligeEditToSend);  
}
```

Figure 11 : méthode permettant de déclencher l'envoi des données vers un composant parent qui lui-même pourra les envoyer vers le back-end de l'application

Ici, je fais appel à une autre méthode (« updateOblige ») pour rafraîchir les informations de l'obligé puis je crée un objet de type « ObligeEdit » pour ensuite remplir ses attributs « oblige » et « obligations » avec les données récupérées après la validation de la modale. La ligne intéressante pour le transfert de données est la dernière « this.modif.emit(obligeEditToSend) ; ». Cette dernière notifie le parent qu'un objet a été transmis. On retrouve plus haut dans le code ceci :

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

Figure 12 : exemple d'utilisation de l'annotation @Output

Un événement est donc envoyé vers le parent grâce au @Output. Il est réceptionné par le parent grâce à cet attribut du html parent :

DOSSIER TECHNIQUE

```
<app-oblige-edit
  (modifier)="editOblige($event)"
  (closed)="onCloseEdit()"
  [oblige]="oblige"
  [obligationsByOblige]="obligationsByOblige">
</app-oblige-edit>
```

Figure 13 : détail d'une balise html pour mettre en valeur la réception d'un évènement avec le composant parent dans le cadre de l'utilisation d'un @Output

A la réception de l'évènement, la fonction editOblige est donc appelée avec comme argument l'évènement qui transite, c'est-à-dire dans ce cas là un objet qui contient les données de l'obligé à modifier. Voici la fonction editOblige :

```
editOblige(value: ObligeEdit) {
  const url = ENVIRONMENT.restContext;
  value.currentUserLogin = this.authService.getUserLogin();
  const headers = new HttpHeaders({
    'Content-Type': 'application/json'
  });
  this.http.put<ObligeEdit>(
    `${url}/oblige/edit_oblige/${this.obligeId}`,
    value,
    {headers}
  ).pipe(
    map(data => data),
    tap(
      data => {
        this.toasterService.success('Modification de l\'obligé avec succès');
        this.ngOnInit();
      },
      error => {
        this.toasterService.error('Erreur lors de la modification de l\'obligé');
      }
    )
  ).subscribe();
  this.ngxSmartModalService.getModal('edit').close();
}
```

Figure 14 : méthode permettant d'envoyer les données vers le back-end de l'application

Elle permet notamment de faire le lien avec le point d'entrée de l'application back-end auquel j'accès grâce à la requête http qui finit par « /oblige/edit_oblige/obligeld ». Le « obligeld » est un paramètre et va changer en fonction du cas. On passe ensuite « value » qui correspond aux données que l'on veut envoyer afin qu'elles soient persistées. On peut noter aussi l'utilisation de headers pour indiquer que le contenu de l'objet envoyé est de type Json. Une fois la réponse reçue depuis le back-end, j'affiche à l'écran une notification grâce à la librairie « ngx-

DOSSIER TECHNIQUE

toastr » qui portera une information sous la forme du string qui varie en cas de succès ou d'échec de la requête.

A la fin de la méthode, je ferme la modale qui était affichée à l'écran de l'utilisateur pour qu'il se retrouve sur le même écran qu'avant d'avoir appelé celle-ci.

Exemple n°2 : Inspecteur – Export obligés – Ajout contacts

Compétences couvertes dans cet exemple

Développer la partie back-end d'une interface utilisateur web

Tâches ou opérations effectuées

Controller :

La fonctionnalité à l'écran ressemble à la capture d'écran suivante :

The screenshot shows a web interface titled 'FICHES-ACTION'. It features a search section 'Recherche d'obligés' with various filters: SIRET, Raison sociale, Code postal, Nom du contact, Commune, Région, % acquis/obligation, Nombre de CEPP acquis, Campagne, Etat, Département, and Nombre d'actions validées. Below the search filters is a table titled 'Liste des obligés' with columns: SIRET, Raison sociale, Commune, Nombre d'obligations, Nombre de CEPP acquis, Campagne, Actions pluriannuelles, and %acquis/obligation. The first two rows of the table are visible, with the first row having a checkbox and the second row having a checkbox. A red circle highlights the 'Editer les actions' button in the top right corner of the table.

	SIRET	Raison sociale	Commune	Nombre d'obligations	Nombre de CEPP acquis	Campagne	Actions pluriannuelles	%acquis/obligation
<input checked="" type="checkbox"/>	39151335500010							
<input type="checkbox"/>	35151734700040							

Figure 15 : capture d'écran représentant la fonctionnalité d'export des informations d'un ou plusieurs obligés

Les données des obligés sont cachées par un souci de confidentialité. Le bouton export est entouré d'un cercle rouge.

Sur la tâche de l'export des obligés, il faut récupérer un obligé en lui attachant une liste de collaborateurs, j'ai donc créé un modèle comportant les informations de l'obligé avec un attribut supplémentaire : la liste de collaborateurs. Ci-dessous l'attribut ajouté au modèle :

```
private List<CredentialsCustomDTO> collaborateursList;
```

Figure 16 : ajout d'un attribut de type List<> à une DTO pour ajouter une liste de collaborateurs attachés à un obligé

Ensuite, j'ai pu modifier le contrôleur existant pour attacher la liste de collaborateurs. Voici le point d'entrée du contrôleur correspondant :

```
@Operation(summary = "Permet la recherche d'obligé suivant plusieurs critères")
@PostMapping(value = "search_v_oblige")
public List<VObligedDTO> searchobligé(@Parameter(description = "Identification d'un obligé", required = true)
                                      @RequestBody @Valid SearchObligedDTO searchObligé) {
    log.debug(new Object() {}.getClass().getName() + " - " + new Object() {
    }.getClass().getEnclosingMethod().getName());
    List<VObligedDTO> vobligedDTOList;
    if (searchObligé.getCampagneId() != null) {
        vobligedDTOList = entityDTOMapper.mapAll(obligeService.searchObligé(searchObligé), VObligedDTO.class);
    } else {
        vobligedDTOList = entityDTOMapper.mapAll(obligeService.searchObligéAll(searchObligé), VObligedDTO.class);
    }
    // boucle sur chacun des obligés trouvés pour leur attacher une liste de 3 collaborateurs
    for (VObligedDTO obligé : vobligedDTOList) {
        //get collaborateurs de l'obligé and add to obligé object of syntheses
        List<CredentialsCustomDTO> collaborateursList = entityDTOMapper.mapAll(credentialsService.findUsersByObligéId(obligé.getObligéId()),
            CredentialsCustomDTO.class);
        if(!collaborateursList.isEmpty()){
            //get 3 first collaborateurs
            obligé.setCollaborateursList(collaborateursList.stream()
                .limit(3)
                .collect(Collectors.toList()));
        }
    }
    return vobligedDTOList;
}
```

Figure 17 : exemple de point d'entrée sur l'application situé dans le contrôleur gérant les obligés

L'annotation `@PostMapping` permet de faire un `@RequestMapping` de type « Post ». C'est donc une méthode qui va permettre d'ajouter ou de modifier les données en base. On voit qu'on attend un objet en entrée avec l'annotation `@RequestBody`. `@Valid` est une annotation qui permettra en combinaison d'autres annotations (`@NotNull`, `@NotBlank` ou des validateurs spécifiques) d'accepter ou de rejeter la donnée en entrée.

On voit ensuite que je récupère les données d'une liste obligé et que je le formate pour correspondre à la classe « `VObligedDTO` » grâce à un mapper.

Ensuite, une boucle `for` est appliquée sur la liste récupérée précédemment. Pour chaque obligé dans la liste, je récupère la liste de ses collaborateurs. Puis cette liste est ajoutée à chacun des obligés en limitant le nombre de collaborateurs à 3 avec « `.limit(3)` ». La liste globale d'obligés est ensuite retournée vers le front-end.

Service :

On permet l'appel des services correspondants grâce à l'injection de dépendance gérée automatiquement par Spring. Cette injection est effectuée grâce à l'annotation `@Autowired`. En l'occurrence :

DOSSIER TECHNIQUE

```
@Autowired
CredentialsService credentialsService;
```

Figure 18 : exemple d'utilisation de l'annotation @Autowired pour faire une injection de dépendance dans le controller

Ensuite le Service ira appeler le Repository pour faire des requêtes en base de données :

```
@Override
public List<TCeppCredentials> findUsersByObligeId(Integer obligeId) {
    List<TCeppCredentials> listUsers = credentialsRepository.findUsersByObligeId(obligeId);
    return listUsers;
}
```

Figure 19 : exemple de service servant à appeler la méthode du Repository correspondant

Ici, rien de spécial à noter, le service se contente de faire transiter l'information entre le Controller et le Repository.

Activité-type 2 : Développer la persistance des données

Exemple n°1 : Affichage des messages

Compétences couvertes dans cet exemple

Développer des composants dans le langage d'une base de données

Tâches ou opérations effectuées

La fonctionnalité à l'écran ressemble à la capture d'écran suivante :

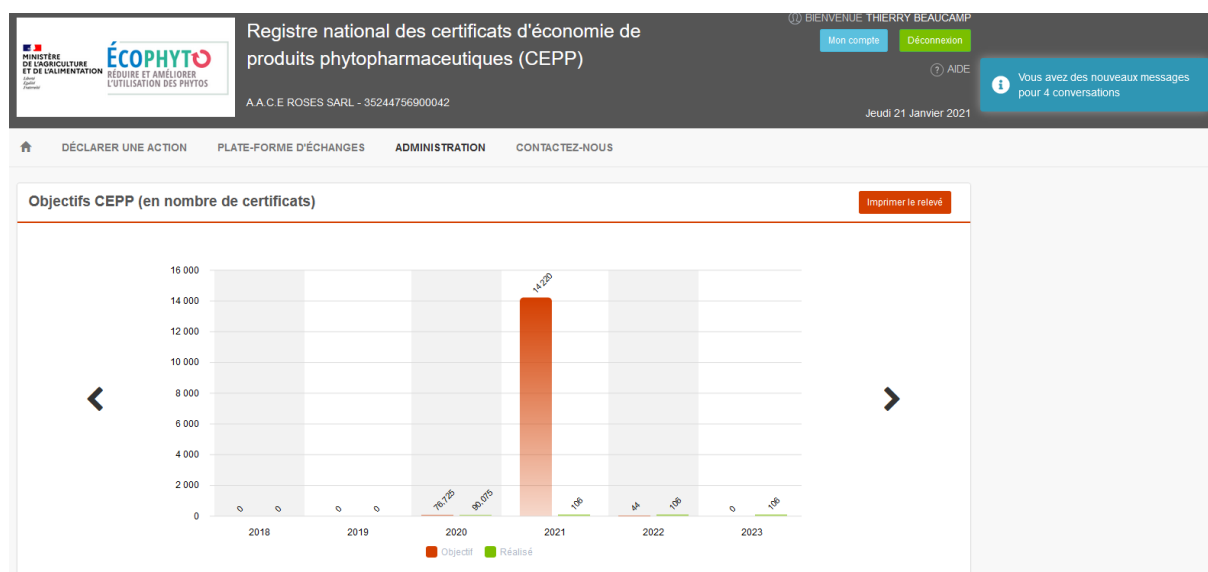


Figure 20 : capture d'écran de la fonctionnalité qui permet d'afficher les messages non lus à la connexion

DOSSIER TECHNIQUE

Sur cet écran on voit en haut à droite le nombre de messages non lus qui s'affiche à la connexion notamment.

Pour résoudre cette tâche, il a été nécessaire de faire des changements sur la partie front-end de l'application mais ce qui nous intéresse ici est la partie back-end. Plus exactement, nous allons nous intéresser à l'accès aux données par le repository concerné. Le but de cette méthode est de compter le nombre de messages non lus d'un utilisateur utilisant un obligé spécifique. L'utilisation de Spring Data permet de faire des requêtes sur la base de données en utilisant seulement le nom des méthodes. Ce système est détaillé dans la partie suivante, dans notre cas, il a été nécessaire de passer par une requête native appliquée à Spring Data. C'est fait avec l'annotation `@Query` comme présenté ci-dessous :

```
public interface TCeppConversationRepository extends JpaRepository<TCeppConversation, Integer> {
    @Query(value = "SELECT COUNT (*) FROM t_cepp_conversation AS conv " +
        "JOIN t_cepp_participant AS part ON conv.conversation_id = part.conversation_id " +
        "WHERE conv.oblige_id = ?1 AND part.utilisateur_id = ?2 AND part.visionne = false",
        nativeQuery = true)
    Integer findUnreadMessagesByOblige(Integer obligeId, Integer userId);
}
```

Figure 21 : exemple de requête native avec l'annotation `@Query` sur une méthode du repository concerné

La requête est de type Select, elle permet de lire des données stockées en base. « SELECT COUNT » retourne le nombre d'entrées correspondant à la suite de la requête. Je cible ensuite une table avec le « FROM ». Le mot clé « AS » donne un alias aux tables utilisées dans cet exemple. Je fais ensuite la jointure entre la table `t_cepp_conversation` et la table `t_cepp_participant` pour récupérer les données voulues, ce qui va de paire avec le mot clé « ON » qui cible 1 colonne dans chacune des 2 tables concernées dans laquelle les deux entrées devront être égales. Enfin, avec le mot « WHERE » je conditionne la sélection des lignes pour que les colonnes `conv.oblige_id`, `part.utilisateur_id` et `part.visionne` soient égales aux paramètres de notre fonction (ou false).

C'est ainsi qu'on peut effectuer une requête native avec Spring Data. Le framework évite beaucoup de code répétitif comme par exemple les ouvertures/fermetures de connexion, la création de Statement ou encore de Resultset.

Activité-type 3 : Développer une application multicouche

Exemple n°1 : Modification d'une fiche-action

Compétences couvertes dans cet exemple

Développer des composants métier

DOSSIER TECHNIQUE

Composant d'accès aux données

Tâches ou opérations effectuées

La fonctionnalité à l'écran ressemble à la capture d'écran suivante :

Liste des actions non conformes par obligé	↓
Campagne, Raison sociale, SIREN, Référence, Numéro de l'action NC, Nombre d'années, Numéro de la fiche-action, Nombre de CEPP total de l'action NC pour la campagne en cours, Nombre de CEPP injustifiés de l'action NC pour la campagne en cours	
Nombre d'obligations et de CEPP par obligé	↓
Campagne, Raison sociale, SIREN, Région, Obligés actif (oui ou non), Nombre d'obligations, Nombre de déclarations effectuées, Nombre de déclarations validées, Nombre de CEPP obtenus au titre des déclarations de l'année, Nombre de CEPP obtenus pour l'année	
Liste des actions à contrôler	↓
Date de déclaration, Campagne, Numéro de l'action, Raison sociale, SIREN, Nombre d'obligations, Nombre de CEPP de l'action, Nombre de CEPP, Numéro de la fiche-action, Nom, Unité, Quantité, Numéro AMM, Commentaire interne MAA, Observations, Proposition cellule, Décision	
Liste des actions déclarées par obligé	↓
Campagne, Région, Raison sociale, SIREN, Obligés actif (oui ou non), Numéro de la fiche-action, Numéro de l'action, Etat, Nombre de CEPP, Nombre d'années	

Figure 22 : capture d'écran montrant la fonctionnalité de statistiques non conformes par obligé

La demande consiste à récupérer les données d'une vue en base et de les afficher à l'utilisateur quand il exporte un certain type de statistiques. C'est une Jira transversale à toute l'application mais nous allons nous intéresser seulement à la partie de l'accès aux données et à l'objet qui va contenir les données de l'entité récupérées en base. Comme on l'a déjà vu précédemment le Repository est assez maigre en termes de code grâce à l'utilisation de Spring Data. Voici le code de celui-ci :

```
public interface VCeppStatistiquesInspectionReferencesRepository
    extends JpaRepository<VCeppStatistiquesInspectionReferences, Integer> {
}
```

Figure 23 : exemple d'une classe de Repository ne comportant aucune méthode

On remarque que c'est une interface qui hérite de « JpaRepository ». Cela permettra de faire des requêtes automatisées avec le nom des méthodes ou encore accéder aux méthodes du crud déjà contenues dans « JpaRepository ». C'est notre cas ici, en effet, c'est la méthode « findAll » qui est appelée, pas besoin de coder de méthode, JpaRepository s'occupe de tout. On sait que les objets retournés seront du type du modèle « VCeppStatistiquesInspectionReferences ». Cependant, pour que tout cela fonctionne, il faut que cette classe ait les bons attributs et autres annotations. Voyons cela avec une capture d'écran simplifiée du modèle :

DOSSIER TECHNIQUE

```
@Entity
@NoArgsConstructor
@AllArgsConstructor
@EqualsAndHashCode
@Subselect("select * from v_cepp_statistiques_inspection_references")
@IdClass(VCeppStatistiquesInspectionReferencesId.class)
public class VCeppStatistiquesInspectionReferences implements Serializable {

    // attributs

    // getters et setters

    // exemple de getter
    @Basic
    @Column(name = "siren")
    public String getSiren() {
        return siren;
    }
    // exemple de getter associé à une clé composite
    @Id
    @Column(name = "raison_sociale")
    public String getRaisonSociale() {
        return raisonSociale;
    }
}
```

Figure 24 : exemple d'une classe métier, correspondant à une entité en base de données

On voit beaucoup d'annotations sur cette classe.

@Entity permet de signifier à Spring que la classe est un bean qui sera associé à une entité de la base de données.

@NoArgsConstructor, @AllArgsConstructor et @EqualsAndHashCode permettent respectivement de générer automatiquement le constructor sans arguments, le constructor avec tous les attributs de la classe en tant qu'argument et enfin les méthodes equals et hashCode à partir des attributs de la classe encore une fois. Ces annotations sont disponibles grâce à la librairie Lombok.

@Subselect permet de cibler et de faire un Select quand cette classe est utilisée par le Repository.

@IdClass permet de créer une clé composite à partir d'une autre classe, ici « VCeppStatistiquesInspectionReferencesId.class », qui reprend certains attributs de notre classe pour en créer une clé primaire.

Les getters des attributs classiques sont marqués par l'annotation @Basic et @Column pour cibler une colonne de la table concernée et la faire correspondre avec ce getter/attribut. A noter que ces annotations peuvent être placées directement sur l'attribut mais que par soucis

DOSSIER TECHNIQUE

de lisibilité sur les attributs, on préfère les placer sur les getters.

Les getters des attributs qu'on utilisera pour la clé composite sont annotés de @Id.

La classe qui permet de créer une clé composite est la suivante :

```
@NoArgsConstructor
@AllArgsConstructor
@Getter
@Setter
@EqualsAndHashCode
@Embeddable
public class VCeppStatistiquesInspectionReferencesId implements Serializable {

    private Integer campagne;
    private String raisonSociale;
    private String reference;
    private BigDecimal nbCeppInjustifies;
}
```

Figure 25 : exemple d'une classe permettant de créer une clé composite pour la classe métier

Comme on peut le voir, elle reprend effectivement des attributs de la classe précédente.

Exemple n°2 : Statistiques – Inspections à contrôler

Compétences couvertes dans cet exemple

Construire une application organisée en couches

Développer des composants métier

Tâches ou opérations effectuées

La fonctionnalité à l'écran ressemble à la capture d'écran suivante :

DOSSIER TECHNIQUE

Liste des actions non conformes par obligé	↓
Campagne, Raison sociale, SIREN, Référence, Numéro de l'action NC, Nombre d'années, Numéro de la fiche-action, Nombre de CEPP total de l'action NC pour la campagne en cours, Nombre de CEPP injustifiés de l'action NC pour la campagne en cours	
Nombre d'obligations et de CEPP par obligé	↓
Campagne, Raison sociale, SIREN, Région, Obligés actif (oui ou non), Nombre d'obligations, Nombre de déclarations effectuées, Nombre de déclarations validées, Nombre de CEPP obtenus au titre des déclarations de l'année, Nombre de CEPP obtenus pour l'année	
Liste des actions à contrôler	↓
Date de déclaration, Campagne, Numéro de l'action, Raison sociale, SIREN, Nombre d'obligations, Nombre de CEPP de l'action, Nombre de CEPP, Numéro de la fiche-action, Nom, Unité, Quantité, Numéro AMM, Commentaire interne MAA, Observations, Proposition cellule, Décision	
Liste des actions déclarées par obligé	↓
Campagne, Région, Raison sociale, SIREN, Obligés actif (oui ou non), Numéro de la fiche-action, Numéro de l'action, Etat, Nombre de CEPP, Nombre d'années	

Figure 26 : capture d'écran montrant la fonctionnalité de statistiques des actions à contrôler

Une application est souvent séparée en 3 couches :

- La couche d'accès aux données servira à accéder aux données à partir de l'application
- La couche métier qui permet de gérer les opérations spécifiques à la demande du client
- La couche présentation qui permettra notamment de mettre en forme les données pour l'afficher à l'utilisateur

Une séparation en couches permet de bien attribuer à chaque couche un rôle spécifique ainsi que de pouvoir remplacer une technologie de l'une des couches si le besoin s'en fait sentir (désuétude de la technologie par exemple). J'ai pu dans le cadre de mon projet, créer tous les éléments qui, bouts à bouts permettent de récupérer une donnée depuis la base pour l'afficher à l'utilisateur.

Couche d'accès aux données :

Dans notre application, elle devient très fine grâce à l'utilisation du framework Spring Data (qui utilise lui-même Hibernate pour communiquer). La plupart du temps, la couche d'accès aux données se résume à l'utilisation de noms de méthodes correspondants à des requêtes SQL automatiquement interprétées. Avec Spring Data, on peut également exécuter des requêtes manuelles, cette possibilité est présentée dans l'activité-type 2, exemple 1.

```
public interface VCeppStatistiquesAControlerRepository
    extends JpaRepository<VCeppStatistiquesActions, Integer> {
    List<VCeppStatistiquesActions> findByStatutAction(String statut);
}
```

Figure 27 : exemple d'une classe de Repository et d'une méthode permettant d'attaquer la base avec Spring Data

DOSSIER TECHNIQUE

Dans l'exemple ci-dessus, on remarque que j'utilise une interface qui hérite de « JpaRepository » épargnant ainsi l'utilisation d'un code répétitif pour l'accès aux données pour les méthodes du CRUD (CREATE, READ, UPDATE, DELETE). Dans le cas de l'exemple, le nommage « findBy » permet d'effectuer un select dans la base sur la vue « VCeppStatistiquesActions ».

Ensuite le reste du nom de la méthode « StatutAction » permet de cibler la colonne « statut_action » de la base de données. On remarque que la méthode prend un argument de type String.

Cette requête récupère donc toutes les entrées qui ont pour « statut_action » la chaîne de caractères passé en argument.

Couche métier :

La couche métier est la couche principale, elle implémente toute la logique spécifique au métier de l'application. Elle comprend notamment les « Controllers » et les « Services ». Les controllers servent à exposer les points d'entrée de l'application à la partie présentation. Les services permettent de faire le lien entre la couche métier et la couche d'accès aux données. Dans ces deux types de classes on peut retrouver des éléments du métier. Les controllers sont particulièrement intéressants car ils vont permettre la communication entre le front-end et le back-end. Un exemple de la méthode que j'ai ajoutée pour répondre au besoin :

```
@Operation(summary = "Retourne les statistiques des actions à contrôler")
@GetMapping(value = "/a_controler")
public List<VStatistiquesActionsAControlerDTO> findAllStatistiquesActionsAControler() {
    log.debug(new Object() {
    }.getClass().getName() + " - " + new Object() {
    }.getClass().getEnclosingMethod().getName());
    if(!userService.currentUserHasPermission(PermissionHelper.GESTION_CEPPE)) {
        String message = getMessage("error.notauthorized");
        log.error(message, userService.getCurrentUserCredential());
        throw new CeppFonctionnelleException(message);
    }
    return entityDTOMapper.mapAll(statistiquesAControlerService.findAllAControler(), VStatistiquesActionsAControlerDTO.class);
}
```

Figure 28 : exemple de classe Controller permettant de réceptionner la requête http et de faire le lien avec un service

L'annotation « @GetMapping » permet d'exposer la méthode aux requêtes http de notre partie front-end. Une requête contenant le chemin initial et terminée par « /a_controler » appelle cette méthode et permet de récupérer des données, en l'occurrence de type « VStatistiquesActionsAControlerDTO ». Le DTO (Data Transfer Object) est utilisé pour formater les données avec les attributs voulus et pour éviter de passer des objets java portant des attributs identiques aux colonnes de l'entité concernée. Cette méthode appellera ensuite un service qui lui-même appellera une méthode de Repository, faisant ainsi le lien avec la couche d'accès aux données.

Couche présentation :

Elle correspond à la plus haute couche qui permettra d'afficher les données à l'utilisateur. C'est là que la mise en forme de l'information interviendra et que tous les éléments graphiques (charte) s'ajouteront aux données envoyées depuis la partie back-end de

DOSSIER TECHNIQUE

l'application pour fournir à l'utilisateur la meilleure expérience possible.

La résolution de cette tâche demandait de respecter l'architecture de chaque couche. Les trois couches sont indépendantes entre elles et la couche présentation connaît seulement la couche métier et non pas la couche d'accès aux données

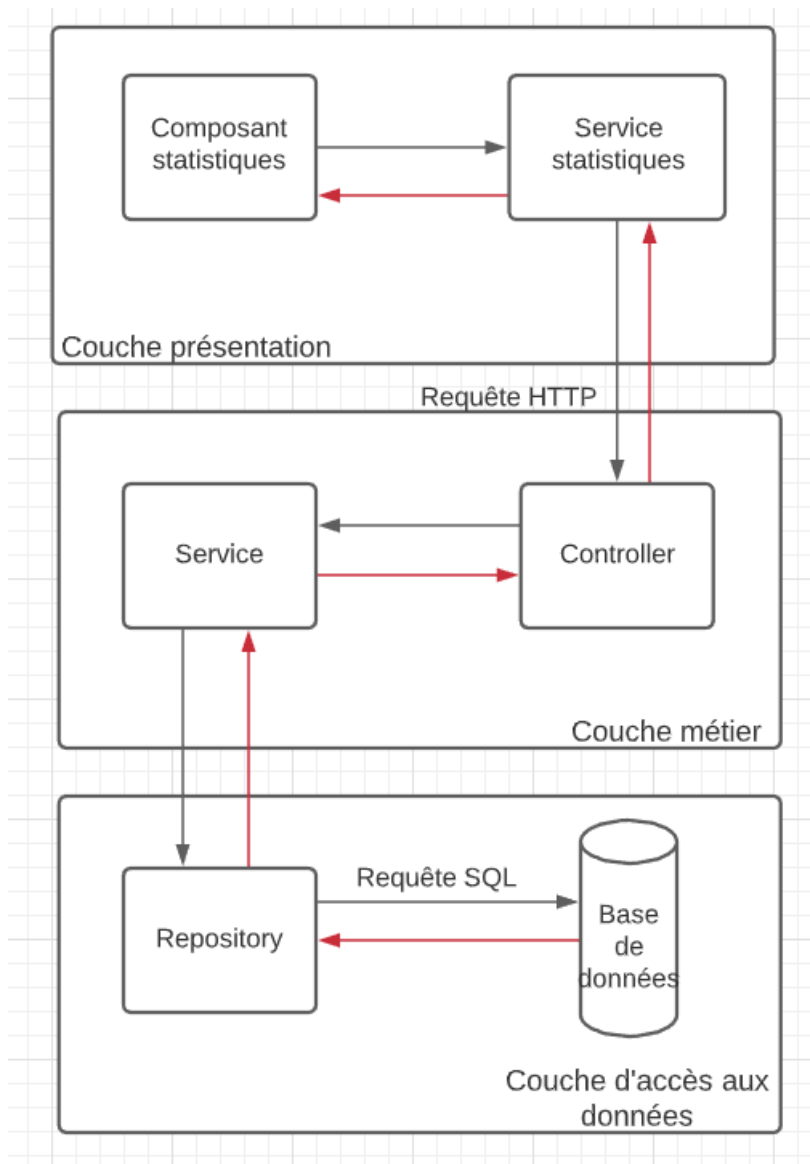


Figure 29 : schéma des différentes couches présentes dans l'application pour le cas des statistiques

Un exemple de requête http faite sur le front permet de comprendre comment l'appel au back est effectué :

DOSSIER TECHNIQUE

```
getStatistiquesAController() {  
    const URL = this.url + '/statistiques/a_controller';  
    return this.http.get<StatistiquesAController[]>(URL);  
}
```

Figure 30 : exemple d'un appel au back via le service du front-end, requête http de type GET

Quand cette méthode est appelée, elle accède à l'url commun pour tous les points d'entrée de l'application suivi de `statistiques/a_controller` permettant de cibler la méthode du controller précédente.















Exemple n°3 : Durée d'affichage pour les inspections

Compétences couvertes dans cet exemple

Préparer et exécuter les plans de tests d'une application

Tâches ou opérations effectuées

La fonctionnalité à l'écran ressemble à la capture d'écran suivante :

N° de l'inspection	Date de l'inspection	Raison sociale	Nombre d'actions non conformes	Etat de l'inspection	Editier courrier
mx45464	07/01/2021	A.A.C.E ROSES SARL	0	Enregistrée	  
1545523165	07/01/2021	A.A.C.E ROSES SARL	2	Validée	 
1842563255	07/01/2021	A.A.C.E ROSES SARL	3	Enregistrée	   
5454578575	06/01/2021	A.A.C.E ROSES SARL	1	Enregistrée	  
0111111222	06/01/2021	A.A.C.E ROSES SARL	1	Validée	 

22 résultat(s)

1 2 3 4 5

Figure 31 : capture d'écran du module des inspections

Pour mener à bien cette tâche, il était nécessaire de créer un nouveau point d'entrée sur l'application en back-end, ainsi que les traditionnels services et repository pour accéder aux données. Mais pour la compétence qui nous intéresse, je vais simplement décrire la méthode de test ajoutée pour tester le point d'entrée.

Voici sur l'image suivante la méthode de test ajoutée pour tester le point d'entrée :

DOSSIER TECHNIQUE

```
@Test
public void getInspectionsToDisplayTest() throws Exception {
    mockMvc.perform(get("/inspections/display/2").header("Authorization", token)
        .contentType(MediaType.APPLICATION_JSON))
        .andExpect(status().isOk());
}
```

Figure 32 : exemple de méthode de test unitaire automatisé

Spring permet ici d'utiliser « @Test » pour estampiller la méthode en tant que test, grâce à cela, lorsqu'on lancera tous les tests, cette méthode sera automatiquement appelée.

La méthode est nommée après la méthode du contrôleur qu'elle devra tester. Ici, le test vérifie simplement que le statut de la réponse http est 200 grâce à « .andExpect(status().isOk()) ». Cela permet de s'assurer que la requête a bien été exécutée. On pourrait aller plus loin en testant le type d'objet retourné par exemple, ou encore la valeur d'un des attributs de cet objet.

A noter qu'on ne teste pas sur les données utilisées lors du développement. C'est une base de données embarquée H2 qui est utilisée sur laquelle on va jouer des scripts SQL pour remplir les données de test. On indique sur la classe de test les annotations suivantes :

```
@Sql("classpath:cleanup.sql")
@Sql("classpath:init.sql")
public class InspectionsControllerTest extends CeppApplicationTest {
    //méthodes de test
}
```

Figure 33 : annotations au-dessus de la classe InspectionsControllerTest

Ces annotations @Sql permettent de jouer des fichiers .sql pour supprimer les données et les remplir avec de nouvelles avant d'appeler les différentes méthodes présentes dans la classe.

On utilise aussi l'annotation @Before :

```
@Before
public void setup() throws Exception {
    BacusIdentityDTO bacusIdentityDTO = new BacusIdentityDTO();
    bacusIdentityDTO.setEmail("test@gmail.com");
    Mockito.when(bacusBouchonServiceImpl.getIdentityById(Mockito.any())).thenReturn(bacusIdentityDTO);
    MvcResult result = mockMvc.perform(get("/auth/cas/validate?ticket=FAKE_TICKET")
        .header("serviceUrl", "123456789")
        .contentType(MediaType.APPLICATION_JSON)).andReturn();
    ObjectMapper mapper = new ObjectMapper();
    Map<String, Object> map
        = mapper.readValue(result.getResponse().getContentAsString(),
            new TypeReference<Map<String, Object>>() {
        });
    token = map.get("access_token").toString();
}
```

Figure 34 : méthode setup annotée d'@Before

DOSSIER TECHNIQUE

De cette manière, la méthode `setup()` est appelée avant toutes les autres. Elle permet de récupérer un token pour le système d'authentification avant de faire des appels aux points d'entrée.

Description d'une situation de travail avec recherche

J'ai eu l'occasion de pouvoir modifier la charte graphique de l'application, ce qui m'a demandé quelques recherches. Notamment parce qu'il fallait changer la police d'écriture des documents pdf ou odt imprimés pour l'utilisateur. Tout d'abord, il a fallu comprendre comment fonctionnait l'export de pdf/odt qui, sur l'application se fait par l'intermédiaire de JasperReports une librairie intégrée au back-end. C'est tout un langage déclaratif qu'il a fallu appréhender pour faire des changements sur les pdf exportés. Mais je vais me pencher tout particulièrement sur l'import de police.

Mes recherches m'ont mené vers une recherche initiale avec les termes « custom font jasperreport java » pour comprendre comment intégrer la police à l'application. La manière usuelle pour ça est d'ajouter un jar au projet, on retrouve cette méthode sur beaucoup de forum. Cependant, ma contrainte était d'ajouter la police sans utiliser de jar, c'est ainsi que je finis par tomber sur un site décrivant comment faire sans celui-ci.

Cela passe par l'ajout des fichiers .ttf (les thèmes de la police) déclinés en plusieurs fichiers pour l'écriture classique, en gras ou encore en italique. Ensuite, on a besoin d'un fichier de propriétés qui permet à JasperReport de supporter une police customisée :

```
net.sf.jasperreports.extension.registry.factory.fonts=net.sf.jasperreports.engine.fonts.SimpleFontExtensionsRegistryFactory
net.sf.jasperreports.extension.registry.factory.simple.font.families=net.sf.jasperreports.engine.fonts.SimpleFontExtensionsRegistryFactory
net.sf.jasperreports.extension.simple.font.families.MyFont=fr/gouv/fonts/fonts.xml
```

Figure 35 : lignes de code à ajouter dans le fichier de propriétés de JasperReport pour accepter une police customisée

On voit également dans ces propriétés qu'on définit un fichier .xml, il faut aussi l'ajouter pour pouvoir cibler les fichiers de thème de la police :

DOSSIER TECHNIQUE

```
<fontFamilies>
  <fontFamily name="Marianne">
    <normal>fr/gouv/fonts/font/Marianne-Regular.ttf</normal>
    <bold>fr/gouv/fonts/font/Marianne-Bold.ttf</bold>
    <italic>fr/gouv/fonts/font/Marianne-RegularItalic.ttf</italic>
    <boldItalic>fr/gouv/fonts/font/Marianne-BoldItalic.ttf</boldItalic>
    <pdfEncoding>Identity-H</pdfEncoding>
    <pdfEmbedded>true</pdfEmbedded>
  </fontFamily>
</fontFamilies>
```

Figure 36 : fichier xml à ajouter pour accepter une police personnalisée

Comme, on le voit ici, on cible les différents fichiers .ttf, on a aussi un encodage avec la balise `<pdfEncoding>` qui indique Identity-H qui correspond à peu près à de l'UTF-8.

Ensuite j'ai bien sûr lancé mon application, pour m'apercevoir que ça n'allait pas fonctionner si facilement. En effet, s'en est suivi de nombreuses erreurs java. J'ai donc recherché des réponses avec les termes suivants :

« java.awt.FontFormatException: bad table, tag »

« java.lang.ExceptionInInitializerError: null jasperreport »

« Failed to execute goal org.apache.maven.plugins:maven-enforcer-plugin:1.4.1:enforce »

Le page indiquant la marche à suivre était bien sur un exemple simple. Seulement, l'application étant beaucoup plus complexe et, à force de recherches et en observant mon pom.xml, j'ai fini par comprendre que l'application filtre les différents fichiers en fonction de leur extension. J'ai donc dû ajouter des exclusions pour les fichiers .ttf à plusieurs endroits dans le pom.xml.

Après la réussite de pouvoir intégrer la police à l'application, j'ai pu modifier dans les fichiers d'export (extension .jrxml) en remplaçant l'ancienne police par la nouvelle.

Bilan personnel

Cette partie clôture mon dossier technique, j'y ai décrit l'application dans son ensemble ainsi que le projet sur lequel j'ai travaillé pendant environ 5 mois. Les tâches que j'ai consignées ici visent à signifier de ma maîtrise des différentes compétences nécessaires à l'obtention du titre. Ces compétences sont séparées en trois catégories, développer des composants d'interface, développer la persistance des données et développer une application multicouche. Cependant, elles ne sont pas toutes adressées notamment parce que dans ce projet je n'ai pas eu l'occasion de travailler beaucoup sur la persistance des données. En revanche, je disserte sur les compétences manquantes dans le dossier professionnel, qui est joint à ce document.

Cette expérience a été incroyablement constructive, j'ai appris beaucoup en peu de temps et cela a été une vraie épreuve à surmonter. Mais je n'en suis pas moins très heureux d'avoir eu la chance de participer à cette réinsertion professionnelle grâce aux personnes qui ont cru en moi. Cette aventure m'a fait découvrir l'ambivalence de la programmation, la frustration lors d'un blocage sur un problème mais surtout le sentiment de réussite une fois l'obstacle surmonté. Sur le plan technique, je me sens bien plus à l'aise sur les technologies que j'ai utilisées, notamment Angular et Spring, ainsi que sur la programmation en général. J'ai acquis non seulement un savoir technique mais aussi une méthodologie plus soucieuse du prochain développeur qui devra faire des évolutions sur le code que j'ai produit. Pour finir, j'ai hâte de continuer dans cette voie, et j'espère que ce sera en collaboration avec Atos !

Tables des illustrations

Figure 1 : compétences traitées dans ce dossier	5
Figure 2 : schéma du cycle en V	10
Figure 3 : schématisation de ce qu'apporte l'environnement Apys	13
Figure 4 : schéma de la cinématique d'authentification sur l'environnement Apys	15
Figure 5 : capture d'écran de la concrétisation de la tâche "Modification d'un obligé"	21
Figure 6 : exemple d'utilisation de la librairie "ngx-smart-modal" pour la création d'une modale dans un fichier html	22
Figure 7 : exemple de création de bouton dans un fichier html	23
Figure 8 : exemple de bouton déclencheur d'une action visible par l'utilisateur dans un fichier html	24
Figure 9 : visuel d'une nouvelle ligne ajoutée suite à l'action de l'utilisateur	24
Figure 10 : code de la méthode qui ajoute une ligne d'obligations au tableau dans le fichier .ts correspondant	24
Figure 11 : méthode permettant de déclencher l'envoi des données vers un composant parent qui lui-même pourra les envoyer vers le back-end de l'application	25
Figure 12 : exemple d'utilisation de l'annotation @Output	25
Figure 13 : détail d'une balise html pour mettre en valeur la réception d'un évènement avec le composant parent dans le cadre de l'utilisation d'un @Output	26
Figure 14 : méthode permettant d'envoyer les données vers le back-end de l'application	26
Figure 15 : capture d'écran représentant la fonctionnalité d'export des informations d'un ou plusieurs obligés	27
Figure 16 : ajout d'un attribut de type List<> à une DTO pour ajouter une liste de collaborateurs attachés à un obligé	28
Figure 17 : exemple de point d'entrée sur l'application situé dans le controller gérant les obligés	28
Figure 18 : exemple d'utilisation de l'annotation @Autowired pour faire une injection de dépendance dans le controller	29
Figure 19 : exemple de service servant à appeler la méthode du Repository correspondant	29
Figure 20 : capture d'écran de la fonctionnalité qui permet d'afficher les messages non lus à la connexion	29
Figure 21 : exemple de requête native avec l'annotation @Query sur une méthode du repository concerné	30
Figure 22 : capture d'écran montrant la fonctionnalité de statistiques non conformes par obligé	31
Figure 23 : exemple d'une classe de Repository ne comportant aucune méthode	31
Figure 24 : exemple d'une classe métier, correspondant à une entité en base de données	32
Figure 25 : exemple d'une classe permettant de créer une clé composite pour la classe métier	33
Figure 26 : capture d'écran montrant la fonctionnalité de statistiques des actions à contrôler	34
Figure 27 : exemple d'une classe de Repository et d'une méthode permettant d'attaquer la base avec Spring Data	34
Figure 28 : exemple de classe Controller permettant de réceptionner la requête http et de faire les lien avec un service	35
Figure 29 : schéma des différentes couches présentes dans l'application pour le cas des statistiques	36
Figure 30 : exemple d'un appel au back via le service du front-end, requête http de type GET	37
Figure 31 : capture d'écran du module des inspections	37

DOSSIER TECHNIQUE

Figure 32 : exemple de méthode de test unitaire automatisé	38
Figure 33 : annotations au-dessus de la classe InspectionsControllerTest.....	38
Figure 34 : méthode setup annotée d'@Before	38
Figure 35 : lignes de code à ajouter dans le fichier de propriétés de JasperReport pour accepter une police customisée	40
Figure 36 : fichier xml à ajouter pour accepter une police customisée	41