

Documentation

Projet ANDROID : Gestion Immobilière

Réalisé en Ateliers Professionnels

Table des matières

1 - Résumé du projet	3
2 - Rappel sur les besoins et fonctionnalités de l'application	4
<u>1) Diagramme de classe</u>	4
<u>2) Diagramme de cas d'utilisation.....</u>	5
<u>3) Maquettes de l'application</u>	6
<u>4) Organisation du code</u>	7
<u>5) Base de données</u>	9
3 - Documentation utilisateur.....	12
<u>1) Les interfaces de l'application</u>	12
4 – Documentation technique	17
<u>1) Ma tâche : Owen</u>	17
<u>2) La classe métier</u>	17
<u>3) La classe DAO</u>	19
<u>4) Les contrôleurs</u>	22
<u>a) ReservationConsultActivity</u>	22
<u>b) ReservationConsultDetailsActivity.....</u>	26
<u>c) ReservationAjoutActivity</u>	30
5 – Mes interfaces	33

1 - Résumé du projet

Le projet Immo'Bill consiste en la conception et la mise en place d'une application mobile de gestion locative pour une agence immobilière. Cette application permettra de faciliter la gestion des réservations de villas ainsi que la visualisation des informations sur les locataires, les villas et les équipements disponibles. Les propriétaires des villas auront également la possibilité d'ajouter, de modifier ou de supprimer certaines informations à propos de leur bien immobilier directement via l'application.

Cette application sera destinée à un usage interne de l'agence Immo'Bill, mais également à un usage externe pour les clients. Elle permettra aux locataires potentiels de consulter les disponibilités des villas, les équipements disponibles, les tarifs et les options de réservation.

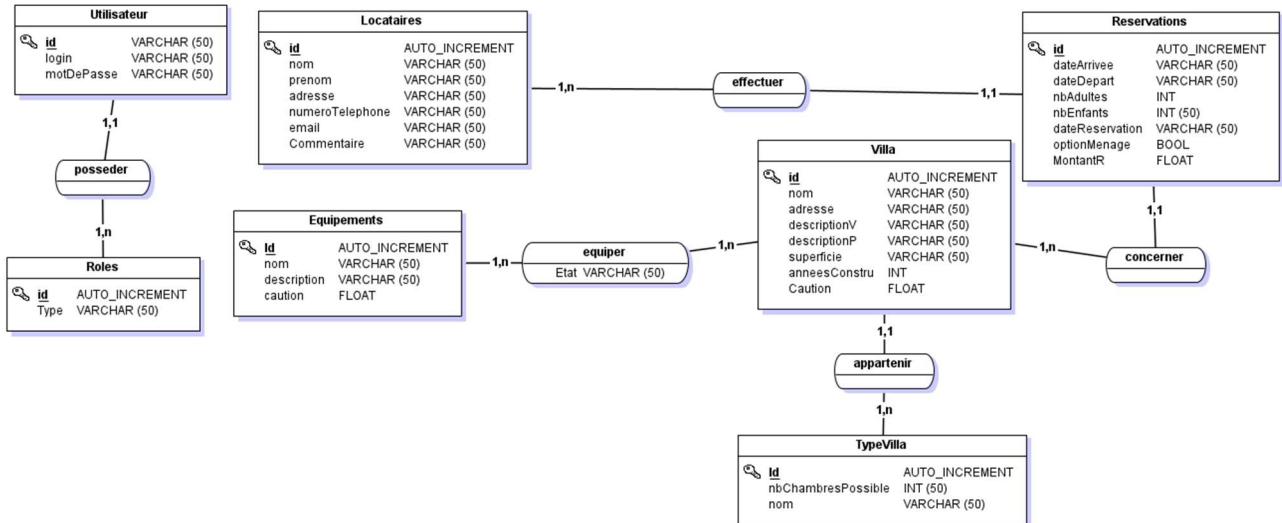
L'application sera conçue pour être facile à utiliser, avec une interface utilisateur intuitive. Le développement de l'application sera effectué en plusieurs étapes, en commençant par la conception d'un diagramme de classe sous forme de MCD pour mieux comprendre les besoins nécessaires au bon développement du projet. Ensuite, une base de données SQLite sera mise en place pour stocker les informations relatives aux villas, aux locataires, aux réservations, aux équipements et aux utilisateurs de l'application.

Dans un souci d'optimisation du temps et d'efficacité, nous avons pris la décision de nous concentrer uniquement sur la gestion interne de l'application, du point de vue de l'administrateur.

2 - Rappel sur les besoins et fonctionnalités de l'application

1) Diagramme de classe

Tout d'abord, afin de bien comprendre les besoins nécessaires pour un développement optimal du projet, nous avons réalisé un diagramme de classes sous forme de modèle conceptuel de données (MCD). Il nous a permis de bien comprendre les relations entre chaque entité ainsi que leur contenu respectif.



Ainsi la base de données est composée des tables :

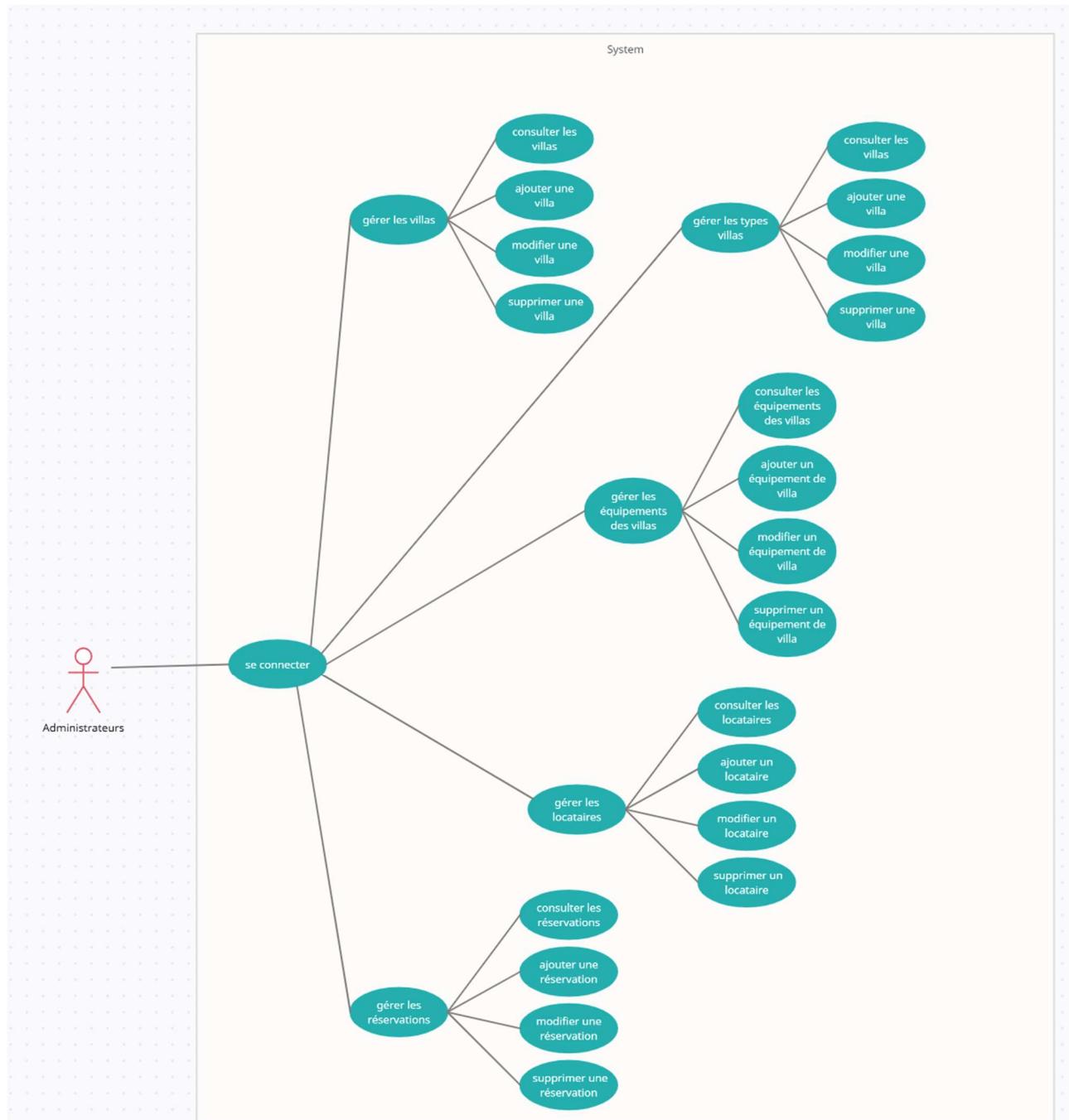
- TypeVilla : contient les informations sur les différents types de villas (nombre de chambres, nom, etc.).
- Locataires : contient les informations sur les locataires (nom, prénom, adresse, numéro de téléphone, etc.).
- Villa : contient les informations sur les villas (nom, adresse, description, etc.), ainsi que l'identifiant du type de villa.
- Equipements : contient les informations sur les équipements (nom, description, caution, etc.).
- Réservations : contient les informations sur les réservations (dates, nombre d'adultes et d'enfants, montant, etc.), ainsi que l'identifiant de la villa réservée et de son locataire.
- Rôles : contient les différents rôles (administrateur, utilisateur, etc.).
- Utilisateur : contient les informations sur les utilisateurs (login, mot de passe, rôle, etc.).
- Equiper : permet de faire correspondre une villa avec les équipements qu'elle contient, ainsi que leur état.

2) Diagramme de cas d'utilisation

Après avoir réalisé notre MCD et avoir compris les besoins de l'administrateur lors de sa gestion immobilière. Nous les avons adapté sous forme de diagramme de cas d'utilisation.

L'administrateur après s'être connecté devra pouvoir gérer :

- Les villas (consulter, ajouter, modifier, supprimer).
- Les types de villas (consulter, ajouter, modifier, supprimer).
- Les équipements des villas (consulter, ajouter, modifier, supprimer).
- Les locataires des villas (consulter, ajouter, modifier, supprimer).
- Les réservations des villas (consulter, ajouter, modifier, supprimer).



3) Maquettes de l'application

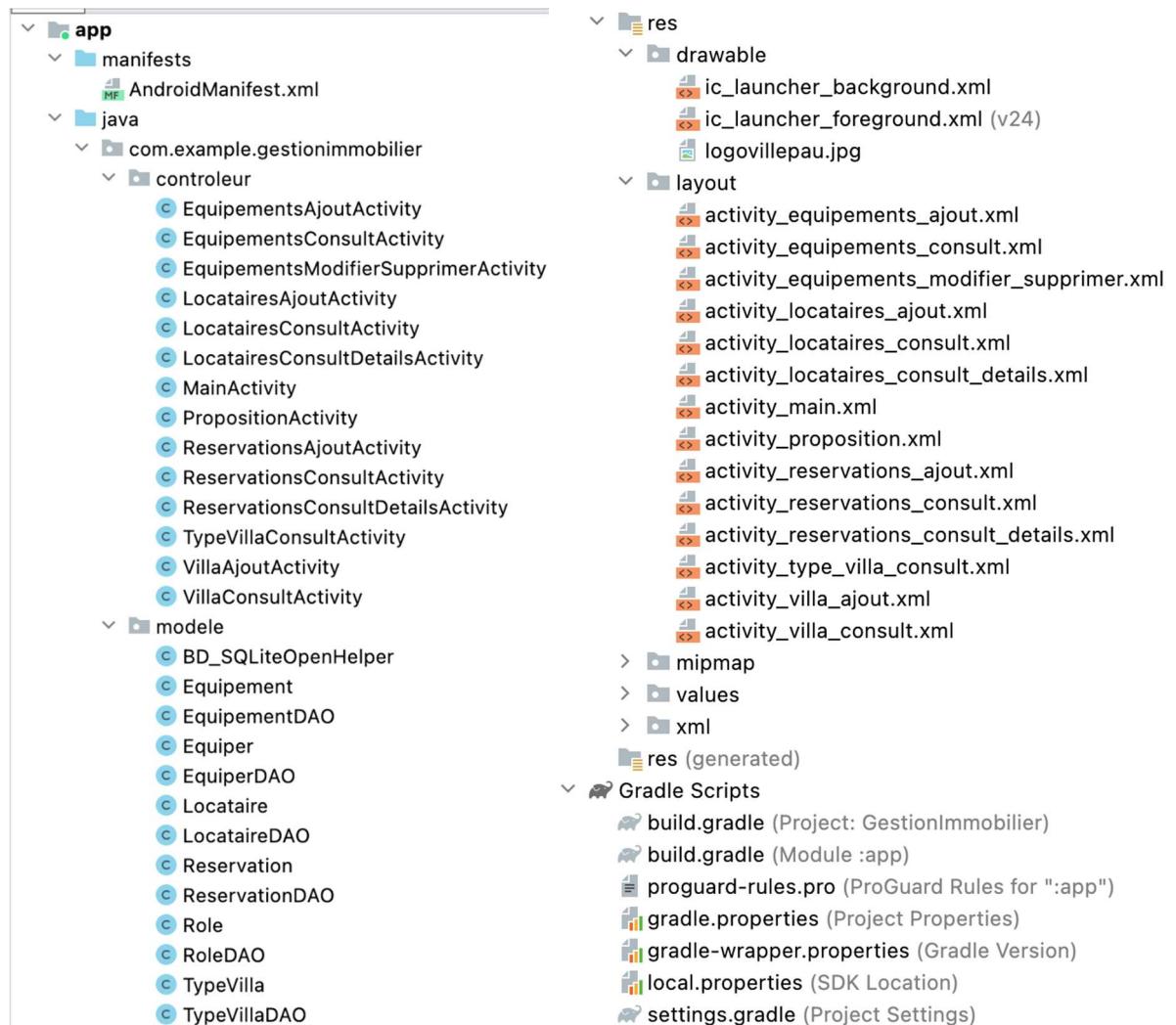
Suite à quoi via le site Figma, nous avons entamé la conception d'une maquette des pages Android nécessaires pour le projet. Nous avons tenu compte des différents besoins des administrateurs internes de l'agence immobilière Immo'Bill et des propriétaires des villas, en veillant à ce que toutes les fonctionnalités soient facilement accessibles et intuitives à utiliser. La maquette que nous avons conçue permettra à l'agence immobilière de gérer efficacement les réservations, les informations sur les locataires, les villas et les équipements.

La maquette que nous avons réalisée se compose de 14 pages fonctionnelles. Parmi ces pages, nous avons prévu une pour la connexion, une pour les propositions, deux pour les différents types de villas, quatre pour les détails des villas et de leurs équipements, trois pour la gestion des locataires et enfin trois autres pour la gestion des réservations.



4) Organisation du code

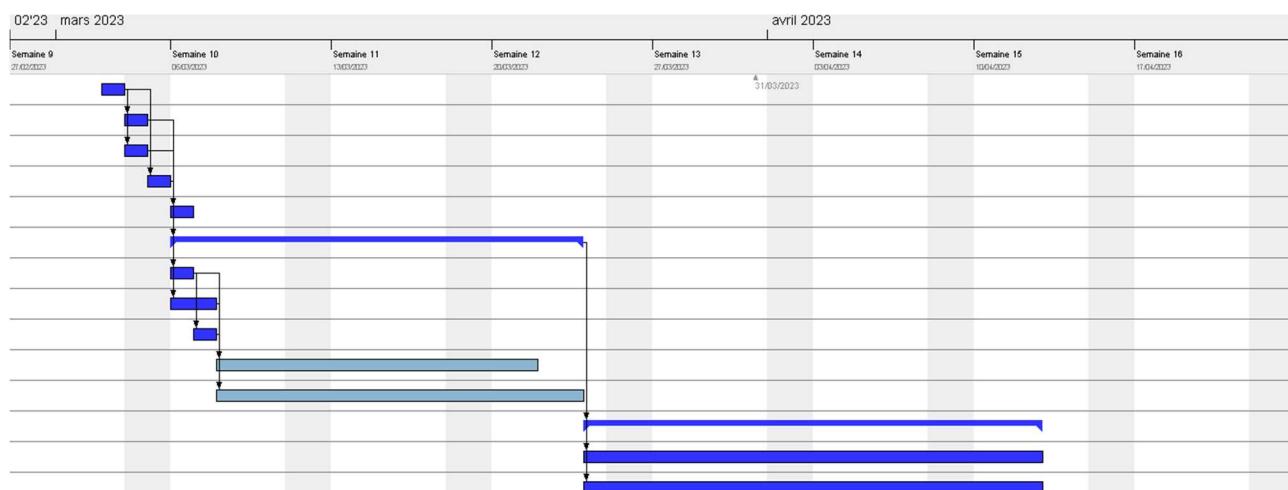
Le projet a été réalisé selon l'architecture MVC (modèle-vue-contrôleur), ce qui permet de séparer et d'organiser efficacement les modèles, les contrôleurs qui les utilisent, ainsi que les vues liées aux contrôleurs.



Proj « Gestion Immobilier »

Afin de réaliser efficacement notre projet, nous avons mis en place un plan de travail détaillé à l'aide d'un projet Gantt. Nous avons ainsi réparti les tâches entre les membres de l'équipe pour maximiser notre efficacité.

Nom	Date de début	Date de fin	Durée	DuréePrecise (heures)	Prédécesseurs	Responsable
Lecture du sujet	03/03/2023	03/03/2023	1	1/2h		Manu, Thibaut, Owen, Lilian
Réalisation du digramme de Gantt	04/03/2023	04/03/2023	1	1h30	0	Owen
Réalisation du MCD (Diagramme de classe)	04/03/2023	04/03/2023	1	1h30	0	Manu, Thibaut, Owen
Réalisation d'exemple de maquette	05/03/2023	05/03/2023	1	1h30	0,76	Thibaut
Créer le diagramme de cas d'utilisation	06/03/2023	06/03/2023	1	1h	33	Manu
▼ Programmation	06/03/2023	23/03/2023	18	10h	33,38,76	Manu, Thibaut, Owen, Lilian
Rédaction de la base de donnée	06/03/2023	06/03/2023	1	1h	38	Owen
Création des vues	06/03/2023	07/03/2023	2	2h	33,38	Thibaut
Création des classes métiers	07/03/2023	07/03/2023	1	30 min	6	Owen
Création des classes DAO	08/03/2023	21/03/2023	14	1h30	48	Manu, Thibaut, Owen, Lilian
Code	08/03/2023	23/03/2023	16	5h	6,7,48	Manu
▼ Créer la documentation du projet	24/03/2023	12/04/2023	20	4h	34	Manu, Thibaut, Owen, Lilian
Documentation technique	24/03/2023	12/04/2023	20	2h	34	Manu, Thibaut, Owen, Lilian
Documentation utilisateur (individuel)	24/03/2023	12/04/2023	20	2h (Libre à chaque personne)	34	Manu, Thibaut, Owen, Lilian



Proj « Gestion Immobilier »

5) Base de données

Pour pouvoir gérer le stockage de toutes les informations nécessaires à notre application nous avons créé un fichier (« BD_SQLiteOpenHelper ») qui nous a permis de créer la base de données dont nous avions besoin. Ainsi nous pourrons utiliser chaque table prévue dans notre MCD.

```
public class BD_SQLiteOpenHelper extends SQLiteOpenHelper{

    //-----
    // Table: TypeVilla
    //-----

    private String createTableTypeVilla = " create table typeVilla ( "
        + " id int, "
        + " nbChambresPossible int, "
        + " nom text, "
        + " primary key (id) "
        + " ); ";

    //-----
    // Table: Locataires
    //-----

    private String createTableLocataires = " create table locataires ( "
        + " id integer primary key AUTOINCREMENT, "
        + " nom text, "
        + " prenom text, "
        + " adresse text, "
        + " numeroTelephone text, "
        + " email text, "
        + " commentaire text "
        + " ); ";

    //-----
    // Table: TypeVilla
    //-----

    private String createTableTypeVilla = " create table typeVilla ( "
        + " id int, "
        + " nbChambresPossible int, "
        + " nom text, "
        + " primary key (id) "
        + " ); ";

    //-----
    // Table: Locataires
    //-----

    private String createTableLocataires = " create table locataires ( "
        + " id integer primary key AUTOINCREMENT, "
        + " nom text, "
        + " prenom text, "
        + " adresse text, "
        + " numeroTelephone text, "
        + " email text, "
        + " commentaire text "
        + " ); ";

    //-----
    // Table: villa
    //-----

    private String createTableVilla = " create table villa ( "
        + " id integer primary key AUTOINCREMENT, "
        + " nom text, "
        + " adresse text, "
        + " descriptionV text, "
        + " descriptionP text, "
        + " superficie text, "
        + " annneesConstru int, "
        + " caution float, "
        + " idTypeVilla int, "
        + " foreign key (idTypeVilla) references typeVilla (id) "
        + " ); ";

    //-----
    // Table: equipements
    //-----

    private String createTableEquipements = " create table equipements ( "
        + " id int, "
        + " nom text, "
        + " description text, "
        + " caution float, "
        + " primary key (id) "
        + " ); ";

    //-----
    // Table: reservations
    //-----

    private String createTableReservations = " create table reservations ( "
        + " id int, "
        + " dateArrivee text, "
        + " dateDepart text, "
        + " nbAdultes int, "
        + " nbEnfants int, "
        + " dateReservation text, "
        + " optionMenage boolean, "
        + " montantR float, "
        + " idVilla int, "
        + " idLocataires int, "
        + " primary key (id), "
        + " foreign key (idVilla) references villa (id), "
        + " foreign key (idLocataires) references locataires (id) "
        + " ); ";

    //-----
    // Table: roles
    //-----

    private String createTableUtilisateur = " create table utilisateur ( "
        + " id text, "
        + " login text, "
        + " motDePasse text, "
        + " idRoles int, "
        + " primary key (id), "
        + " foreign key (idRoles) references roles (id) "
        + " ); ";

    //-----
    // Table: equiper
    //-----

    private String createTableEquiper = " create table equiper ( "
        + " idVilla int, "
        + " idEquipements int, "
        + " etat text, "
        + " primary key (idVilla, idEquipements), "
        + " foreign key (idVilla) references villa (id), "
        + " foreign key (idEquipements) references equipements (id) "
        + " ); ";
}
```

Proj « Gestion Immobilier »

Ensuite pour pouvoir vérifier et tester le stockage de ces informations, nous avons inséré directement depuis le fichier des données dans chaque table pour ne pas les laisser vide.

```

public BD_SQLiteOpenHelper(Context context, String name, SQLiteDatabase.CursorFactory factory, int version) {
    super(context, name, factory, version);
}

@Override
public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
}

@Override
public void onCreate(SQLiteDatabase db){

    db.execSQL("createTableVilla");
    db.execSQL("createTableTypeVilla");
    db.execSQL("createTableRoles");
    db.execSQL("createTableEquipements");
    db.execSQL("createTableReservations");
    db.execSQL("createTableLocataires");
    db.execSQL("createTableUtilisateur");
    db.execSQL("createTableEquiper");

    db.execSQL("INSERT INTO `typeVilla` ('id', 'nbChambresPossible', 'nom') VALUES (1, 1, 'T1')");
    db.execSQL("INSERT INTO `typeVilla` ('id', 'nbChambresPossible', 'nom') VALUES (2, 2, 'T2')");
    db.execSQL("INSERT INTO `typeVilla` ('id', 'nbChambresPossible', 'nom') VALUES (3, 3, 'T3')");
    db.execSQL("INSERT INTO `typeVilla` ('id', 'nbChambresPossible', 'nom') VALUES (4, 1, 'STUDIO')");

    db.execSQL("INSERT INTO `locataires` ('id', 'nom', 'prenom', 'adresse', 'numeroTelephone', 'email', 'commentaire') " +
        "VALUES (3, 'Rousseau', 'Pierre', '85 rue de Genève AMIENS 80000', '06 96 09 17 71', " +
        "'pierre.rousseauchevalier.com', 'Bon accueil')");
    db.execSQL("INSERT INTO `locataires` ('id', 'nom', 'prenom', 'adresse', 'numeroTelephone', 'email', 'commentaire') " +
        "VALUES (30, 'Monnier', 'Margaud', '2 Place du Jeu de Paume VILLEJUIF 94800', '07 40 58 53 14', " +
        "'margaud.monnierberlin.com', 'Dagu des chambres')");
    db.execSQL("INSERT INTO `locataires` ('id', 'nom', 'prenom', 'adresse', 'numeroTelephone', 'email', 'commentaire') " +
        "VALUES (58, 'Lefèvre', 'Sylvie', '96 boulevard Albin Durand CERGY 95800', '06 33 21 59 95', " +
        "'sylvie.lefeuvre@isotc.com', 'Satisfait')");
    db.execSQL("INSERT INTO `locataires` ('id', 'nom', 'prenom', 'adresse', 'numeroTelephone', 'email', 'commentaire') " +
        "VALUES (90, 'Carpentier', 'Thierry', '8 Place Napoléon LAON 02000', '07 60 09 28 90', " +
        "'thierry.carpentier@osso.com', 'Très bon accueil')");
    db.execSQL("INSERT INTO `locataires` ('id', 'nom', 'prenom', 'adresse', 'numeroTelephone', 'email', 'commentaire') " +
        "VALUES (115, 'Goncalves', 'Brigitte', '78 Place de la Madeleine PARIS 75011', '06 28 77 90 15', " +
        "'brigitte.goncalves@ibero.com', 'Bien')");
    db.execSQL("INSERT INTO `equipements` ('id', 'nom', 'description', 'caution') VALUES " +
        "(1, 'réfrigérateur', 'Americain', 100.50)");
    db.execSQL("INSERT INTO `equipements` ('id', 'nom', 'description', 'caution') VALUES " +
        "(2, 'cafetière', 'Kaffait', 25.99)");
    db.execSQL("INSERT INTO `equipements` ('id', 'nom', 'description', 'caution') VALUES " +
        "(3, 'lave-linge', 'Avec Hublot', 150.99)");
    db.execSQL("INSERT INTO `equipements` ('id', 'nom', 'description', 'caution') VALUES " +
        "(4, 'table', 'Ikea', 74.99)");
    db.execSQL("INSERT INTO `equipements` ('id', 'nom', 'description', 'caution') VALUES " +
        "(5, 'sèche-linge', 'Miele', 125.99)");
    db.execSQL("INSERT INTO `villa` ('nom', 'adresse', 'descriptionV', 'descriptionP', 'superficie', " +
        "'anneesConstru', 'caution', 'idTypeVilla') VALUES ('Villa ALIZE', '77, Rue Joseph Heuzé', " +
        "'Villa de petite taille, parfaite pour une petite famille en vacances.', 'Il y a 1 chambres spacieuses.'," +
        "'100', '2003', '1000', 1)");
    db.execSQL("INSERT INTO `villa` ('nom', 'adresse', 'descriptionV', 'descriptionP', 'superficie', " +
        "'anneesConstru', 'caution', 'idTypeVilla') VALUES ('Villa des BTS SIO', '77, Rue John-Persé', " +
        "'Villa de grande taille, parfaite pour une classe de BTS SIO en vacances.', 'Il y a 4 chambres spacieuses.'," +
        "'100', '2003', '1000', 4)");
    db.execSQL("INSERT INTO `villa` ('nom', 'adresse', 'descriptionV', 'descriptionP', 'superficie', " +
        "'anneesConstru', 'caution', 'idTypeVilla') VALUES ('Villa Thibaut', '77, Rue Thibaut', " +
        "'Villa de petite taille, parfaite pour un couple en vacances.', 'Il y a 1 chambres spacieuses.'," +
        "'100', '2003', '1000', 1)");
    db.execSQL("INSERT INTO `villa` ('nom', 'adresse', 'descriptionV', 'descriptionP', 'superficie', " +
        "'anneesConstru', 'caution', 'idTypeVilla') VALUES ('Villa du Crous', '51, Route du Crous', " +
        "'Villa de taille normale, parfaite pour un solitaire en vacances.', 'Il y a 1 chambres spacieuses.'," +
        "'150', '2005', '3000', 1)");
    db.execSQL("INSERT INTO `villa` ('nom', 'adresse', 'descriptionV', 'descriptionP', 'superficie', " +
        "'anneesConstru', 'caution', 'idTypeVilla') VALUES ('Villa MALMAISON', '8, Rue des Champs', " +
        "'Villa de taille moyenne, parfaite pour une famille normale en vacances.', 'Il y a 2 chambres spacieuses.'," +
        "'125', '1999', '2000', 2)");
    db.execSQL("INSERT INTO `villa` ('nom', 'adresse', 'descriptionV', 'descriptionP', 'superficie', " +
        "'anneesConstru', 'caution', 'idTypeVilla') VALUES ('Villa du Tarbes', '51, Route tarbes', " +
        "'Villa de taille normale, parfaite pour un couple en vacances.', 'Il y a 2 chambres spacieuses.'," +
        "'150', '2005', '3000', 2)");
    db.execSQL("INSERT INTO `villa` ('nom', 'adresse', 'descriptionV', 'descriptionP', 'superficie', " +
        "'anneesConstru', 'caution', 'idTypeVilla') VALUES ('Villa du Cros', '51, Route du Cros', " +
        "'Villa de taille normale, parfaite pour une famille en vacances.', 'Il y a 3 chambres spacieuses.'," +
        "'150', '2005', '3000', 3)");
    db.execSQL("INSERT INTO `villa` ('nom', 'adresse', 'descriptionV', 'descriptionP', 'superficie', " +
        "'anneesConstru', 'caution', 'idTypeVilla') VALUES ('Villa des Marseillais', '51, Route de une émission nul', " +
        "'Villa de taille normale, parfaite pour une série nul en vacances.', 'Il y a 3 chambres spacieuses.'," +
        "'150', '2005', '3000', 3)");
    db.execSQL("INSERT INTO `villa` ('nom', 'adresse', 'descriptionV', 'descriptionP', 'superficie', " +
        "'anneesConstru', 'caution', 'idTypeVilla') VALUES ('Villa CAUCHOISE', '9, Chemin de Bellevue', " +
        "'Villa de grande taille, parfaite pour une grande famille ou entre amis en vacances.', 'Il y a 2 chambres spacieuses.'," +
        "'150', '2009', '5000', 2)");
    db.execSQL("INSERT INTO `villa` ('nom', 'adresse', 'descriptionV', 'descriptionP', 'superficie', " +
        "'anneesConstru', 'caution', 'idTypeVilla') VALUES ('Villa GENEVIEVE', '11, Sente des Douaniers', " +
        "'Villa immense parfaite pour une grande famille ou entre amis.', 'Il y a 2 chambres spacieuses.'," +
        "'200', '2009', '5000', 2)");
    db.execSQL("INSERT INTO `villa` ('nom', 'adresse', 'descriptionV', 'descriptionP', 'superficie', " +
        "'anneesConstru', 'caution', 'idTypeVilla') VALUES ('Villa de Bordes', '5, Chemin Hilari', " +
        "'Villa de taille normale, parfaite pour un couple en vacances.', 'Il y a 2 chambres spacieuses.'," +
        "'150', '2005', '3000', 2)");
    db.execSQL("INSERT INTO `villa` ('nom', 'adresse', 'descriptionV', 'descriptionP', 'superficie', " +
        "'anneesConstru', 'caution', 'idTypeVilla') VALUES ('Villa de Bayonne', '51, Route de Bayonne', " +
        "'Villa de taille normale, parfaite pour un couple en vacances.', 'Il y a 2 chambres spacieuses.'," +
        "'150', '2005', '3000', 2)");
    db.execSQL("INSERT INTO `reservations` ('id', 'dateArrivee', 'dateDepart', 'nbAdultes', 'nbEnfants', 'dateReservation', " +
        "'optionMenage', 'montantR', 'idVilla', 'idLocataires') VALUES (1, '18-07-2022', '30-07-2022', 3, 0, '10-02-2022', " +
        "'oui', '300', 1, 1)");
    db.execSQL("INSERT INTO `reservations` ('id', 'dateArrivee', 'dateDepart', 'nbAdultes', 'nbEnfants', 'dateReservation', " +
        "'non', '400', 2, 2)");
    db.execSQL("INSERT INTO `reservations` ('id', 'dateArrivee', 'dateDepart', 'nbAdultes', 'nbEnfants', 'dateReservation', " +
        "'oui', '1500', 3, 3)");
    db.execSQL("INSERT INTO `reservations` ('id', 'dateArrivee', 'dateDepart', 'nbAdultes', 'nbEnfants', 'dateReservation', " +
        "'non', '3000', 4, 4)");
    db.execSQL("INSERT INTO `reservations` ('id', 'dateArrivee', 'dateDepart', 'nbAdultes', 'nbEnfants', 'dateReservation', " +
        "'oui', '1500', 5, 5)");
    db.execSQL("INSERT INTO `reservations` ('id', 'dateArrivee', 'dateDepart', 'nbAdultes', 'nbEnfants', 'dateReservation', " +
        "'non', '4000', 1, 2)");
    db.execSQL("INSERT INTO `reservations` ('id', 'dateArrivee', 'dateDepart', 'nbAdultes', 'nbEnfants', 'dateReservation', " +
        "'oui', '1500', 3, 3)");
    db.execSQL("INSERT INTO `reservations` ('id', 'dateArrivee', 'dateDepart', 'nbAdultes', 'nbEnfants', 'dateReservation', " +
        "'non', '3000', 4, 4)");
    db.execSQL("INSERT INTO `reservations` ('id', 'dateArrivee', 'dateDepart', 'nbAdultes', 'nbEnfants', 'dateReservation', " +
        "'oui', '1500', 5, 5)");
    db.execSQL("INSERT INTO `reservations` ('id', 'dateArrivee', 'dateDepart', 'nbAdultes', 'nbEnfants', 'dateReservation', " +
        "'non', '4000', 1, 2)");
    db.execSQL("INSERT INTO `reservations` ('id', 'dateArrivee', 'dateDepart', 'nbAdultes', 'nbEnfants', 'dateReservation', " +
        "'oui', '1500', 3, 3)");
    db.execSQL("INSERT INTO `reservations` ('id', 'dateArrivee', 'dateDepart', 'nbAdultes', 'nbEnfants', 'dateReservation', " +
        "'non', '3000', 4, 4)");
    db.execSQL("INSERT INTO `reservations` ('id', 'dateArrivee', 'dateDepart', 'nbAdultes', 'nbEnfants', 'dateReservation', " +
        "'oui', '1500', 5, 5)");
    db.execSQL("INSERT INTO `reservations` ('id', 'dateArrivee', 'dateDepart', 'nbAdultes', 'nbEnfants', 'dateReservation', " +
        "'non', '4000', 1, 2)");
    db.execSQL("INSERT INTO `reservations` ('id', 'dateArrivee', 'dateDepart', 'nbAdultes', 'nbEnfants', 'dateReservation', " +
        "'oui', '1500', 3, 3)");
    db.execSQL("INSERT INTO `reservations` ('id', 'dateArrivee', 'dateDepart', 'nbAdultes', 'nbEnfants', 'dateReservation', " +
        "'non', '3000', 4, 4)");
    db.execSQL("INSERT INTO `reservations` ('id', 'dateArrivee', 'dateDepart', 'nbAdultes', 'nbEnfants', 'dateReservation', " +
        "'oui', '1500', 5, 5)");
    db.execSQL("INSERT INTO `utilisateur` ('id', 'login', 'motDePasse', 'idRoles') VALUES " +
        "(1, 'admin', 'admin', 1)");
    db.execSQL("INSERT INTO `utilisateur` ('id', 'login', 'motDePasse', 'idRoles') VALUES " +
        "(2, 'utilisateurauthentifie', 'utilisateurauthentifie', 2)");
    db.execSQL("INSERT INTO `utilisateur` ('id', 'login', 'motDePasse', 'idRoles') VALUES " +
        "(3, 'monier', 'monier', 1)");
    db.execSQL("INSERT INTO `utilisateur` ('id', 'login', 'motDePasse', 'idRoles') VALUES " +
        "(4, 'lefeuvre', 'lefeuvre', 2)");
    db.execSQL("INSERT INTO `utilisateur` ('id', 'login', 'motDePasse', 'idRoles') VALUES " +
        "(5, 'thierry', 'thierry', 2)");
    db.execSQL("INSERT INTO `utilisateur` ('id', 'login', 'motDePasse', 'idRoles') VALUES " +
        "(6, 'goncalves', 'goncalves', 2)");
    db.execSQL("INSERT INTO `equiper` ('idVilla', 'idEquipements', 'etat') VALUES (1, 1, 'Bon état')");
    db.execSQL("INSERT INTO `equiper` ('idVilla', 'idEquipements', 'etat') VALUES (2, 2, 'Très bon état')");
    db.execSQL("INSERT INTO `equiper` ('idVilla', 'idEquipements', 'etat') VALUES (3, 3, 'Bon état')");
    db.execSQL("INSERT INTO `equiper` ('idVilla', 'idEquipements', 'etat') VALUES (4, 4, 'Très bon état')");
    db.execSQL("INSERT INTO `equiper` ('idVilla', 'idEquipements', 'etat') VALUES (5, 5, 'Bon état')");
    db.execSQL("INSERT INTO `equiper` ('idVilla', 'idEquipements', 'etat') VALUES (6, 1, 'Très bon état')");
    db.execSQL("INSERT INTO `equiper` ('idVilla', 'idEquipements', 'etat') VALUES (7, 2, 'Bon état')");
    db.execSQL("INSERT INTO `equiper` ('idVilla', 'idEquipements', 'etat') VALUES (8, 3, 'Très bon état')");
    db.execSQL("INSERT INTO `equiper` ('idVilla', 'idEquipements', 'etat') VALUES (9, 4, 'Bon état')");
    db.execSQL("INSERT INTO `equiper` ('idVilla', 'idEquipements', 'etat') VALUES (10, 5, 'Très bon état')");
    db.execSQL("INSERT INTO `equiper` ('idVilla', 'idEquipements', 'etat') VALUES (12, 2, 'Bon état')");
    db.execSQL("INSERT INTO `equiper` ('idVilla', 'idEquipements', 'etat') VALUES (13, 3, 'Très bon état')");
    Log.d("tag: \"log\", msg: \"base de test cree\"");
}

```

3 - Documentation utilisateur

1) Les interfaces de l'application

Tout d'abord nous avons eu à réaliser la page main du projet, celle de démarrage, soit celle de connexion. Simplement avec trois TextView (« Connectez Vous », « Utilisateur » et « Mot de passe »), deux EditText pour entrer le nom d'utilisateur et son mot de passe puis un bouton « se connecter » qui permettra à l'utilisateur de se connecter et d'accéder à la page suivante. Nous avons aussi ajouté une image en haut de l'écran y insérer le logo de l'entreprise.



Après avoir appuyé sur le bouton « se connecter » l'utilisateur accède donc à la page suivante qui est la page « Proposition ». Celle-ci donnera le choix à son utilisateur (soit l'administrateur) de cliquer sur l'un des trois boutons afficher « Villa », « Locataire » ou encore « Réservations ». Ces trois boutons lui permettront de gérer la gestion complète des villas de leur locataire ainsi que de leur réservation. Sur cette page apparaît également un bouton « retour » pour comme son nom l'indique revenir à la page précédente (Page de connexion). On retrouve un nouvelle fois le logo de l'entreprise.

Proj « Gestion Immobilier »



Si l'utilisateur décide de cliquer sur le bouton « Villa » il tombera sur la page suivante qui lui affichera chacune des villas enregistrées dans la base de données en affichant son nom, son adresse et sa superficie. Une liste déroulante sera présente sous le logo de l'entreprise permettant de filtrer les villas selon leur type (T1, T2, etc...). En bas de la pages deux boutons seront présent le bouton « Retour » pour revenir à la page d'avant et « Ajouter » pour confirmer l'ajout de la villa qu'il aura créé.

Si l'utilisateur souhaite ajouter une villa et clique donc sur le bouton « Ajouter » la page suivante s'affichera. Lui permettant de remplir un formulaire avec toutes les caractéristiques d'une villa. Une liste déroulante situé en bas de page lui donnera la possibilité de sélectionner le type de villa qu'il souhaite ajouter.



Proj « Gestion Immobilier »



En revanche si l'utilisateur clique sur une des villas affichées dans la page « Villa » il ouvrira la page suivante. Ou s'affichera un formulaire remplie des informations de la villa sélectionné. Lui donnant la capacité de changer les informations de celle-ci et de confirmer la modification dans la base de données en appuyant sur « Modifier ». Il pourra également supprimer la villa avec le bouton « Supprimer ». Il restera enfin deux bouton « Retour » et « Voir les équipements » qui lui permettront de voir de quels équipements dispose la villa sélectionnée.

Proj « Gestion Immobilier »



Si sur la page « Proposition » l'utilisateur clique sur « Locataire » il sera redirigé vers cette page. Ici il trouvera une liste de tous les locataires enregistrés dans la base et pourra apercevoir leur nom, prénom et adresse. Deux boutons en bas de pages, « Retour » pour revenir à la page d'avant et « Ajouter » pour ajouter un locataire.



Si l'utilisateur clique sur "Ajouter", il sera redirigé vers une nouvelle page comportant un formulaire qui lui permettra de créer un nouveau locataire en renseignant toutes les informations nécessaires. Enfin, deux boutons seront disponibles en bas de la page, « Retour » pour revenir en arrière et « Ajouter » pour confirmer la création du locataire.

Proj « Gestion Immobilier »

Si l'utilisateur clique sur l'un des locataires affichés sur la page « Locataires », il sera redirigé vers une nouvelle page où il pourra voir toutes les informations relatives au locataire sélectionné. Un formulaire sera également affiché, prérempli avec les informations existantes du locataire, ce qui permettra à l'utilisateur de modifier ces informations si nécessaire. Pour enregistrer les modifications dans la base de données, il suffira de cliquer sur le bouton « Modifier ». L'utilisateur pourra également supprimer le locataire en cliquant sur le bouton « Supprimer ». Enfin, un autre bouton sera disponible, « Retour » pour revenir en arrière.



Proj « Gestion Immobilier »



C'est le même schéma d'enchaînement de pages pour les autres propositions. Si depuis la page « Proposition » l'utilisateur sélectionne le bouton « Réservation », la liste de toutes les réservations enregistrées s'affichera avec une liste déroulante pour filtrer les réservations selon leur villas. Deux boutons en bas de pages « Retour » comme pour revenir en arrière et « Ajouter » pour ajouter une réservation.

Si l'utilisateur clique sur « Ajouter » la page qui suit apparaîtra. Avec un formulaire lui permettant de créer une nouvelle réservation en insérant toutes les valeurs nécessaires à sa création. Ici nous avons deux listes déroulantes pour que l'utilisateur puisse spécifier le locataire qui souhaite cette réservation et la villa qu'il souhaite réserver. Enfin deux boutons en bas de page, « Retour » et « Ajouter » pour confirmer la création de cette réservation.



Sinon comme toujours l'utilisateur peut sélectionner l'une des réservations affichées sur la page « Réservation » et en afficher son contenu, encore une fois dans un formulaire remplie des informations de la réservation cliqué. Lui permettant encore une fois de modifier ces informations ou de supprimer la réservation avec les boutons « Modifier » et « Supprimer ». Toujours en bas de page le bouton « Retour » pour revenir à la page précédente.

4 – Documentation technique

1) Ma tâche : Owen

Pour ma part, j'ai été chargé de faire toute la partie « **Réservations** ». Pour cela, j'ai dû réaliser 3 interfaces.

- Une pour gérer la consultation des réservations
- Une pour gérer la modification et la suppressions des réservations
- Une pour gérer l'ajout de nouvelles réservations.

De plus j'ai dû réaliser 3 contrôleurs pour pouvoir gérer les 3 aspects des réservations qui me permet aussi de naviguer entre chaque page de l'application. Enfin, j'ai réalisé une classe métier et une classe DAO.

2) La classe métier

Pour commencer, j'ai réalisé la classe métier « **Réservation** » pour pouvoir utiliser les informations de ma base de données :

```
public class Reservation {  
    private int id;  
    private String dateArrivee;  
    private String dateDepart;  
    private int nbAdultes;  
    private int nbEnfants;  
    private String dateReservation;  
    private boolean optionMenage;  
    private float montantR;  
    private int idVilla;  
    private int idLocataires;  
  
    //Constructeur  
    public Reservation(int id, String dateArrivee, String dateDepart, int nbAdultes, int nbEnfants,  
                      String dateReservation, boolean optionMenage, float montantR, int idVilla, int idLocataires) {  
        this.id = id;  
        this.dateArrivee = dateArrivee;  
        this.dateDepart = dateDepart;  
        this.nbAdultes = nbAdultes;  
        this.nbEnfants = nbEnfants;  
        this.dateReservation = dateReservation;  
        this.optionMenage = optionMenage;  
        this.montantR = montantR;  
        this.idVilla = idVilla;  
        this.idLocataires = idLocataires;  
    }  
}
```

Proj « Gestion Immobilier »

```
//Getter et Setter
public int getId() { return id; }

public void setId(int id) { this.id = id; }

public String getDateArrivee() { return dateArrivee; }

public void setDateArrivee(String dateArrivee) { this.dateArrivee = dateArrivee; }

public String getDateDepart() { return dateDepart; }

public void setDateDepart(String dateDepart) { this.dateDepart = dateDepart; }

public int getNbAdultes() { return nbAdultes; }

public void setNbAdultes(int nbAdultes) { this.nbAdultes = nbAdultes; }

public int getNbEnfants() { return nbEnfants; }

public void setNbEnfants(int nbEnfants) { this.nbEnfants = nbEnfants; }

public String getDateReservation() { return dateReservation; }

public void setDateReservation(String dateReservation) { this.dateReservation = dateReservation; }

public boolean getOptionMenage() { return optionMenage; }

public void setOptionMenage(boolean optionMenage) { this.optionMenage = optionMenage; }

public float getMontantR() { return montantR; }

public void setMontantR(float montantR) { this.montantR = montantR; }

public int getIdVilla() { return idVilla; }

public void setIdVilla(int idVilla) { this.idVilla = idVilla; }

public int getIdLocataires() { return idLocataires; }

public void setIdLocataires(int idLocataires) { this.idLocataires = idLocataires; }

@Override
public String toString() {
    return " nbAdultes : " + nbAdultes +
           "\n nbEnfants : " + nbEnfants +
           "\n dateReservation : " + dateReservation + "";
}

}
```

Proj « Gestion Immobilier »

3) La classe DAO

Ensuite, j'ai créé « **ReservationDAO** » qui me permet d'accéder aux données des réservations se trouvant dans la base de donnée. De plus, c'est dans ce fichier aussi que j'ai réalisé toutes mes fonctions que j'utiliserais plus tard dans mes modèles.

```
public class ReservationDAO {

    // Classe permettant d'accéder aux données des réservations de la BDD
    private static String base = "gestionimmo";
    private static int version = 1;
    BD_SQLiteOpenHelper accesBD;

    public ReservationDAO(Context ct){
        accesBD = new BD_SQLiteOpenHelper(ct, base, factory: null, version);
    }

    /* Fonction qui récupère l'id le plus grand des réservations et lui ajoute +1 dans la BDD
       Retourne l'id le plus petit disponible pour les réservations
    */
    public int dernierId() {
        Cursor curseur;
        curseur = accesBD.getReadableDatabase().rawQuery( sql: "select max(id)+1 from reservations;",
            selectionArgs: null);
        curseur.moveToFirst();
        int lastId = curseur.getInt( i: 0);
        return lastId;
    }

    // Fonction qui ajoute une réservation dans la BDD
    public long addReservation(Reservation uneReservation){
        long ret;
        SQLiteDatabase bd = accesBD.getWritableDatabase();

        ContentValues value = new ContentValues();
        value.put("id",uneReservation.getId());
        value.put("dateArrivee", uneReservation.getDateArrivee());
        value.put("dateDepart",uneReservation.getDateDepart());
        value.put("nbAdultes", uneReservation.getNbAdultes());
        value.put("nbEnfants",uneReservation.getNbEnfants());
        value.put("dateReservation", uneReservation.getDateReservation());
        value.put("optionMenage", uneReservation.getOptionMenage());
        value.put("montantR", uneReservation.getMontantR());
        value.put("idVilla", uneReservation.getIdVilla());
        value.put("idLocataires", uneReservation.getIdLocataires());
        // Exécution de l'ordre SQL d'insertion
        ret = bd.insert( table: "reservations", nullColumnHack: null, value);

        return ret;
    }

    //Fonction qui récupère une collection de réservations
    public ArrayList<Reservation> getLesReservations(){
        Cursor curseur;
        String req = "select * from reservations";
        curseur = accesBD.getReadableDatabase().rawQuery(req, selectionArgs: null);
        return cursorToReservationArrayList(curseur);
    }
}
```

Proj « Gestion Immobilier »

```
// Fonction qui supprime une réservation de la BDD
public long supprimerReservation(Reservation uneReservation){
    long ret;
    SQLiteDatabase bd = accesBD.getWritableDatabase();
    String condition = "nbAdultes ='" +uneReservation.getNbAdultes()+
        "' AND nbEnfants='"+uneReservation.getNbEnfants()+"'";
    Log.d( tag: "réservation supprimer", condition);
    //Exécution de l'ordre SQL de suppression
    ret = bd.delete( table: "reservations", condition , whereArgs: null);
    return ret;
}

//Fonction qui récupère une réservation en fonction de l'id récupéré
public Reservation getReservations(int idR){
    Reservation laReservation = null;
    Cursor curseur;
    curseur = accesBD.getReadableDatabase().rawQuery( sql: "select * from reservations where id="
        +idR+";", selectionArgs: null);
    if (curseur.getCount() > 0) {
        curseur.moveToFirst();
        laReservation = new Reservation(idR, curseur.getString( i: 1), curseur.getString( i: 2),
            curseur.getInt( i: 3), curseur.getInt( i: 4), curseur.getString( i: 5),
            Boolean.parseBoolean(curseur.getString( i: 6)), curseur.getFloat( i: 7),
            curseur.getInt( i: 8), curseur.getInt( i: 9));
    }
    return laReservation;
}

// Fonction qui modifie une réservation de la BDD
public int modifierReservation(Reservation nvReservation, Reservation ancReservation) {
    int ret;

    SQLiteDatabase bd = accesBD.getWritableDatabase();
    ContentValues value = new ContentValues();

    value.put("dateArrivee", nvReservation.getDateArrivee());
    value.put("dateDepart", nvReservation.getDateDepart());
    value.put("nbAdultes", nvReservation.getNbAdultes());
    value.put("nbEnfants", nvReservation.getNbEnfants());
    value.put("dateReservation", nvReservation.getDateReservation());
    value.put("optionMenage", nvReservation.getOptionMenage());
    value.put("montantR", nvReservation.getMontantR());
    value.put("idVilla", nvReservation.getIdVilla());
    value.put("idLocataires", nvReservation.getIdLocataires());

    String condition = "id =" + ancReservation.getId() + "";

    //Exécution de l'ordre SQL de modification
    ret = bd.update( table: "reservations", value, condition, whereArgs: null);
    return ret;
}
```

Proj « Gestion Immobilier »

```
/*
Fonction qui récupère un id de la table Villa et affiche les informations liées aux
réservation en fonction de l'id de la villa choisi
Retourne une collection de réservation
*/
public ArrayList<Reservation> getVillaFiltre(int idR) {
    Cursor curseur;
    String req = "select * from reservations where idVilla = " + idR + "";
    Log.d( tag: "msg", msg: "requete " + req);
    curseur = accesBD.getReadableDatabase().rawQuery(req, selectionArgs: null);
    ArrayList<Reservation> listeReservation = new ArrayList<>();

    int id;
    String dateArrivee;
    String dateDepart;
    int nbAdultes;
    int nbEnfants;
    String dateReservation;
    boolean optionMenage;
    float montantR;
    int idVilla;
    int idLocataires;

    curseur.moveToFirst();
    while (!curseur.isAfterLast()) {
        id = curseur.getInt( i: 0);
        dateArrivee = curseur.getString( i: 1);
        dateDepart = curseur.getString( i: 2);
        nbAdultes = curseur.getInt( i: 3);
        nbEnfants = curseur.getInt( i: 4);
        dateReservation = curseur.getString( i: 5);
        optionMenage = Boolean.parseBoolean(curseur.getString( i: 6));
        montantR = curseur.getFloat( i: 7);
        idVilla = curseur.getInt( i: 8);
        idLocataires = curseur.getInt( i: 9);
        dateReservation = curseur.getString( i: 5);
        listeReservation.add(new Reservation(id, dateArrivee, dateDepart, nbAdultes,
            nbEnfants, dateReservation, optionMenage, montantR, idVilla, idLocataires));
        curseur.moveToNext();
    }
    return listeReservation;
}
```

```

/*
Fonction qui récupère un curseur et implémente les informations liées aux équipements en
fonction du contenu du curseur
Retourne une collection de réservations
*/
private ArrayList<Reservation> cursorToReservationArrayList(Cursor curseur){
    ArrayList<Reservation> listeReservation = new ArrayList<>();
    int id;
    String dateArrivee;
    String dateDepart;
    int nbAdultes;
    int nbEnfants;
    String dateReservation;
    boolean optionMenage;
    float montantR;
    int idVilla;
    int idLocataires;

    curseur.moveToFirst();
    while (!curseur.isAfterLast()){
        id = curseur.getInt(0);
        dateArrivee = curseur.getString(1);
        dateDepart = curseur.getString(2);
        nbAdultes = curseur.getInt(3);
        nbEnfants = curseur.getInt(4);
        dateReservation = curseur.getString(5);
        optionMenage = Boolean.parseBoolean(curseur.getString(6));
        montantR = curseur.getFloat(7);
        idVilla = curseur.getInt(8);
        idLocataires = curseur.getInt(9);
        listeReservation.add(new Reservation(id, dateArrivee, dateDepart, nbAdultes, nbEnfants,
                dateReservation, optionMenage, montantR, idVilla, idLocataires));
        curseur.moveToNext();
    }

    return listeReservation;
}
}

```

4) Les contrôleurs

a) ReservationConsultActivity

J'ai donc dû réaliser 3 contrôleurs

Le premier contrôleur que j'ai codé est « **ReservationConsultActivity** ».

Pour gérer la réservation des villas, j'ai utilisé un spinner qui m'affichait les réservations pour les villas concernés sous forme d'une liste. A chaque fois, qu'on change de villa, un message s'affiche.

Proj « Gestion Immobilier »

Voici le contrôleur pour les consultations des réservations :

```
public class ReservationsConsultActivity extends AppCompatActivity {

    //Déclaration des variables de classe
    private Spinner spinVilla;
    private ArrayList<Villa> listeVilla;
    private int idVilla;
    private ListView listeViewRervationConsult;
    private Button retour, ajouter;
    private ArrayList<Reservation> listeReservation;
    private ImageView image;

    /*
        Méthode permettant d'afficher la liste des réservations en fonction de l'id de la villa
        choisi dans le spinner
        Actualise la liste des réservations à chaque changement d'id dans le spinner
    */
    public void actualiser(int id){
        ReservationDAO rAcces = new ReservationDAO( ct: this );
        listeReservation = rAcces.getVillaFiltre(id);
        for (Reservation uneReservation : listeReservation){
            Log.d( tag: "reservation", msg: "*****" + uneReservation.toString());
        }
        ArrayAdapter<String> monAdapteur = new ArrayAdapter( context: this,
            android.R.layout.simple_list_item_1,listeReservation);
        listeViewRervationConsult.setAdapter(monAdapteur);
    }

    @Override
    protected void onCreate(Bundle savedInstanceState) {

        // Méthode permettant de récupérer l'activité en cas de crash de l'application
        super.onCreate(savedInstanceState);
        // Méthode associant cette page au fichier XML choisi
        setContentView(R.layout.activity_reservations_consult);

        // Lien avec les fichiers de l'interface XML
        ajouter = findViewById(R.id.buttonAjouterReservationsConsult);
        retour = findViewById(R.id.buttonRetourReservationsConsult);
        listeViewRervationConsult = findViewById(R.id.listViewReservationsAjout);
        spinVilla = findViewById(R.id.spinnerChoixVillaReservationsConsult);
        image = findViewById(R.id.imageView10);

        //Permet l'affichage de l'image choisi dans le fichier XML
        image.setImageResource(R.drawable.logovillepau);

        // Gestion du bouton 'Ajouter' qui renvoie vers la page 'ReservationAjoutActivity'
        ajouter.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                Intent i = new Intent ( packageContext: ReservationsConsultActivity.this,
                    ReservationsAjoutActivity.class );
                startActivity(i);
            }
        });
    }
}
```

Proj « Gestion Immobilier »

```
// Gestion du bouton 'Retour' qui renvoie vers la page 'PropositionActivity'
retour.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {
        Intent i2 = new Intent ( packageContext: ReservationsConsultActivity.this,
                               PropositionActivity.class );
        startActivity(i2);
    }
});

//Création d'objet de la classe VillaDAO
VillaDAO villaAcces = new VillaDAO( ct: this);
//Création d'objet de la classe ReservationDAO
ReservationDAO reservationAcces = new ReservationDAO( ct: this);

// Récupération de la liste des villas
listeVilla = villaAcces.recupVilla();
// Récupération de la liste des réservations
listeReservation = reservationAcces.getLesReservations();

// Crédation d'un adapteur pour les villas
ArrayAdapter<Villa> spinVillaAdapter = new ArrayAdapter<~>( context: this,
    android.R.layout.simple_spinner_item);

// Parcours des villas et ajout de ceux-ci dans l'adapteur
for (int i=0; i<listeVilla.size(); i++){
    spinVillaAdapter.add(listeVilla.get(i));
    Log.d( tag: "message", msg: "***" +listeVilla.get(i));
}
// Applique l'adapteur au spinner des Villas
spinVilla.setAdapter(spinVillaAdapter);

// Gestion de la valeur du spinner villa
spinVilla.setOnItemSelectedListener(new AdapterView.OnItemSelectedListener() {
    @Override
    public void onItemSelected(AdapterView<?> adapterView, View view, int position,
        long id) {
        Villa uneVilla = (Villa) spinVilla.getSelectedItem();
        Toast.makeText(getApplicationContext(), text: "Voici les réservations de la "
            + uneVilla.getNom(), Toast.LENGTH_SHORT).show();
        actualiser(uneVilla.getId());
    }
    @Override
    public void onNothingSelected(AdapterView<?> adapterView) {
    }
});

// Crédation d'un adapteur pour les réservations
ArrayAdapter lesReservations = new ArrayAdapter( context: this,
    android.R.layout.simple_list_item_1, listeReservation);

// Applique l'adapteur à la listView des Réservations
listViewRervationConsult.setAdapter(lesReservations);
```

Proj « Gestion Immobilier »

```
// Affichage d'une listView contenant toutes les réservation de la BDD
listViewRervationConsult.setOnItemClickListener(new AdapterView.OnItemClickListener() {
    @Override
    public void onItemClick(AdapterView<?> parent, View view, int position, long id) {
        Intent i = new Intent( packageContext: ReservationsConsultActivity.this,
            ReservationsConsultDetailsActivity.class);
        Reservation clickedItem = (Reservation) listViewRervationConsult.getAdapter().
            getItem(position);
        // Récupère toutes les informations de la réservation quand on clique dessus
        i.putExtra( name: "id", clickedItem.getId());
        i.putExtra( name: "dateArrivee", clickedItem.getDateArrivee());
        i.putExtra( name: "dateDepart", clickedItem.getDateDepart());
        i.putExtra( name: "nbAdultes", clickedItem.getNbAdultes());
        i.putExtra( name: "nbEnfants", clickedItem.getNbEnfants());
        i.putExtra( name: "dateReservation", clickedItem.getDateReservation());
        i.putExtra( name: "optionMenage", clickedItem.getOptionMenage());
        i.putExtra( name: "montantR", clickedItem.getMontantR());
        i.putExtra( name: "idVilla", clickedItem.getIdVilla());
        i.putExtra( name: "idLocataires", clickedItem.getIdLocataires());
        startActivity(i);
    }
});
```

Proj « Gestion Immobilier »

b) ReservationConsultDetailsActivity

Le second contrôleur que j'ai codé est « **ReservationConsultDetailsActivity** ».

Pour vérifier le bon fonctionnement de mes deux fonctions réalisées au préalable dans ReservatoindAO, j'ai réalisé des conditions. Si la modification ou la suppression se déroule bien, l'appli revient sur la page « **ReservationConsultActivity** » avec un message compris. Dans le cas contraire, l'appli reste sur la page « **ReservationConsultDetailsActivity** ».

Voici le contrôleur pour les modifications et les suppressions des réservations :

```
public class ReservationsConsultDetailsActivity extends AppCompatActivity {

    //Déclaration des variables de classe
    private Button retour, supprimer, modifier;
    private EditText txtDateArriveeRecu, txtDateDepartRecu, txtNbAdultesRecu, txtNbEnfantsRecu,
    txtDateReservationRecu, txtOptionMenageRecu, txtMontantRRecu, txtIdVillaRRecu;
    private Spinner spinLocataire;
    private ReservationDAO recuperation;
    private LocataireDAO locataireAcces = new LocataireDAO( ct: this);
    private ArrayList<Locataire> listeLocataire;
    private Reservation laReservation;
    private ImageView image;
    private String dateArrivee, dateDepart, dateReservation;
    private int idR, nbAdultes, nbEnfants, idLocataire, idVilla;
    private float montantR;
    private boolean optionMenage;

    @Override
    protected void onCreate(Bundle savedInstanceState) {

        // Méthode permettant de récupérer l'activité en cas de crash de l'application
        super.onCreate(savedInstanceState);
        // Méthode associant cette page au fichier XML choisi
        setContentView(R.layout.activity_reservations_consult_details);

        // Lien avec les fichiers de l'interface XML
        txtDateArriveeRecu = findViewById(R.id.editTextDateArriveeReservationsConsultDetails);
        txtDateDepartRecu = findViewById(R.id.editTextDateDepartReservationsConsultDetails);
        txtNbAdultesRecu = findViewById(R.id.editTextNombreAdultesReservationsConsultDetails);
        txtNbEnfantsRecu = findViewById(R.id.editTextNombreEnfantsReservationConsultDetails);
        txtDateReservationRecu = findViewById(R.id.editTextDateReservationsConsultDetails);
        txtOptionMenageRecu = findViewById(R.id.editTextOptionMenageReservationsConsultDetails);
        txtMontantRRecu = findViewById(R.id.editTextMontantRReservationsConsultDetails);
        txtIdVillaRRecu = findViewById(R.id.editTextIdVillaReservationsConsultDetails);
        retour = findViewById(R.id.buttonRetourReservationsConsultDetails);
        supprimer = findViewById(R.id.buttonSupprimerReservationsConsultDetails);
        modifier = findViewById(R.id.buttonModifierReservationsConsultDetails);
        spinLocataire = findViewById(R.id.spinnerLocatairesReservationsConsultDetails);
        image = findViewById(R.id.imageView11);

        //Permet l'affichage de l'image choisi dans le fichier XML
        image.setImageResource(R.drawable.logovillepau);
    }
}
```

Proj « Gestion Immobilier »

```
// Récupération des informations de la réservation cliquée au préalable
recupReservation = new ReservationDAO( ct: this);
idR = getIntent().getIntExtra( name: "id", defaultValue: 1);
dateArrivee = getIntent().getStringExtra( name: "dateArrivee");
dateDepart = getIntent().getStringExtra( name: "dateDepart");
nbAdultes = getIntent().getIntExtra( name: "nbAdultes", defaultValue: -1);
nbEnfants = getIntent().getIntExtra( name: "nbEnfants", defaultValue: -1);
dateReservation = getIntent().getStringExtra( name: "dateReservation");
optionMenage = getIntent().getBooleanExtra( name: "optionMenage", defaultValue: true);
montantR = getIntent().getFloatExtra( name: "montantR", defaultValue: -1);
idLocataire = getIntent().getIntExtra( name: "idLocataire", defaultValue: 0);
idVilla = getIntent().getIntExtra( name: "idVilla", defaultValue: 0);

// Récupération de l'id de la réservation cliquée au préalable
laReservation = recuperation.getReservations(idR);

// Modification des données de la réservation en chaîne de caractère
txtDateArriveeReçu.setText(dateArrivee);
txtDateDepartReçu.setText(dateDepart);
txtNbAdultesReçu.setText(String.valueOf(nbAdultes));
txtNbEnfantsReçu.setText(String.valueOf(nbEnfants));
txtDateReservationReçu.setText(dateReservation);
txtOptionMenageReçu.setText(String.valueOf(optionMenage));
txtMontantRReçu.setText(String.valueOf(montantR));
txtIdVillaRReçu.setText(String.valueOf(idVilla));
//Création d'objet de la classe LocataireDAO
LocataireDAO locaAcces = new LocataireDAO( ct: this);
// Récupération de la liste des locataires
listeLocataire = locaAcces.recupLocataire();

// Création d'un adaptateur pour les locataires
ArrayAdapter<Locataire> spinLocataireAdapter = new ArrayAdapter<~>( context: this,
    android.R.layout.simple_spinner_item);

// Parcours des locataires et ajout de ceux-ci dans l'adaptateur
for (int i=0; i<listeLocataire.size(); i++){
    spinLocataireAdapter.add(listeLocataire.get(i));
}
// Applique l'adaptateur au spinner des Locataires
spinLocataire.setAdapter(spinLocataireAdapter);

// Gestion de la valeur du spinner locataire
spinLocataire.setOnItemSelectedListener(new AdapterView.OnItemSelectedListener() {
    @Override
    public void onItemSelected(AdapterView<?> adapterView, View view, int position,
        long id) {
        idLocataire=listeLocataire.get(position).getId();
    }
    @Override
    public void onNothingSelected(AdapterView<?> adapterView) {
    }
});
```

Proj « Gestion Immobilier »

```
// Gestion du bouton 'Modifier'
modifier.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {

        dateArrivee=txtDateArriveeRecu.getText().toString();
        dateDepart=txtDateDepartRecu.getText().toString();
        nbAdultes=Integer.parseInt(txtNbAdultesRecu.getText().toString());
        nbEnfants=Integer.parseInt(txtNbEnfantsRecu.getText().toString());
        dateReservation=txtDateReservationRecu.getText().toString();
        optionMenage= Boolean.parseBoolean(txtOptionMenageRecu.getText().toString());
        montantR= Float.parseFloat(txtMontantRRecu.getText().toString());

        // Création de la nouvelle réservation modifiée
        Reservation newReservation = new Reservation(idR, dateArrivee,dateDepart, nbAdultes,
            nbEnfants, dateReservation, optionMenage, montantR, idVilla, idLocataire);
        Log.d( tag: "message", msg: "*****"+ newReservation.toString());

        // Modification dans la BDD grâce à la fonction prévue
        int res = recuperation.modifierReservation(newReservation, laReservation);

        /*
            Si la modification a échoué, reste sur la page + message d'erreur s'affiche
            Sinon si modification réussie, retour sur la page 'ReservationConsultActivity'
            + message de suppression
        */
        if (res != 1){
            Toast.makeText(getApplicationContext(), text: "Veuillez saisir une date de " +
                "réservation.",Toast.LENGTH_LONG).show();
        }
        else{
            Intent i = new Intent( packageContext: ReservationsConsultDetailsActivity.this,
                ReservationsConsultActivity.class);
            recuperation.modifierReservation(newReservation, laReservation);
            Toast.makeText(getApplicationContext(), text: "Votre réservation a été modifiée.",
                Toast.LENGTH_LONG).show();
            startActivity(i);
        }
    }
});
```

Proj « Gestion Immobilier »

```
// Gestion du bouton 'Supprimer'
supprimer.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {

        // Suppression dans la BDD grâce à la fonction prévue
        long res = recuperation.supprimerReservation(laReservation);

        /*
            Si la suppression a échoué, reste sur la page + message d'erreur s'affiche
            Sinon si suppression réalisée, retour sur la page 'ReservationConsultActivity'
            + message de suppression
        */
        if (res != 0){
            Intent i2 = new Intent( packageContext: ReservationsConsultDetailsActivity.this,
                ReservationsConsultActivity.class);
            Toast.makeText(getApplicationContext(), text: "La réservation du "
                + laReservation.getDateReservation() +
                " a été supprimée.", Toast.LENGTH_LONG).show();
            startActivity(i2);
        }else{
            Toast.makeText(getApplicationContext(), text: "Echec de la suppression !",
                Toast.LENGTH_LONG).show();
        }
    }
});

// Gestion du bouton 'Retour' qui renvoie vers la page 'ReservationConsultActivity'
retour.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {
        Intent i = new Intent ( packageContext: ReservationsConsultDetailsActivity.this,
            ReservationsConsultActivity.class );
        startActivity(i);
    }
});
}
```

c) ReservationAjoutActivity

Pour vérifier le bon fonctionnement de ma fonction ajout réalisé au préalable dans ReservatoinDAO, j'ai réalisé une condition. Si l'ajout se déroule bien, l'appli revient sur la page « ReservationConsultActivity » avec un message compris. Dans le cas contraire, l'appli reste sur la page « ReservationAjoutActivity ». De plus, j'ai dû gérer l'ajout d'un nouveau locataire et d'une nouvelle villa. Pour cela, j'ai réalisé deux spinner qui permettra de choisir un locataire et une villa existant déjà dans la base de données.

Voici le contrôleur pour l'ajout de réservations :

```
public class ReservationsAjoutActivity extends AppCompatActivity {

    //Déclaration des variables de classe
    private Button ajouter, retour;
    private ArrayList<Locataire> listeLocataire;
    private ArrayList<Villa> listeVilla;
    private Spinner spinVilla, spinLocataire;
    private EditText txtDateArrivee, txtDateDepart, txtNbAdultes, txtNbEnfants, txtDateReservation,
        txtOptionMenage, txtMontantR;
    private int idVilla,idLocataire;
    private Reservation uneReservation;
    private String dateArrivee, dateDepart, dateReservation;
    private int nbAdultes, nbEnfants;
    private float montantR;
    private boolean optionMenage;
    private ImageView image;

    protected void onCreate(Bundle savedInstanceState) {

        // Méthode permettant de récupérer l'activité en cas de crash de l'application
        super.onCreate(savedInstanceState);
        // Méthode associant cette page au fichier XML choisi
        setContentView(R.layout.activity_reservations_ajout);

        // Lien avec les fichiers de l'interface XML
        txtDateArrivee = findViewById(R.id.editTextDateArriveeReservationsAjout);
        txtDateDepart = findViewById(R.id.editTextDateDepartReservationsAjout);
        txtNbAdultes = findViewById(R.id.editTextNombreAdultesReservationsAjout);
        txtNbEnfants = findViewById(R.id.editTextNombreEnfantsReservationsAjout);
        txtDateReservation = findViewById(R.id.editTextDateReservationsAjout);
        txtOptionMenage = findViewById(R.id.editTextOptionMenageReservationsAjout);
        txtMontantR = findViewById(R.id.editTextMontantReservationsAjout);
        spinLocataire = findViewById(R.id.spinnerAjoutLoc);
        spinVilla = findViewById(R.id.spinnerAjoutVilla);
        ajouter = findViewById(R.id.buttonAjouterReservationsAjout);
        retour = findViewById(R.id.buttonRetourReservationsAjout);
        image = findViewById(R.id.imageView12);

        //Permet l'affichage de l'image choisi dans le fichier XML
        image.setImageResource(R.drawable.logovillepau);
    }
}
```

Proj « Gestion Immobilier »

```
//Création d'objet de la classe ReservationDAO
ReservationDAO reservationAcces = new ReservationDAO( ct: this);
//Création d'objet de la classe LocataireDAO
LocataireDAO locaAcces = new LocataireDAO( ct: this);
//Création d'objet de la classe VillaDAO
VillaDAO villaAcces = new VillaDAO( ct: this);

// Récupération de la liste des locataires
listeLocataire = locaAcces.recupLocataire();
// Récupération de la liste des villas
listeVilla = villaAcces.recupVilla();

// Création d'un adaptateur pour les locataires
ArrayAdapter<Locataire> spinLocataireAdapter = new ArrayAdapter<~>( context: this,
    android.R.layout.simple_spinner_item);

// Parcours des locataires et ajout de ceux-ci dans l'adaptateur
for (int i=0; i<listeLocataire.size(); i++){
    spinLocataireAdapter.add(listeLocataire.get(i));
}

// Applique l'adaptateur au spinner des Réervations
spinLocataire.setAdapter(spinLocataireAdapter);

// Création d'un adaptateur pour les villas
ArrayAdapter<Villa> spinVillaAdapter = new ArrayAdapter<~>( context: this,
    android.R.layout.simple_spinner_item);

// Parcours des villas et ajout de ceux-ci dans l'adaptateur
for (int i=0; i<listeVilla.size(); i++){
    spinVillaAdapter.add(listeVilla.get(i));
}

// Applique l'adaptateur au spinner des Villas
spinVilla.setAdapter(spinVillaAdapter);

// Gestion de la valeur du spinner Locataire
spinLocataire.setOnItemSelectedListener(new AdapterView.OnItemSelectedListener() {
    @Override
    public void onItemSelected(AdapterView<?> adapterView, View view, int position,
        long id) {
        idLocataire=listeLocataire.get(position).getId();
    }
    @Override
    public void onNothingSelected(AdapterView<?> adapterView) {
    }
});

// Gestion de la valeur du spinner Villa
spinVilla.setOnItemSelectedListener(new AdapterView.OnItemSelectedListener() {
    @Override
    public void onItemSelected(AdapterView<?> adapterView, View view, int position,
        long id) {
        idVilla=listeVilla.get(position).getId();
    }
    @Override
    public void onNothingSelected(AdapterView<?> adapterView) {
    }
});
```

Proj « Gestion Immobilier »

```
// Gestion du bouton 'Ajouter'
ajouter.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {

        dateArrivee=txtDateArrivee.getText().toString();
        dateDepart=txtDateDepart.getText().toString();
        nbAdultes=Integer.parseInt(txtNbAdultes.getText().toString());
        nbEnfants=Integer.parseInt(txtNbEnfants.getText().toString());
        dateReservation=txtDateReservation.getText().toString();
        optionMenage= Boolean.parseBoolean(txtOptionMenage.getText().toString());
        montantR= Float.parseFloat(txtMontantR.getText().toString());

        // Création d'une nouvelle réservation
        uneReservation = new Reservation(reservationAcces.dernierId(), dateArrivee,
            dateDepart, nbAdultes, nbEnfants, dateReservation, optionMenage, montantR,
            idVilla, idLocataire);

        // Insertion dans la base de données de la nouvelle réservation
        reservationAcces.addReservation(uneReservation);

        /* Si le champ de la date d'arrivée est vide, reste sur la page + message d'erreur
         * s'affiche
        */
        if (uneReservation.getDateArrivee().equals("")){
            Toast.makeText(getApplicationContext(), text: "Veuillez saisir une date de" +
                " réservation.",Toast.LENGTH_LONG).show();
            Intent i = new Intent ( packageContext: ReservationsAjoutActivity.this,
                ReservationsConsultActivity.class );
        }

        /* Si l'ajout s'est bien réalisée, retour sur la page 'ReservationsConsultActivity'
         * + message d'ajout
        */
        else {
            Intent i = new Intent ( packageContext: ReservationsAjoutActivity.this,
                ReservationsConsultActivity.class );
            reservationAcces.addReservation(uneReservation);
            Toast.makeText(getApplicationContext(), text: "La réservation du " +
                uneReservation.getDateReservation() + " a été ajoutée.",
                Toast.LENGTH_LONG).show();
            startActivity(i);
        }
    });

    // Gestion du bouton 'Retour' qui renvoie vers la page 'ReservationsConsultActivity'
    retour.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View view) {
            Intent i = new Intent ( packageContext: ReservationsAjoutActivity.this,
                ReservationsConsultActivity.class );
            startActivity(i);
        }
    });
}
});
```

5 – Mes interfaces

Voici les interfaces finales de ma partie.

Pour la consultation :



Proj « Gestion Immobilier »

Pour la modification et la suppression :



Proj « Gestion Immobilier »

Pour l'ajout :

