

## Dossier de projet pour l'obtention du Titre Professionnel Concepteur Développeur d'Applications

### ServiceCoPro

Logiciel de proposition de services et de prêt d'objets entre voisins



**Projet réalisé par Geoffrey ROBELOT**

Supervisé par M. MARTIN : Formateur Java

2019 – 2020

## Remerciements

Je tiens à remercier le Greta des Yvelines pour m'avoir permis de suivre cette formation de Concepteur Développeur d'Applications et l'équipe administrative qui m'a suivi tout au long de ces huit mois.

Je remercie aussi les intervenants professionnels qui ont assuré la partie théorique. Leurs conseils, toujours pertinents, m'ont été précieux.

Je tiens à remercier le département informatique de la CPAM des Yvelines qui avait accepté de me prendre en stage dans le cadre de cette formation. Malheureusement, en raison de la crise sanitaire « Covid-19 », ce stage n'a pas pu avoir lieu.

## Table des matières

1. Abstract.....	4
2. Présentation de l'application .....	5
3. Contexte du projet.....	6
3.1. Cahier des charges.....	6
3.2. Livrables.....	6
3.3. Besoins et contraintes .....	6
3.4. Résultats attendus.....	6
4. Gestion de projet .....	7
4.1. Planning et suivi.....	7
4.2. Environnement humain et technique .....	7
4.3. Objectifs de qualité.....	7
5. Analyse du besoin .....	8
6. Analyse fonctionnelle.....	8
6.1. Diagramme de cas d'utilisation .....	8
6.2. Description textuelle .....	9
6.3. Maquettage .....	10
7. Conception .....	12
7.1. Diagramme d'activité .....	12
7.2. Diagramme de classe.....	13
7.3. Diagramme de séquence.....	13
7.4. Création de la base de données .....	13
7.4.1. Conception de la base de données .....	13
7.4.2. Déploiement de la base de données.....	14
8. Codage .....	16
8.1. Langage de programmation et environnement de développement .....	16
8.2. De la maquette à l'interface utilisateur sous SceneBuilder .....	16
8.3. Application N-tiers.....	17
8.4. EJB.....	18
8.5. Composants d'accès aux données.....	19
9. Jeu d'essai d'une fonctionnalité de l'application.....	21
9.1. Ajouter un utilisateur grâce aux Web-Services .....	21
9.2. Récupérer un utilisateur grâce aux Web-Services .....	24
10. Mise en place de tests .....	26
11. Exemple de recherche .....	27
12. Conclusion.....	28
13. Annexes.....	29

## 1. Abstract

Because of the health context (Covid-19 and the containment measures), internships have been cancelled. Consequently, I had to design and develop a project on my side.

The application « ServiceCoPro » aims to enable the exchange of services and the loan of objects between neighbours. The person who is interested can download this desktop application, create an account through a contact form and finally log in. Then, the user accesses to the main menu, where he can consult the latest posts from other users: “the latest services” and “the latest objects”. By clicking on one of these two propositions and then on “more information”, he will access to the comment of the author’s post and he will see the date from which the object or service will become available.

If the user wants to offer an object/service, he has to click on “post” and to choose the period of the loan or the date of the service.

On the contrary, if he is interested in an object/service amongst those available, he can book from the space “reservation”. The borrower will be notified of the date of return. (In this presentation, this function has not been developed).

To manage the application, an administrator will have access to all functionality, such as:

- add a user,
- change information about a user,
- delete a user,
- manage users posts and reservations (In this presentation, this function has not been developed).

## 2. Présentation de l'application

L'objectif de l'application est de rendre possible le prêt d'objets et l'échange de services entre voisins. La personne qui le souhaite peut télécharger cette application de bureau, créer son compte via un formulaire de renseignement, puis s'authentifier.

Une fois la connexion valide, l'utilisateur a accès au menu principal de l'application d'où il pourra consulter les derniers « post » des différents utilisateurs : « derniers objets en prêt » ainsi que « derniers services proposés ».

En cliquant sur un service ou un objet depuis le menu principal, puis en cliquant sur le bouton « + d'infos », il aura accès au commentaire de l'auteur du « post » ainsi que la date à partir de laquelle l'objet/service est mis à disposition.

Si l'utilisateur souhaite partager un service et/ou un objet, il peut le faire depuis le menu « poster » de l'application et choisir la durée du prêt ou la date du service. Si un objet ou un service l'intéresse parmi ceux proposés, il peut réserver depuis l'espace « réserver » du menu principal.

Une alerte devra avertir l'emprunteur de la date de restitution de l'objet (fonctionnalité non développée dans le cadre de cette présentation).

Pour gérer l'application, un administrateur aura accès à toutes les fonctionnalités de l'application. Il pourra ainsi :

- Ajouter un utilisateur (type client ou administrateur)
- Modifier les informations concernant un utilisateur
- Supprimer un utilisateur (seulement si celui-ci ne possède pas de réservations en cours)
- Gérer les « post » et « réservations » des utilisateurs (fonctionnalité non développée dans le cadre de cette présentation).

### 3. Contexte du projet

#### 3.1. Cahier des charges

L'application ServiceCoPro a pour objectif de proposer une plateforme d'échanges de biens et de services entre voisins.

Pour ce projet, je tente de conceptualiser les attentes du client même si l'expérience utilisateur révélera, sans doute, des améliorations à apporter.

#### 3.2. Livrables

Lors de la première itération de développement, seules les fonctionnalités basiques de l'application seront fonctionnelles. Dans une deuxième itération, de nouvelles fonctionnalités feront leur apparition.

Un rapport documenté du projet, une notice d'utilisation du logiciel et un fichier exécutable seront attendus.

#### 3.3. Besoins et contraintes

Besoins fonctionnels : Le client doit pouvoir consulter la liste des objets en prêt et les services mis à disposition à partir de l'application ServiceCoPro. Si un objet l'intéresse et qu'il souhaite le réserver, il doit pouvoir le faire depuis le menu correspondant. Il en est de même pour la réservation d'un service. Si un client souhaite proposer un objet ou un service, il doit pouvoir le faire depuis le menu correspondant.

Contraintes : Il n'y a pas de contrainte de développement en termes de langage de programmation et d'IHM (Interface Homme-Machine).

Je choisis donc d'utiliser la plateforme Java EE pour le développement de ce projet.

Java sera utilisé comme langage de programmation pour sa portabilité et le Framework Java Fx permettra de bâtir les IHM.

#### 3.4. Résultats attendus

- Création des maquettes de l'application,
- Conception et codage d'une partie de l'application.

En effet, compte tenu du peu de temps accordé pour le développement de ce projet, toutes les fonctionnalités de l'application ne seront pas opérationnelles. Un fichier exécutable de l'application ServiceCoPro sera fonctionnel.

## 4. Gestion de projet

### 4.1. Planning et suivi

J'ai choisi d'utiliser le diagramme de GANTT afin d'avoir une vision globale du projet.

(Annexe 1)

Je décide d'étaler l'étape de codage sur dix-sept jours volontairement car je suis conscient que je rencontrerai des difficultés lors de la programmation de l'application. Des tests unitaires seront mis en place tout au long du codage de l'application.

Il est aussi possible que je me rende compte, à mesure de l'avancée du projet, que la base de données n'intègre pas tous les champs nécessaires au stockage des données provenant de l'application.

Je m'autorise donc à la retravailler lors d'une deuxième itération de développement. Il en est de même pour certaines fonctionnalités de ServiceCoPro.

### 4.2. Environnement humain et technique

Environnement humain : La conception et le développement de ce projet seront réalisés par un seul acteur : moi-même. Mon formateur Java durant cette année supervise le projet 1h par semaine à distance via une plateforme de classe virtuelle.

Environnement technique : Je réalise cette application depuis mon ordinateur portable Asus sous Windows 10 équipé d'un processeur IntelCore i7.

Eclipse, SceneBuilder et XAMPP sont disponibles sur ce poste de travail et me permettent de développer ServiceCoPro.

### 4.3. Objectifs de qualité

Le code de l'application doit être maintenable et compréhensible pour de futurs administrateurs.

L'application se voudra en quête de nouvelles fonctionnalités (connexion grâce aux réseaux sociaux, chat, etc.).

## 5. Analyse du besoin

Résidant actuellement dans un appartement en copropriété, certains copropriétaires souhaitent parfois emprunter des objets à leurs voisins plutôt que de les acheter.

D'autres souhaitent rendre des services (petits travaux par exemple).

D'autres encore, veulent bénéficier des services et des connaissances de leurs voisins.

L'application ServiceCoPro, à travers ses fonctionnalités, tente de répondre aux besoins que nous venons d'identifier.

## 6. Analyse fonctionnelle

### 6.1. Diagramme de cas d'utilisation

Afin de modéliser le diagramme d'utilisation, j'utilise l'outil Modelio open source.

Le diagramme de cas d'utilisation (DUC) permet d'organiser les besoins et de regrouper les fonctionnalités principales d'un système. (Annexe 2).

Dans mon application ServiceCoPro, je choisis d'avoir deux types d'utilisateurs : Administrateur et Client.

Le diagramme en annexe indique qu'un administrateur peut :

- Gérer les utilisateurs,
- Gérer les réservations,
- Gérer les propositions.

La relation « include » indique que le cas d'utilisation au départ de la flèche inclut le cas d'utilisation « s'authentifier ».

Ces actions sont réalisables par l'administrateur, si et seulement si, celui-ci s'authentifie au préalable.

Un client peut :

- Consulter les services et objets proposés par ses voisins,
- Réserver un service ou un objet
- Proposer un service ou un objet
- Supprimer un objet ou un service proposé.

Ces actions sont possibles, à condition que le client se soit authentifié au préalable.



## 6.2. Description textuelle

J'ai choisi de détailler les cas d'utilisation « gérer les utilisateurs », « consulter les services et objets » et « réserver services ou objets » qui me semblent être les plus pertinents dans le cadre de ce projet.

### Cas 1 :

Nom : Gérer les utilisateurs

Acteur : Administrateur

Description : La gestion des utilisateurs doit être réalisée par l'administrateur de l'application

Précondition : L'administrateur doit être authentifié

#### Scénario nominal :

1. Le système affiche la page de connexion de l'application.
2. L'administrateur entre ses informations de connexion.
3. Le système vérifie les informations et valide la connexion.
4. Le système affiche la page « administration » de l'application.
5. L'administrateur affiche tous les utilisateurs de l'application.
6. L'administrateur sélectionne un utilisateur et peut le modifier ou le supprimer.
7. L'administrateur peut ajouter un utilisateur.

#### Scénario alternatif :

- 1.1. L'administrateur entre des informations d'authentification erronées.
- 1.2. L'administrateur saisit à nouveau ses informations de connexion.
- 1.3. L'administrateur quitte l'application.

### Cas 2 :

Nom : consulter les services et objets

Acteur : Client

Description : Le client doit pouvoir consulter les services et objets mis à disposition

Précondition : Le client doit être authentifié

#### Scénario nominal :

1. Le système affiche la page de connexion de l'application.
2. Le client entre ses informations de connexion.
3. Le système vérifie les informations et valide la connexion.
4. Le système affiche la page principale de l'application. Deux listes contiennent les objets disponibles et les services mis à disposition.

#### Scénario alternatif :

- 1.1. Le client entre des informations d'authentification erronées.
- 1.2. Le client saisit à nouveau ses informations de connexion.
- 1.3. Le client quitte l'application.

### Cas 3 :

Nom : réserver services ou objets

Acteur : Client

Description : Le client doit pouvoir réserver un ou plusieurs service(s) / objet(s)

Précondition : Le client doit être authentifié

#### Scénario nominal :

1. Le système affiche la page de connexion de l'application.
2. Le client entre ses informations de connexion.
3. Le système vérifie ces informations et valide la connexion.
4. Le système affiche la page principale de l'application. Deux listes contiennent les objets disponibles et les services mis à disposition.
5. Le client clique sur le bouton permettant d'afficher la page de réservation.
6. Le client sélectionne l'objet ou le service qu'il souhaite réserver.
7. Le système lui demande les informations nécessaires à la réservation.
8. Le client valide la réservation en appuyant sur le bouton correspondant.

#### Scénario alternatif :

- 1.1. L'objet ou le service n'est pas disponible à la date souhaitée.
- 1.2. Le client entre des informations d'authentification erronées.
- 1.3. Le client saisi à nouveau ses informations de connexion.
- 1.4. Le client quitte l'application.

### 6.3. Maquettage

Pour la réalisation du maquettage de l'application, je préfère faire des croquis sur papier.

Les maquettes présentées ici se veulent plutôt basiques. J'ai privilégié la simplicité d'utilisation pour le client. Il sera donc possible d'améliorer le design de l'application par la suite si celui-ci ne convient pas.

L'utilisateur démarre l'application et arrive sur la page authentification. (Annexe 3).

Si l'utilisateur a un compte, il entre ses informations de connexion :

- Son nom d'utilisateur,
- Son mot de passe.

Il poursuit en cliquant sur « login » pour valider sa saisie et se connecter.

Si l'utilisateur n'a pas encore de compte, il clique sur « sign up ». Il ouvre ainsi une nouvelle page de l'application. Celle-ci se présente sous la forme d'un formulaire. (Annexe 4)

L'utilisateur doit entrer les informations suivantes :

- Son genre,
- Son nom d'utilisateur,
- Son mot de passe,
- Son courriel,

- Le numéro de son bâtiment,
- Le numéro de son appartement.

Il clique ensuite sur « ajouter » et son compte est persisté dans la base de données.

Lorsque la connexion de l'utilisateur est validée, celui-ci se retrouve sur la page « menu principal » de l'application. (Annexe 5)

Depuis cette interface utilisateur, il peut choisir l'action de son choix. Deux listes lui présenteront :

- Les objets en prêt,
- Les services proposés.

En cliquant sur « + d'infos », il obtiendra le commentaire de l'utilisateur qui a posté l'objet ou le service ainsi que la date à partir de laquelle il est disponible.

Le menu de gauche permet de naviguer entre les différentes interfaces de l'application. En haut à droite se trouve un bouton « Admin » qui permettra à l'administrateur d'afficher un panneau de contrôle de l'application après authentification de celui-ci. Un utilisateur de type client n'aura donc pas accès à ce panneau de contrôle.

Une fois la connexion de l'administrateur validée, le panneau de contrôle (Annexe 6) s'affiche. Le menu de gauche permet d'afficher dans le panneau de contrôle :

- La liste des utilisateurs,
- Les services proposés,
- Les objets proposés,
- Les services réservés,
- Les objets réservés.

Je vous présente ici la fonction du bouton « utilisateur » qui permet d'afficher la liste des utilisateurs de l'application depuis le menu « admin ». Il dispose des actions « Ajouter », « Modifier » et « Supprimer ».

Pour ajouter un utilisateur, l'administrateur va remplir un formulaire (création de compte « Sign Up » vu précédemment). Lorsque le bouton « valider » du formulaire est cliqué, des vérifications ont lieu pour s'assurer qu'il n'y a pas de champ vide et que le mot de passe contient au minimum quatre caractères. Si ces conditions ne sont pas remplies, une alerte indique qu'il y a des erreurs sur la saisie du formulaire. Sinon, l'ajout est validé et l'utilisateur est persisté dans la base de données.

Avant de supprimer un utilisateur, une alerte demande à l'administrateur s'il est sûr de vouloir poursuivre. D'autre part, un utilisateur ayant réservé un objet ou un service « en cours » ne peut pas être supprimé. Il faut que l'objet soit restitué avant de pouvoir supprimer l'utilisateur.

## 7. Conception

Les différents diagrammes présentés sont réalisés à l'aide de l'outil Modelio open source.

### 7.1. Diagramme d'activité

J'ai choisi de présenter les diagrammes d'activités du cas d'utilisation « authentification » et du cas d'utilisation « ajouter un utilisateur ».

**Cas : authentification.** C'est le cas d'utilisation qui détermine l'accès à l'application pour le client et l'administrateur. (Annexe 7)

Acteur : Utilisateur (Client ou Administrateur)

Précondition : L'utilisateur doit avoir créé son compte

Scénario principal :

L'utilisateur entre ses informations de connexion via la page de démarrage de l'application. Les informations saisies sont récupérées puis traitées afin de déterminer si l'utilisateur existe dans la base de données. Le mot de passe saisi est crypté et comparé avec le mot de passe crypté qui est stocké dans la base de données. Validation de la connexion et ouverture du menu principal de l'application.

Scénario alternatif :

L'utilisateur entre des informations de connexion erronées. S'il le souhaite, l'utilisateur peut saisir à nouveau ses informations de connexion.

**Cas : ajouter un utilisateur.** (Annexe 8)

Acteur : Utilisateur (client)

Précondition : Posséder une adresse e-mail.

Scénario principal :

Les informations personnelles concernant l'utilisateur sont saisies. Le mot de passe de l'utilisateur est crypté. Le système récupère les informations du formulaire. Le nom d'utilisateur saisi est comparé avec les noms d'utilisateurs stockés dans la base de données. L'utilisateur est persisté dans la base.

Scénario alternatif :

Le nom d'utilisateur est déjà utilisé. L'utilisateur est invité à saisir un autre nom d'utilisateur.

## 7.2. Diagramme de classe

Ce diagramme contient les classes avec leurs méthodes respectives. (Annexe 9)

Il permet une description statique du système puisqu'il n'intègre pas la façon d'utiliser les méthodes.

Le diagramme d'interaction qu'est le diagramme de séquence, permet de montrer l'aspect dynamique du système.

## 7.3. Diagramme de séquence

Ce diagramme permet d'afficher les échanges entre les objets du système selon une « ligne de vie ».

Je vous présente ici un affinage du cas d'utilisation « gérer les utilisateurs » et plus particulièrement la fonction « ajouter un utilisateur » de l'application ServiceCoPro. (Annexe 10)

Les informations saisies dans le formulaire du contrôleur de la vue « Sign Up » sont récupérées. La méthode « isEmpty () » a la charge de vérifier les champs saisis par l'utilisateur. Elle détermine si les champs correspondent au résultat attendu pour persister l'utilisateur dans la base de données. Le mot de passe saisi dans le formulaire est crypté par la classe en charge du hachage. Le mot de passe crypté est récupéré. Il sera utilisé pour stocker l'utilisateur dans la base de données.

Le contrôleur de la vue « Sign Up » appelle la méthode de la DAO « ajouterUtilisateur ». La fonction de la DAO fait appel aux Web-Services de la classe ServiceUtilisateur et plus particulièrement la méthode « addUser » de cette classe. Ensuite, la méthode « persistUtilisateur » de l'interface UtilisateurEjbLocal entre en jeu. L'objet utilisateur est ensuite stocké dans la base de données. Il est retourné au contrôleur de la vue afin d'afficher les informations de l'utilisateur créé.

## 7.4. Création de la base de données

### 7.4.1. Conception de la base de données

Afin de concevoir la base de données de l'application, j'utilise la méthode MERISE. Le modèle conceptuel de données (MCD) est l'un des outils de cette méthode. Celui-ci me permet de déterminer les relations entités – associations.

L'application compte 6 entités : Administrateur, Utilisateur, Propositionobjet, Propositionservice, Reservationobjet et Reservationsservice.

(Annexe 11).

Ci-dessous, les relations entités / associations :

- Utilisateur / réservation : Un utilisateur peut réserver 1 ou plusieurs objet(s) / service(s).
- Utilisateur / proposition : Un utilisateur peut proposer 1 ou plusieurs objet(s) / service(s).
- Proposition / réservation : Chaque proposition, service ou objet, a 1 ou plusieurs réservation(s), service ou objet.

Les cardinalités minimales et maximales sont représentées sur le lien entre deux entités.

Quelques exemples dans notre cas :

- Un administrateur peut créer un ou plusieurs utilisateurs (1,n).
- Un utilisateur peut être créé par un et un seul administrateur (1,1).
- Un utilisateur peut effectuer 0 ou plusieurs reservationobjet (0,n).
- Une reservationobjet peut être effectuée par 1 ou plusieurs utilisateurs (1,n).

Il faut maintenant construire le MLD (Modèle Logique de Données). (Annexe 12)

Il convertit le MCD en un ensemble compréhensible pour un SGBD (Système de Gestion de Base de Données).


Les entités deviennent ainsi des éléments de base de données : des tables.

L'identifiant de l'entité devient la clef primaire de la table.

#### 7.4.2. Déploiement de la base de données

Je choisis de vous présenter ici les tables Utilisateur, Propositionobjet et Reservationobjet. Les autres tables ont une structure similaire. Seule la table Administrateur diffère quelque peu. En effet, elle a pour attributs : un num (clé primaire), un login, un mot de passe et un rôle.

Lors de la création de mes tables, je remplis les champs : nom, type, longueur, le champs « null » : nul ou non, et le champ clef (primaire / étrangère / sans valeur).

#	Nom	Type	Interclassement	Attributs	Null	Valeur par défaut	Commentaires	Extra
<input type="checkbox"/> 1	num 	int(11)			Non	Aucun(e)		AUTO_INCREMENT
<input type="checkbox"/> 2	genre	varchar(255) utf8_bin			Non	Aucun(e)		
<input type="checkbox"/> 3	nom	varchar(255) utf8_bin			Non	Aucun(e)		
<input type="checkbox"/> 4	mdp	varchar(255) utf8_bin			Non	Aucun(e)		
<input type="checkbox"/> 5	batiment	varchar(255) utf8_bin			Non	Aucun(e)		
<input type="checkbox"/> 6	appartement	varchar(255) utf8_bin			Oui	NULL		
<input type="checkbox"/> 7	email	varchar(255) utf8_bin			Oui	NULL		

Structure de la table Utilisateur.

	#	Nom	Type	Interclassement	Attributs	Null	Valeur par défaut	Commentaires	Extra
<input type="checkbox"/>	1	num 🔑	int(11)			Non	Aucun(e)		AUTO_INCREMENT
<input type="checkbox"/>	2	objet	varchar(50)	utf8_bin		Non	Aucun(e)		
<input type="checkbox"/>	3	commentaire	varchar(255)	utf8_bin		Non	Aucun(e)		
<input type="checkbox"/>	4	dateDebut	date			Non	Aucun(e)		
<input type="checkbox"/>	5	delai	varchar(50)	utf8_bin		Oui	NULL		
<input type="checkbox"/>	6	numUtilisateur 🔑	int(11)			Non	Aucun(e)		

### Structure de la table Propositionobjet.

	#	Nom	Type	Interclassement	Attributs	Null	Valeur par défaut	Commentaires	Extra
<input type="checkbox"/>	1	num 🔑	int(11)			Non	Aucun(e)		AUTO_INCREMENT
<input type="checkbox"/>	2	objet	varchar(25)	utf8_bin		Non	Aucun(e)		
<input type="checkbox"/>	3	dateEmprunt	date			Non	Aucun(e)		
<input type="checkbox"/>	4	dateRendu	date			Oui	NULL		
<input type="checkbox"/>	5	lieuRestitution	varchar(25)	utf8_bin		Oui	NULL		
<input type="checkbox"/>	6	numUtilisateur 🔑	int(11)			Non	Aucun(e)		
<input type="checkbox"/>	7	numObjet 🔑	int(11)			Non	Aucun(e)		

### Structure de la table Reservationobjet.

Dans phpMyAdmin, je crée la base de données MySQL « ServiceCoPro ». (Annexe 13)

## 8. Codage

### 8.1. Langage de programmation et environnement de développement

ServiceCoPro : Une application Java EE (*Java Enterprise Edition*). L'intérêt de cette plateforme est sa facilité de déploiement d'application web sur un serveur d'application. Pour ce projet, j'utilise Payara.

Les Web-Services de type REST (REpresentational State Transfer) basés sur le protocole http permettront de gérer les opérations CRUD (Create, Read, Update, Delete) de l'application.

Celle-ci est codée en Java, un langage de programmation que j'affectionne pour sa portabilité. En effet, il sera possible de faire fonctionner cette application aussi bien sur Windows, Linux ou encore Mac (à condition d'avoir un SDK installé sur la machine). Le code source de l'application ne changera pas d'un poste à un autre, c'est la machine virtuelle qui ne sera pas la même selon l'environnement (Windows, Linux, Mac).

La partie programmation du projet se fait grâce à l'IDE (Environnement de développement intégré) Eclipse.

Pour la partie graphique de l'application, j'utilise le Framework Java Fx et l'outil SceneBuilder. Celui-ci permet de créer des IHM.

### 8.2. De la maquette à l'interface utilisateur sous SceneBuilder

Pour développer la partie front-end de mon application, je choisis d'utiliser SceneBuilder. Cet outil, sous Java Fx, permet de bâtir les différentes fenêtres, scènes et nœuds graphiques de l'application. (Annexe 14).

Je propose ici de lister les composants de la vue présentée en annexe :

- TextField : permet de récupérer le nom d'utilisateur,
- PasswordField : récupère le mot de passe de l'utilisateur,
- Buttons : permettent de « se connecter », « s'enregistrer » et « indiquer si le mot de passe a été oublié ». La dernière fonctionnalité ne sera pas opérationnelle dans cette itération de développement.
- Labels : permettent d'indiquer du texte,
- ImageView : présentent les logos « nom d'utilisateur » et « mot de passe ».

Afin d'appeler cette vue dans l'application, il faut créer une classe Main qui hérite de la classe Application. (Annexe 15).

Cette classe contient deux méthodes :

- La méthode « main » qui lance l'application grâce à la fonction « launch ».
- La méthode « start » qui prend en argument l'objet « Stage ».



Précisons la classe Main afin de comprendre le mécanisme qui permet d'afficher la vue réalisée en amont.

D'abord, j'instancie la classe FXMLLoader qui permet de charger des fichiers au format « fxml », puis j'indique le chemin relatif de la ressource.

Ensuite, je crée un conteneur « AnchorPane » qui contiendra tous les objets de la vue.

Puis, j'ajoute un objet « Scene » qui prend comme paramètres le conteneur et la taille de la fenêtre en pixels.

Enfin, j'indique que je souhaite « montrer » l'objet « Stage ».

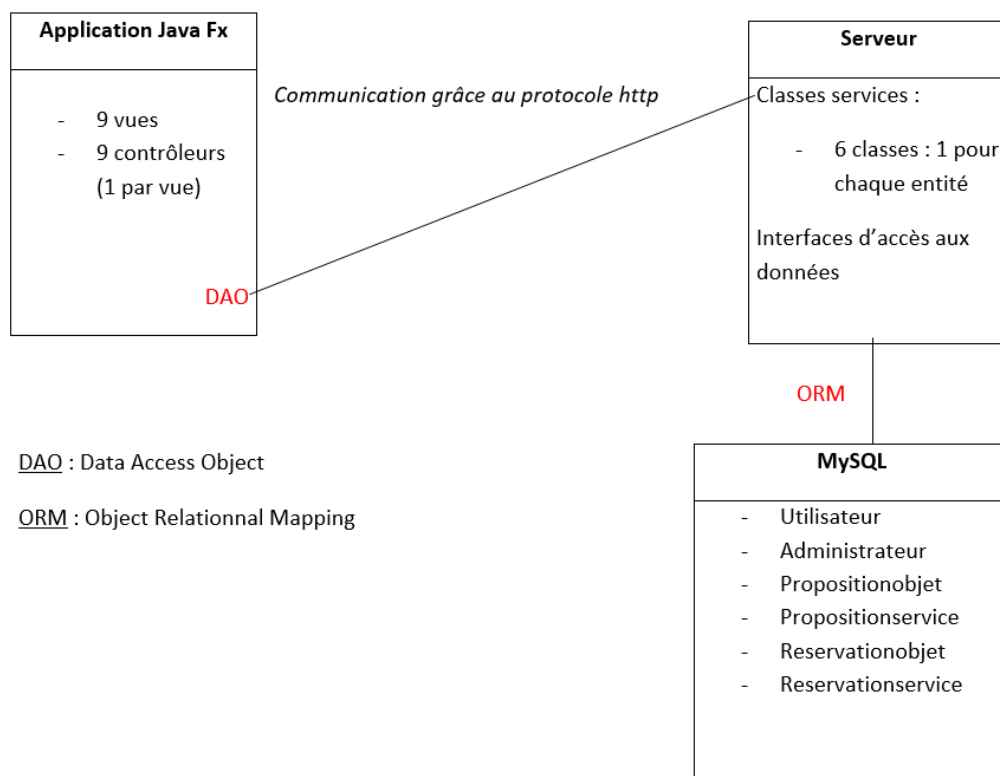
En compilant la classe Main, la vue précédemment créée doit s'afficher. Afin de contrôler les composants à manipuler, un contrôleur est nécessaire. J'indique donc sous SceneBuilder, le nom de celui-ci pour le connecter à la vue.

### 8.3. Application N-tiers

L'application ServiceCoPro est construite en multi couches (ou multi tiers) :

- La couche métier,
- La couche de stockage des données,
- La couche de présentation.

#### Architecture en couches de l'application ServiceCoPro



#### Architecture de l'application ServiceCoPro.

Le code de la couche métier est indépendant de la couche de présentation. Ceci permet, dans une nouvelle itération de développement par exemple, de déployer l'application sur un support mobile comme Android sans avoir à changé le code métier. Seules les vues (Layout en Android) seront à revoir.

La couche DAO (Data Access Object) permet les opérations de type CRUD (Create, Read, Update and Delete). Elle permet aussi d'isoler le stockage des données. Les objets métier vont communiquer avec la couche DAO, qui elle-même va communiquer avec le système de stockage. Le but étant d'encapsuler le code responsable du stockage des données.

La DAO a aussi un rôle de sécurité : elle protège (à faible niveau) des éventuelles injections SQL. Dans l'application ServiceCoPro, la couche DAO utilise le protocole http pour communiquer avec le serveur (web services REST).

La couche de présentation est développée grâce au Framework Java Fx.

#### 8.4. EJB

La plateforme Java EE propose l'architecture EJB pour développer les couches métier et la persistance des données en association avec JPA (*Java Persistence API*). L'application ServiceCoPro utilise des EJB et doit être développée dans un conteneur EJB. Le serveur d'application Payara jouera ce rôle.

L'accès aux EJB se fait à travers deux interfaces : une interface Locale et une interface Remote. Dans le cadre de cette présentation, les EJB implémentent l'interface Locale puisque tous les objets s'exécutent dans le même conteneur EJB. (Annexe 16). Voyons plus en détails la classe UtilisateurEjb :

- L'annotation `@Stateless` définit un EJB Stateless dont le nom JNDI (Java Naming and Directory Interface) est UtilisateurEjb.
- L'annotation `@LocalBean` signifie que toutes les méthodes publiques du bean sont accessibles dans le conteneur.
- L'annotation `@Ressource` marque une ressource nécessaire à l'EJB. Le serveur d'application va injecter une référence sur l'attribut `sessionContext` lors de l'exécution du composant EJB.
- L'annotation `@PersistenceContext` demande au conteneur d'injecter dans l'attribut « em » une instance de la classe `EntityManager` associée au contexte persistant utilisé.

L'EJB UtilisateurEjb fournit les méthodes CRUD (Create, Read, Update, Delete).

## 8.5. Composants d'accès aux données

Côté client, la couche DAO permet de communiquer avec le serveur par le biais du protocole http.

```
////////// méthode GET ////////////
public List<Utilisateur> afficherListeUtilisateur() {

    //////////// service web ////////////
    System.out.println("");
    System.out.println("Service qui permet d'afficher tous les utilisateurs");
    Client client = ClientBuilder.newClient();
    WebTarget service = client.target("http://localhost:8080/serviceCoPro/jaxrs/utilisateur");

    List<Utilisateur> liste = service.request(MediaType.APPLICATION_JSON_TYPE)
        .get(new GenericType<List<Utilisateur>>()) {
        };
    liste.forEach(a -> System.out
        .println("num : " + a.getNum() + " / " + "nom : " + a.getNom() + " / " + "email : " + a.getEmail()));

    return liste;
}
```

### Méthode « afficherListeUtilisateur » de la classe DAO.

Cette méthode de la DAO communique avec le serveur et avec la classe service utilisateur.

- L'utilisation du type GET permet de récupérer une ressource : ici la liste des utilisateurs de l'application.
- L'objet retourné sera du type JSON.
- L'URL du service ciblé (WebTarget).

Voyons maintenant la méthode getUsers de la classe service utilisateur.

```
@GET
@Produces(MediaType.APPLICATION_JSON)
public List<Utilisateur> getUsers() {
    return utilisateurEjb.getUtilisateurFindAll();
}
```

### Méthode « getUsers » de la classe service utilisateur.

L'annotation @GET identifie la méthode Java exécutée pour traiter la méthode GET du protocole http. L'annotation @Produces définit le type MIME pour la réponse renvoyée au client (ici au format JSON).

```

////////// méthode POST //////////
public void ajouterUtilisateur(Utilisateur utilisateur) {
    System.out.println("");
    System.out.println("\n***Invoyer le service ajouter un utilisateur***");
    ////////// service web //////////
    Form forme = new Form();
    forme.param("lenom", utilisateur.getNom());
    forme.param("email", utilisateur.getEmail());
    forme.param("batiment", utilisateur.getBatiment());
    forme.param("genre", utilisateur.getGenre());
    forme.param("appartement", utilisateur.getAppartement());
    forme.param("mdp", utilisateur.getMdp());
    Client client = ClientBuilder.newClient(); // Obtenir le service cible
    WebTarget service = client.target("http://localhost:8080/serviceCoPro/jaxrs/utilisateur/adduser");
    Invocation.Builder lesutilisateurs = service.request(MediaType.APPLICATION_JSON);
    Response response = lesutilisateurs.post(Entity.entity(forme, MediaType.APPLICATION_FORM_URLENCODED_TYPE),
        Response.class);
    Utilisateur monutilisateur = response.readEntity(Utilisateur.class);
    System.out.println(monutilisateur.toString());
}

```

### Méthode « ajouterUtilisateur » de la classe DAO.

- L'utilisation du type POST correspond à l'envoi de données d'un formulaire.
- L'objet retourné sera du type JSON.
- L'URL du service ciblé (WebTarget).

Voyons maintenant la méthode « creerUtilisateur » de la classe service utilisateur.

```

@Path("/adduser")
@POST
@Consumes(MediaType.APPLICATION_FORM_URLENCODED)
@Produces(MediaType.APPLICATION_JSON)
public Utilisateur creerUtilisateur(@FormParam("lenom") String n, @FormParam("email") String e,
    @FormParam("batiment") String b, @FormParam("genre") String g, @FormParam("appartement") String a,
    @FormParam("mdp") String m) {
    Utilisateur utilisateur = new Utilisateur(g, n, m, b, a, e);
    utilisateurEjb.persistUtilisateur(utilisateur);
    return utilisateur;
}

```

### Méthode « creerUtilisateur » de la classe service utilisateur.

L'annotation `@Path` indique la partie relative à ajouter à l'URL de base pour accéder à la ressource. Ainsi, dans notre cas, il faudra saisir l'URL suivante pour atteindre la ressource : « <http://localhost:8080/serviceCoPro/jaxrs/utilisateur/adduser> », « **adduser** » étant la partie variable.

Les données reçues du formulaire ont le type MIME

`MediaType.APPLICATION_FORM_URLENCODED`.

Chaque paramètre transmis est récupéré grâce à l'annotation `@FormParam` suivie du nom du champ du formulaire mis entre parenthèses puis affecté à l'argument de la méthode qui suit. Dans notre cas : `@FormParam("lenom") String n, etc.`

Le contenu du champ « lenom » du formulaire est mis dans l'argument « n » de la méthode.

```

/*
 * l'interface métier locale est définie dans cette interface
 */
@Local
public interface UtilisateurEjbLocal {

    public Utilisateur persistUtilisateur(Utilisateur utilisateur);

    public Utilisateur mergeUtilisateur(Utilisateur utilisateur);

    public Utilisateur removeUtilisateur(String nom);

    public List<Utilisateur> getUtilisateurFindAll();

    public Utilisateur getUtilisateur(String utilisateur);

}

```

### Interface Locale « UtilisateurEjbLocal ».

Enfin, l'utilisateur est sauvé grâce à la méthode « persistUtilisateur » de l'interface utilisateurEjbLocal.

## **9. Jeu d'essai d'une fonctionnalité de l'application**

Je propose ici de tester les fonctionnalités :

- Ajouter un utilisateur grâce à la méthode POST.
- Récupérer un utilisateur grâce à la méthode GET.

Ces fonctions font appel aux Web-Services.

### 9.1. Ajouter un utilisateur grâce aux Web-Services

Premièrement, je code une classe gestionnaire de base de données que je place dans le package « dbConnection » du projet.

```

Connection dbconnection;

public Connection getConnection() {

    String connectionString = "jdbc:mysql://localhost:3306/servicecopro";

    try {
        Class.forName("com.mysql.cj.jdbc.Driver");
    } catch (ClassNotFoundException e) {
        e.getMessage();
        e.printStackTrace();
    }

    try {
        dbconnection = DriverManager.getConnection("jdbc:mysql://localhost:3306/servicecopro", "admin", "admin");
    } catch (SQLException e) {
        e.getMessage();
        e.printStackTrace();
    }

    return dbconnection;

}

```

### Code de la classe « DatabaseHandler ».

Ensuite, je programme le formulaire de la vue « Sign Up » qui permet de récupérer les paramètres (informations personnelles) de l'utilisateur que l'on souhaite créer.

```
@FXML
public void creer() throws GeneralSecurityException, IOException, SQLException {

    if (isEmpty()) {
        // si le radiobutton "homme" est sélectionné
        if (homme.isSelected()) {
            String genreButton = "Homme";
            String nom = utilisateur.getText();
            // cryptage du mot de passe saisi dans le formulaire
            Crypto crypto = new ClassCrypto();
            String code = mdp.getText();
            String encrypt = new String(crypto.encrypt(code.getBytes()));
            String mail = email.getText();
            String bat = batiment.getText();
            String apt = appartement.getText();
            // créer un utilisateur avec comme paramètres les données du formulaire
            Utilisateur utilisateur = new Utilisateur(genreButton, nom, encrypt, bat, apt, mail);
            // vérifie si l'utilisateur existe dans la base de données
            Connection con = DriverManager.getConnection("jdbc:mysql://localhost:3306/servicecopro", "admin",
                "admin");
            Statement stmt = con.createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE, ResultSet.CONCUR_READ_ONLY);
            // requête dans la base de données
            ResultSet rs = stmt.executeQuery("SELECT * FROM Utilisateur WHERE nom like '" + nom + "'");
            if (rs.next()) {
                Alert alert = new Alert(AlertType.ERROR);
                alert.setHeaderText("L'utilisateur : " + nom + " existe déjà");
                alert.setContentText("Veuillez choisir un autre nom d'utilisateur");
                alert.show();
                System.out.println("l'utilisateur existe");
            } else {
                // persiste l'utilisateur dans la base de données
                dao.ajouterUtilisateur(utilisateur);
                System.out.println("ajout: " + utilisateur + " / mot de passe : " + encrypt);
                Stage dialogStage = (Stage) annuler.getScene().getWindow();
                dialogStage.close();
            }
        }
    }
}
```

#### Code de la classe « ControleurVueSignUp ».

Avant d'ajouter un nouvel utilisateur, plusieurs actions sont mises en place. D'abord, je vérifie que les champs du formulaire correspondent bien à ce que j'attends en résultat grâce à la méthode booléenne « isEmpty () ». (Annexe 17)

Ensuite, je crypte le mot de passe saisi par l'utilisateur afin qu'il ne transite pas du côté client au côté serveur « en clair ».

Avant d'appeler la méthode de la DAO pour persister l'utilisateur, je vérifie que le nom saisi n'existe pas dans la base de données. S'il n'existe pas, alors la méthode « ajouterUtilisateur » de la DAO est appelée. Sinon, une alerte signale que le nom d'utilisateur existe déjà et qu'il faut choisir un autre nom.

Se reporter à la méthode « ajouterUtilisateur » de la classe DAO au point 8.4 *composant d'accès aux données*. C'est dans cette méthode que l'appel aux Web-Services a lieu. En effet l'URL du service ciblé est invoquée dans la « WebTarget ».

Intervient ensuite la classe « ServiceUtilisateur » et plus particulièrement la méthode « creerUtilisateur ». Voir au point 8.4 *composant d'accès aux données*.

Enfin la méthode « persistUtilisateur » de l'interface UtilisateurEjbLocal sauvegarde l'utilisateur dans la base de données. Voir au point 8.4 *composant d'accès aux données*.

J'utilise ARC (Advanced Rest Client) afin de tester les requêtes http de mes Web-Services. En effet, il permet de tester toutes les méthodes http (GET, POST, PUT, DELETE).

The screenshot shows the ARC interface for configuring a POST request. The 'Method' is set to 'POST' and the 'Host' is 'http://localhost:8080'. The 'Path' is '/serviceCoPro/jaxrs/utilisateur/adduser'. Under 'Query parameters', there is a table with the following data:

Parameter	Value	Action
lenom	Dupont	X
email	dupont@gmail.com	X
batiment	D	X
genre	Homme	X
appartement	A	X
mdp	dupont	X

Below the table is an 'ADD' button. To the right of the configuration fields is a blue 'SEND' button and a vertical ellipsis menu icon.

Requête http de type « post » pour ajouter un utilisateur sous ARC.

Résultat après avoir envoyé la requête :

The screenshot shows the ARC interface displaying the result of a successful POST request. The 'Method' is 'POST' and the 'Request URL' is 'http://localhost:8080/serviceCoPro/jaxrs/utilisateur/adduser?lenom=Dupont&email=dupont@gmail.com&batiment=D'. The 'Parameters' tab is selected, showing a table with 'Header name' and 'Header value'.

Header name	Header value
content-type	application/x-www-form-urlencoded

Below the table is an 'ADD HEADER' button. A green checkmark icon and the text 'Headers are valid' are displayed. The status bar shows '200 OK' and '71.69 ms'. The 'Body' tab is selected, showing a JSON response:

```
{
  "appartement": "A",
  "batiment": "D",
  "email": "dupont@gmail.com",
  "genre": "Homme",
  "mdp": "dupont",
  "nom": "Dupont",
  "num": 65
}
```

Validation de la requête sous ARC.

Comme on peut le voir, le résultat de la requête est valide puisqu'un code « 200 » est retourné. L'utilisateur est donc sauvegardé grâce à un Web-Service.

Maintenant, je souhaite récupérer l'utilisateur créé précédemment.

## 9.2. Récupérer un utilisateur grâce aux Web-Services

```
package service;

import java.util.List;

@DeclareRoles({ "admin", "client", "user" })
@Path("utilisateur")
public class ServiceUtilisateur {

    @EJB
    UtilisateurEjbLocal utilisateurEjb;

    @PersistenceContext(unitName = "serviceCoPro")
    private EntityManager em;

    // @RolesAllowed("admin")

    @GET
    @Produces(MediaType.APPLICATION_JSON)
    public List<Utilisateur> getUsers() {
        return utilisateurEjb.getUtilisateurFindAll();
    }

    @Path("/{nomutilisateur}")
    @GET
    @Produces(MediaType.APPLICATION_JSON)
    public Utilisateur getUser(@PathParam("nomutilisateur") String nomutilisateur) {
        return utilisateurEjb.getUtilisateur(nomutilisateur);
    }
}
```

### Méthode « getUser » de la classe ServiceUtilisateur.

Depuis la classe ServiceUtilisateur, je code la méthode « getUser » afin de récupérer un utilisateur en fonction du nom d'utilisateur passé en paramètre de la requête http. La ressource est donc accessible à l'URL

suivante : « <http://localhost:8080/serviceCoPro/jaxrs/utilisateur/+nomutilisateur> ».

Pour s'assurer que le service web fonctionne, il suffit de remplacer le « nomutilisateur » par Dupont (l'utilisateur créé en amont). Nous pouvons tester la méthode GET sous ARC ou directement depuis le navigateur.



## Sous ARC :

Method: GET, Request URL: http://localhost:8080/serviceCoPro/jaxrs/utilisateur/Dupont

Parameters: ^

Headers: content-type: application/x-www-form-urlencoded

Variables:

ADD HEADER

Headers are valid. Headers size: 47 bytes

200 OK 15.69 ms

DETAILS

```
{
  "appartement": "A",
  "batiment": "D",
  "email": "dupont@gmail.com",
  "genre": "Homme",
  "mdp": "dupont",
  "nom": "Dupont",
  "num": 65
}
```

### Requête GET pour récupérer un utilisateur de l'application sous ARC.

On observe que le résultat de la requête est valide puisqu'un code « 200 » est retourné. De plus, les informations concernant l'utilisateur Dupont sont retournées au format JSON. Le Service-Web est donc fonctionnel.

Depuis le navigateur :

localhost:8080/serviceCoPro/jaxrs/utilisateur/Dupont

```
{
  "appartement": "A",
  "batiment": "D",
  "email": "dupont@gmail.com",
  "genre": "Homme",
  "mdp": "dupont",
  "nom": "Dupont",
  "num": 65
}
```

### Requête GET pour récupérer un utilisateur de l'application depuis le navigateur.

## 10. Mise en place de tests

Afin de vérifier le bon fonctionnement d'extraits de code développés, je mets en place des tests unitaires. Je choisis de vous présenter le test du Web-Service GET utilisateurs qui doit permettre l'affichage de la liste des utilisateurs de l'application.

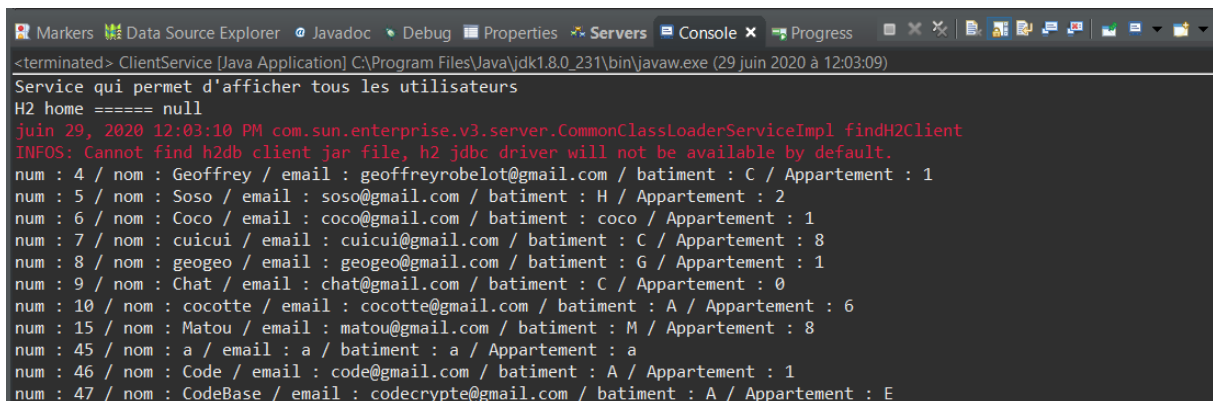
Pour commencer, je crée un nouveau projet qui servira à tester certains morceaux de code. Je code une classe de test « ClientService » afin de vérifier que mes Web-Services fonctionnent correctement.

```
public class ClientService {
    public static void main(String[] args) {
        getLesUtilisateursJson();
        // ajouterUtilisateur();
        // supprimerUtilisateur();
    }

    private static void getLesUtilisateursJson() {
        System.out.println("Service qui permet d'afficher tous les utilisateurs");
        Client client = ClientBuilder.newClient();
        WebTarget service = client.target("http://localhost:8080/serviceCoPro/jaxrs/utilisateur");
        List<Utilisateur> liste = service.request(MediaType.APPLICATION_JSON_TYPE)
            .get(new GenericType<List<Utilisateur>>()) {
        };
        liste.forEach(a -> System.out
            .println("num : " + a.getNum() + " / " + "nom : " + a.getNom() + " / " + "email : " + a.getEmail()
                + " / " + "batiment : " + a.getBatiment() + " / " + "Appartement : " + a.getAppartement()));
    }
}
```

### Classe de test ClientService.

Le point d'entrée du programme est la méthode « main » de la classe. Celle-ci invoque la méthode « getLesUtilisateursJson () » qui va donc s'exécuter.



```
<terminated> ClientService [Java Application] C:\Program Files\Java\jdk1.8.0_231\bin\javaw.exe (29 juin 2020 à 12:03:09)
Service qui permet d'afficher tous les utilisateurs
H2 home ===== null
juin 29, 2020 12:03:10 PM com.sun.enterprise.v3.server.CommonClassLoaderServiceImpl findH2Client
INFOS: Cannot find h2db client jar file, h2 jdbc driver will not be available by default.
num : 4 / nom : Geoffrey / email : geoffreyrobelot@gmail.com / batiment : C / Appartement : 1
num : 5 / nom : Soso / email : soso@gmail.com / batiment : H / Appartement : 2
num : 6 / nom : Coco / email : coco@gmail.com / batiment : coco / Appartement : 1
num : 7 / nom : cuicui / email : cuicui@gmail.com / batiment : C / Appartement : 8
num : 8 / nom : geogeo / email : geogeo@gmail.com / batiment : G / Appartement : 1
num : 9 / nom : Chat / email : chat@gmail.com / batiment : C / Appartement : 0
num : 10 / nom : cocotte / email : cocotte@gmail.com / batiment : A / Appartement : 6
num : 15 / nom : Matou / email : matou@gmail.com / batiment : M / Appartement : 8
num : 45 / nom : a / email : a / batiment : a / Appartement : a
num : 46 / nom : Code / email : code@gmail.com / batiment : A / Appartement : 1
num : 47 / nom : CodeBase / email : codecrypte@gmail.com / batiment : A / Appartement : E
```

### Résultat du test de la méthode dans la console d'Eclipse.

D'après le résultat en console, j'en conclus que la fonction « getLesUtilisateursJson () » est opérationnelle puisque je récupère correctement la liste de tous les utilisateurs de l'application. Le test unitaire de cette fonction est donc positif.

## 11. Exemple de recherche

Lors de la phase de développement du projet, il m'est arrivé de me questionner. D'abord, sur la façon de faire la plus appropriée à la compréhension du code, et ensuite sur la facilité de la maintenance ultérieure.

J'ai choisi de vous présenter ici le raisonnement et la recherche du stockage du type des attributs « dateDebut / dateEmprunt » des différentes tables « propositions » et « réservations ».

Dans les vues « réserver » et « poster », j'utilise un DatePicker sous Java Fx afin de récupérer la date à partir de laquelle l'objet ou le service sera mis à disposition. Un autre DatePicker doit permettre de récupérer la date de réservation de l'objet ou du service. Les valeurs retournées par le DatePicker sont du type LocalDate.

Mes attributs « dateDebut / dateEmprunt » de mes tables de base de données ont un type Date. Il faut donc qu'à partir du formulaire de ma vue, je récupère ce même type afin de stocker l'information dans ma base. Cela revient à convertir le type LocalDate en type Date.

J'ai donc effectué une recherche sur le site web BeginnersBook accessible depuis l'URL suivante : <https://beginnersbook.com/>. Ce site propose de petits tutoriels sur les langages de programmation tels que Java, C, C++ ou encore Kotlin.

Je m'inspire de leur méthode pour convertir un type LocalDate en Date. Ci-dessous, un extrait de mon code afin de récupérer ma dateDebut dans un type Date et le persister dans la base de données.

```
// récupère la valeur saisie dans le DatePicker
LocalDate localDate = choixDate.getValue();
Instant instant = Instant.from(localDate.atStartOfDay(ZoneId.systemDefault()));
// stocke dans la variable dateDebut la date en type date
Date dateDebut = Date.from(instant);
System.out.println(localDate + " / " + instant + " / " + dateDebut);

String duree = delai.getText();

Propositionobjet propObjet = new Propositionobjet(commentaire, dateDebut,
    duree, obj, selUtilisateur);
```

Conversion du type LocalDate en Date : Application ServiceCoPro.

## 12. Conclusion

En raison de la crise sanitaire actuelle (covid-19), je n'ai pas pu réaliser comme prévu mon stage de huit semaines au sein du département informatique de la CPAM des Yvelines situé à Versailles. Cette expérience aurait été, j'en suis sûr, très enrichissante.

Ce stage ayant été annulé au dernier moment, il a fallu concevoir et développer seul un projet personnel, écrire ce mémoire et préparer une présentation, le tout sur une période très courte.

Ce fût une expérience intense et du fait du peu de temps imparti, je n'ai pas pu développer toutes les fonctionnalités de l'application. Cela a un « côté frustrant ».

Cependant, ce projet m'a permis d'approfondir mes connaissances, tant sur la partie conception que sur la partie développement. Les erreurs lors de la phase de codage ont aiguisé ma curiosité et m'ont permis de progresser.

Je vais continuer le développement de cette application. Je suis conscient que la structure de la base de données est à revoir pour développer toutes les fonctionnalités que l'application pourra proposer.

J'envisage, quand l'application de bureau sera pleinement fonctionnelle, de développer le projet sous la plateforme Android et de la mettre sur le store Google Play. Rappelons que l'OS Android représente 80% des parts de marché en France. C'est donc la plateforme de développement la plus intéressante pour atteindre un large public.

### 13. Annexes

<u>Annexe 1</u> : Diagramme de GANTT.....	p.30
<u>Annexe 2</u> : Diagramme de cas d'utilisation.....	p.31
<u>Annexe 3</u> : Maquettage de l'authentification.....	p.32
<u>Annexe 4</u> : Maquettage de création de compte.....	p.33
<u>Annexe 5</u> : Maquettage de l'accueil de l'application.....	p.34
<u>Annexe 6</u> : Maquettage de la partie admin de l'application.....	p.35
<u>Annexe 7</u> : Diagramme d'activité du cas d'utilisation « authentification » .....	p.36
<u>Annexe 8</u> : Diagramme d'activité du cas d'utilisation « ajout d'un utilisateur » .....	p.37
<u>Annexe 9</u> : Diagramme de classe de l'application ServiceCoPro.....	p.38
<u>Annexe 10</u> : Diagramme de séquence « ajouter un utilisateur » .....	p.39
<u>Annexe 11</u> : MCD (Modèle Conceptuel de Données) de l'application.....	p.40
<u>Annexe 12</u> : MLD (Modèle Logique de Données) de l'application.....	p.41
<u>Annexe 13</u> : Vue relationnelle des tables de la base de données.....	p.42
<u>Annexe 14</u> : Création de la vue « VueLogin.fxml » sous SceneBuilder.....	p.43
<u>Annexe 15</u> : Code de la classe « Main » de l'application Java Fx.....	p.44
<u>Annexe 16</u> : Extrait du code de la classe « UtilisateurEjb » .....	p.45
<u>Annexe 17</u> : Extrait de la méthode « isEmpty () » du « ControleurVueSignUp » .....	p.46

## Annexe 1 : Diagramme de GANTT

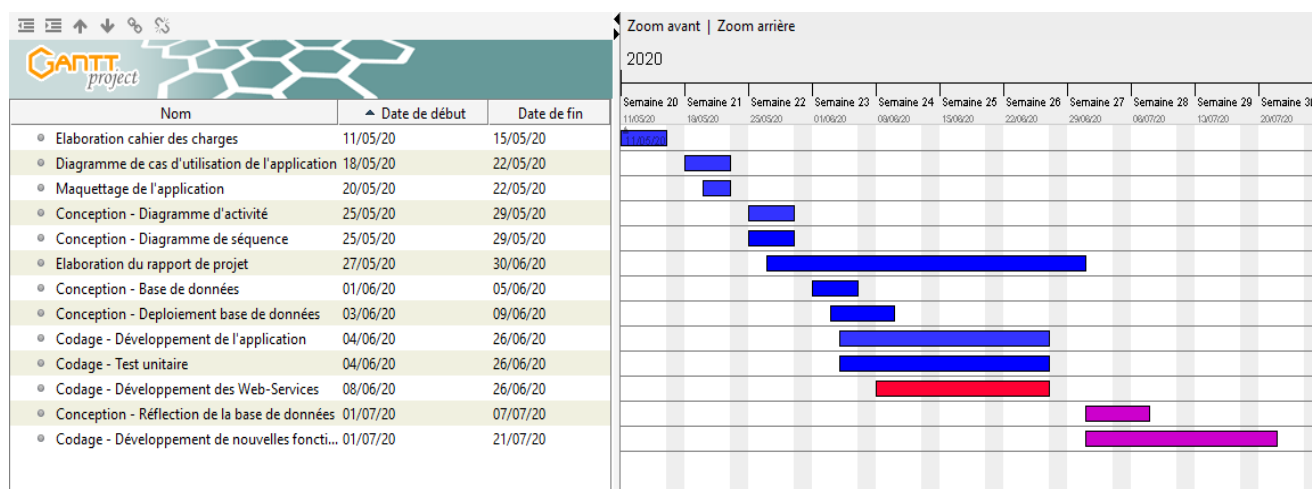


Diagramme de GANTT réalisé sous GANTTPROJECT.

- En bleu : les tâches réalisées,
- En rouge : les tâches en retard sur le planning,
- En violet : les tâches à venir.

## Annexe 2 : Diagramme de cas d'utilisation

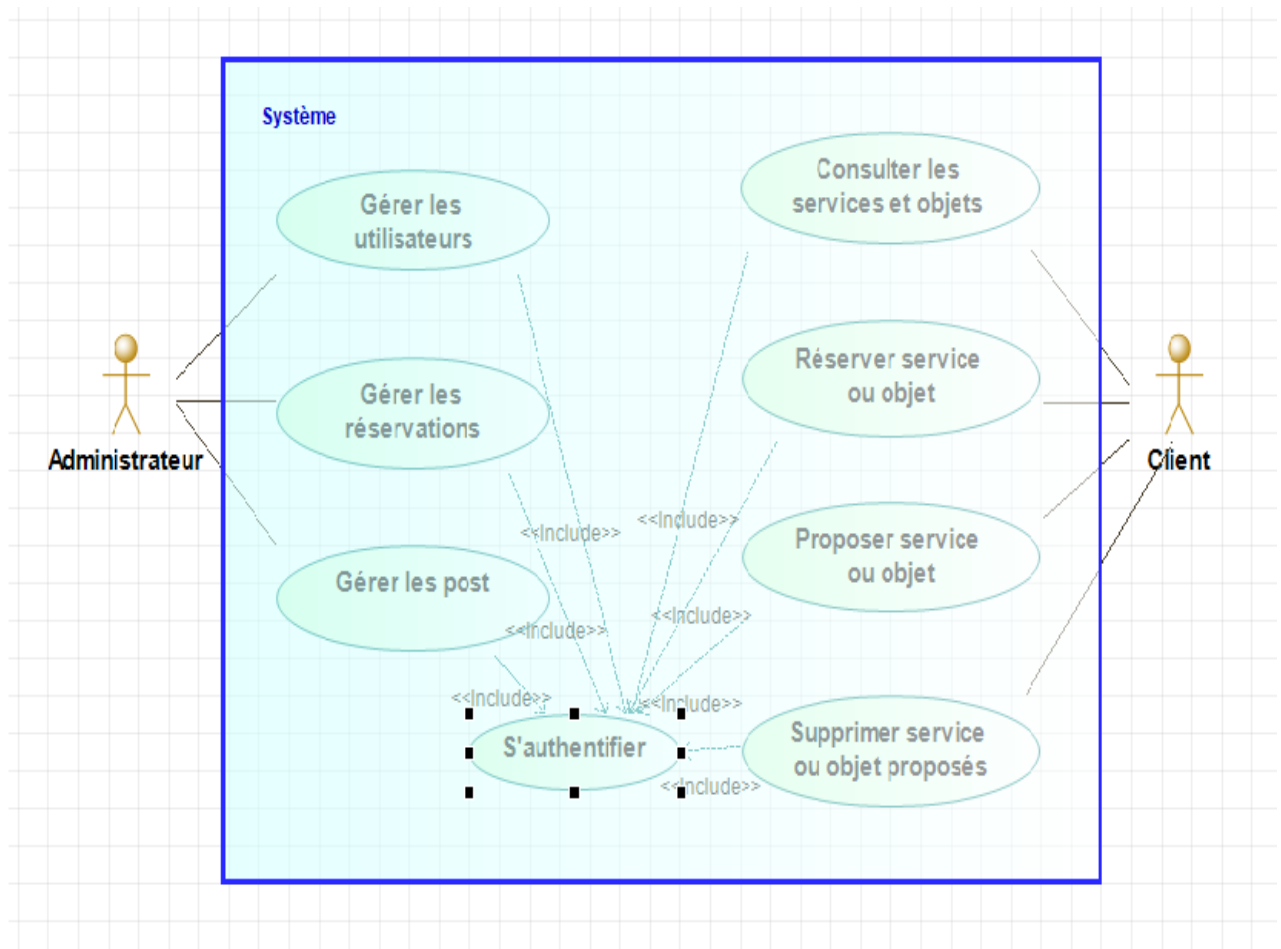
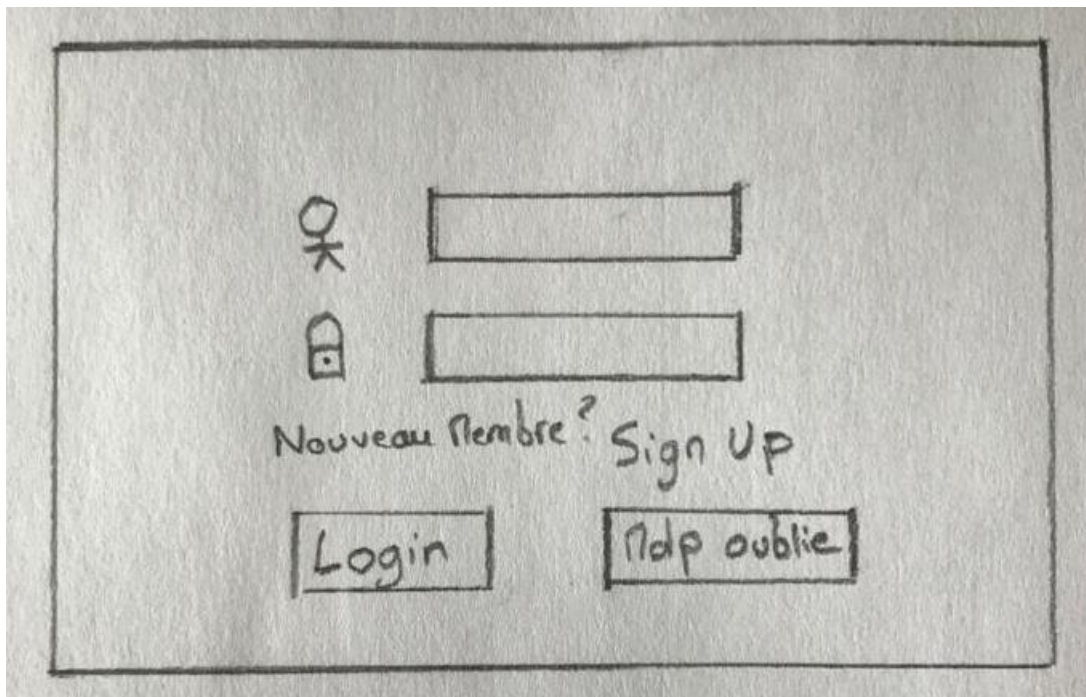


Diagramme de cas d'utilisation.

### Annexe 3 : Maquettage de l'authentification



Maquettage de l'authentification.

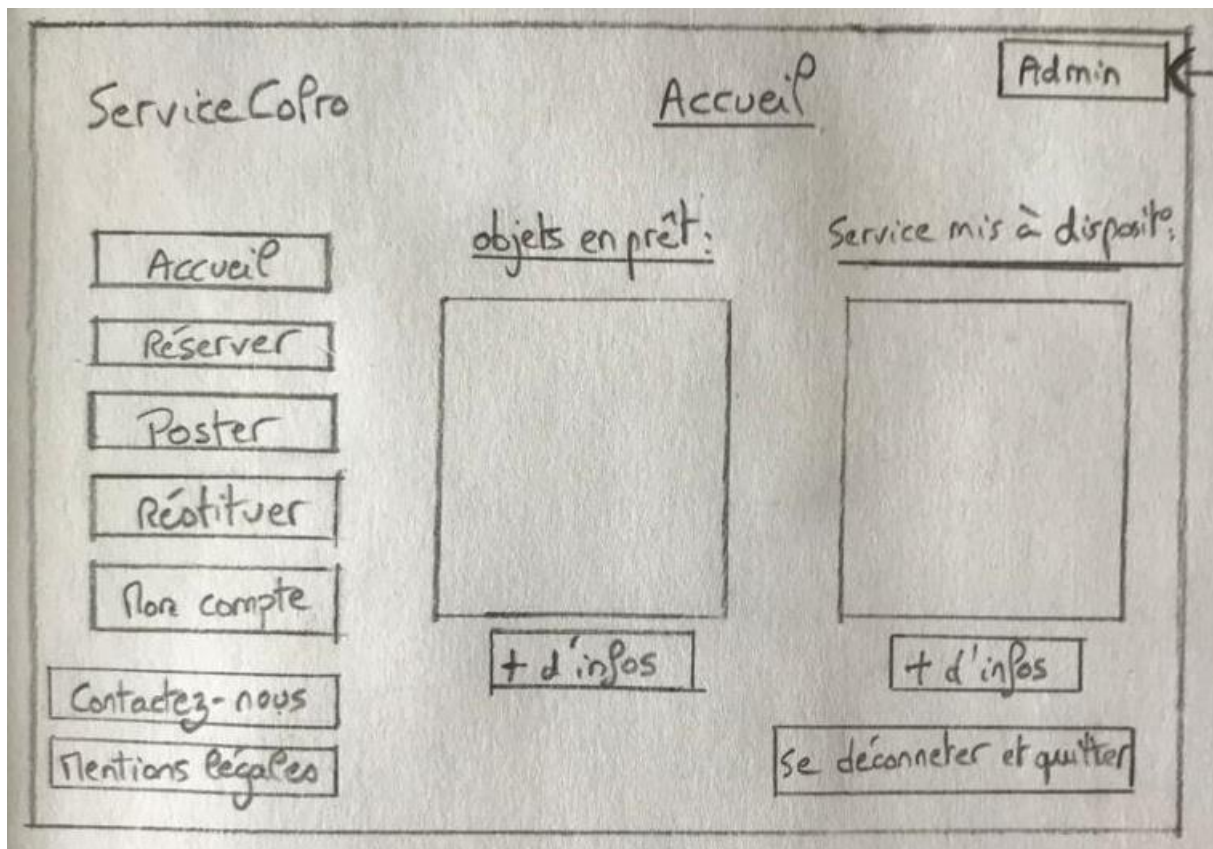


Annexe 4 : Maquettage de création de compte

A hand-drawn wireframe for an account creation form, enclosed in a rectangular border. At the top, there are two radio buttons with labels "Homme" and "Femme". Below these are five input fields, each preceded by a small icon: a stick figure for "Nom utilisateur", a key for "Mot de passe", an envelope for "email", a square for "Batiment", and another square for "Appartement". At the bottom of the form are two buttons labeled "Créer" and "Annuler".

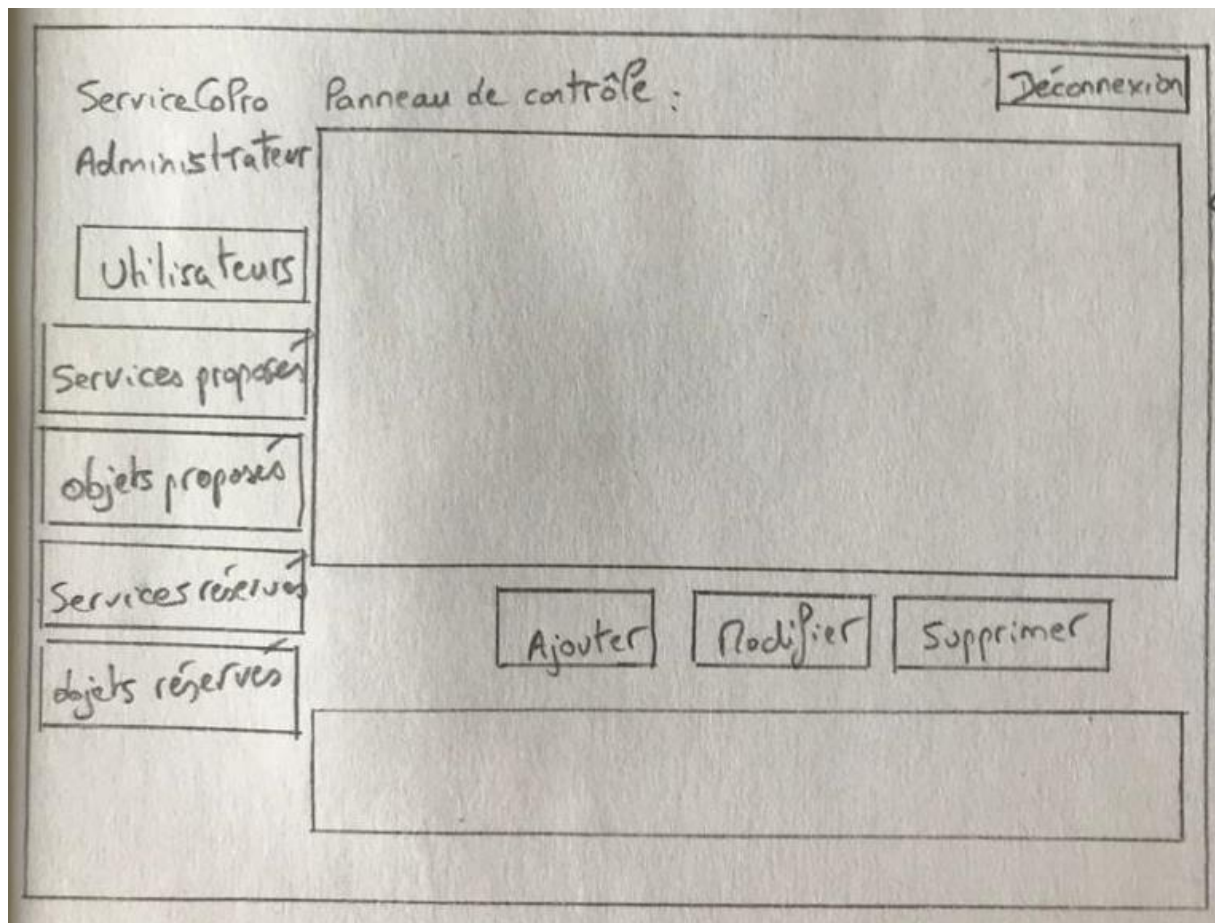
Maquettage de « création compte ».

## Annexe 5 : Maquettage de l'accueil de l'application



Maquettage « accueil » de l'application.

## Annexe 6 : Maquettage de la partie admin de l'application



Maquettage de la partie « admin » de l'application.

Annexe 7 : Diagramme d'activité du cas d'utilisation « authentification »

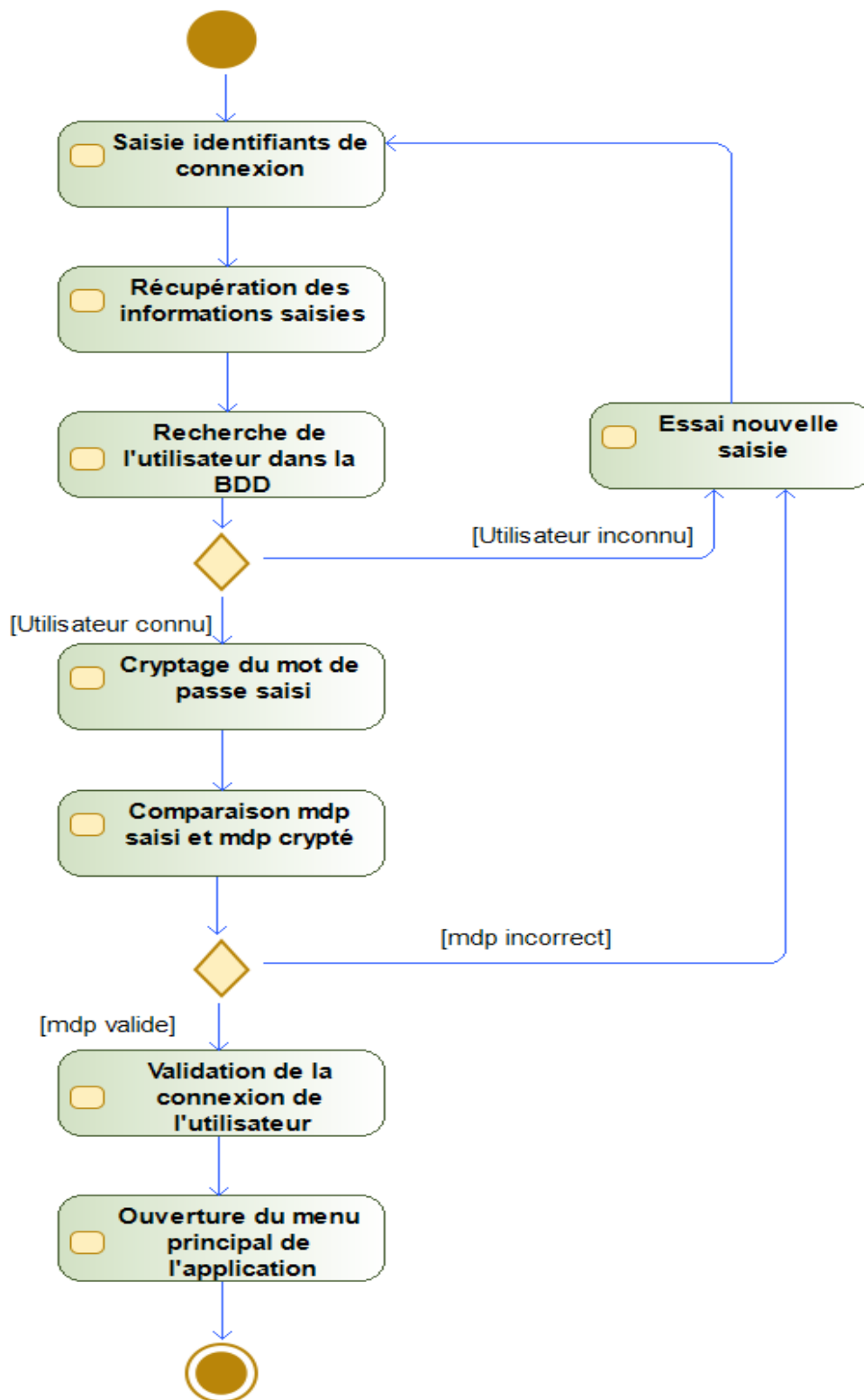


Diagramme d'activité du cas d'utilisation : authentification.

Annexe 8 : Diagramme d'activité du cas d'utilisation « ajout d'un utilisateur »

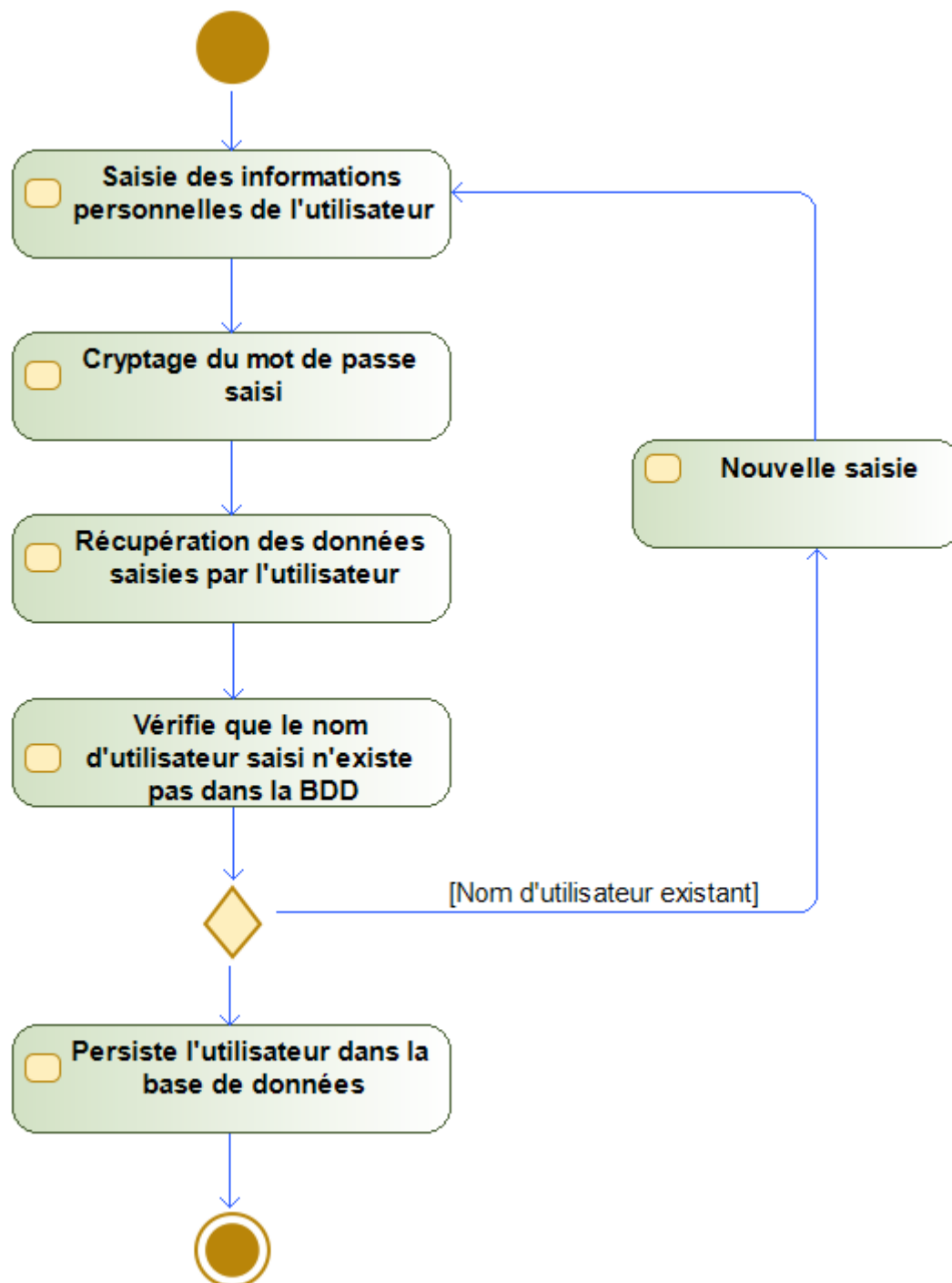


Diagramme d'activité « ajout d'un utilisateur ».

## Annexe 9 : Diagramme de classe de l'application ServiceCoPro

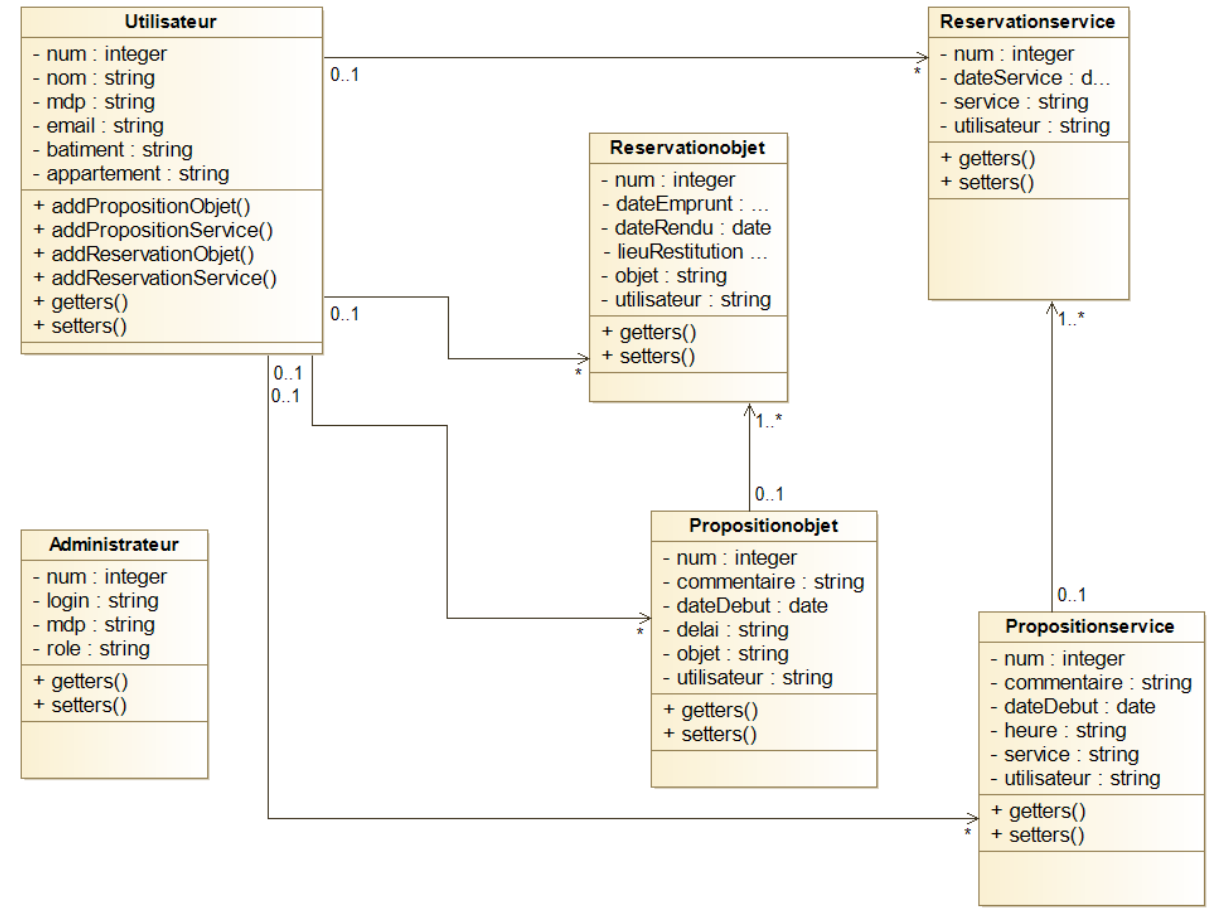


Diagramme de classe de l'application ServiceCoPro.

## Annexe 10 : Diagramme de séquence « ajouter un utilisateur »

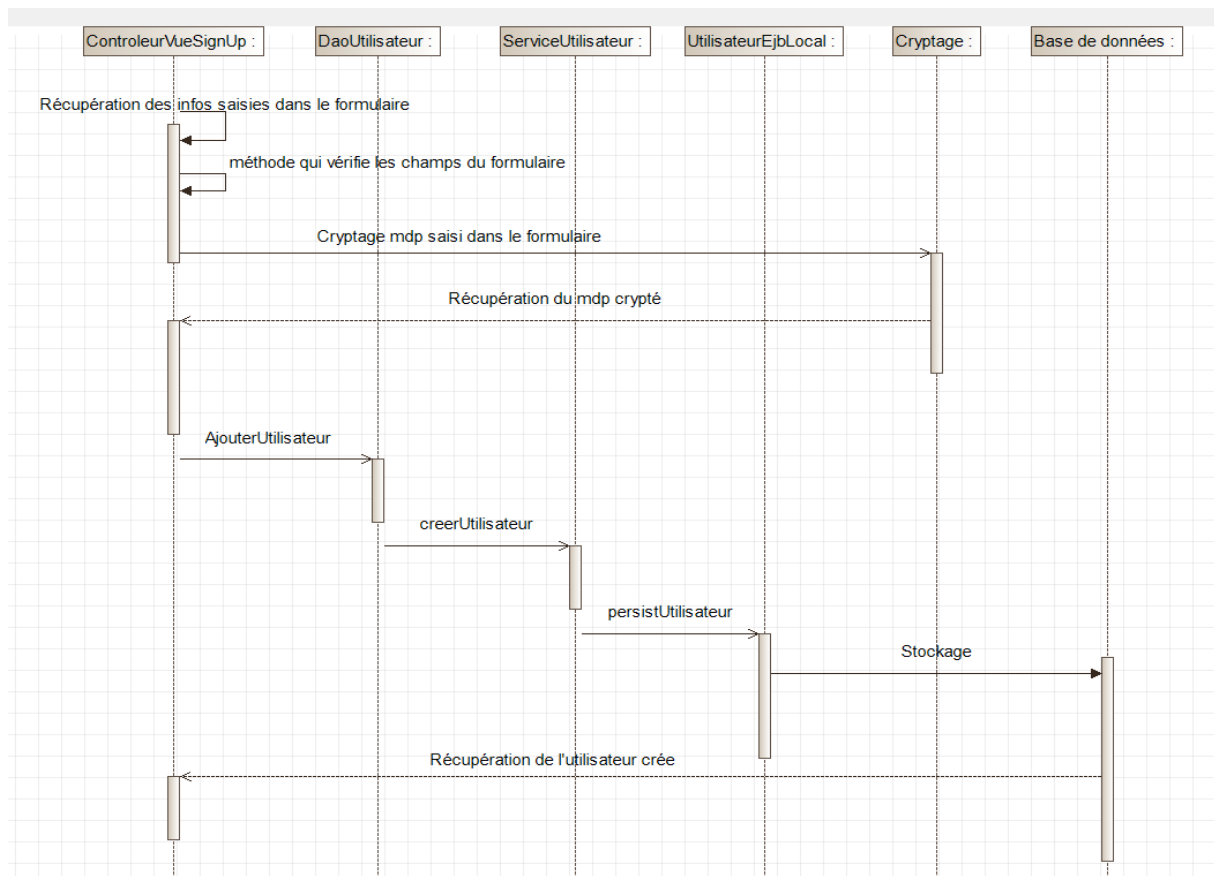
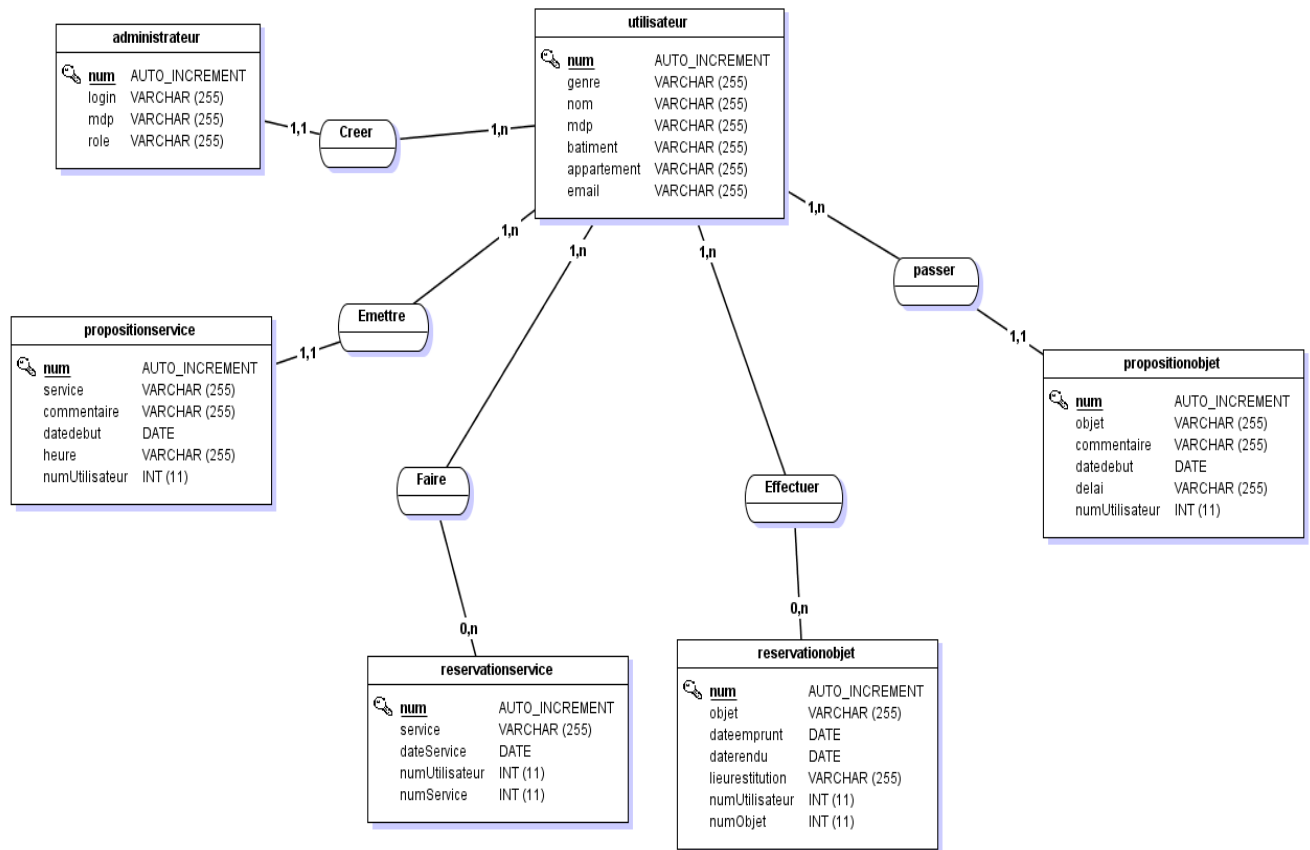


Diagramme de séquence « Ajouter un utilisateur ».

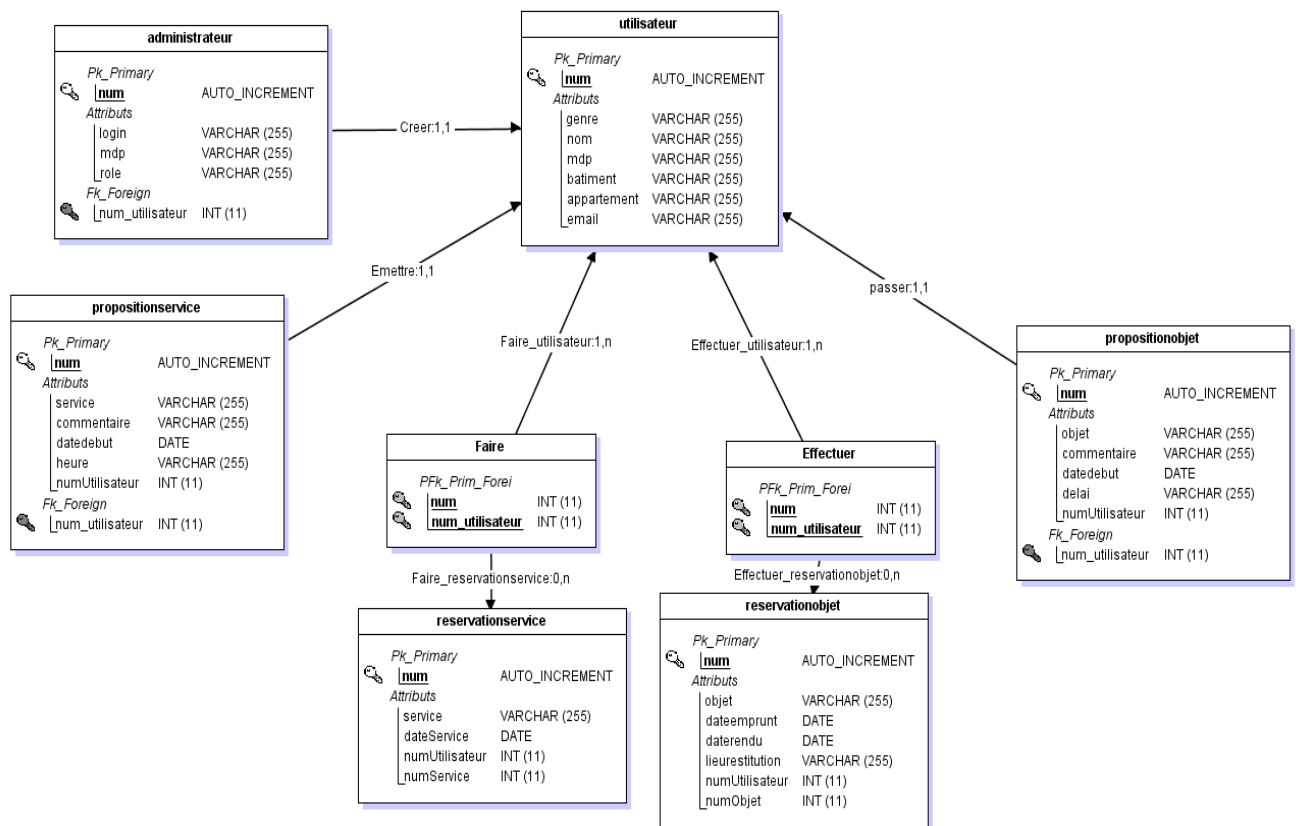
## Annexe 11 : MCD (Modèle Conceptuel de Données) de l'application



MCD de l'application ServiceCoPro.

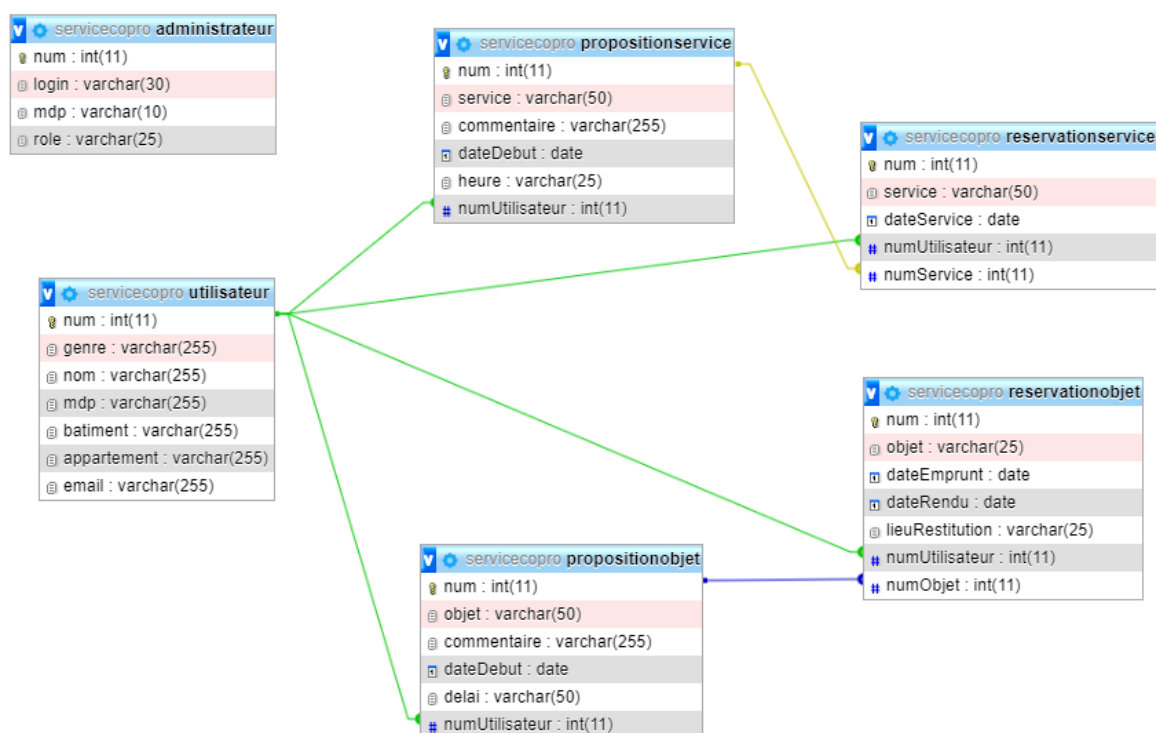


## Annexe 12 : MLD (Modèle Logique de Données) de l'application



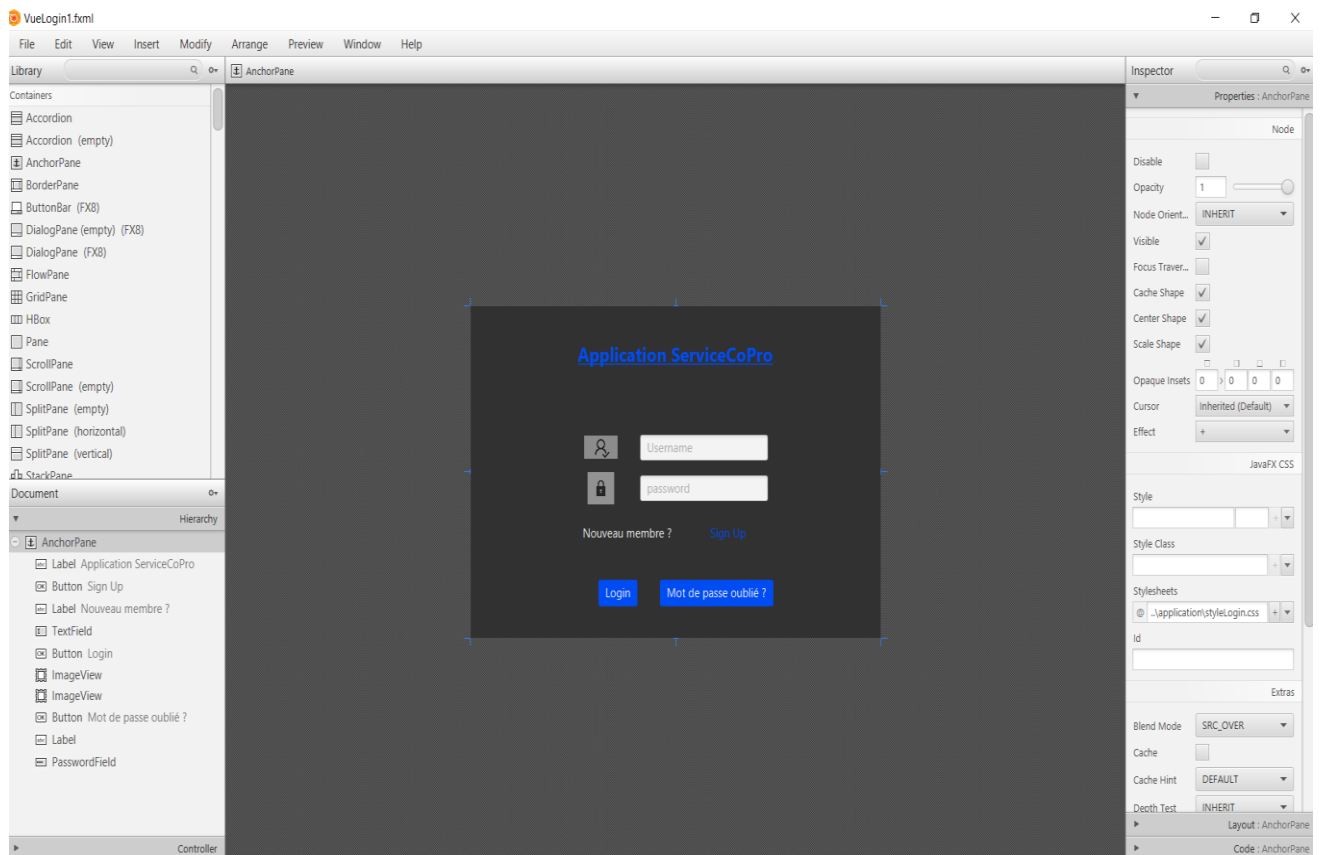
MLD de l'application ServiceCoPro.

## Annexe 13 : Vue relationnelle des tables de la base de données



Vue relationnelle des tables de la base de données.

## Annexe 14 : Création de la vue « VueLogin.fxml » sous SceneBuilder



Création de la vue « VueLogin.fxml » sous SceneBuilder.

## Annexe 15 : Code de la classe « Main » de l'application Java Fx

```
package application;

import controller.ControleurVueLogin;

public class Main extends Application {

    @Override
    public void start(Stage primaryStage) throws Exception {
        try {
            FXMLLoader loader = new FXMLLoader();
            loader.setLocation(Main.class.getResource("/view/VueLogin1.fxml"));
            AnchorPane root = (AnchorPane) loader.load();

            Scene scene = new Scene(root, 600, 400);
            scene.getStylesheets().add(getClass().getResource("styleLogin.css").toExternalForm());
            primaryStage.setTitle("ServiceCoPro");
            primaryStage.setScene(scene);
            primaryStage.setResizable(false);

            MainMenuPrincipal mainMenuPrincipal = new MainMenuPrincipal(primaryStage);
            ControleurVueLogin controleur = loader.getController();
            controleur.setMainMenuPrincipal(mainMenuPrincipal);

            MainSignUp mainSignUp = new MainSignUp(primaryStage);
            ControleurVueLogin controleurSignUp = loader.getController();
            controleurSignUp.setMainSignUp(mainSignUp);

            primaryStage.show();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    public static void main(String[] args) {
        launch(args);
    }
}
```

Code de la classe « Main » de l'application Java Fx.

## Annexe 16 : Extrait du code de la classe « UtilisateurEjb »

```
1  /*
2  * Cette classe est implémentée sous la forme d'un EJB de type Stateless.
3  */
4
5  @Stateless(mappedName = "utilisateurEjb")
6  @LocalBean
7  public class UtilisateurEjb implements UtilisateurEjbLocal, UtilisateurEjbRemote {
8
9      @Resource
10     SessionContext sessionContext;
11
12     @PersistenceContext(unitName = "serviceCoPro")
13     private EntityManager em;
14
15     /**
16      * Default constructor.
17      */
18
19     public UtilisateurEjb() {
20         // TODO Auto-generated constructor stub
21     }
22
23     @Override
24     public Utilisateur persistUtilisateur(Utilisateur utilisateur) {
25         em.persist(utilisateur);
26         return utilisateur;
27     }
28
29     @Override
30     public Utilisateur mergeUtilisateur(Utilisateur utilisateur) {
31         return em.merge(utilisateur);
32     }
33
34     @Override
35     public Utilisateur getUtilisateur(String nomutilisateur) {
36         List<Utilisateur> listUtilisateur = getUtilisateurFindAll();
37
38         for (Utilisateur u : listUtilisateur)
39             if (u.getNom().equals(nomutilisateur)) {
40                 return u;
41             }
42         return null;
43     }
44 }
```

Extrait du code de la classe « UtilisateurEjb ».

## Annexe 17 : Extrait de la méthode « isEmpty () » du « ControleurVueSignUp »

```
/*
 * méthode qui vérifie si les champs du formulaire sont corrects
 */
private boolean isEmpty() throws SQLException {
    boolean Bouton = (homme.isSelected() || femme.isSelected());
    String Nom = utilisateur.getText();
    String Mdp = mdp.getText();
    String Mail = email.getText();
    // vérifie le format de mail saisi par l'utilisateur
    String masque = "^([a-zA-Z]+[a-zA-Z0-9\\._-]*[a-zA-Z0-9]@[a-zA-Z]+\"
        + \"[a-zA-Z0-9\\._-]*[a-zA-Z0-9]+\\.\"[a-zA-Z]{2,4})$\";
    Pattern pattern = Pattern.compile(masque);
    Matcher controler = pattern.matcher(Mail);

    String Bat = batiment.getText();
    String Apt = appartement.getText();
    if (Bouton == false) {
        System.out.println(\"Saisir votre genre\");
        Alert alert = new Alert(AlertType.ERROR);
        alert.setContentText(\"Veuillez sélectionner votre genre\");
        alert.showAndWait();
        return false;
    }
    if (Nom.length() < 3) {
        System.out.println(\"Le nom d'utilisateur doit contenir au moins 3 caractères\");
        Alert alert = new Alert(AlertType.ERROR);
        alert.setContentText(\"Le nom d'utilisateur doit contenir au moins 3 caractères\");
        alert.showAndWait();
        return false;
    }
    if (!controler.matches()) {
        Alert alert = new Alert(AlertType.ERROR);
        alert.setContentText(\"Le mail saisi est incorrect\");
        alert.showAndWait();
        System.out.println(\"le mail est incorrect\");
        return false;
    }
    if (Mdp.length() < 4) {
        Alert alert = new Alert(AlertType.ERROR);
        alert.setContentText(\"Le mot de passe doit contenir au moins 4 caractères\");
        alert.showAndWait();
        System.out.println(\"le mot de passe doit contenir au moins 4 caractères\");
        return false;
    }
}
```

Extrait de la méthode « isEmpty () » de la classe « ControleurVueSignUp ».