# Dossier de projet

## Concepteur Développeur d'Applications


Octobre 2023 à février 2024

CENTRE DE RÉADAPTATION MULHOUSE
Rééducation et Formation Professionnelle

David CRAVO

46262 CARACTERES

# Table des matières

# Remerciements

Je souhaite remercier toute l'équipe d'Apside Saint-Pierre-Des-Corps qui m'a aidé à monter en compétences. Notamment Sébastien AUPETIT, mon tuteur, qui m'a accompagné et conseillé tout au long de mon stage.

Je tiens également à remercier ma femme, Sandrine, mes enfants Calixte et Océane, pour leur soutien et leur relecture de ce dossier.

Je souhaite également remercier toute l'équipe du Centre de Réadaptation de Mulhouse, Francine COMMUNOD, responsable de formation, Sophie THIRY, formatrice référente, Michaël DEVOLDERE, Franck CHATELOT, Catherine ULMER, formateurs, Lydia PELLOUX, conseillère en insertion professionnelle, pour leur disponibilité et leur pédagogie dont ils ont fait preuve tout au long de la période de formation.

# 1. Présentation de l'entreprise

## 1.1. Apside

**1976 Naissance d'Apside**

La société Apside a été fondée par M. Michel Klar. Ce fut l'une des premières entreprises indépendantes françaises de conseil en ingénierie.

Apside est un environnement de partage, de curiosité et d'échange qui met en relation une communauté de 3 000 consultants partageant les mêmes passions. En tant que société de service, communément appelée ESN, son rôle est double. Apside accompagne ses clients dans le développement de leurs projets grâce aux compétences technologiques de ses talents. Le Groupe stimule cette intelligence collective en mettant au cœur de son développement la formation interne et l'intrapreneuriat, favorisant ainsi, la construction du plan de carrière de ses collaborateurs.

**Cf 12.1 Présentation de l'entreprise**

## 1.2. Apside Top

Créée en 2004, Apside TOP, filiale du groupe Apside, est une Entreprise de Service Numérique spécialisée dans l'informatique de gestion. Elle propose des prestations en assistance technique et au forfait auprès de clients grands comptes et de clients locaux dans divers domaines (banque, mutuelle, assurance, industrie…).

Apside TOP, au travers de ses quatre agences de Tours, Orléans, Le Mans et Niort et de ses 200 collaborateurs, est ancrée en région Centre Val de Loire dans de nombreuses villes comme Niort, Poitiers, Le Mans, Chartres, Bourges, Blois, etc.

## 1.3. Apside Saint-Pierre-Des-Corps

**Quentin ROULET** – Développeur Web et Web Mobile

C'est un jeune développeur de 24 ans qui a envie d'apprendre et a soif de découverte. Il possède un bac STI 2D (Technologies informatiques et développement durable) avec option SIM (Systèmes Informatiques et Numériques). Après s'être cherché au travers d'une inscription en Faculté des Sciences (Mathématiques, Physique Chimie Informatique) et ensuite au travers d'une faculté en Informatique, il s'inscrit au Digital Campus de Rennes où il suit une formation de développeur Web et Web Mobile. Il effectue son stage de fin de formation chez Apside, qui se conclura par un contrat à durée indéterminée jusqu'à aujourd'hui. Il travaille directement pour Harmonie Mutuelle, pour laquelle il s'occupe du développement du « parcours de vente en ligne » qui permet aux clients de souscrire en ligne pour une offre de prévoyance, de santé, ou les deux. Il développe la partie front-end sur VueJS et le back-end sur Drupal. Il a appris tout le vocabulaire de ce secteur d'activité depuis son arrivée chez Apside. Il y est heureux et compte bien y poursuivre son aventure.

Ne pouvant citer tout le monde, j'ai dû me restreindre à un petit groupe de personnes ayant participer directement au projet :

**Baptiste DAVID** – Etudiant en école d'ingénieur SUPINFO

Après un bac S, il s'oriente vers la Faculté de Sciences de Tours, où il travaillera sur les matières : mathématiques, physique et informatique. Cette dernière matière mettra à jour son gout prononcé pour cette discipline. Il continue donc dans cette voie en obtenant une licence d'informatique. Aujourd'hui, il est en école d'ingénieur par alternance, ce qui lui permet d'intégrer une team avec des missions client pour Harmonie Mutuelle. Il est soutenu par Corentin et Robin, deux Tech Lead. Il aspire maintenant à faire de la mise en production.

**Antoine PINARD** – Business Analyst

Issu d'une reconversion professionnelle après un master en biologie végétale, il a travaillé pendant 2 ans pour le ministère des armées. Dans le projet Source Solde en tant que recetteur, il s'est spécialisé dans l'automatisation des tests dans le même projet pendant une période équivalente. Il a ensuite travaillé comme business analyst spécialisé dans la recette fonctionnelle pour plusieurs projets pendant un an pour le ministère des armées et pour Harmonie Mutuelle, avant d'atterrir sur le projet interne Apside Trombine en tant que BA. Sur ce projet naissant, il travaille sur la conception de diagrammes d'activités, de maquettes et de spécifications fonctionnelles.

**Simon PILLON** – Ingénieur d'étude et développement

Il est diplômé du Master Of Engineering de l'école Supinfo. Sa formation très généraliste lui a permis d'acquérir un large panel de connaissances de notre monde numérique. Il devait se spécialiser à partir de la 4ème année dans un des domaines suivants : Intelligence Artificielle, Ingénierie Data, Développement cloud et mobile, Systèmes et réseaux ou Cybersécurité. Il a opté pour la spécialisation Intelligence Artificielle qui est un domaine d'actualité. Le cursus met également en avant nos expériences professionnelles via des stages chaque année et offre la possibilité d'être en alternance à partir de la 3ème année. Avant de rejoindre Apside, il a pu suivre sa formation en alternance au sein de la Banque Populaire Val France de sa 3ème année jusqu'à sa 5ème année. Cette expérience lui a permis de se confronter au monde du travail et de renforcer ses connaissances théoriques sur le développement Web. Dans un intérêt de recherche constante de connaissances, il a choisi d'intégrer une ESN tel qu'Apside pour élargir ses possibilités d'évolutions.

**Maxence GRIAS** – Ingénieur études et développement

C'est un développeur qui reste à disposition pour des missions chez des clients. Il a obtenu un master en informatique et système d'information à Supinfo. À la suite de ses cinq ans d'études, il a rejoint Apside depuis un an.

**Vincent DUGUET** – Ingénieur d'Etude et Développement

Avant de rejoindre Apside, il y a deux ans, il a travaillé chez Citya en tant que Développeur Fullstack pendant un an et demi. Son expertise se concentre principalement sur le développement côté front-end, avec une préférence marquée pour le langage JavaScript. Son parcours professionnel est le fruit d'une reconversion. En effet, en 2019, il a entrepris une transition vers le développement web et mobile, qu'il a accompli avec succès à la Wild Code School. Dans ses fonctions actuelles, en tant qu'ingénieur d'Étude et de Développement, il doit concevoir et de réaliser des solutions logicielles innovantes pour répondre aux besoins des clients.

**Hugo PIERRE** – Alternant en Etudes et Développement

Issu d'une formation en école d'ingénieur informatique au Cesi de Pau, il était chez Apside Top à St-Pierre-Des-Corps. Au cours de son stage chez Apside, il a travaillé sur une mission en C# pendant six mois, puis sur une mission en Flutter sur une application web et mobile pendant six mois. Enfin, il a développé un algorithme de recommandation pendant neuf mois. Lors de ses études, il a travaillé sur différents projets, que ce soit du web, du réseau, de l'IA ou encore la création d'une application windows. Pour finaliser ses études, il est amené à gérer ce projet en tant que chef de projet et développeur.

# 2. English Abstract

Following my training as an Application Designer Developper at Centre de Réadaptation de Mulhouse, I did a four-month internship as a developer at Apside Top in Saint-Pierre-Des-Corps, near Tours, in the region Centre Val-De-Loire. It is a Digital Service Company who works for Health insurance company like MGEN or Harmonie Mutuelle.

My project was to create a new application called Trombine, which would permit at all collaborators of the company to find other collaborators with means of contacts and functions. It will permit to manage collaborators'information. The application must manage right areas and tags, in addition to civil status and contacts data. It's an internal project, which concerns principally Apside Top, but may be used in all the Apside companies. It's not meant to be sent to other companies, due to a special organization. Collaborators work with a manager and other collaborators directly in the client's companies.

I was in charge of the project organization, analysis, design and system development across functional specifications.

I have deepened my technical knowledge of Java programming language, helped by Spring for the back-end and Angular for the front-end. I also use other technologies, such as WSL and a customized development environment for the application made by another developer. This environment was associated with PostGreSql for the database.

# 3. Liste des compétences couvertes

Voici les compétences couvertes en stage par mes projets :

| | Compétence couverte **totalement** ou **partiellement** | Compétence non couverte |
|---|---|---|
| Maquetter une application | X | |
| Développer une interface utilisateur de type desktop | | X |
| Développer des composants d'accès aux données | X | |
| Développer la partie front-end d'une interface utilisateur web | X | |
| Développer la partie back-end d'une interface utilisateur web | X | |
| Concevoir une base de données | X | |
| Mettre en place une base de données | X | |
| Développer des composants dans le langage d'une base de données | X | |
| Collaborer à la gestion d'un projet informatique et à l'organisation de l'environnement de développement | X | |
| Concevoir une application | X | |
| Développer des composants métiers | X | |
| Construire une application organisée en couches | X | |
| Développer une application mobile | | X |
| Préparer et exécuter les plans de tests d'une application | | X |
| Préparer et exécuter le déploiement d'une application | | X |

# 4. Le cahier des charges du projet Trombine

## 4.1. Résumé du projet

Le projet TROMBINE est un projet interne qui a pour but la réalisation d'un annuaire amélioré permettant à chaque collaborateur de l'entreprise de retrouver les coordonnées d'autres collaborateurs suivant plusieurs critères de recherche : les noms, les tags et les zones de droits.

## 4.2. Contexte du projet

L'entreprise APSIDE n'a pas une organisation pyramidale classique. Certains collaborateurs sont encadrés par un chef de projet, avec une organisation classique, mais beaucoup d'entre eux sont sous l'égide de scrum-masters basés chez les clients ; d'autres rendent comptes directement au client pour les petits projets. L'entreprise met donc à disposition ses collaborateurs pour la réalisation de projets ponctuels ou à long terme. Certains collaborateurs travaillent chez le client, et d'autres sur le site de l'agence. Il faut donc permettre à chacun de retrouver d'autres collaborateurs suivant des tags, des sites d'agence, des clients, des fonctions et d'autres à encore définir.

Le projet a donc pour but de développer une application qui sera l'une des sources d'informations pour les collaborateurs Apside. Grâce à cette application un collaborateur pourra :

- Retrouver son supérieur hiérarchique ainsi que les membres de son équipe
- Trouver des informations utiles à propos de ses collaborateurs (fiche métier, agence de rattachement, fonction au sein d'Apside, etc.)

## 4.3. Objectifs du projet

Le projet a comme objectif de centraliser les données des collaborateurs ce qui permettra de faciliter l'accès à leurs informations. Par exemple, il sera possible de retrouver les collaborateurs possédant un rôle annexe à leur fonction.

Pour répondre à ces objectifs, la solution cible devra respecter les critères suivants :

- La solution doit être simple d'utilisation,
- La solution doit pouvoir gérer des habilitations différentes définies en fonction des types de profils,
- La solution doit permettre de stocker, modifier et supprimer les données des collaborateurs,
- La solution doit conserver la trace de toutes les modifications opérées sur les informations des collaborateurs : auteur, horodatage, ancienne et nouvelle valeur.

## 4.4. Spécifications techniques

### 4.4.1. Choix des technologies

Back



Front



Base de données


IDE


Versioning


Communication



Gestion de projet


### 4.4.2. Environnement

Des environnements de travail basés sur des conteneurs de développement, conçus par Baptiste DAVID, m'ont été fournis. Un pour le front, et un pour le back avec :




## 4.5. Spécifications fonctionnelles

Cette section définit les différents termes et concepts qui seront utilisés dans ce document et qui nécessitent une clarification préalable.

- Zone de droits : agence (ou groupe d'agences) Apside. Abréviation ZDD,
- Zone de droits 0 : Zone de droits mère, commune à tous les collaborateurs Apside,
- Tag : mots-clés servant à définir un collaborateur, qui peut être sa zone de droits, son métier (développeur, business analyst...), ses spécialités (dba, Angular...) ou encore son rôle au sein d'Apside (Référent Harcèlement...),
- Catégorie : groupe de tags (fonctions, compétences, client...).

# 5. Analyse « système fermé »

## 5.1. Analyse des acteurs et des uses cases

### 5.1.1. Définir les acteurs

**Utilisateur** : personne connectée à l'application Trombine.

**Lecteur :** utilisateur ayant un privilège de lecteur permettant de consulter toutes les données de tous les profils dans leur zone de droits.

**Collaborateur** : personne travaillant pour Apside.

**Manager de tag** : utilisateur responsable de tags lui étant affecté, peut au sein de son périmètre, affecter des collaborateurs à ses tags ou au contraire leur retirer ces tags.

**Administrateur de tag** : utilisateur ayant tous les droits sur la gestion de son/ses tags et utilisateurs de son/ses tags.

**Administrateur de zone** : utilisateur ayant tous les droits sur la gestion des tags et utilisateurs de sa zone.

**Super administrateur** : administrateur ayant accès à toute la plateforme, il peut configurer la plateforme et accorder les privilèges administrateur et manager à un collaborateur.

### 5.1.2. Définir les uses cases

**Gérer les droits** : le super administrateur affecte les différents rôles aux collaborateurs.

**Gérer les préférences d'affichage** : l'administrateur de zone modifie les informations liées à l'affichage.

**Gérer les utilisateurs** : l'administrateur de zone crée, modifie ou supprime des utilisateurs.

**Gérer les tags** : l'administrateur de tag crée, supprime des tags.

**Gestion des zones de droits** : l'administrateur de zone crée, modifie ou supprime des zones de droits.

**Gérer les tags qui lui sont affectés** : le manager de tag apporte des modifications aux tags qui lui sont affectés.

**Gérer les collaborateurs** : le manager de tag affecte ou supprime des collaborateurs sur les tags dont il est responsable.

**Gérer les inscriptions** : le manager de tag accepte ou refuse les collaborateurs sur les tags dont il est responsable.

**Gérer le profil** : un collaborateur peut mettre ou non ses informations visibles, modifier ses informations personnelles (état civil, moyens de contact, tags, réseaux, photo de profil).

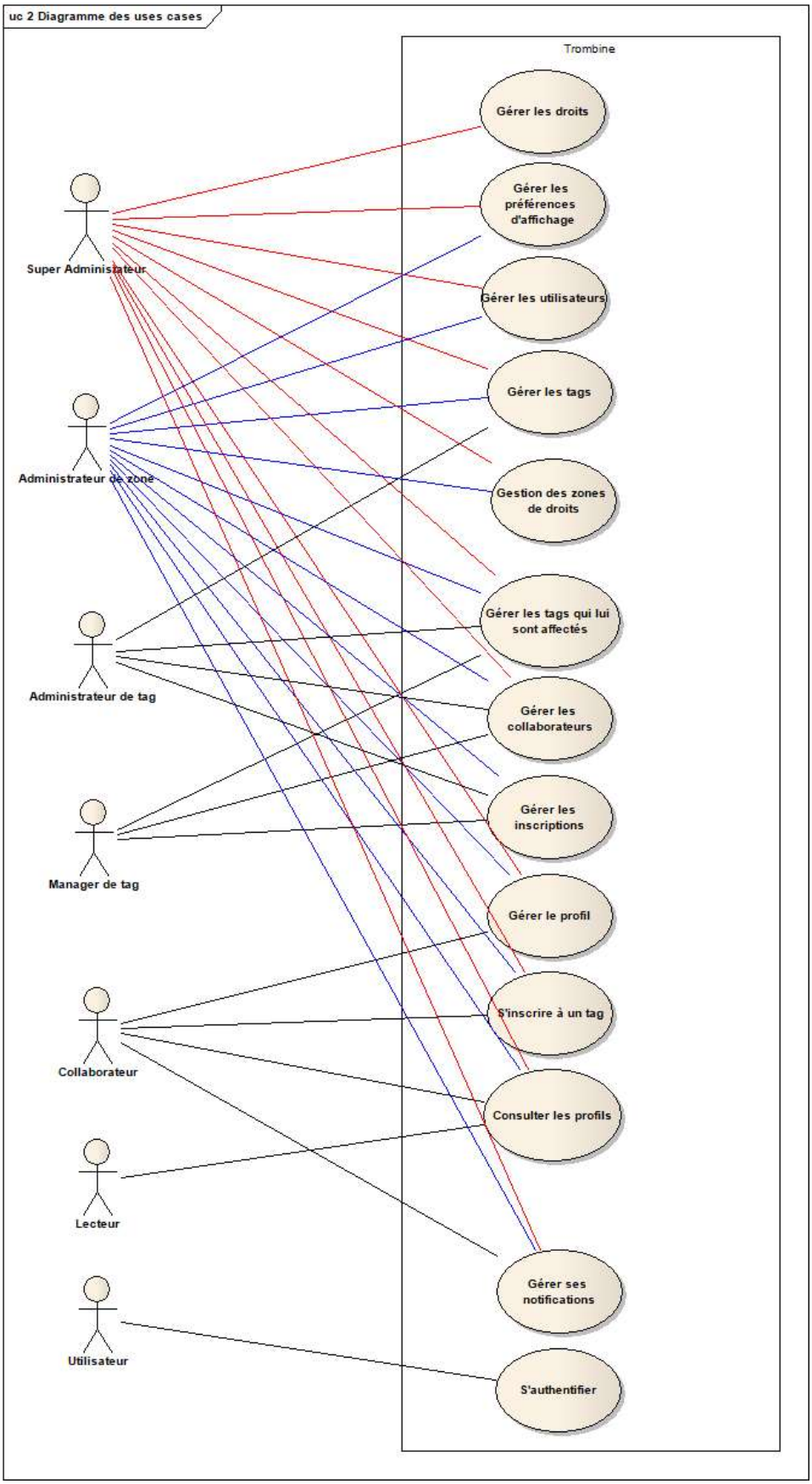
**S'inscrire à un tag** : un collaborateur peut s'affecter un tag provisoirement jusqu'à validation d'un responsable.

**Consulter les profils** : un lecteur peut consulter les profils complets des collaborateurs Apside, mais ne peut apporter aucune modification.

**Gérer ses notifications** : un collaborateur peut consulter ses notifications et ses demandes en cours.

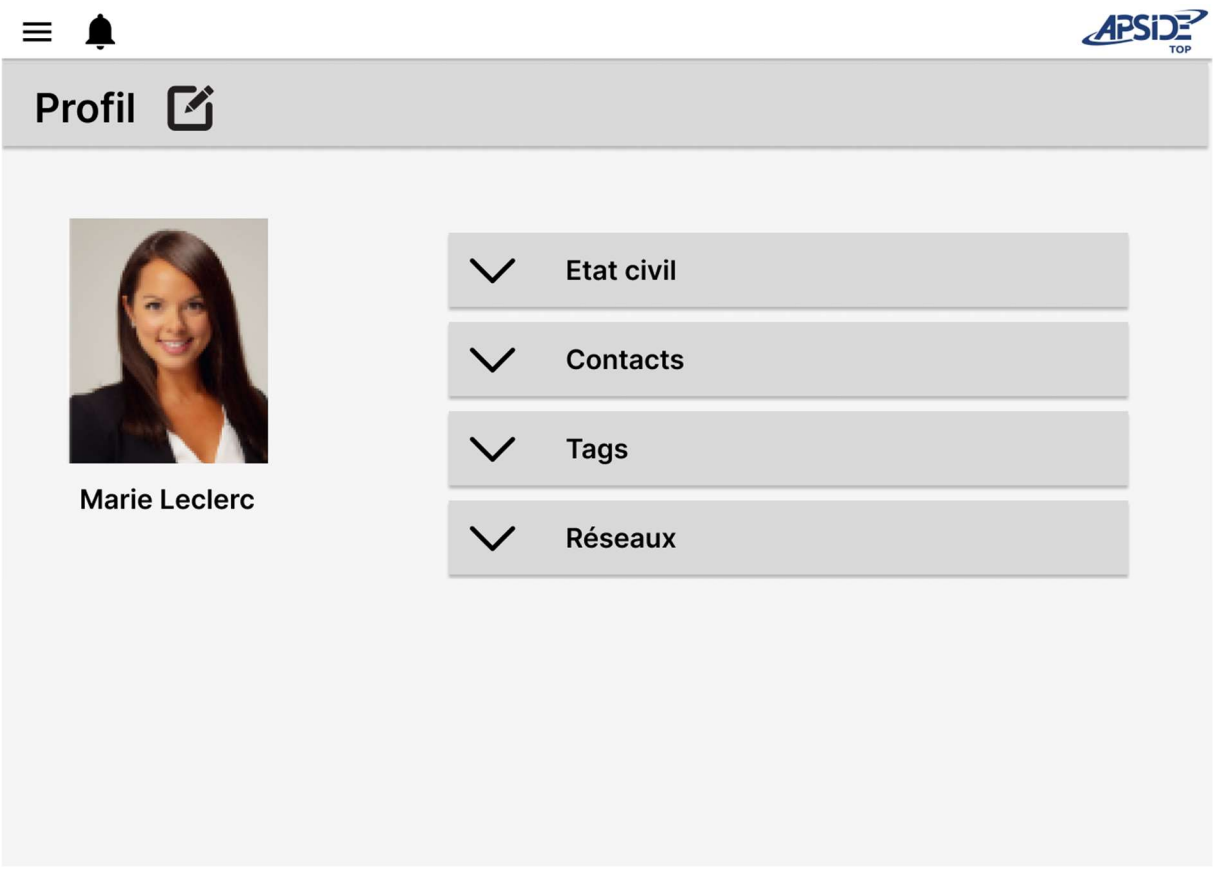
**S'authentifier** : l'utilisateur saisit son identifiant et son mot de passe afin de se connecter à l'application. Ce use case est pris en charge par Keycloak, le portail interne de connexion aux différentes applications déjà existantes chez Apside.

### 5.1.3. Diagramme des uses cases



## 5.2. Maquettage

### 5.2.1. Maquettes les plus représentatives

# Profil

Marie Corbeau

## Etat civil

| | |
|---|---|
| Nom : | Corbeau |
| Prénom : | Marie |
| Sexe : | F |
| Situation familiale | Marié(e) |
| Adresse postale | 26 rue des écoles |
| Ville : Tours | Code Postal : 37000 |

## Contacts

## Tags

## Réseaux

# Profil

Marie Corbeau

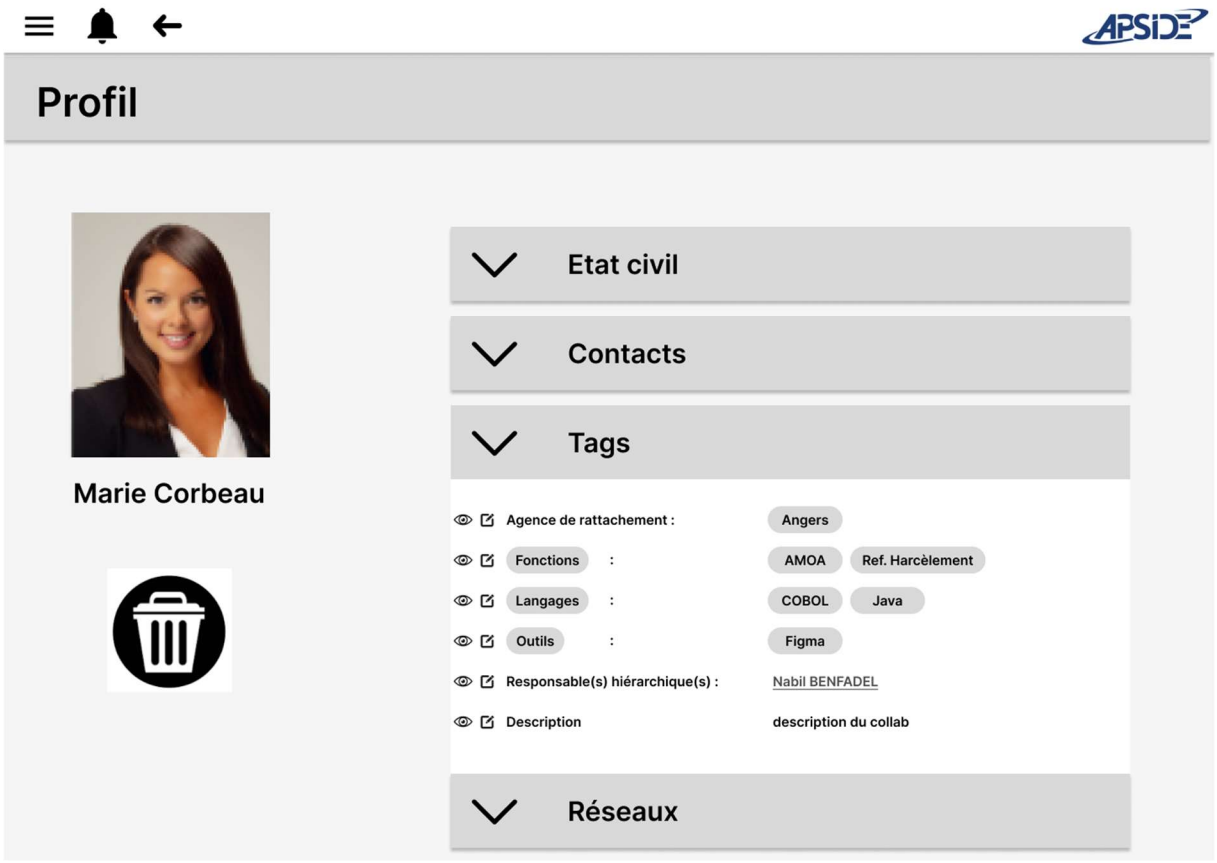
## Etat civil

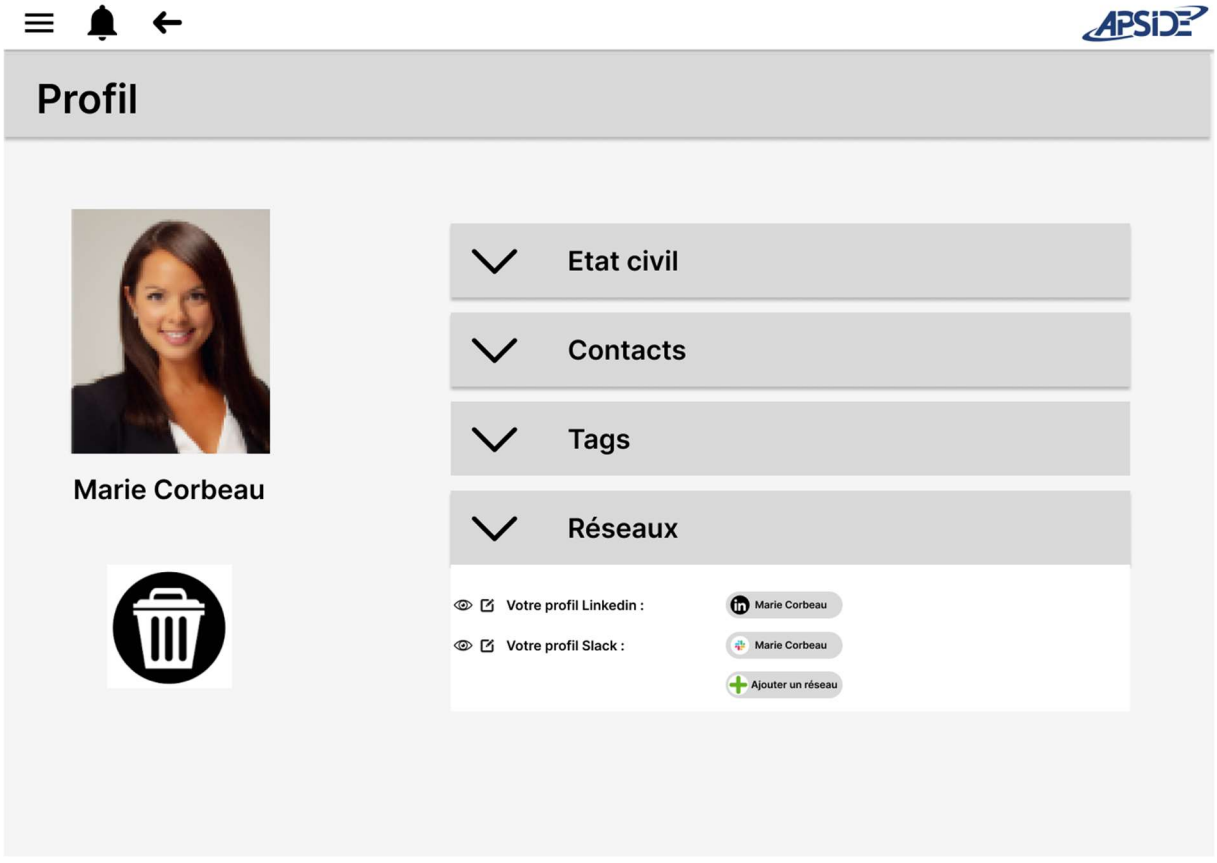
## Contacts

| | |
|---|---|
| Numéro de téléphone professionnel : | 01 23 45 67 89 |
| Numéro de téléphone personnel : | 01 23 45 67 89 |
| Ajouter n° de téléphone | |
| Mail Apside : | marie.corbeau@apside.com |
| Mail Client : | Cmarie@exemple.fr |
| Mail personnel : | mariedu37@gmail.com |
| Ajouter adresse mail | |

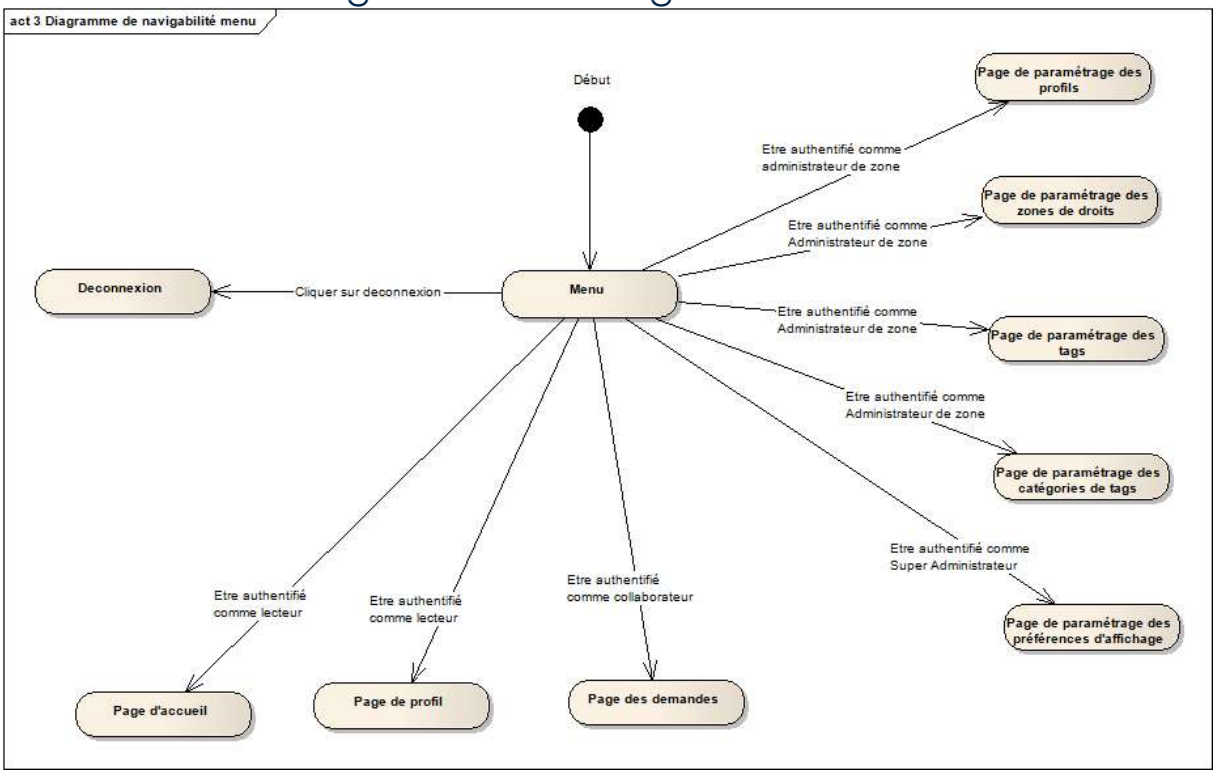
## Tags

## Réseaux


Profil
Marie Corbeau
Etat civil
Contacts
Tags
Agence de rattachement :
Angers
Fonctions
:
AMOA
Ref. Harcèlement
Langages
:
COBOL
Java
Outils
:
Figma
Responsable(s) hiérarchique(s) :
Nabil BENFADEL
Description
description du collab
Réseaux


Profil
Marie Corbeau
Etat civil
Contacts
Tags
Réseaux
Votre profil Linkedin :
Marie Corbeau
Votre profil Slack :
Marie Corbeau
Ajouter un réseau

### 5.2.2. Diagramme de navigabilité du menu



## 5.3. Sélection d'un use case

Use case : gérer le profil

### 5.3.1. Scénarios

Acteur : Collaborateur

Précondition : Être sur la page de profil en mode édition

Postcondition : Avoir sauvegardé les modifications

Scénario nominal :

1. Le collaborateur clique sur Tags.
2. L'application ouvre la section Tags.
3. Le collaborateur clique sur Ajout d'un Tag.
4. L'application affiche la liste des Tags.
5. Le collaborateur clique sur le Tag à ajouter.
6. L'application invite à enregistrer ou annuler l'opération.
7. Le collaborateur clique sur enregistrer.
8. L'application enregistre la modification.

Scénarios alternatifs :

3A1 Le collaborateur clique sur suppression d'un Tag

1. Reprise du scénario nominal en 6

1A2 Le collaborateur clique sur Etat civil

1. L'application ouvre la section Etat civil
2. Le collaborateur saisie une modification
3. Reprise du scénario nominal en 6

1A3 Le collaborateur clique sur Contacts

1. L'application ouvre la section Contacts
2. Le collaborateur saisie une modification
3. Le collaborateur sauvegarde
4. Reprise du scénario nominal en 6

1A4 Le collaborateur clique sur Réseaux

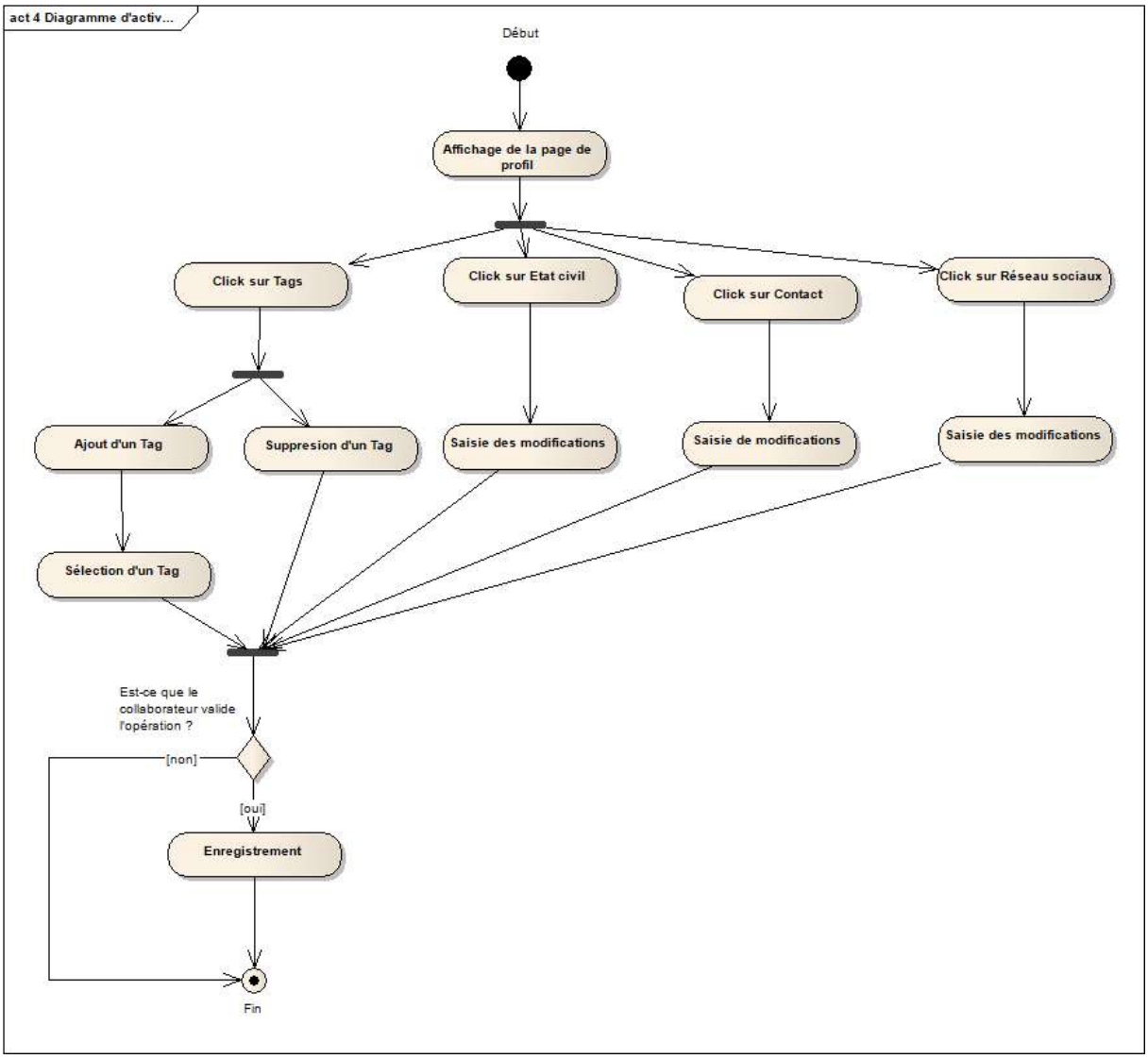
1. L'application ouvre la section Réseaux
2. Le collaborateur saisie une modification
3. Reprise du scénario nominal en 6

Scénarios exceptionnels :

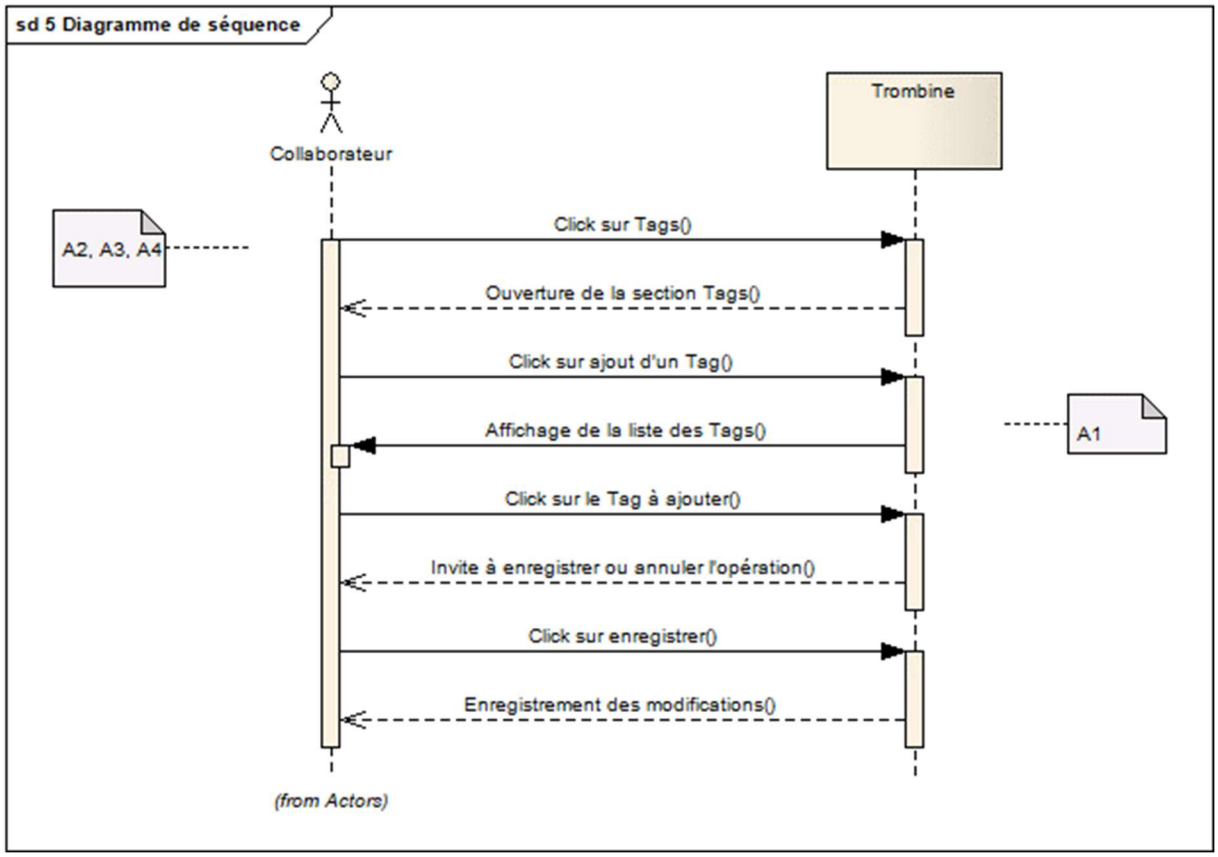
6E1 Le collaborateur clique sur annuler

1. L'application annule l'opération et revient à l'affichage de la page de profil

### 5.3.2. Diagramme d'activité

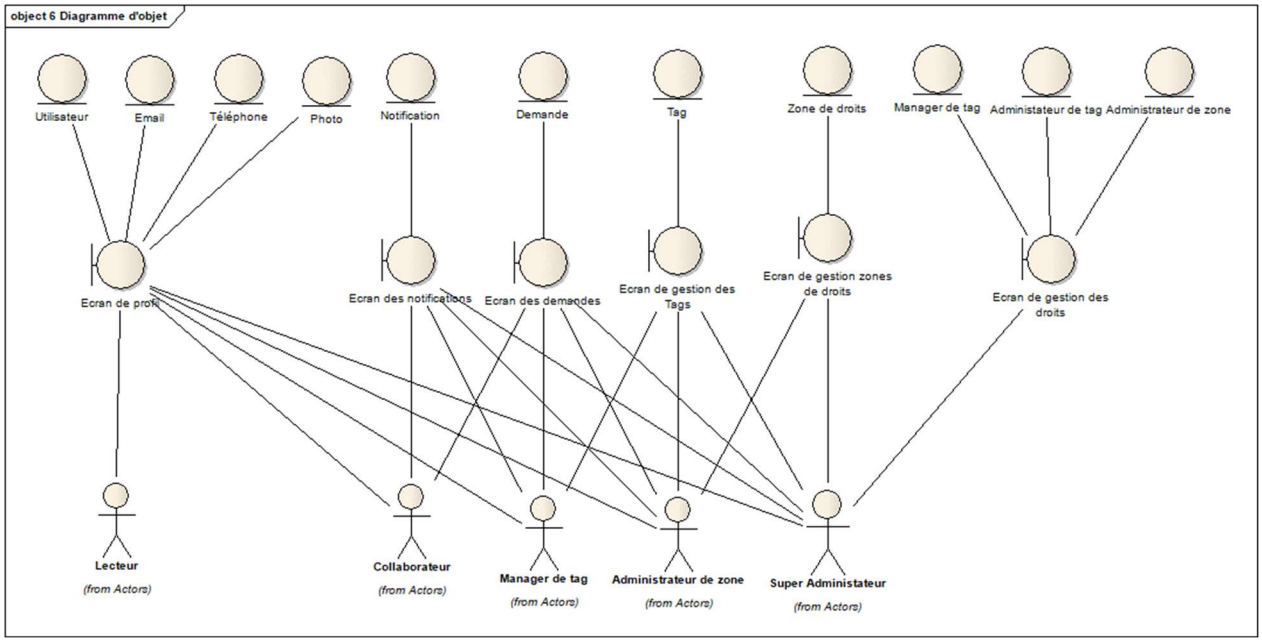


### 5.3.3. Diagramme de séquence système fermé

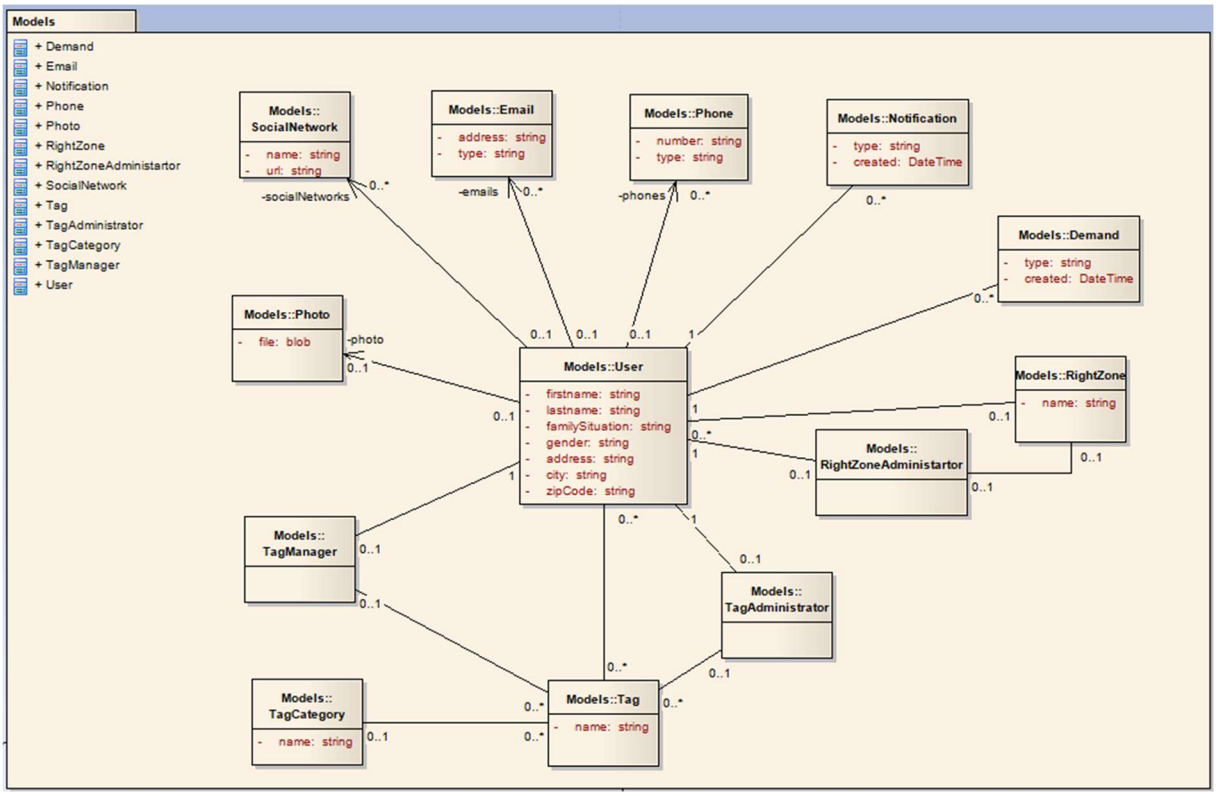



# 6. Conception

## 6.1. Diagramme d'objet



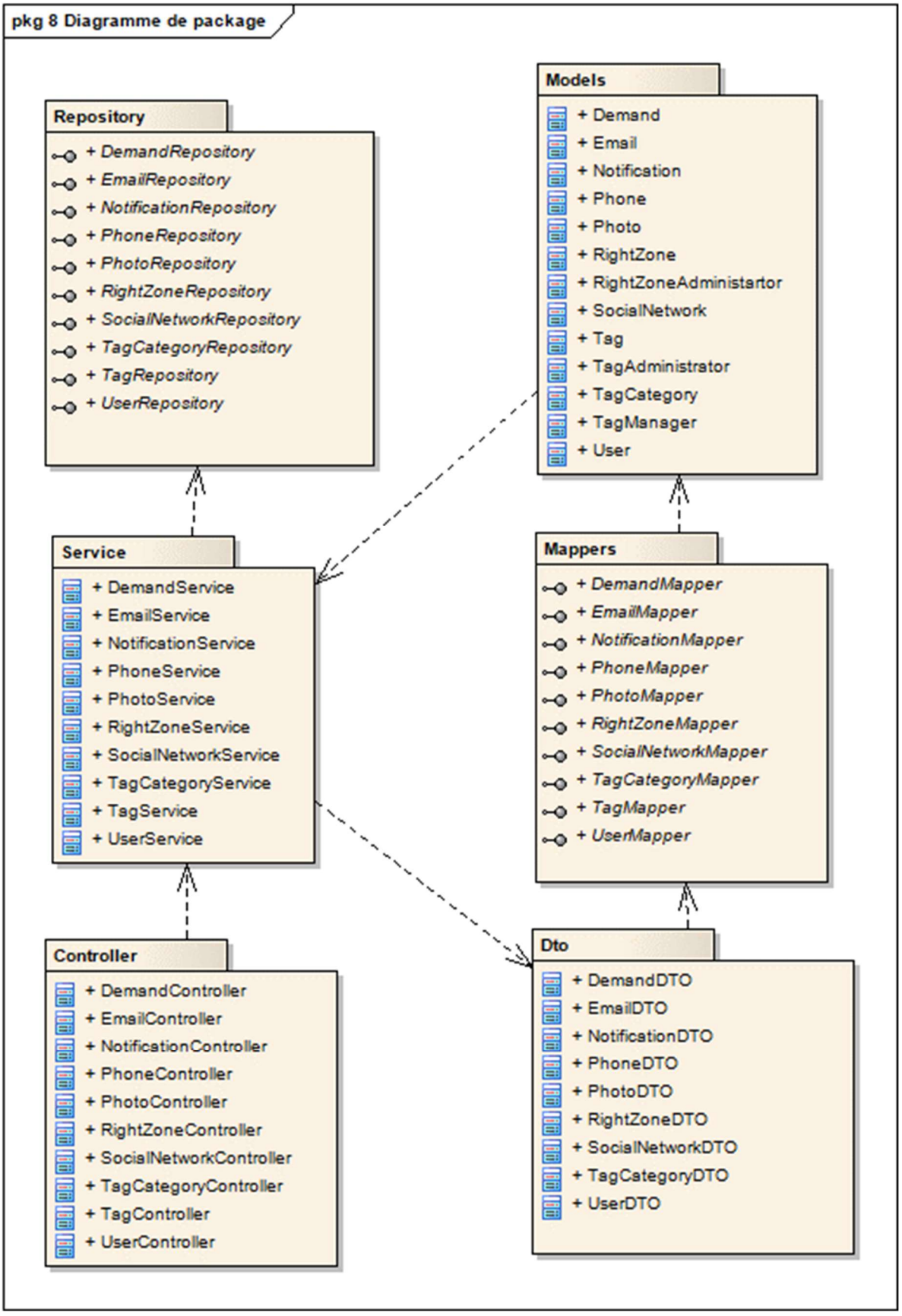
## 6.2. Diagramme de classe



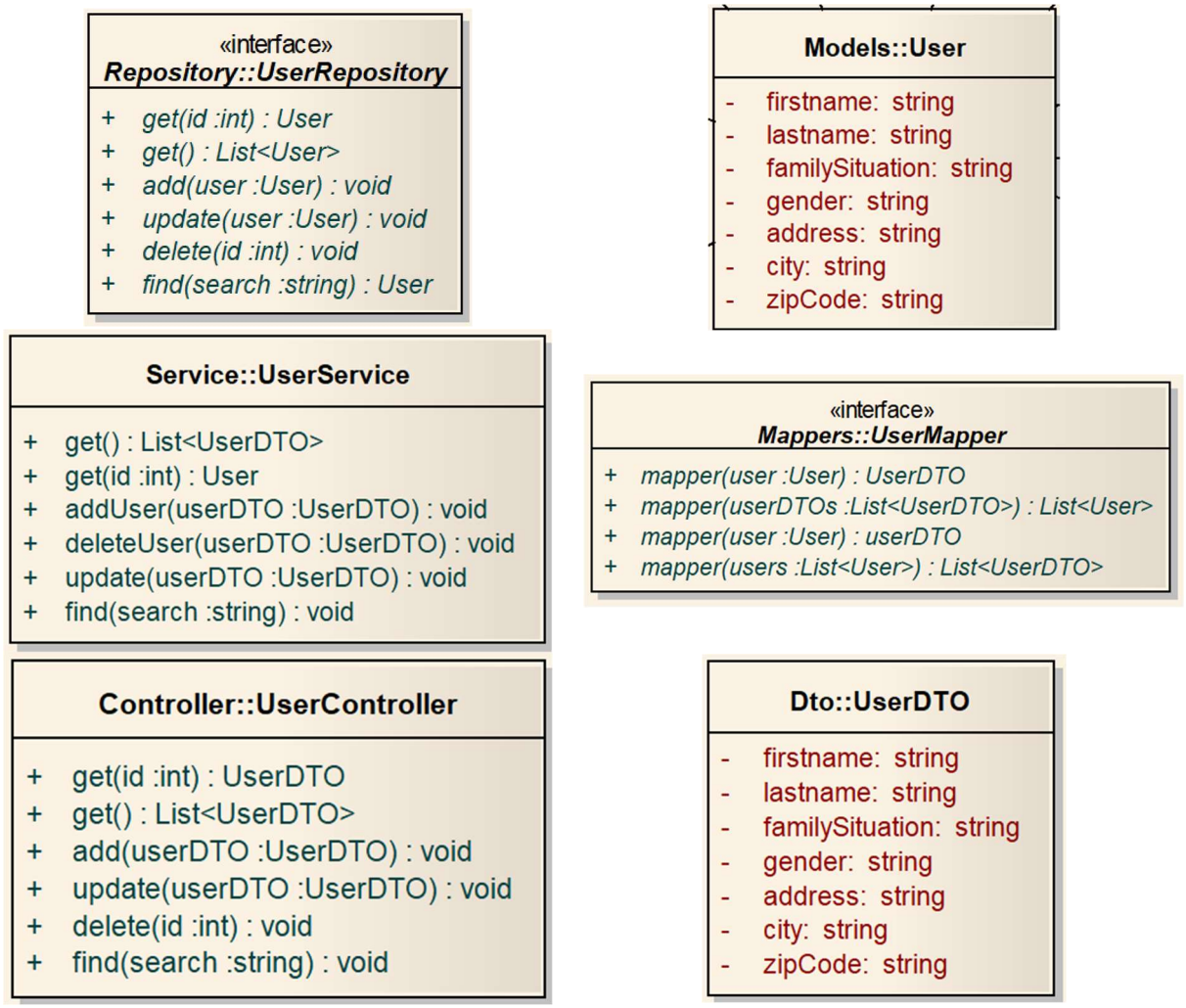

## 6.3. Détermination des couches

- **Controller**, qui va être la porte d'entrée du Frontend pour communiquer avec l'API.
- **Service,** qui va regrouper toute la logique métier, et qui va assurer la communication entre le Controller et le Repository.
- **Repository**, qui va avoir la charge de récupérer les données, dans notre cas, depuis la base de données.
- **Models**, qui va regrouper toutes les objets métiers nécessaires au fonctionnement de l'API.
- **Mapper**, qui va assurer la conversion des objets DTO en objets Models et inversement.
- **DTO** (Data Transfer Object), qui va permettre le transport des données à travers les couches, sans exposer les Models.

## 6.4. Diagramme de package



## 6.5. Classes et interfaces en détails



## 6.6. Base de données

### 6.6.1. Dictionnaire des données

| Entités | Mnémonique | Signification | Type(longueur) | Contraintes |
|---|---|---|---|---|
| trombine_user | id_user | Identifiant | Int(11) | Identifiant,ai |
| | lastname | Nom de famille | Varchar(50) | obligatoire |
| | firstname | Prénom | Varchar(50) | Obligatoire |
| | family_situation | Situation familiale(marié, pacsé, célibataire, non renseigné) | Varchar(50) | Obligatoire |
| | gender | Genre(feminin, masculin, autre, non renseigné) | Varchar(50) | Obligatoire |
| | description | Description | Varchar(255) | Facultatif |
| | address | Adresse | Varchar(255) | Facultatif |
| | city | Ville | Varchar(50) | Obligatoire |
| | zip_code | Code postal | Char(5) | obligatoire |
| tag | id_tag | Identifiant | Int(11) | Identifiant,ai |
| | name | Nom | Varchar(50) | Obligatoire |
| tag_category | id_tag_category | Identifiant | Int(11) | Identifiant,ai |
| | name | Nom | Varchar(50) | Obligatoire |
| social_network | id_social_network | Identifiant | Int(11) | Identifiant,ai |
| | name | Nom | Varchar(50) | Obligatoire |
| | url | Chemin d'accès | Varchar(255) | Obligatoire |
| email | Id_email | Identifiant | Int(11) | Identifiant,ai |
| | Address | Adresse | Varchar(50) | Obligatoire |
| | type | Type (personnel, professionnel) | Varchar(50) | Obligatoire |
| phone | id_phone | Identifiant | Int(11) | Identifiant,ai |
| | number | Numéro | Char(10) | Obligatoire |
| | type | Type(personnel, professionnel) | Varchar(50) | Obligatoire |
| photo | id_photo | Identifiant | Int(11) | Identifiant,ai |
| | file | Fichier | Bytea | Obligatoire |
| notification | id_notification | Identifiant | Int(11) | Identifiant,ai |
| | type | Type(tag, desscription, zone de droits) | Varchar(50) | Obligatoire |
| | created | Date de création | DateTime | Obligatoire, auto-générée |
| demand | id_demand | Identifiant | Int(11) | Identifiant,ai |
| | type | Type(en attente, validée, refusé) | Varchar(50) | Obligatoire |
| | created | Date de création | DateTime | Obligatoire, auto-générée |
| right_zone | id_right_zone | Identifiant | Int(11) | Identifiant,ai |
| | name | nom | Varchar(50) | Obligatoire |
| right_zone_administrator | id_right_zone_administrator | identifiant | Int(11) | Identifiant,ai |
| tag_manager | id_tag_manager | identifiant | Int(11) | Identifiant,ai |
| Tag_administrator | id_tag_administrator | identifiant | Int(11) | Identifiant,ai |

### 6.6.2. Règles de gestion

1 trombine_user possède 1 seule photo
1 photo est possédée par 1 trombine_user

1 trombine_user possède 0 ou plusieurs social_network
1 social_network est possédé par 0 ou 1 trombine_user

1 trombine_user possède 0 ou plusieurs email
1 email est possédé par 0 ou 1 trombine_user

1 trombine_user possède 0 ou plusieurs phone
1 phone est possédé par 0 ou 1 trombine_user

1 trombine_user est associé à 0 ou plusieurs tag
1 tag est associé à 0 ou plusieurs trombine_user

1 tag intègre 0 ou 1 tag_category
1 tag_category est intégrée par 0 ou plusieurs tag

1 tag_manager est 1 seul trombine_user
1 trombine_user est 0 ou 1 tag_manager

1 tag_manager gère 0 ou plusieurs tag
1 tag est géré par 0 ou 1 tag_manager

1 tag_manager est 1 seul trombine_user
1 trombine_user est 0 ou 1 tag_manager

1 tag_administrator administre 0 ou plusieurs tag
1 tag est administré par 0 ou 1 tag_administrator

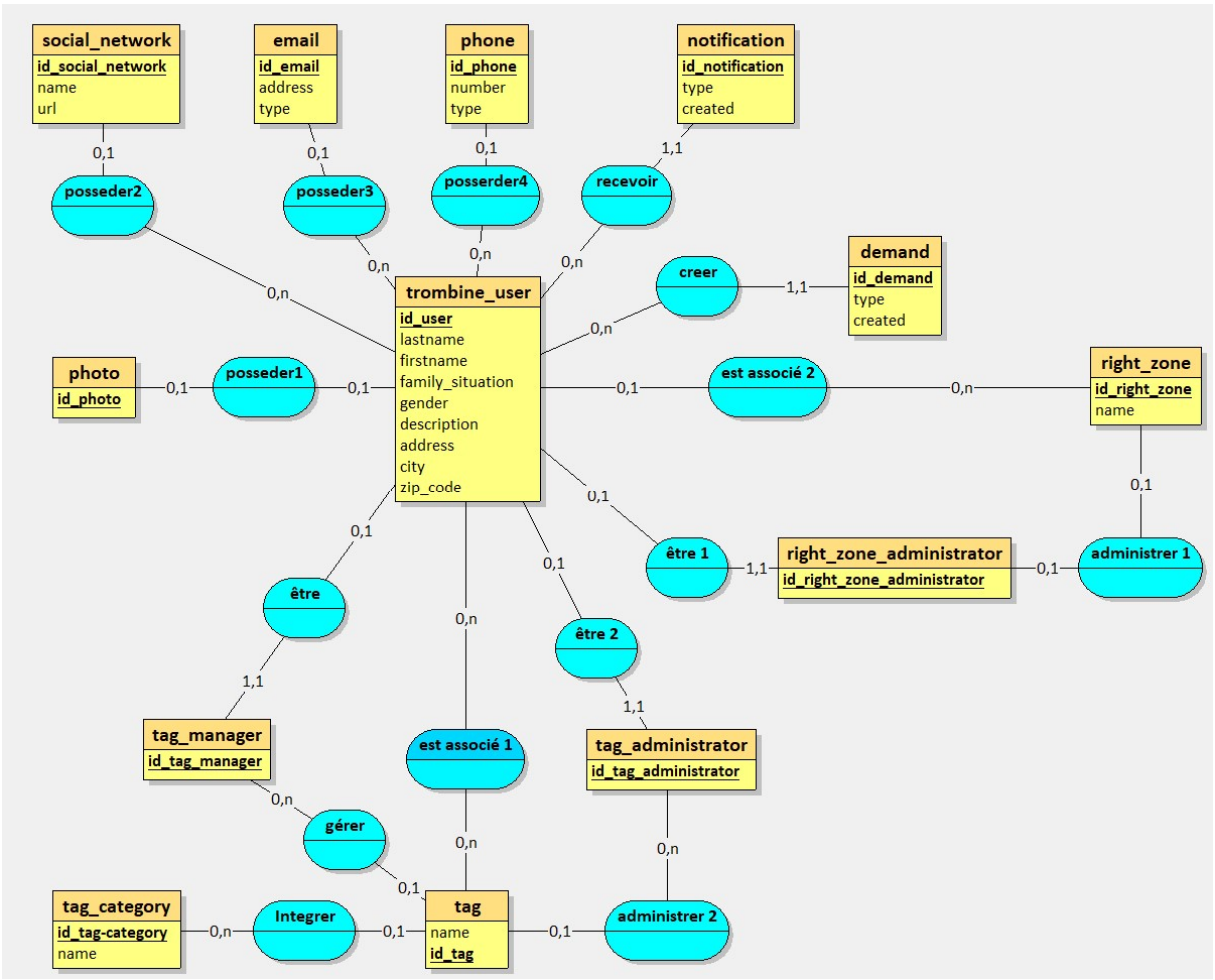
1 right_zone_administrator est 1 seule trombine_user
1 trombine_user est 0 ou 1 right_zone_administrator

1 right_zone_administrator administre 0 ou 1 right_zone
1 right_zone est administré par 0 ou 1 right_zone_administrator

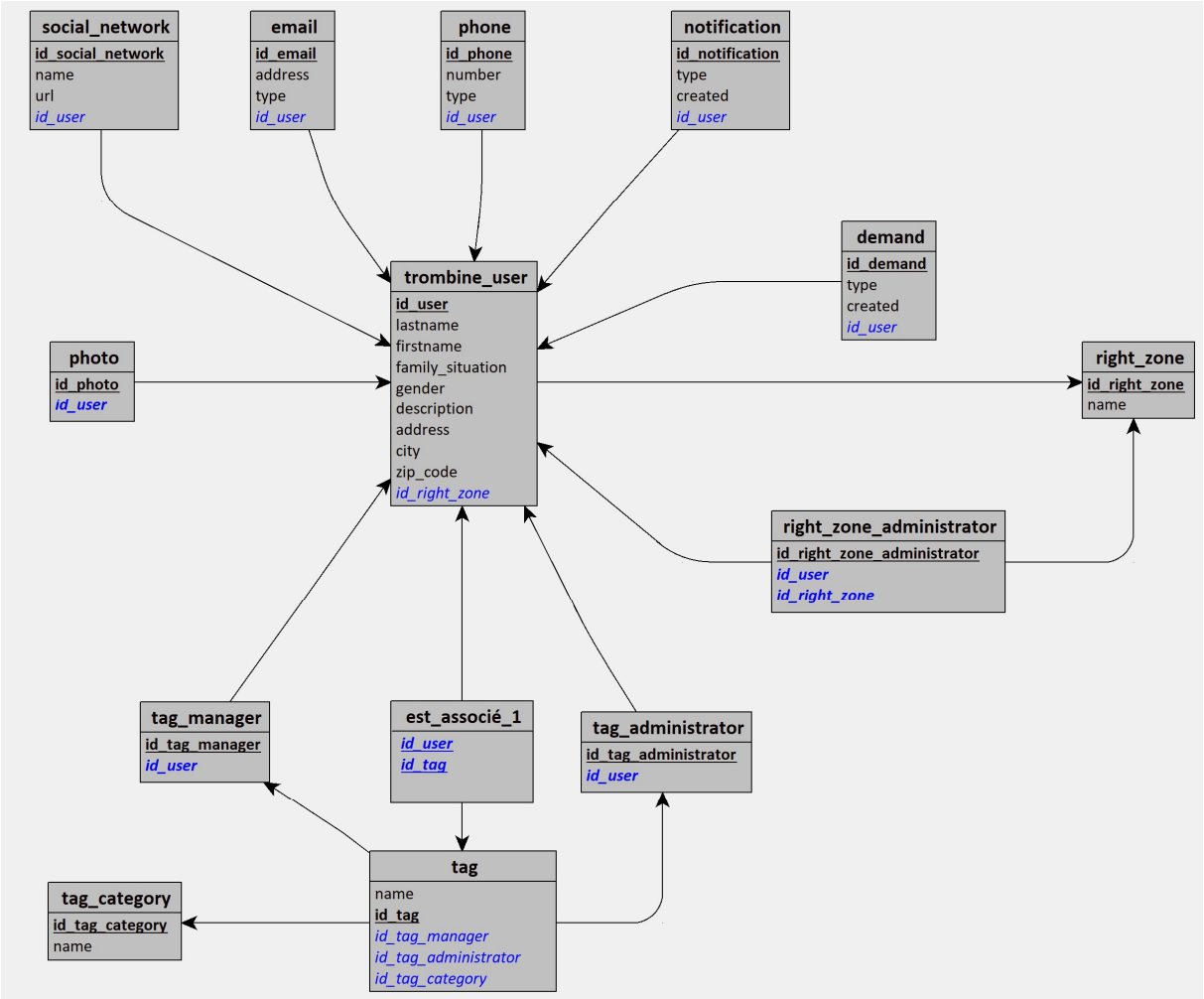
1 trombine_user reçoit 0 ou plusieurs notification
1 notification est reçu par 1 seul trombine_user

1 trombine_user crée 0 ou plusieurs demand
1 demand est créé par 1 seul trombine_user

### 6.6.3. Modèle conceptuel des données (MCD)



### 6.6.4. Modèle logique des données (MLD)



### 6.6.5. Diagramme Entités Relations (ERD)

J'ai débuté la conception seul mais lorsque Baptiste DAVID est arrivé sur le projet, il m'a mis sur Lucidchart afin de permettre un travail d'équipe et ainsi coller au plus près de l'organisation en entreprise.

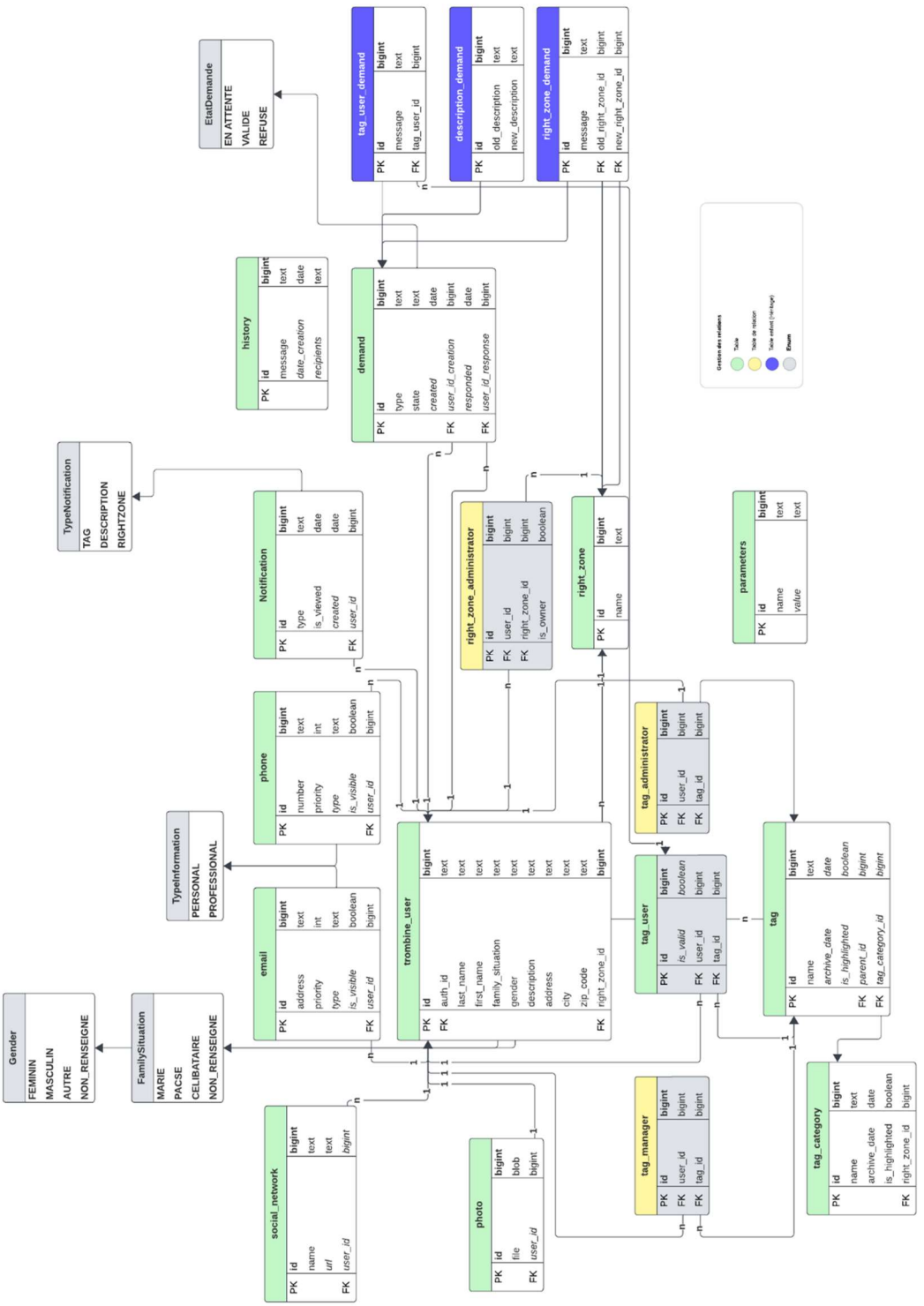
L'ERD a évolué en fonction des éclaircissements et des retours de Sébastien AUPETIT qui jouait le rôle du Product Owner. A l'arrivée de Simon PILLON et de Hugo PIERRE, les évolutions ont été encore plus rapides pour arriver au diagramme ci-dessous.

Si la plupart des entités (trop nombreuses pour les citer) ont été faciles à élaborer, il en a été tout autrement des relations entre les entités tag et trombine_user à cause de l'organisation non hiérarchique de l'entreprise. Il fallait tenir compte qu'un tag avait un manager, qui gère uniquement l'attribution de tags, à la différence d'un tag_administrator qui lui a la possibilité en plus d'en créer, d'en modifier ou d'en supprimer.

S'ajoute à cela l'existence de right_zone qui définissent des zones de droits pour les tags_manager et les tags_administrator. Il fallait aussi ajouter un right_zone_administrator qui gère les right_zone. Les entités tag_user_demand, description_demand, et right zone_demand servent à gérer les différentes demandes d'attribution de tag, d'attribution de right_zone, et enfin les demandes de modification de description.

Indépendamment des autres entités, on retrouve history qui se justifie par la nécessité de garder toute trace de modification des autres entités, ainsi que l'entité parameters pour garder en mémoire des configurations de paramètres afin de permettre d'avoir des gestions de paramètres différentes suivant les choix des différents administrateurs(rôle).

Enfin, contrairement à ce qui m'avait été enseigné en formation concernant le type Varchar, Sébastien nous a demandé de les passer en Text afin de tenir compte des futures évolutions du projet. En effet, les limitations en nombre de caractères pourraient présenter une limitation nécessitant une reprise de la base de données. De plus, les performances des équipements actuels ne nécessitent plus ce genre de limitations.


EtatDemande
EN ATTENTE
VALIDE
REFUSE
tag_user_demand
description_demand
right_zone_demand
history
demand
TypeNotification
TAG
DESCRIPTION
RIGHTZONE
Notification
right_zone_administrator
right_zone
parameters
phone
TypeInformation
PERSONAL
PROFESSIONAL
tag_administrator
email
trombine_user
tag_user
tag
Gender
FEMININ
MASCULIN
AUTRE
NON_RENSEIGNE
FamilySituation
MARIE
PACSE
CELIBATAIRE
NON_RENSEIGNE
social_network
tag_manager
tag_category
photo

# 7. Gestion du projet

## 7.1. Environnement humain

J'ai débuté seul sur le projet, sous la direction de Sébastien AUPETIT, avec la nécessité de me former sur quatre technologies que je ne connaissais pas, à savoir Java, Spring, Angular et Material. Mes premières semaines dans l'entreprise ont été rythmées par une première formation sur Java et sur Spring. A l'arrivée de Baptiste DAVID sur le projet, celui-ci a vraiment démarré avec la création des conteneurs de développement, mais sans réelle organisation du travail, car Baptiste n'était là que pour répondre à mes questions de code les lundis et mardis. Il a ensuite réellement participé au projet et m'a aidé à revoir mon diagramme Entités Relations (ERD). J'ai alors pu faire mes premières implémentations sous Java, Spring et en SQL.

Puis, sont arrivés Simon PILLON en tant que développeur, et Hugo PIERRE en tant que chef de projet. Les éclaircissements sur le projet se faisant, j'ai effectué une nouvelle implémentation back du projet avec un nouvel ERD. Une organisation des tâches a alors été mise en place pour dispatcher le travail par Hugo. Une tentative de premier sprint a alors été effectuée, sans succès, dû, à une complexité de compréhension des difficultés du projet. Il a fallu renoncer à ce premier sprint, mais le découpage des tâches a cependant été achevé. Des réunions quotidiennes ont été mises en place afin de discuter de l'avancement du projet.

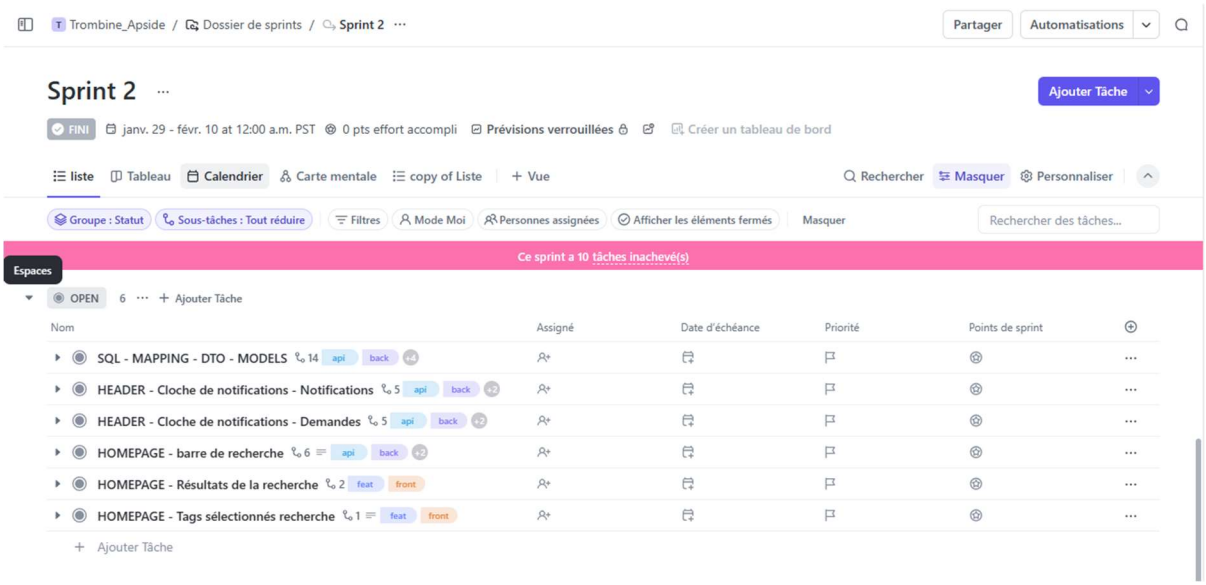
A l'arrivée d'Antoine PINARD, le travail sur les spécifications fonctionnelles a été revu. Sont ensuite arrivés Christopher VALLOT, dans un premier temps, Maxence GRIAS et Vincent DUGUET, un nouveau sprint de quinze jours a été mis en place. L'implémentation que j'avais précédemment faite, a été reprise pour pouvoir la découper en petites Pull-request afin d'être validées par un tecklead avant de pouvoir la merge sur la branche develop du Repository Github créé à cet effet. Mais le début de ce sprint s'est accompagné du départ de Simon et d'Hugo.

A ce moment du projet, il m'a fallu me former sur Angular, je n'ai donc pu faire que trois implémentations et pull-requests associées. De plus, les implémentations se sont limitées au Models, DTO, Mapper, et aux requêtes SQL de création des bases.

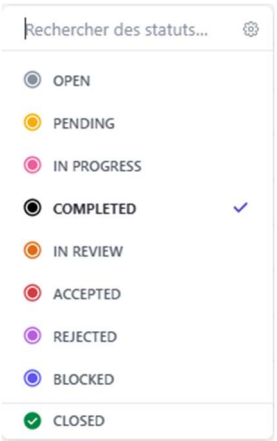
## 7.2. Organisation générale du planning

Le processus de développement est organisé de façon AGILE, via une variation de la méthode SCRUM. Les sprints sont d'une durée de deux semaines.

Le sprint 1 a été un tour de chauffe, et n'a pas été très rentable. Le sprint 2 a été plus efficace avec un meilleur découpage des tâches. Il a été organisé par Hugo PIERRE sur Clickup.

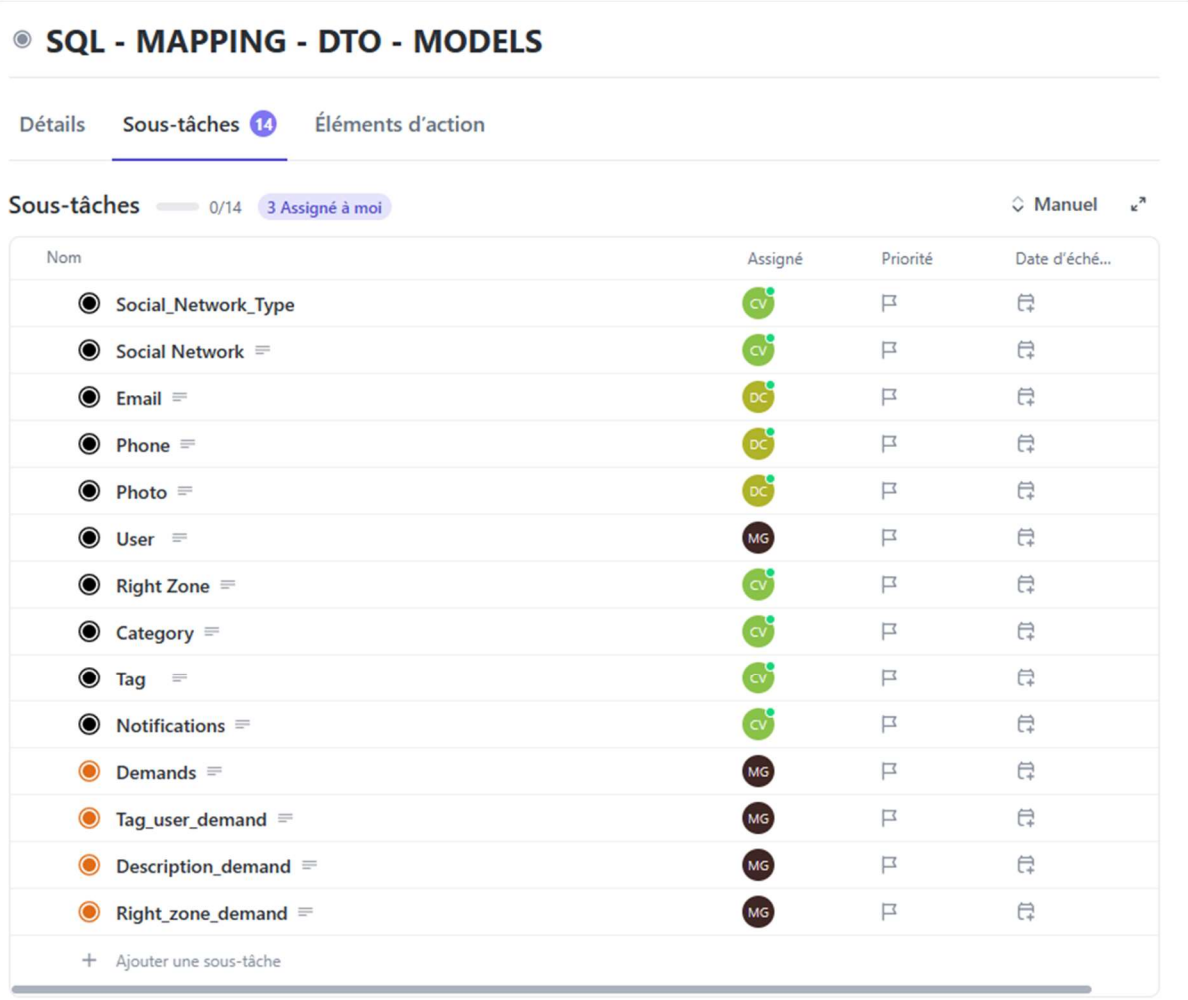



Chaque tâche est ensuite découpée en sous-tâches que chacun s'assigne. Comme étant tous en mode découverte, aucun temps n'avait été programmé pour chaque sous-tâche. Après s'être attribué une sous-tâche, son statut était modifié pour passer par différentes étapes :




- Open pour indiquer qu'elle était disponible.
- Pending quand on se l'assignait.
- In Progress quand elle était en cours d'implémentation.
- In review quand elle était le sujet d'une pull-request, et en attente de validation de cette dernière.
- Accepted quand la pull-request avait été validée.
- Completed quand le merge sur github avait été fait.

La liste des sous-tâches pour la tâche SQL – MAPPING – DTO -MODELS était la suivante :




Le sprint 2 est ponctué d'une Review de sprint où chacun peut faire le point sur ce qui s'était bien passé, sur les points d'amélioration, et sur son ressenti.

# 8. Le cahier des charges du projet Annuaire

Le projet Trombine étant très important par sa taille et sa complexité, il m'a fallu partir sur ce projet afin de pouvoir mener un projet à terme et remplir un maximum des conditions d'obtention du titre.

## 8.1. Objectif

Le projet Annuaire a pour but de développer une application qui permettra de d'afficher les informations de contact des employés d'une entreprise. On pourra ainsi

- Trouver les nom, prénom, adresse d'une personne,
- Trouver ses emails personnels et professionnels,
- Trouver ses téléphones personnels et professionnels,

## 8.2. Exigences

Le projet Annuaire permettra :

- De rechercher des contacts par nom et prénom,
- D'afficher les informations des contacts de façon claire,
- D'ajouter, modifier ou supprimer des contacts,
- D'être compatible avec les navigateurs web,

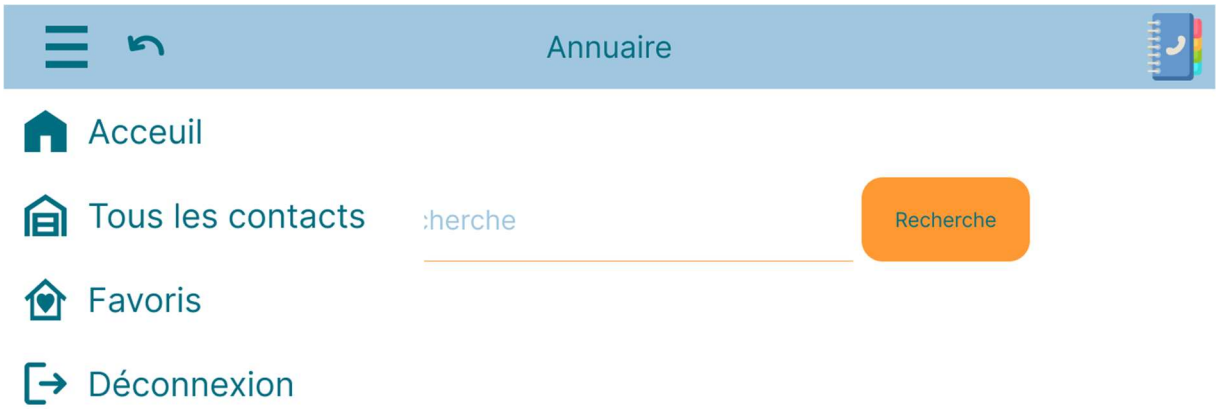
## 8.3. Spécifications techniques

Le développement du Backend se fera sous Java et Spring Boot par l'intermédiaire de l'IDE IntelliJ. Le développement du Frontend se fera sous Angular et utilisera Material par l'intermédiaire de l'IDE Visual Studio Code. La persistance des données sera gérée par PostGreSQL. Le versioning sera assuré par Git

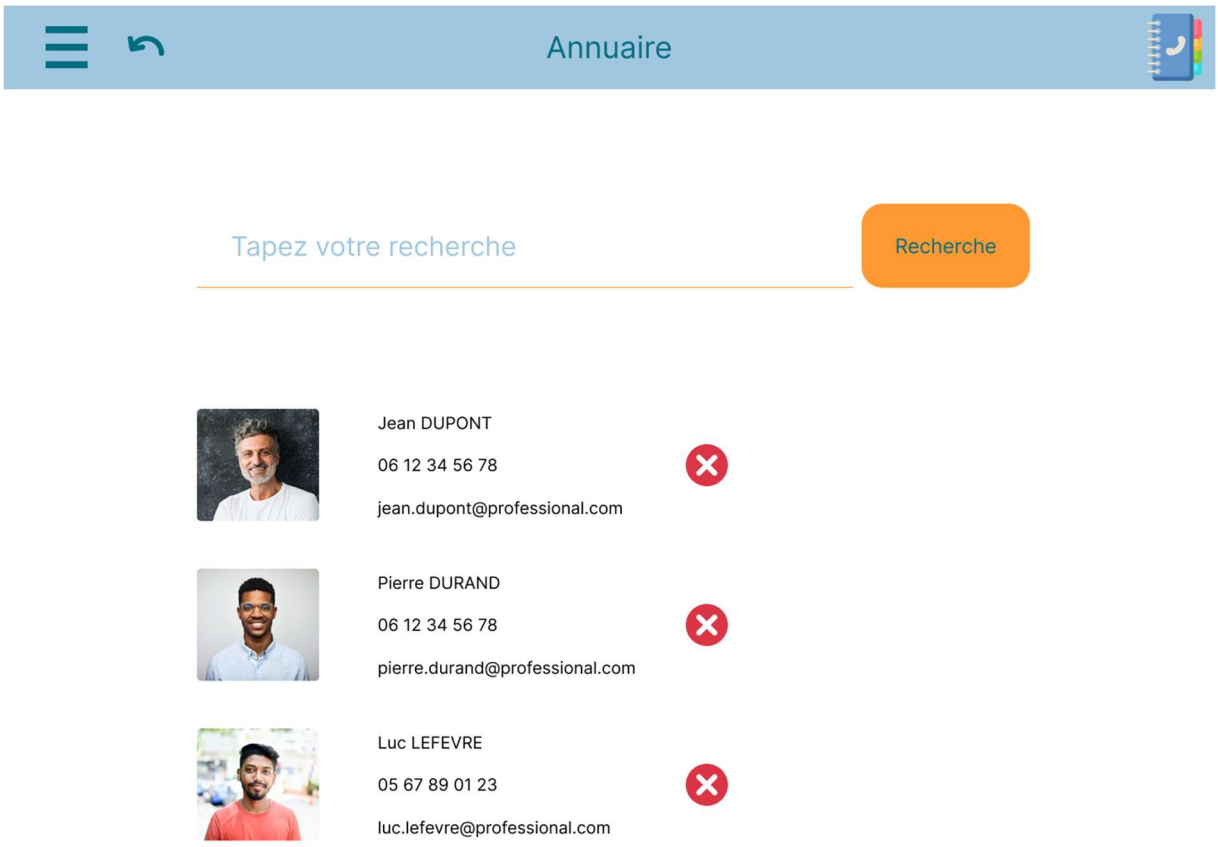
## 8.4. Spécifications fonctionnelles

### 8.4.1. Maquette des écrans

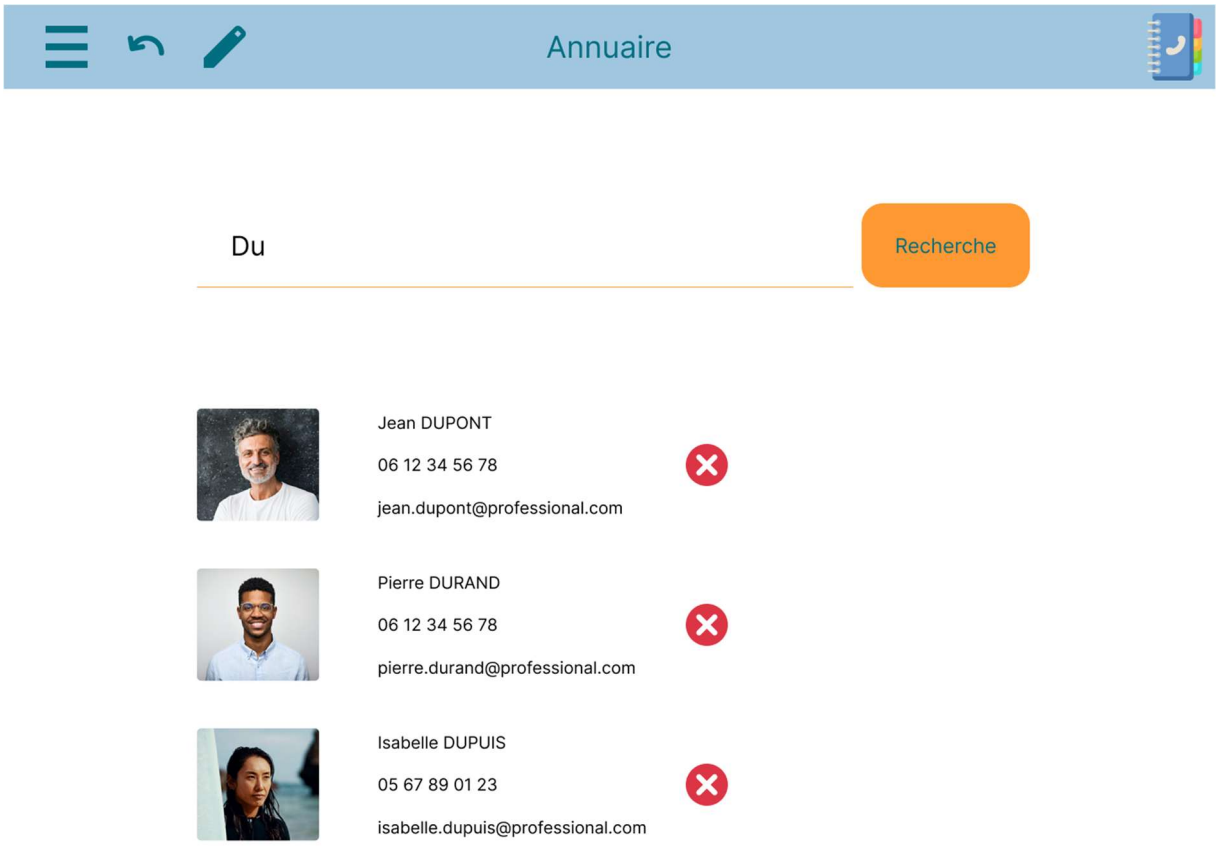

**PAGE D'ACCUEIL**




PAGE D'ACCUEIL AVEC MENU




PAGE D'ACCUEIL AVEC TOUS LES CONTACTS

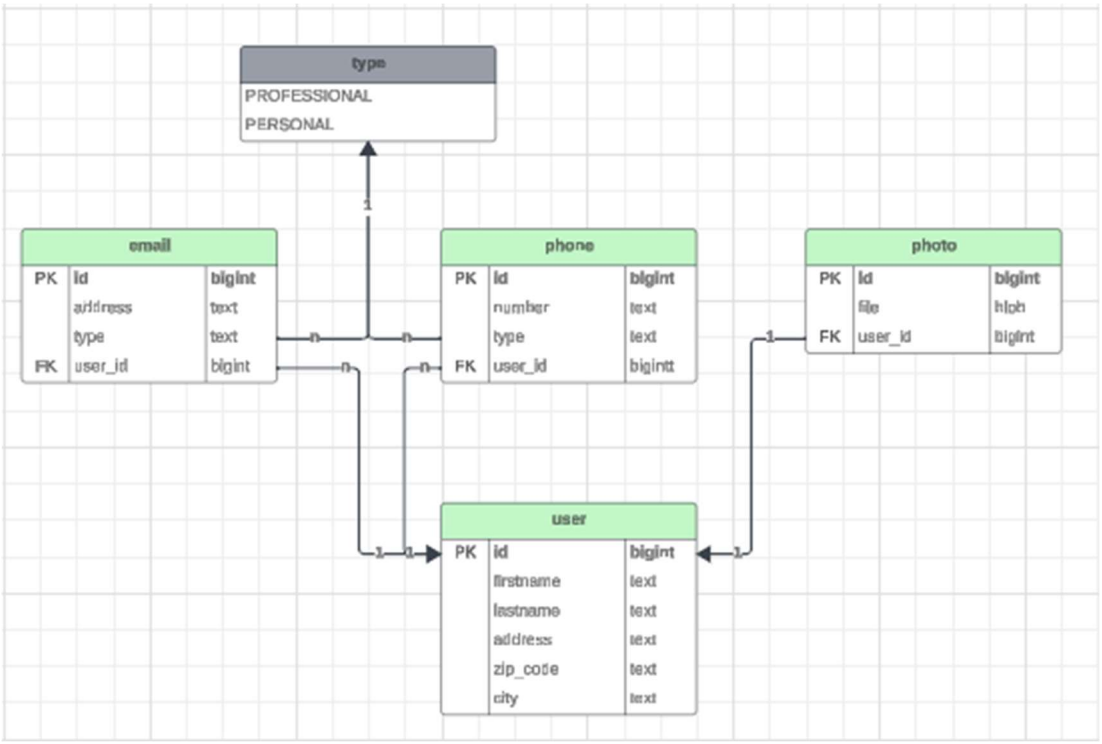



## PAGE D'ACCUEIL AVEC DES RESULTATS DE RECHERCHE


## PAGE DE PROFIL


**PAGE DE PROFIL EN MODE EDITION**

### 8.4.2. Diagramme Entités Relations (ERD)



## 8.5. Implémentation du Backend du projet Annuaire

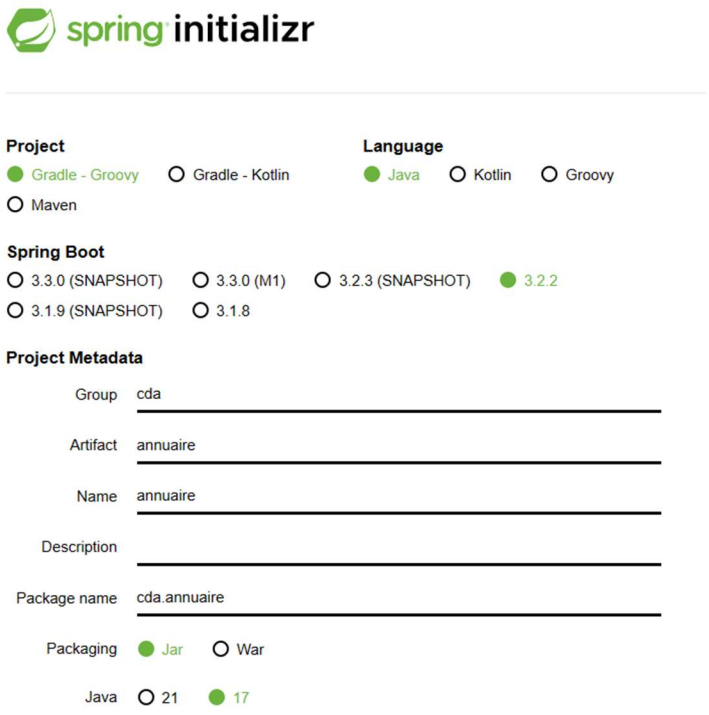
### 8.5.1. Configuration du projet

Le Backend est implémenté avec l'IDE IntelliJ. Sa création a nécessité un paramétrage différent du projet Trombine, que je vais détailler.

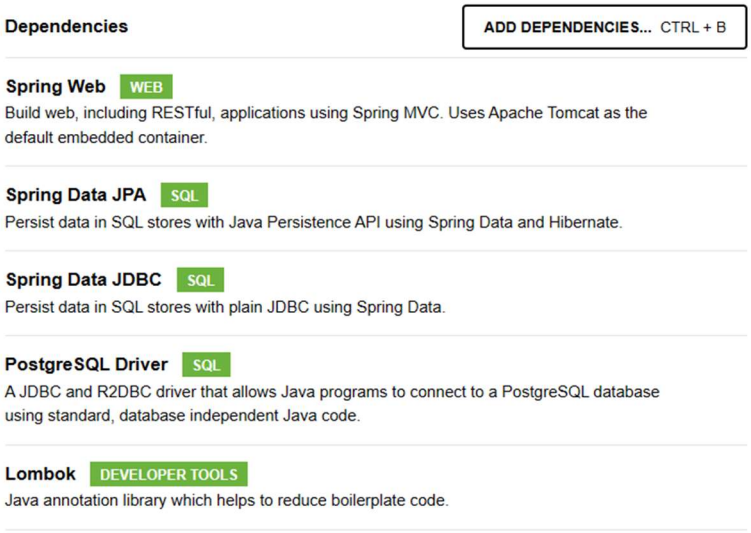
Tout commence par spring initalizr où l'on choisit :

- Le type de projet : Maven,
- Le langage : Java,
- La version de Spring Boot : 3.2.2,

On remplit ensuite les metadata, on sélectionne le packaging jar et la version de Java, 17 dans notre cas.




Viennent ensuite la sélection des dépendances nécessaires au projet




On génère le tout, ce qui nous crée un fichier zip à décompresser que l'on va ouvrir ensuite dans IntelliJ avec un pom.xml déjà configuré.

Reste encore à configurer le fichier application.properties :

```
spring.datasource.url=jdbc:postgresql://localhost:5432/postgres
spring.datasource.username = postgres
spring.datasource.password = azer

spring.jpa.properties.hibernate.dialect = org.hibernate.dialect.PostgreSQLDialect
spring.jpa.properties.hibernate.jdbc.lob.non_contextual_creation = true

spring.jpa.hibernate.ddl-auto=update
spring.jpa.show-sql=true

spring.liquibase.change-log=classpath:/changelog/changelog-master.xml
spring.liquibase.enabled=true
logging.level.liquibase = INFO
```

On saisit l'url pour accéder à postgreSQL, avec le nom d'utilisateur et le mot de passe associé pour que l'application puisse savoir où se situe la base de données.

On configure ensuite hibernate qui assure la communication entre la base de données et l'application.

Reste enfin à configurer liquidbase en lui indiquant le chemin du fichier changelog.xml (que l'on crée dans la volée dans le dossier ressources) qui va participer à la création des tables de la base de données. A savoir que sous Spring Boot les tables sont automatiquement générées une fois l'accès à PostgreSQL configuré, mais dans un souci de contrôle, on l'associe à liquibase, ce qui nous permet d'écrire les scripts SQL de création des tables.

Le projet est organisé en couches avec :

- Un package **controller**, va être la porte d'entrée du Frontend pour communiquer avec l'API,
- Un package **service**, va regrouper toute la logique métier, et va assurer la communication entre le controller et le repository,
- Un package **repository**, va être en charge de récupérer les données, dans notre cas, depuis la base de données
- Un package **models**, va regrouper tous les objets métiers nécessaires au fonctionnement de l'API
- Un package **dto**, va servir une version allégée des models au service et qui va éviter d'exposer toutes les données en sortie de l'API
- Un package **mapper**, va assurer la conversion des objets DTO en objets models et inversement

Maintenant que tout est configuré, on peut passer à l'implémentation.

### 8.5.2. Classe User

```
@Getter
@Setter
@Entity
@Component
@Table(name = "a_user")
```

Avant la déclaration de la classe, on trouve cinq annotations :

**@Getter** va générer automatiquement les accesseurs.

**@Setter** va générer automatiquement les mutateurs.

**@Entity** signifie que la classe est mappée avec une table de la base de données.

**@Component** indique qu'une classe est un composant Spring. C'est une directive pour le conteneur Spring, lui indiquant de prendre en charge l'instanciation, la configuration et l'assemblage de ces classes automatiquement. Les classes annotées avec @Component sont automatiquement détectées et gérées par Spring lors de l'analyse du chemin de classe.

**@Table** spécifie que la classe est mappée plus précisément avec la table « a_user ». On utilise cette annotation uniquement lorsqu'il faut personnaliser le mappage par défaut fourni par JPA.

```java
public class User {

    /** Identifiant de l'utilisateur */
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    /** Prénom de l'utilisateur */
    @Column(columnDefinition = "TEXT")
    private String firstname;
    /** Nom de famille de l'utilisateur */
    @Column(columnDefinition = "TEXT", nullable = false)
    private String lastname;
    /** Adresse de l'utilisateur */
    @Column(columnDefinition = "TEXT")
    private String address;
    /** Code postal de l'utilisateur */
    @Column(columnDefinition = "TEXT")
    private String zipCode;
    /** Ville de l'utilisateur */
    @Column(columnDefinition = "TEXT")
    private String city;
    /** Liste des téléphones associés à l'utilisateur */
    @OneToMany(cascade = CascadeType.ALL, mappedBy = "user")
    private List<Phone> phones;
    /** Liste des emails associés à l'utilisateur */
    @OneToMany(cascade = CascadeType.ALL, mappedBy = "user")
    private List<Email> emails;
    /** Photo de l'utilisateur */
    @OneToOne(mappedBy = "user")
    private Photo photo;
}
```

Dans la déclaration de la classe, on retrouve les annotations suivantes :

**@Id** est utilisée dans le contexte de la Java Persistence API (JPA) pour indiquer qu'un champ particulier d'une classe d'entité représente la clé primaire de cette entité dans la base de données.

**@GeneratedValue** avec la stratégie **GenerationType.IDENTITY** est utilisée dans le contexte de la Java Persistence API (JPA) pour indiquer que la valeur de la clé primaire d'une entité doit être générée automatiquement par la base de données.

**@Enumerated** est utilisée avec JPA (Java Persistence API) pour spécifier comment une valeur d'énumération est persistée dans une base de données. EnumType.STRING indique que l'énumération doit être stockée sous forme de chaîne de caractères dans la base de données.

**@Column** permet de définir précisément quel type doit être renseigné lors de la création de la base lorsqu'il est associé à **columnDefinition**. En effet, par défaut, le type généré pour un String en Java est un Varchar(255). Voulant que ce soit un type Text, je l'indique. S'ajoute à cela **nullable = false** qui indique que la valeur dans colonne ne peut être nulle.

**@OneToMany** est utilisée dans Hibernate et JPA (Java Persistence API) pour établir une relation de type "un à plusieurs" entre deux entités. Elle indique qu'un seul objet d'une entité est associé à plusieurs objets d'une autre entité. Cette annotation est

souvent utilisée dans le cadre des relations entre les tables dans une base de données relationnelle.

**@OneToOne** est utilisée pour établir une relation de type "un à un" entre deux entités. Cela signifie qu'un objet d'une entité est associé à exactement un objet d'une autre entité.

### 8.5.3. Classe UserDTO

```java
@Getter
@Setter
@AllArgsConstructor
public class UserDTO {
    /** Identifiant de l'utilisateur */
    private Long id;
    /** Prénom de l'utilisateur */
    private String firstname;
    /** Nom de l'utilisateur */
    private String lastname;
    /** Adresse de l'utilisateur */
    private String address;
    /** Code postal de l'utilisateur */
    private String zipCode;
    /** Ville de l'utilisateur */
    private String city;
    /** Liste des téléphones de l'utilisateur */
    private List<PhoneDTO> phones;
    /** Liste des emails de l'utilisateur */
    private List<EmailDTO> emails;
    /** Photo de l'utilisateur */
    private Photo photo;
}
```

**@AllArgsConstructor** est une fonctionnalité de Lombok. Cette annotation est utilisée pour générer automatiquement un constructeur avec tous les arguments pour la classe.

### 8.5.4. Interface UserMapper

```java
@Mapper(uses = {EmailMapper.class, PhoneMapper.class, PhotoMapper.class})
public interface UserMapper {
    /** Convertit le User en UserDTO ...*/
    1 implementation  CRAVO David
    UserDTO map(User user);
    /** Convertit la liste de User en liste de UserDTO ...*/
    1 implementation  CRAVO David
    List<UserDTO> map(List<User> users);
    /** Convertit le UserDTO en User ...*/
    1 implementation  CRAVO David
    User update(UserDTO userDTO);
    /** Convertit un UserDTO en User ...*/
    1 implementation  CRAVO David
    List<User> update(List<UserDTO> userDTOS);
    1 usage  CRAVO David
    @AfterMapping
    default void afterUpdate(UserDTO dto, @MappingTarget User user){
        safe(user.getPhones())
                .forEach(phone -> phone.setUser(user));
        safe(user.getEmails())
                .forEach(email -> email.setUser(user));
    }
}
```

**@Mapper** dans le contexte de MapStruct est une annotation puissante utilisée pour définir une interface qui agit comme un mapper de données.

**map** est une méthode de Spring Boot qui est utilisée pour transformer les éléments d'un flux de données en utilisant Spring Data JPA.

**update** est une méthode de Spring Boot qui est utilisée pour mettre à jour des données dans la base de données en utilisant Spring Data JPA.

**@AfterMapping** est utilisée dans MapStruct. MapStruct facilite la conversion automatique entre différents types d'objets, par exemple entre des entités de base de données et des objets de transfert de données (DTO). Elle est utile lorsque vous avez besoin de manipuler manuellement les objets après que MapStruct a effectué le mapping automatique des propriétés.

**@MappingTarget** est appliquée à un paramètre de méthode dans une interface de mapper pour indiquer que ce paramètre est l'objet cible dans lequel les données doivent être mappées.

### 8.5.5. Controlleur UserControlleur

```java
@CrossOrigin(origins = "*")
@RestController
public class UserController {
    6 usages
    @Autowired
    private UserService userService;
    /** Demande la récupération de tous les utilisateurs à l'UserService ...*/
    no usages  CRAVO David
    @RequestMapping("/users")
    public List<UserDTO> getUsers(){
        return userService.getUsers();
    }
    /** Demande la récupération d'un utilisateur à l'UserService ...*/
    no usages  CRAVO David
    @RequestMapping("/user/{id}")
    public UserDTO getUser(@PathVariable long id){
        return userService.getUser(id);
    }
    /** Demande la suppression d'un utilisateur à l'UserService ...*/
    no usages  CRAVO David
    @RequestMapping(method = RequestMethod.DELETE, value = "/user/{id}")
    public void deleteUser(@PathVariable long id) { userService.deleteUser(id); }
    /** Demande l'ajout d'un utilisateur à l'UserService ...*/
    no usages  CRAVO David
    @RequestMapping(method = RequestMethod.POST, value = "/users")
    public void addUser(@RequestBody UserDTO userDTO) { userService.addUser(userDTO); }
    /** Demande la mise à jour d'un utilisateur à l'UserService ...*/
    no usages  CRAVO David
    @RequestMapping(method = RequestMethod.PUT, value = "/user/{id}")
    public void updateUser(@RequestBody UserDTO userDTO){
        userService.updateUser(userDTO);
    }
    /** Demande la récupération d'une liste d'utilisateurs auprès de l'UserService ...*/
    no usages  CRAVO David
    @RequestMapping(method = RequestMethod.GET, value="/{search}")
    public ResponseEntity<List<UserDTO>> searchUsers(@PathVariable String search){
        return ResponseEntity.ok(userService.searchUsers(search));
    }
}
```

**@CrossOrigin,** dans le contexte de Spring Framework, notamment avec Spring MVC et Spring WebFlux, est utilisée pour activer le Cross-Origin Resource Sharing (CORS) sur les contrôleurs ou les méthodes de gestionnaire spécifiques. CORS est un mécanisme de sécurité permettant à des ressources restreintes sur une page web d'être récupérées par un autre domaine en dehors du domaine d'origine de la ressource.

**@RestController** indique qu'une classe sert de contrôleur RESTful dans le modèle MVC de Spring.

Un service web RESTful implémente généralement ces principes en utilisant les méthodes HTTP standard (GET, POST, PUT, DELETE, etc.) pour effectuer des opérations CRUD (Créer, Lire, Mettre à jour, Supprimer) sur les ressources.

**@Autowired** est une fonctionnalité de Spring Framework, utilisée pour l'injection de dépendances.

L'injection de dépendances (DI) est un patron de conception (design pattern) où les objets ne créent pas leurs dépendances directement. Au lieu de cela, ils reçoivent leurs dépendances de l'extérieur, souvent par un conteneur de gestion des dépendances.

**@RequestMapping** est utilisée pour mapper les requêtes web à des méthodes dans les contrôleurs.

**@PathVariable** permet d'extraire les valeurs des variables du chemin d'URL et de les passer aux méthodes de contrôleur.

**@RequestBody** dans Spring Framework est utilisée pour lier le corps de la requête HTTP à un objet dans une méthode de contrôleur.

Ainsi, dans la méthode deleteUser, on va récupérer l'id de l'utilisateur dans l'Url qui communique avec l'API afin de demander au userService de le supprimer d'après son id.

Dans la méthode updateUser, on récupère la requête qui communique avec l'API pour demander au userService de mettre à jour les informations récupérées.

### 8.5.6. Service UserService

```java
@Service
public class UserService {

    6 usages
    @Autowired
    private UserRepository userRepository;

    5 usages
    private final UserMapper userMapper = (UserMapper) Mappers.getMapper(UserMapper.class);

    /** Demande la récupération de tous les utilisateurs au UserRepository ...*/
    3 usages  CRAVO David
    public List<UserDTO> getUsers() { return userMapper.map(userRepository.findAll()); }

    /** Demande la récupération d'un utilisateur au UserRepository ...*/
    1 usage  CRAVO David
    public UserDTO getUser(long id) { return userMapper.map(userRepository.findById(id).orElse( other: null)); }

    /** Demande l'ajout d'un utilisateur au UserRepository ...*/
    1 usage  CRAVO David *
    public void addUser(UserDTO userDTO) { userRepository.save(userMapper.update(userDTO)); }

    /** Demande la suppression d'un utilisateur au UserRepository ...*/
    1 usage  CRAVO David
    public void deleteUser(long id) { userRepository.deleteById(id); }

    /** Demande la mise à jour de l'utilisateur au UserRepository ...*/
    1 usage  CRAVO David
    public void updateUser(UserDTO userDTO){
        userRepository.save(userMapper.update(userDTO));
    }

    /** Demande au UserRepository de rechercher les utilisateurs ...*/
    1 usage  CRAVO David
    public List<UserDTO> searchUsers(String search) {
        return userMapper.map(userRepository.findByFirstnameContainsOrLastnameContains(search, search));
    }
}
```

**@Service**, dans Spring Framework est utilisée pour marquer une classe comme un service. Un service en Spring est une classe qui encapsule la logique métier ou les opérations commerciales.

### 8.5.7. Repository UserRepository

```java
@Repository
public interface UserRepository extends ListCrudRepository<User, Long> {

    /** Recherche les utilisateurs suivant leur prénom et leur nom ...*/
    1 usage  CRAVO David
    List<User> findByFirstnameContainsOrLastnameContains(String firstname, String lastname);

}
```

**@Repository** dans le cadre du Spring Framework est utilisée pour indiquer qu'une classe est un "repository", c'est-à-dire une classe qui est responsable de l'interaction avec la source de données ou la couche de persistance.

### 8.5.8. Script de création de la table

```sql
CREATE TABLE a_user(
id SERIAL PRIMARY KEY,
firstname VARCHAR(50),
lastname VARCHAR(50),
address VARCHAR(255),
zip_code CHAR(5),
city VARCHAR(50)
);
```

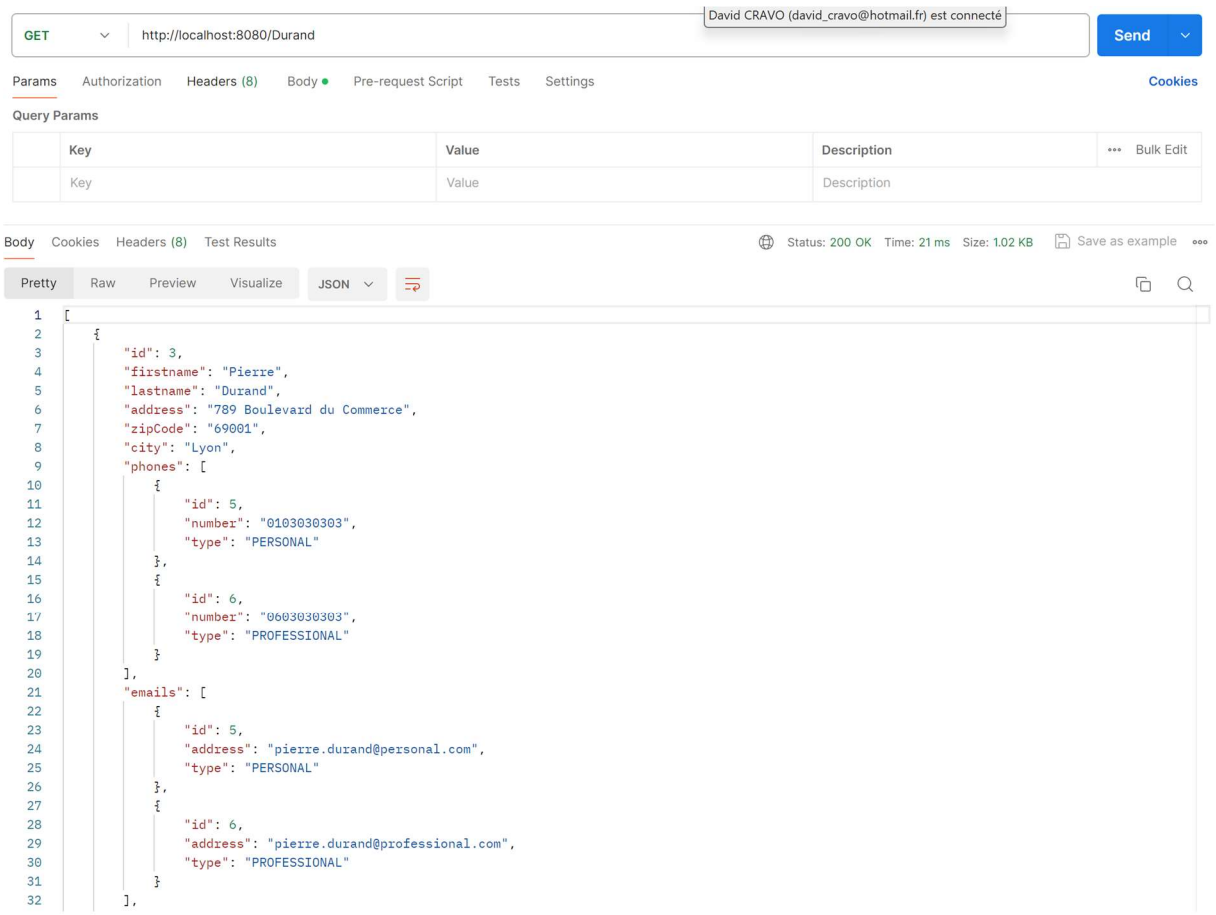
### 8.5.9. Script d'insertion de données

```sql
INSERT INTO "a_user" (firstname, lastname, address, zip_code, city)
VALUES
    ('Jean', 'Dupont', '123 Rue de la Poste', '75001', 'Paris'),
    ('Marie', 'Martin', '456 Avenue des Fleurs', '13001', 'Marseille'),
    ('Pierre', 'Durand', '789 Boulevard du Commerce', '69001', 'Lyon'),
    ('Sophie', 'Leclerc', '101 Rue de la Mairie', '31000', 'Toulouse'),
    ('Luc', 'Lefebvre', '222 Chemin des Bois', '06000', 'Nice'),
    ('Isabelle', 'Dupuis', '789 Avenue des Roses', '59000', 'Lille'),
    ('Thomas', 'Robert', '555 Rue de la République', '33000', 'Bordeaux'),
    ('Elodie', 'Girard', '999 Boulevard de la Liberté', '44000', 'Nantes'),
    ('Antoine', 'Petit', '777 Rue de la Paix', '67000', 'Strasbourg'),
    ('Julie', 'Dubois', '123 Avenue des Champs-Élysées', '83000', 'Toulon');
```

### 8.5.10. Vérification de l'accessibilité des données

N'ayant plus le temps de revoir la préparation et l'exécution de plans de tests, j'ai dû me contenter de vérifier le bon fonctionnement du back-end de mon application avec Postman.

En plus de la vérification du CRUD, j'ai vérifié le bon fonctionnement de la recherche dans mon application.



# 9. Implémentation du Frontend

## 9.1. Configuration du projet

Tout d'abord, l'installation de la **version LTS de Node.js et npm** (Gestionnaire de paquets Node.js) sous Windows via le téléchargement de l'installateur : **node-v20.11.1-x64.**
C'est un environnement d'exécution open-source pour JavaScript construit sur le moteur JavaScript V8 de Chrome.

Vient ensuite l'installation de **Angular CLI 17** en ligne de commande avec l'instruction : **npm install -g @angular/cli.**
C'est un framework de développement d'applications Web open-source maintenu par Google et une communauté de développeurs et d'entreprises individuelles. Il est écrit en TypeScript et utilisé pour développer des applications Web.

Toujours en ligne de commande pour créer le projet Annuaire via la commande **ng new annuaire**.

Reste enfin à installer **Angular Material**. C'est une bibliothèque de conception d'interface utilisateur pour Angular, un framework de développement d'applications web front-end. Elle fournit un ensemble de composants d'interface utilisateur réutilisables, bien testés et accessibles, basés sur les principes du Material Design de Google. Elle s'installe via la commande : **ng add @angular/material.**

Les différents composants du projet peuvent alors être créés.

## 9.2. Composant user-list

Dans le fichier HTML, on retrouve le code :

```
<app-menu></app-menu>
<app-autocomplete-filter></app-autocomplete-filter>
<div class="container">
    <button routerLink="/user/add" mat-raised-button color="primary">Ajouter</button>
</div>
<mat-card class="mat-card" *ngFor="let user of users">
    <mat-card-content class="mat-card-content" >
        <a (click)="goToUser(user.id)">
            {{user.photo}}
            {{user.lastname}}
            {{user.firstname}}
            {{user.phones[1].number}}
            {{user.emails[1].address}}
        </a>
    </mat-card-content>
</mat-card>
```

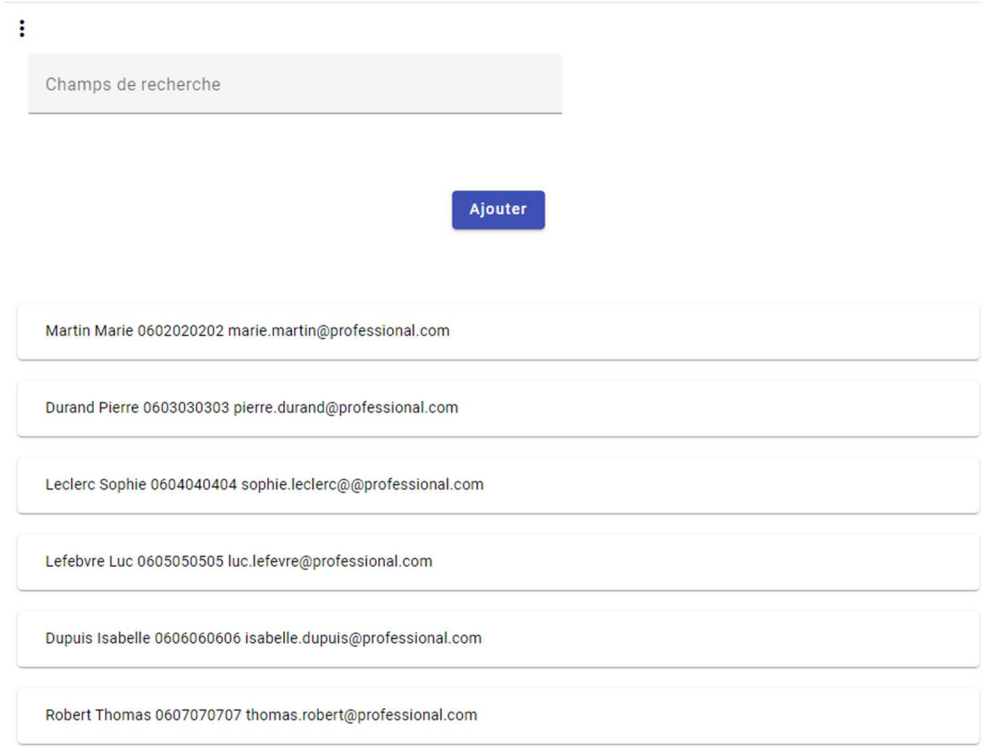
Sous le menu et le champ de recherche se positionne un bouton Material qui renvoie vers la page de création d'un nouveau contact. Sous ce bouton viennent s'afficher les différents contacts au travers d'une carte mat-card couplée à une boucle permise par Angular *ngFor qui nous permet de parcourir l'ensemble des contacts fournis par le fichier composant-list.ts. Dans cette carte, est placé un lien qui nous renvoie vers le contact sélectionné au travers de la fonction goToUser au click. Dans ce lien, on place les différentes propriétés du contact au travers d'une référence permise par Angular.

Le fichier user-list.component.ts :

```
@Component({
  selector: 'app-user-list',
  templateUrl: './user-list.component.html',
  styleUrl: './user-list.component.css'
})
export class UserListComponent implements OnInit, OnDestroy{
  users: any
  subscription: Subscription

  constructor(
    private userService: UserService,
    private router: Router
  ){
    this.subscription = new Subscription
  }
  ngOnInit(): void{
    this.subscription = this.userService.getUsers().subscribe(users => this.users = users)
  }
  ngOnDestroy(): void {
      this.subscription.unsubscribe()
  }
  goToUser(userId: number){
    this.router.navigate(['/user', userId])
  }
}
```

Ici notre composant implémente OnInit et OnDestroy qui sont des interfaces de cycle de vie de Angular. Ainsi à l'initialisation du composant, dans ngOnInit(), Angular récupère les utilisateurs en souscrivant au flux proposé par le service userService via sa méthode getUsers(). On assigne cette action à subscription qui est du type Subscription de la librairie rxjs pour pouvoir annuler la souscription dans ngOnDestroy() afin de libérer les ressources matérielles lors de la fin de l'utilisation du composant.
Dans le constructeur, on injecte les dépendances vers UserService et Router. Le Router va nous permettre de rediriger la page en cours de consultation vers le profil du contact sélectionné grâce à son identifiant dans la méthode goToUser(). Dans ce même constructeur on initialise aussi notre variable subscription.

Le résultat est la page suivante :




On y retrouve le menu, la barre de recherche, et le bouton pour ajouter un nouveau contact. S'ajoute à cela la liste des contacts se trouvant dans la base de données en faisant appel au BackEnd via le UserService.

## 9.3. Composant user-details

Afin de pouvoir retrouver tous les détails d'un contact, j'ai créé un composant qui va les afficher, les supprimer ou rediriger vers son édition.

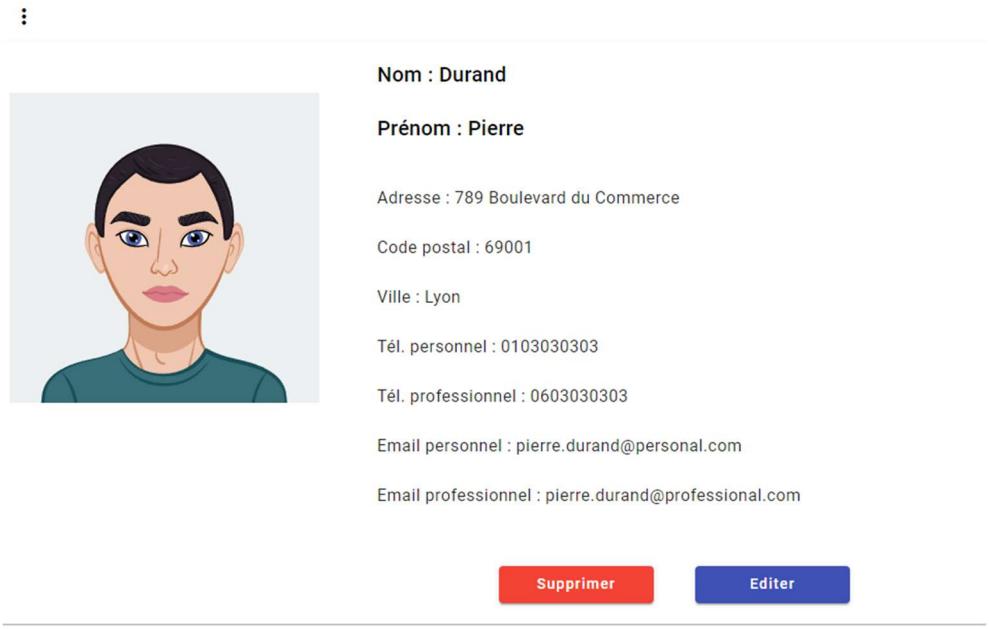
```
<div class="center-part">
    <mat-card-header>
        <mat-card-title class="mat-card-title" *ngIf="user">
            Nom : {{user.lastname}}
        </mat-card-title>
        <mat-card-title class="mat-card-title" *ngIf="user">
            Prénom : {{user.firstname}}
        </mat-card-title>
    </mat-card-header>
    <mat-card-content>
        <mat-list>
            <mat-list-item class="mat-list-item" *ngIf="user">Adresse : {{user.address}}</mat-list-item>
            <mat-list-item class="mat-list-item" *ngIf="user">Code postal : {{user.zipCode}}</mat-list-item>
            <mat-list-item class="mat-list-item" *ngIf="user">Ville : {{user.city}}</mat-list-item>
            <mat-list-item class="mat-list-item" *ngIf="user">Tél. personnel : {{user.phones[0].number}}</mat-list-item>
            <mat-list-item class="mat-list-item" *ngIf="user">Tél. professionnel : {{user.phones[1].number}}</mat-list-item>
            <mat-list-item class="mat-list-item" *ngIf="user">Email personnel : {{user.emails[0].address}}</mat-list-item>
            <mat-list-item class="mat-list-item" *ngIf="user">Email professionnel : {{user.emails[1].address}}</mat-list-item>
        </mat-list>
    </mat-card-content>
</div>
```

Ici, la directive *ngIf vérifie avant d'afficher les informations que l'on a bien un contact envoyé par user-details.component.ts qui suit :

```
export class UserDetailsComponent implements OnInit, OnDestroy{
  user: any;
  isAddForm: any
  private subscription: Subscription
  constructor(
    private route: ActivatedRoute,
    private router: Router,

    private userService: UserService,
  ){
    this.subscription = new Subscription
  }
  ngOnInit(): void{
    this.isAddForm = this.router.url.includes('add')

    if(!this.isAddForm){
      const userId : string | null = this.route.snapshot.paramMap.get('id')

      if (userId){
        this.subscription = this.userService.getUserById(+userId).subscribe(user => this.user = user)
      }
    }
  }
  ngOnDestroy(): void {
    this.subscription.unsubscribe()
  }
  goToUserList(){
    this.router.navigate(['/users'])
  }
  goToEditUser(user: any){
    this.router.navigate(['edit/user', user.id])
  }
  deleteUser(user: any){
    this.userService.deleteUserById(user.id).subscribe(_ => this.goToUserList());
  }
}
```

Dans ngOnInit, on instancie le booléen isAddForm avec la valeur retournée par la présence ou non de la chaine de caractère add dans l'url.
Si la valeur est à true, on récupère la valeur de l'id  dans l'url et si on obtient bien un id alors on s'abonne au flux permettant de récupérer le contact en fonction de son identifiant.
Enfin dans deleteUser on demande au UserService de demander la suppression du contact auprès du BackEnd.

Le résultat à l'écran est :



## 9.4. Service user

Via la commande ng generate service services/user, on demande à Angular de nous générer le service user dans un dossier services. Le code complet du service est le suivant :

```
@Injectable({
  providedIn: 'root'
})
export class UserService {
  readonly ENDPOINT_USERS = "/users";
  readonly ENDPOINT_USER = "/user";
  constructor(private httpClient: HttpClient, private annuaireService: AnnuaireService) { }
  getUsers(){
    return this.httpClient.get(this.annuaireService.API_URL + this.ENDPOINT_USERS);
  }
  getUserById(id: number){
    return this.httpClient.get(this.annuaireService.API_URL + this.ENDPOINT_USER + `/${id}`).pipe(
      tap((response) => this.log(response)),
      catchError((error) => this.handleError(error, undefined))
    )
  }
  updateUser(user: any){
    const httpOptions = {
      headers: new HttpHeaders({'Content-Type': 'application/json'})
    }
    return this.httpClient.put(this.annuaireService.API_URL, user, httpOptions).pipe(
      tap((response) => this.log(response)),
      catchError((error) => this.handleError(error, undefined))
    )
  }
  deleteUserById(id: number){
    return this.httpClient.delete(this.annuaireService.API_URL + this.ENDPOINT_USER + `/${id}`).pipe(
      tap((response) => this.log(response)),
      catchError((error) => this.handleError(error, undefined))
    )
  }
  addUser(user: any){
    const httpOptions = {
      headers: new HttpHeaders({'Content-Type': 'application/json'})
    }
    return this.httpClient.post(this.annuaireService.API_URL, user, httpOptions).pipe(
      tap((response) => this.log(response)),
      catchError((error) => this.handleError(error, undefined))
    )
  }
  searchUserList(term: string){
    return this.httpClient.get(`${this.annuaireService.API_URL}/{${term}}`).pipe(
      tap((response) => this.log(response)),
      catchError((error) => this.handleError(error, undefined))
    )
  }
```

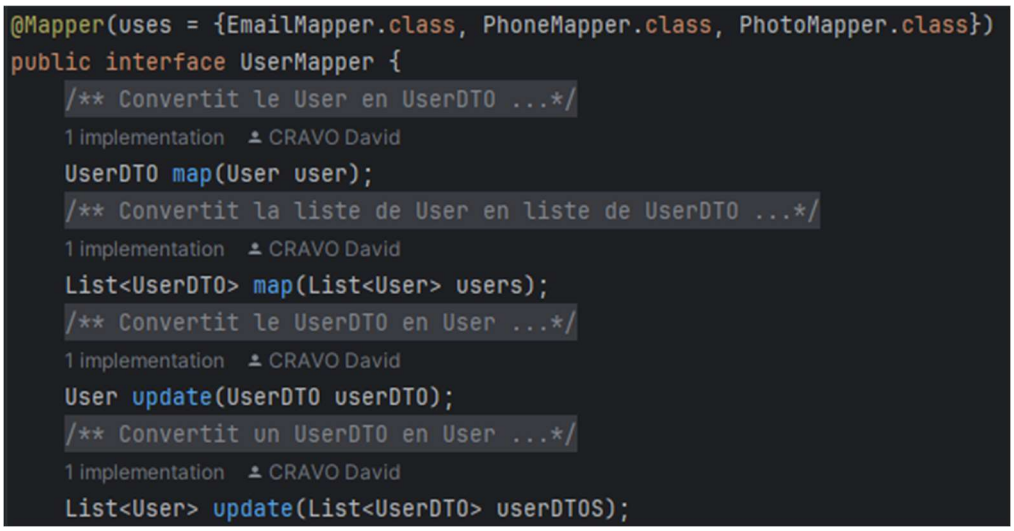
On commence par définir les endpoints de connexion à l'API proposé par le BackEnd. Ensuite, dans le constructeur, on injecte un **HttpClient**, élément fourni par le HttpClientModule de Angular, qui va permettre de communiquer avec le serveur http fourni par l'API du BackEnd.

On retrouve après le constructeur, les deux appels pour la récupération de contacts, un pour la totalité des contacts, et un pour un contact en fonction de son identifiant. Suivre les appels pour les opérations CRUD vont permettre la mise à jour, la suppression, l'ajout d'un contact. Il y a encore la fonction de recherche qui va faire appel à l'API de recherche du BackEnd.

La fonction updateUser permet de mettre à jour un contact. Elle fait appel à la méthode put de l'httpClient et on lui passe en paramètres l'url de l'API du BackEnd, la donnée qu'il faut mettre à jour, ici notre contact, et enfin des options pour la requête http (on spécifie que le corps de la requête est au format JSON). La méthode **pipe** permet de combiner plusieurs opérateurs **Rxjs**. L'opérateur **tap** permet d'enregistrer la réponse de la requête et fait appel à la fonction log pour l'afficher dans la console. L'opérateur **catchError** intercepte une éventuelle erreur pour l'afficher dans la console en faisant appel à la fonction handleError.

# 10. Recherche

Le but de ma recherche est de trouver un moyen de savoir comment mapper les classes Phone et Email qui, avec la structure existante :

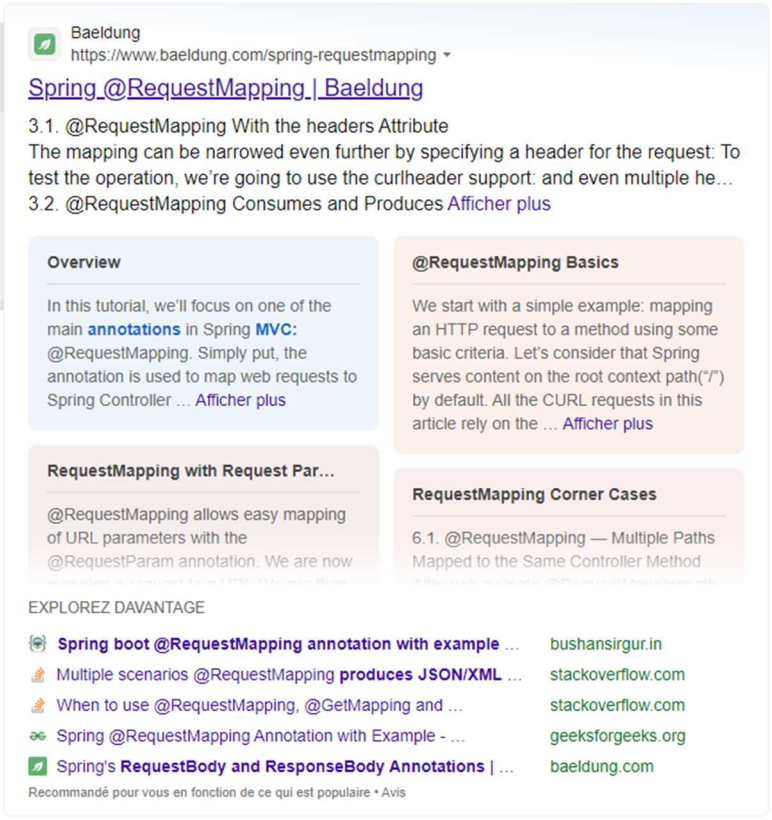



Le mappage n'est pas assuré. En effet, lorsque je fais une requête GET dans Postman, j'ai bien tous les détails de mon User (firstname, lastname, etc.), mais le retour pour les emails et les téléphones est un tableau vide.

Dans mon navigateur, j'effectue donc une recherche en utilisant trois mots-clés :

- Mapping puisque que je cherche à mapper mes deux dernières entités,
- Java parce que je développe mon BackEnd en java,
- Spring pour affiner ma recherche puisque j'utilise le framework Spring.

J'obtiens alors les résultats suivants :




À la suite de nombreuses recherches, j'ai souvent fini par lire les explications proposées par le site Baeldung. Je m'oriente donc naturellement vers ce site. D'après le conseil de Thomas LACOUTURE et Lilian GAYET qui m'ont aidé sur la résolution de problèmes dans mon code, je recherche plus précisément « before mapping » et « after mapping ».

Cela me mène au chapitre 10 de la page :

## 10. Before-Mapping and After-Mapping Annotations

Here's another way to customize *@Mapping* capabilities by using *@BeforeMapping* and *@AfterMapping* annotations. **The annotations are used to mark methods that are invoked right before and after the mapping logic.**

They are quite useful in scenarios where we might want this **behavior to be applied to all mapped super-types.**

Let's take a look at an example that maps the sub-types of *Car ElectricCar* and *BioDieselCar* to *CarDTO*.

While mapping, we would like to map the notion of types to the *FuelType* enum field in the DTO. Then after the mapping is done, we'd like to change the name of the DTO to uppercase.

Comme mon but est bien de mapper des entités liées comme des « sub-types », je continue de parcourir le document, ce qui me mène au chapitre :

### 10.2. Defining the Mapper

Now let's proceed and write our abstract mapper class that maps *Car* to *CarDTO*:

```
@Mapper
public abstract class CarsMapper {
    @BeforeMapping
    protected void enrichDTOWithFuelType(Car car, @MappingTarget CarDTO carDto) {
        if (car instanceof ElectricCar) {
            carDto.setFuelType(FuelType.ELECTRIC);
        }
        if (car instanceof BioDieselCar) {
            carDto.setFuelType(FuelType.BIO_DIESEL);
        }
    }

    @AfterMapping
    protected void convertNameToUpperCase(@MappingTarget CarDTO carDto) {
        carDto.setName(carDto.getName().toUpperCase());
    }

    public abstract CarDTO toCarDto(Car car);
}
```

***@MappingTarget*** is a parameter annotation that **populates the target mapping DTO right before the mapping logic is executed in case of *@BeforeMapping*** and **right after in case of *@AfterMapping*** annotated method.

Le code obtenu, avec l'aide de Thomas est alors :

```
@AfterMapping
default void afterUpdate(UserDTO dto, @MappingTarget User user){
    safe(user.getPhones())
            .forEach(phone -> phone.setUser(user));
    safe(user.getEmails())
            .forEach(email -> email.setUser(user));
}
```

Je teste à nouveau alors sur Postman et le résultat est le suivant


```
"id": 1,
"firstname": "Jean",
"lastname": "Durand Dupont",
"address": "123 Rue de la Poste",
"zipCode": "75001",
"city": "Paris",
"phones": [
    {
        "id": 1,
        "number": "0101010101",
        "type": "PERSONAL"
    },
    {
        "id": 2,
        "number": "0601010101",
        "type": "PROFESSIONAL"
    }
],
"emails": [
    {
        "id": 1,
        "address": "jean.dupont@personal.com",
        "type": "PERSONAL"
    },
    {
        "id": 2,
        "address": "jean.dupont@professional.com",
        "type": "PROFESSIONAL"
    }
],
"photo": null
```


Le résultat est celui escompté, en effet, je n'ai plus de tableau vide pour mes téléphones et mes emails.

# 11. Conclusion

## Le projet Trombine

Le projet Trombine a été une rude épreuve pour moi. Il m'a fallu dans un premier temps me former aux technologies employées par l'entreprise, ce qui m'a pris beaucoup temps. J'ai dû beaucoup travailler pour monter en compétences et être opérationnel pour l'implémentation du Back et du Front. Je me suis formé par moi-même en utilisant les ressources disponibles sur le web.

Il m'a fallu commencer à analyser le projet seul avec les connaissances acquises en formation, mais ce n'était pas le type d'analyse attendu par l'entreprise. En effet, les diagrammes effectués n'avaient pas beaucoup d'intérêt pour eux, contrairement à l'ERD. J'ai donc dû m'adapter à ce mode de fonctionnement. Le projet paraissait simple au début, mais au fur et à mesure des échanges avec le Product Owner, celui-ci s'est complexifié et a nécessité de reprendre l'analyse à plusieurs reprises. En effet, avec l'ajout des Tags, des Zones de droits, des Administrateurs de Tag, des Responsables de Tag et des Administrateurs de Zones de droits, dus à une organisation peu commune de l'entreprise, il a fallu reconsidérer l'organisation des classes à plusieurs reprises.

Même si je savais dès le départ que j'aurai du mal à voir la fin du projet, il a été difficile de le reléguer au deuxième plan lors des dernières semaines afin de me consacrer au projet Annuaire et au dossier de projet que je n'avais pas eu le temps de travailler.

## Le projet Annuaire

Il est né de la nécessité de pouvoir mener un projet web de A à Z. Le projet Trombine était en attente des retours de pull-request et je devais pouvoir finir de développer un Back-end complet, et développer un début de Front-end avec le peu de connaissances acquises sur Angular. S'il a été moins éprouvant intellectuellement, il m'a néanmoins obligé à accélérer la cadence afin de pouvoir produire des résultats pour le dossier de projet. J'avais aussi prévu de m'en servir pour effectuer des tests, mais je n'ai pas eu le temps de me replonger dans ces derniers.

## Le ressenti en cet fin de stage

Je regrette de ne pas avoir eu plus de temps pour monter en compétences et mener à termes mon projet Annuaire qui reprenait toutes les connaissances acquises avec le projet Trombine. J'aurais aimé avoir deux mois de stage en plus.

L'expérience en entreprise, dans la vie réelle d'une ESN, m'a beaucoup apporté en termes de suivi de projet. J'ai pu faire de la conception et mettre en pratique les méthodes agiles. Il a vraiment été plaisant de travailler en équipe, de respecter une organisation d'entreprise en termes de temps et de communication. J'ai pu observer les développeurs (presque tous ingénieurs) évoluer au gré des deadlines et des Daily meetings. J'en ressors grandi, et convaincu que je suis dans la bonne voie dans mon projet de reconversion professionnelle. Je sais que je vais encore devoir continuer à me former, comme l'entreprise me l'a conseillé, mais cela n'est pas un obstacle à mes ambitions.

# 12. Annexe

## 12.1. Présentation de l'entreprise

Fort de ses agences en France et à l'international, le groupe Apside est capable d'intervenir sur l'ensemble du cycle de vie d'un projet : de l'idéation au maintien opérationnel en passant par le développement informatique et industriel.

**1985 Apside Technologies**

Pour développer ses activités et se rapprocher de ses clients aux profils industriels, Apside crée sa filiale spécialisée dans le domaine de l'informatique technique, scientifique et électronique.

**1990 Toulouse**

La proximité de ses clients étant indispensable, Apside commence son déploiement à travers la France par l'ouverture d'une première agence dans la ville rose : Toulouse.

**1995 Bordeaux**

La relation de confiance ne pouvant s'établir à distance, Apside déploie ses activités sur le bassin bordelais.

**1996 Aix-En-Provence**

Le Sud-Ouest conquis, Apside se rapproche de la Méditerranée et ouvre son agence à Aix-en-Provence.

**1998 Rennes**

Un véritable groupe est en train de se construire grâce à l'indépendance dans la gestion de chacune des agences. Cette liberté de gestion, permet à l'entreprise de continuer son développement en ouvrant une nouvelle agence à Rennes.

**2001 Nantes & Nice**

Une année double pour Apside qui réussit à ouvrir deux agences dans deux belles régions : Nantes et Nice. La localisation va définir les activités clés des agences en fonction des écosystèmes offrant un bassin plutôt industriel pour Nice et IT pour la région Nantaise.

**2002 Apside HTI Automobile**

Pour continuer sa croissance et développer ses expertises, Apside fait l'acquisition de l'entreprise HTi spécialisée dans le domaine de l'ingénierie automobile.

**2003 Apside Infogérance**

L'évolution grandissante de l'ingénierie et de l'informatique proposent de nouveaux métiers plus spécialisés pour une sécurité des données. Pour offrir à ses clients un

accompagnement complet, Apside crée sa filiale spécialisée dans le domaine de l'infrastructure et de la production : Apside Infogérance.

**2004 Lille, Lyon & Tours**

La couverture géographique de ses agences permet à Apside d'être présent à travers toutes les régions de France. Les agences de Lille, Lyon et Tours signent l'ouverture de la huitième, neuvième et dixième implantation du groupe.

**2007 Brest & Orléans**

L'accompagnement de carrière des consultants est optimal lorsque celui-ci est personnalisé. Apside fait le choix de développer plusieurs agences pour offrir à ses collaborateurs une meilleure écoute et proximité.

**2011 Belgique**

Le développement d'une nouvelle agence est un nouveau défi, pour les équipes Apside mais surtout pour ses collaborateurs qui vont venir construire sa personnalité. Pour ce nouveau challenge, destination Bruxelles en Belgique

**2014 Strasbourg**

Apside favorise la mobilité par ses multiples agences et s'implante proche du traditionnel marché de noël Strasbourgeois.

**2015 Suisse**

Apside décide de relever le défi et passe les frontières pour ouvrir une première agence à l'international en s'installant à Genève.

**2017 Allemagne**

L'Allemagne, grand bassin industriel, permet à nos consultants de participer aux développements de projets innovants.

**2018 Maroc**

Apside décide de suivre l'un de ses grands acteurs de l'automobile et développe un véritable plateau au Maroc.

**2018 Montpellier**

Chaque ouverture d'agence est un nouveau challenge stimulant, à l'initiative de l'agence de Toulouse, Montpellier vient compléter la famille Apside.

**2018 Siparex**

M. Michel Klar transmet Apside à une centaine de ses cadres et managers, lesquels sont entrés au capital de l'entreprise. Ils sont guidés par M. Pierre Gauthier, nouveau PDG d'Apside. Siparex a accompagné le projet en capital investissement.

**2019 Portugal**

Région ensoleillée et proche de la côte, les plateaux d'Aveiro et Porto bénéficient d'un cadre naturel propice aux nouvelles idées.

**2019 Keley & Elanz**

Une entreprise est une conjugaison de personnalités et de talents qui la rende unique. Apside affirme sa dynamique de croissance en intégrant les société Keley et Elanz.

Keley, cabinet de conseil en transformation digitale, permet au groupe Apside de proposer des prestations de haut niveau de conseil en stratégie, data et ingénierie.

Elanz, Microsoft Partner depuis 2014, apporte au groupe sa spécialisation dans les technologies Microsoft ainsi qu'un renforcement de sa couverture territoriale, notamment à Clermont-Ferrand.

**2019 Wecanopy embedded powertrain**

Le développement des start-ups apporte une expertise de pointe sur le marché. La filiale du groupe Apside, Hti Automobile, renforce son positionnement avec l'acquisition de la société Wecanopy Embedded Powertrain qui devient une branche spécialisée dans l'innovation autour du groupe motopropulseur : technologies thermiques, hybrides et électriques.

**2019 Apside'EA**

Chaque talent est important. La conjugaison de la performance et de la bienveillance est au cœur des préoccupations du groupe Apside. L'entreprise donne naissance à son entreprise adaptée intégrée Apsid'EA pour favoriser l'insertion professionnelle des personnes en situation de handicap.

**2022 Canada**

Avec son agence au Canada, Apside part à la conquête du territoire nord-américain et s'implante à Montréal afin d'accompagner ses clients (et de nouveaux !) outre Atlantique.