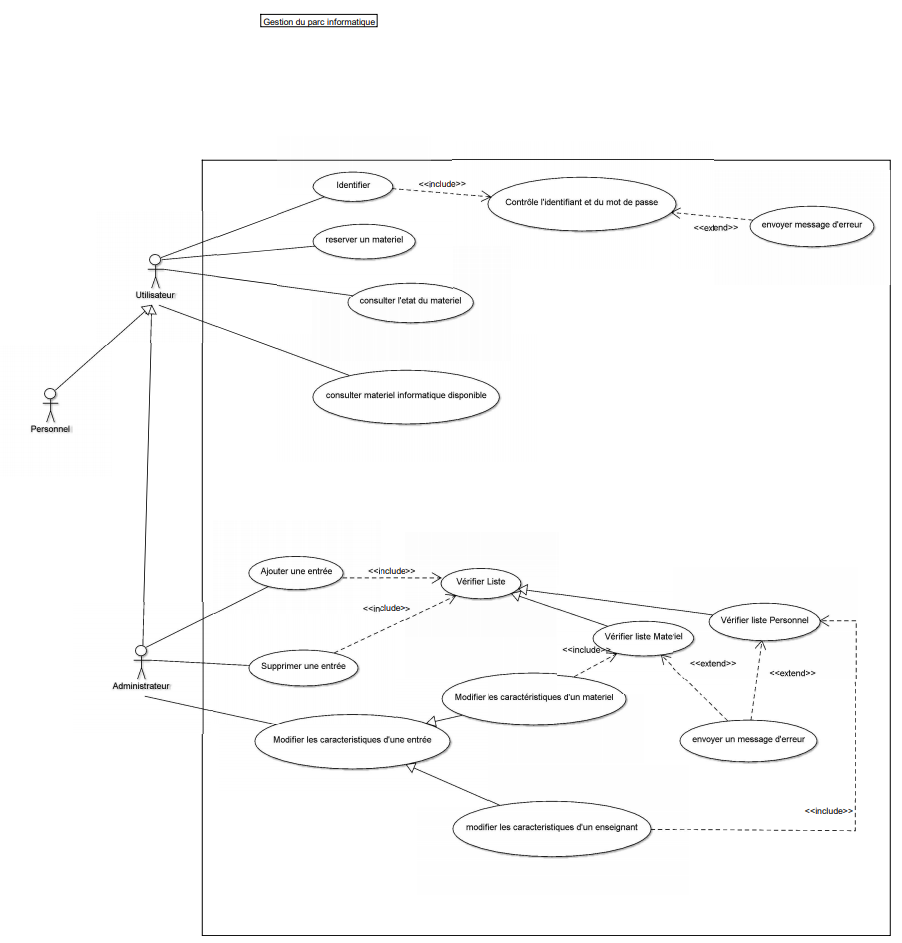
# Rapport projet – AP4B

# Introduction :

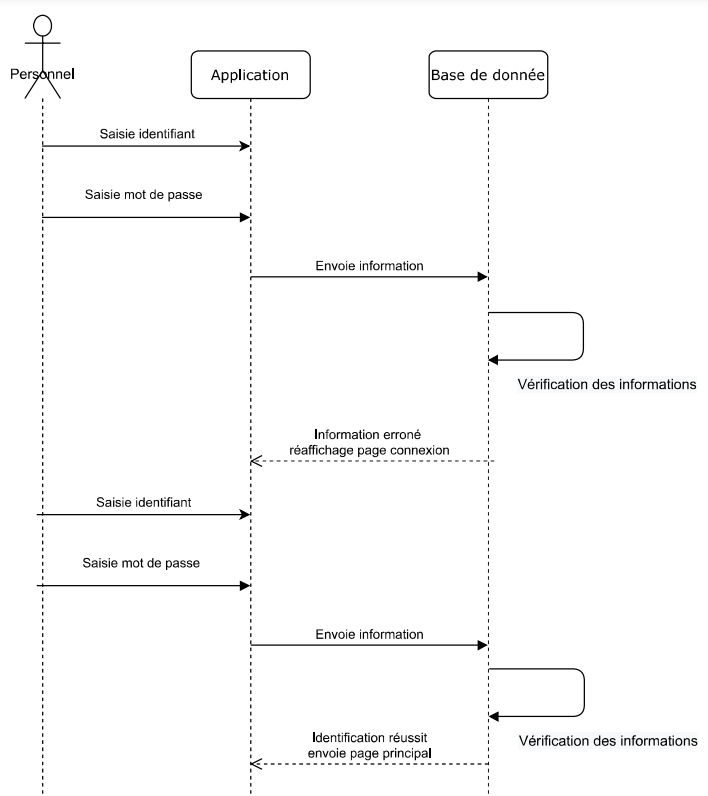
Dans ce projet, nous avions pour consigne de réaliser une application permettant à un établissement de gérer son parc informatique. Pour se faire, nous avons utilisé le langage de programmation JAVA ainsi que la librairie sqlite-jdbc-3.34.0.

# Analyse du cahier des charges

Nous avons commencé par réaliser une analyse du cahier des charges en nous demandant quelles étaient les différentes utilisations fonctionnelles de notre application, et quels allaient être les différents types d’acteur qui l’utiliseront.

Nous en avons donc tiré de cette analyse un diagramme de cas d’utilisation que nous avons mis ci-dessous :

Nous avons ensuite réalisé quelques scénarios afin de mieux penser la manière dont l’utilisateur va interagir avec notre application.

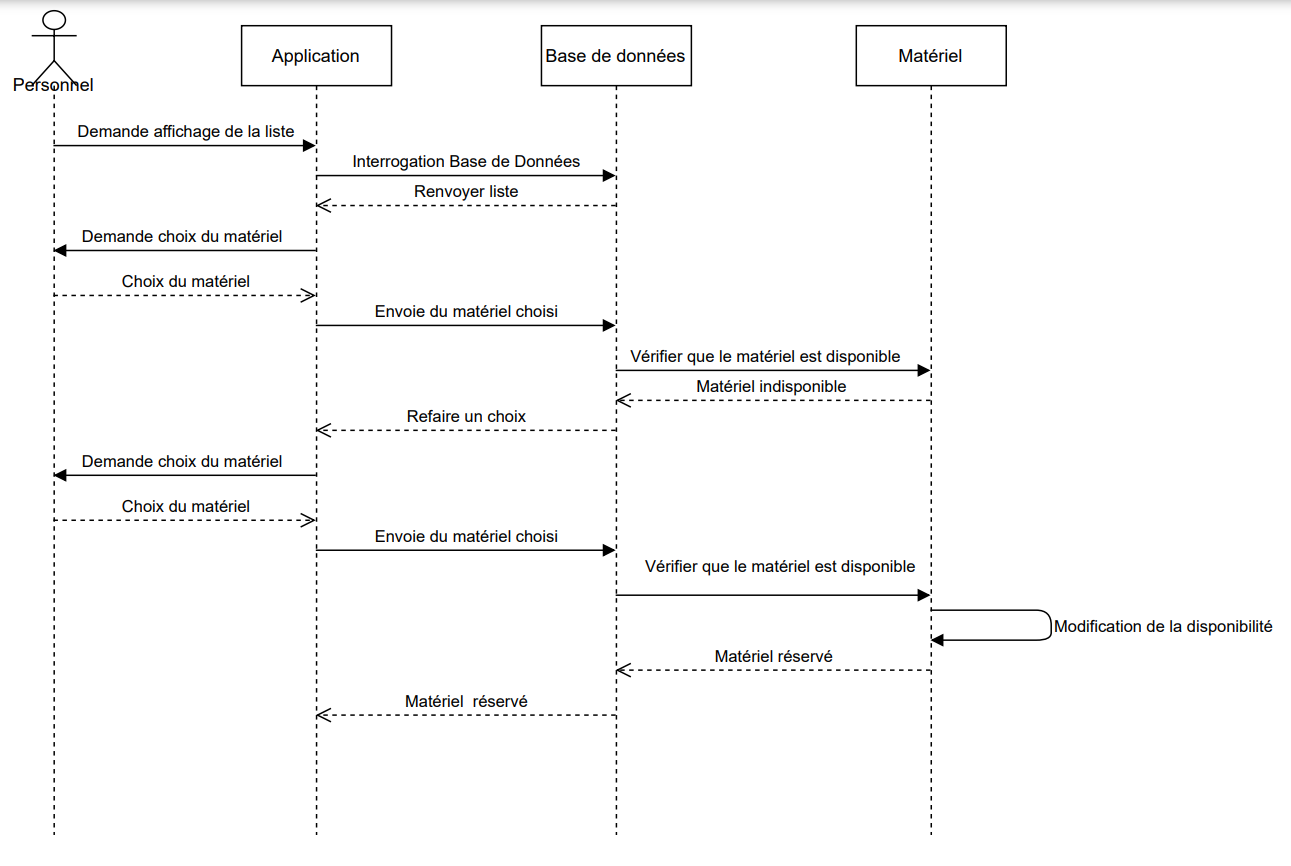
Nous avons commencé par réaliser le scénario de connexion :

Dans ce scénario, nous prenons un utilisateur qui va dans un premier temps se tromper sur le mot de passe ce que l’application lui fait savoir en lui demandant de recommencer. Enfin l’utilisateur entre le bon mot de passe ce qui lui donne accès au menu principal.

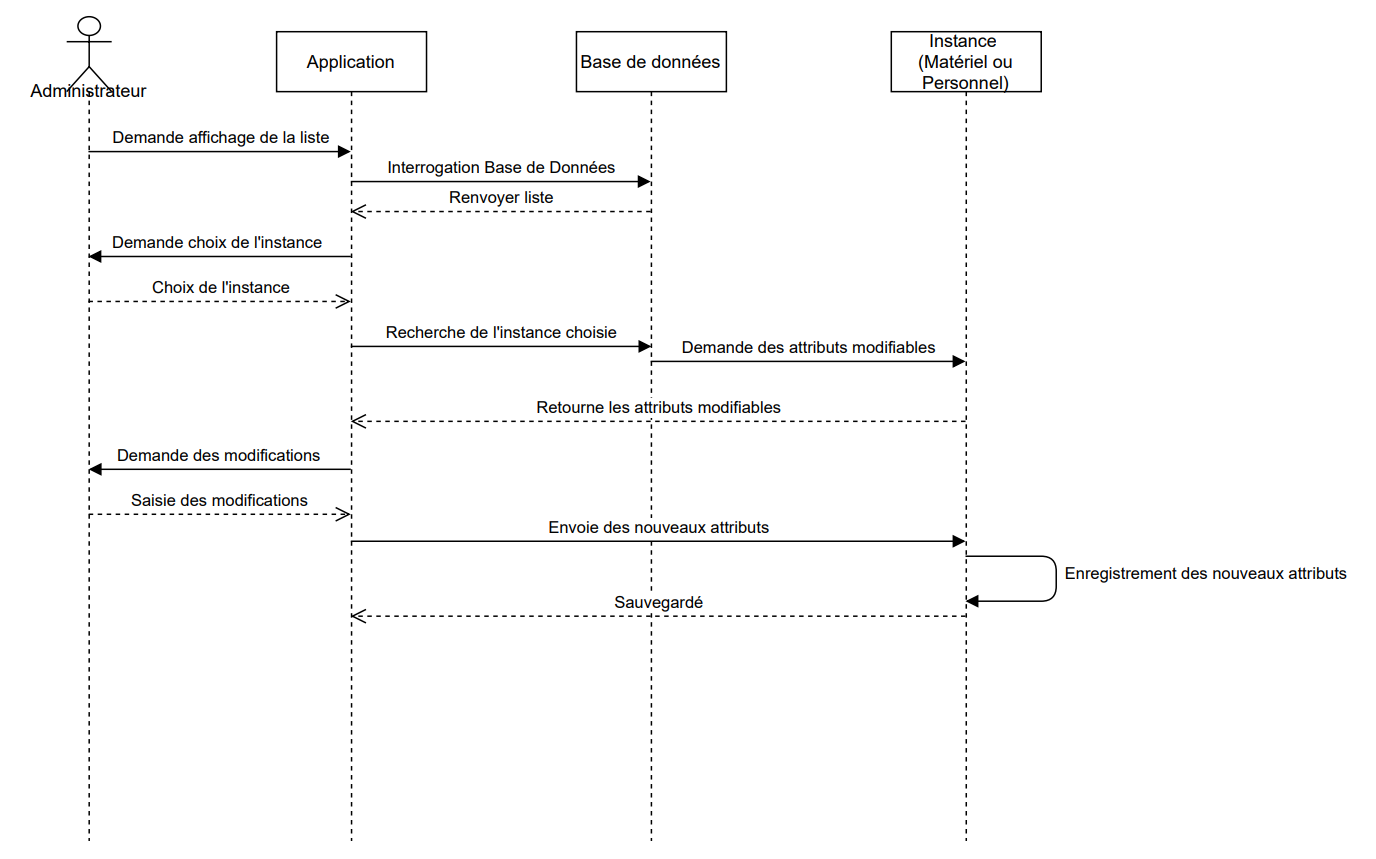
Ensuite, nous nous sommes arrêtés sur le scénario consistant à la réservation d’un matériel ce que nous avons représenté dans le diagramme présent sur la page suivante.

Dans ce scénario, nous prenons en considération un utilisateur choisissant un matériel parmi la liste de matériel, il décide ensuite la date et l’heure de sa réservation. Cependant, après vérification, l’application lui indique que le matériel n’est plus disponible à cette période. C’est pourquoi il recommence sa réservation.

Voici le diagramme de scénario correspondant :



Nous avons terminé par réaliser le scénario de modification des attributs d’une instance. Une instance peut être un utilisateur ou un matériel enregistré dans la base de données. Les seuls utilisateurs pouvant apporter des modifications sont les administrateurs, c’est pourquoi nous avons décidé de mettre un administrateur dans ce scénario.

Voici le diagramme associé à ce scénario.

# Conception de notre projet

Enfin, à partir de ces différents scénarii, et de notre diagramme de cas d’utilisation, nous avons dessiné le diagramme de classe de notre application (sur la page suivante).

L’ensemble des informations du projet sont enregistrés sur une base de données SQLite. C’est pourquoi dans notre diagramme de classe n’est présent que la classe permettant la connexion et l’interrogation de cette base de données : ConnexionSQL.

De plus, nous avons décidé de partir sur une conception suivant le modèle MVC. C’est pourquoi nous avons réparti nos classes dans trois packages différents. Nous avons séparé les classes ne se rapportant qu’au modèle (les informations dont le programme a besoin pour fonctionner), les classes ne se rapportant qu’à la vue (nous y reviendrons plus tard) et les classes qui ne se rapportent qu’aux différents Controllers permettant les interactions avec le logiciel.

Les classes présentes dans le Controller sont toutes, à l’exception de la classe main, des classes héritant de MouseListener. Nous avons en effet choisi d’implémenter ce listener sur l’ensemble de nos Controller car nous manipulerons des boutons. De plus, nous avons fait le choix de réaliser un Controller par type de vue. C’est ainsi que nous avons presque le même nombre de classe dans le package vue que dans le package Controller. Cependant, ayant dans la majorité des Controllers la même définition et dans un souci de lisibilité, nous avons décidé dans le diagramme de classe de n’en représenter qu’un. Nous avons, pour les Controllers particuliers prenant certains attributs, pris la décision de les représenter sur le diagramme.

Ensuite, nous avons remarqué que nous avons de nombreuses classes permettant des vues différentes. Cependant, lorsque nous avons imaginé le design de nos différentes pages, nous avons remarqué que nous nous orientons vers un ensemble de formulaire (les listes étant des formulaires non modifiables). C’est pourquoi, à la place de réaliser de nombreuses fois la même chose, nous avons créé quatre classes permettant de générer rapidement des formulaires. Ces classes : Fenêtre, Formulaires, Marges, et Pop-up permettent respectivement de créer une fenêtre dans laquelle nous pourrons insérer des éléments ; créer un formulaire : nom donnée à un ensemble de champs (qu’il soit fixe comme des labels ou modifiables comme des TextFields) ordonné verticalement et centré sur l’écran (cette page a pour particularité d’être en pleine écran) ; créer des JPannels vierges permettant de créer un espace vide sur l’écran ; créer des pages sous forme de formulaire apparaissant en superposition aux autres pages et n’éteignant pas le programme lorsque nous les fermons.

Nous avons, dans un souci de lisibilité du diagramme, décidé de ne pas représenter sur le model l’ensemble de getters et des setters de nos classes.

Enfin, nous avons décidé, comme vu précédemment, de réaliser une base de données tournant sous le SGBD SQLite. Nous avons ainsi créé, pour représenter cette base de données, un diagramme d’entité association que vous pouvez retrouver en page 6. Nous avons décidé dans cette base de regrouper l’ensemble des utilisateurs sous une seule table. Ainsi, pour différencier les deux types d’utilisateur, un simple champ booléen.

Voici le diagramme de classe :

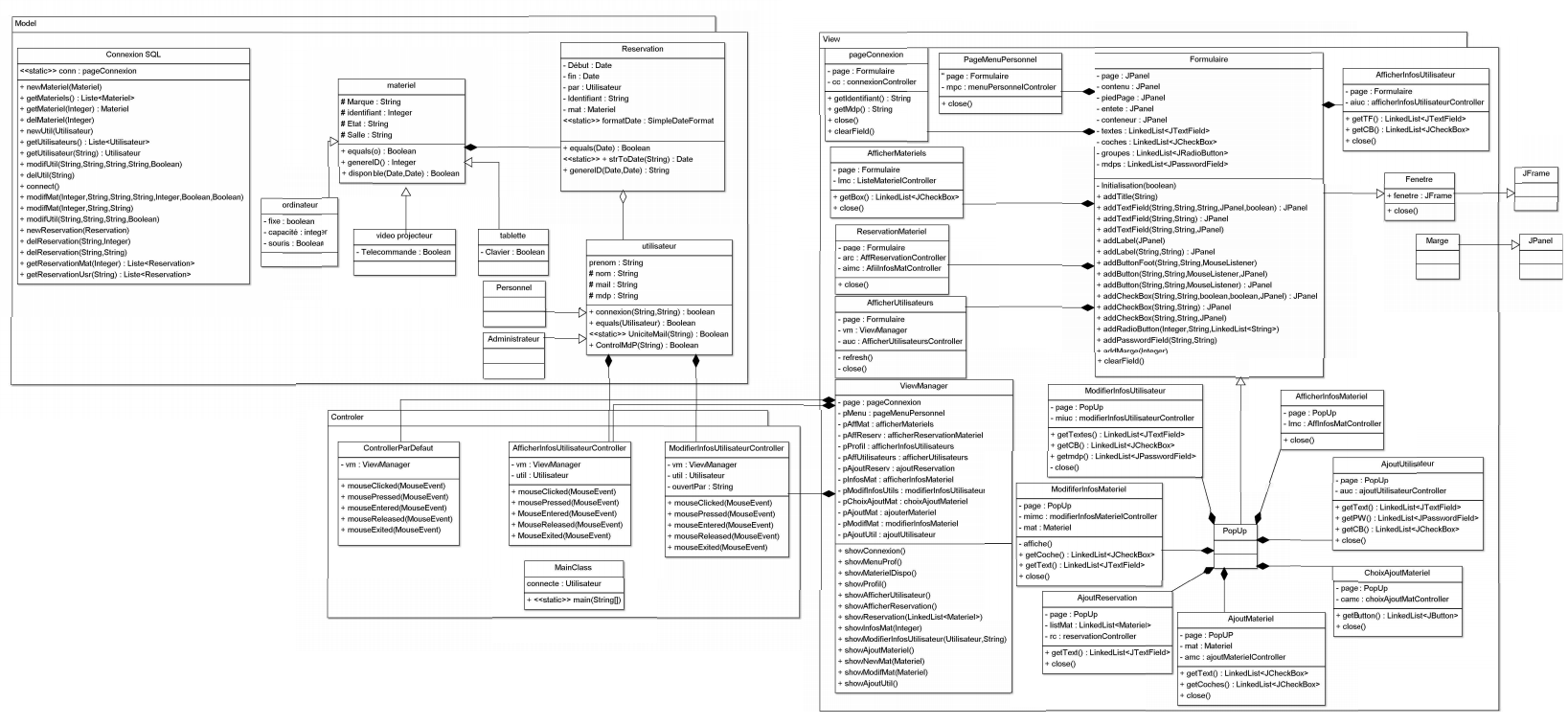


Diagramme d’entité association pour la base de données :

Une image contenant texte, intérieur

Description générée automatiquement

Enfin, pour apporter plus de lisibilité à notre code, nous avons décidé de générer la documentation vis Javadoc, jointe à ce rapport

# Conclusion :

En conclusion, ce projet nous aura permis d’approfondir nos connaissances dans la programmation orientée objet tout en nous permettant d’adopter une démarche d’analyse-conception tirée de la méthode du génie logiciel. Il nous aura aussi permis de découvrir le lien entre la gestion de base de données sous SQLite et le développement JAVA.