Gr’immo

Une image contenant logo, Graphique, Police, symbole

Description générée automatiquement

Table des matières

[Introduction 3](#_Toc183437323)

[Diagramme de cas d’utilisation 3](#_Toc183437324)

[Diagramme de classe 4](#_Toc183437325)

[Base de données 5](#_Toc183437326)

[Base de donnée V1 5](#_Toc183437327)

[MLD 5](#_Toc183437328)

[Mld textuel 6](#_Toc183437329)

[Script de creation 7](#_Toc183437330)

[Base de données V2 9](#_Toc183437331)

[MLD 9](#_Toc183437332)

[Mld textuel 10](#_Toc183437333)

[Script de creation 11](#_Toc183437334)

[Solution mise en place 13](#_Toc183437335)

[Table des illustrations 15](#_Toc183437336)

# Introduction

Gr’immo est une agence immobilière située à Grenoble qui prévoit de se développer dans d’autres villes. L’agence nous a donc demandé de développer un client lourd permettant de gérer leur base de données contenant l'ensemble de leurs biens, ainsi qu'un agenda.

# Diagramme de cas d’utilisation

Le diagramme de cas d'utilisation représente les fonctionnalités principales d’un système de gestion immobilière, avec deux types d’utilisateurs identifiés : **l’agent** et **le responsable**. Voici une description des cas d’utilisation associés à chaque acteur :

**1. L’agent**

L’agent, en tant qu’utilisateur du système, dispose des fonctionnalités suivantes :

* **Ajouter un bien** : possibilité d’ajouter de nouveaux biens immobiliers dans le système, avec leurs caractéristiques (adresse, type, prix, etc.).
* **Afficher ses biens** : visualiser la liste des biens qu’il gère personnellement.
* **Gérer son agenda** : organiser et planifier ses rendez-vous, visites ou événements professionnels.
* **Se connecter à l’AD (Annuaire LDAP)** : s’authentifier via un serveur LDAP pour accéder au système de manière sécurisée.
* **Gérer ses clients** : suivre et gérer les informations des clients qui lui sont attribués, tels que leurs coordonnées ou leurs demandes.
* **Gérer ses propriétaires** : gérer les informations des propriétaires associés aux biens sous sa responsabilité.

**2. Le responsable**

Le responsable dispose d’un rôle élargi avec des privilèges supplémentaires, incluant toutes les fonctionnalités des agents, ainsi que les suivantes :

* **Gérer tous les biens** : accès global à la gestion de l’ensemble des biens du système, indépendamment des agents responsables.
* **Gérer tous les clients** : possibilité de gérer les informations et interactions de tous les clients, sans restriction.
* **Ajouter un agent** : capacité d’ajouter de nouveaux agents au système, leur permettant ainsi d’utiliser la plateforme pour leurs activités.

**Relation entre les acteurs :**

Le **responsable** hérite des droits de l’**agent**, ce qui signifie qu’il peut réaliser toutes les actions possibles pour un agent tout en ayant des privilèges supplémentaires pour superviser et administrer le système.

**Conclusion :**

Ce diagramme met en évidence une organisation hiérarchique des rôles et des responsabilités, avec des fonctionnalités adaptées aux besoins spécifiques de chaque utilisateur. Il garantit un contrôle efficace des biens, des clients et des utilisateurs dans le cadre d’une gestion immobilière.

Une image contenant texte, diagramme, ligne, Police

Description générée automatiquement

Figure 1: diagramme de cas d'utilisation

# Diagramme de classe

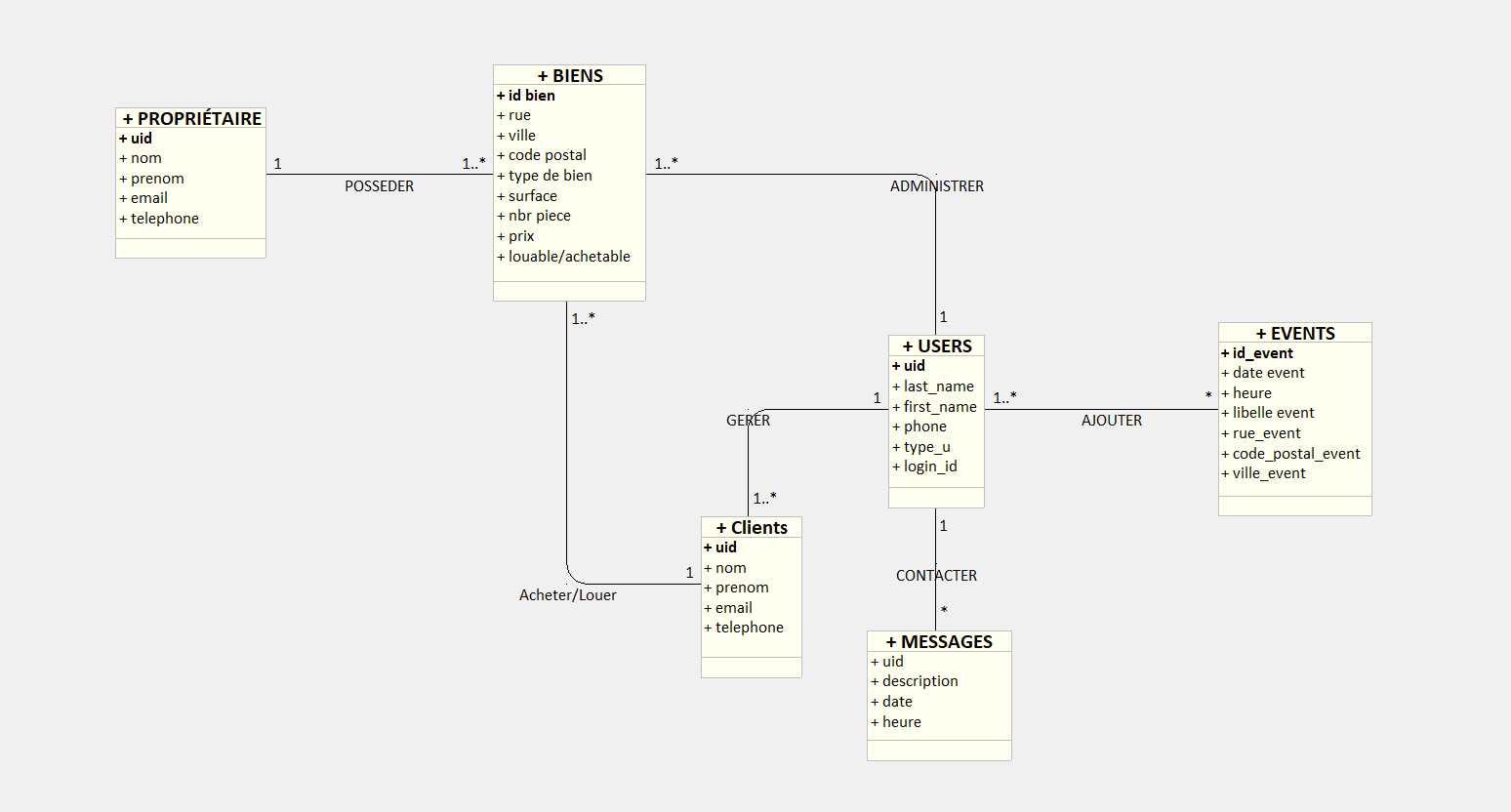


Figure 2 : diagramme de classe

## 1. Entités et attributs

La classe PROPRIÉTAIRE comporte les attributs : uid (identifiant unique), nom, prenom, email, telephone. Il représente les biens immobiliers des propriétaire de l’application

La classe BIENS comporte les attributs : id\_bien (identifiant unique), rue, ville, code postal, type de bien, surface, nbr\_piece, prix, louable/achetable. Il représente les biens immobiliers gérés dans le système.

La classe USERS comporte les attributs : uid, last\_name, first\_name, phone, type\_u (type d'utilisateur), login\_id. Il représente les utilisateurs qui gèrent le systhème en tant qu'administrateur ou agents.

La classe CLIENTS comporte les attributs : uid, nom, prenom, email, telephone. Il représenteles clients qui sont (ou pourrais être intéressés) par les biens qu’il soit acheteur ou locataires.

La classe EVENTS comporte les attributs : id\_event, date\_event, heure, libelle\_event, rue\_event, code\_postal\_event, ville\_event. Il représente les événements en liens avec les biens cela peut être des visites, des rendez-vous, etc... .

La classe MESSAGES comporte les attributs : uid, description, date, heure. Il représente les messages échangés entre les utilisateurs et les clients.

## 2. Relations

POSSEDER (entre PROPRIÉTAIRE et BIENS) :

La relation avec la multiplicité de type 1 à plusieurs (1 à \*). Un propriétaire peut posséder plusieurs biens, mais un bien appartient à un seul propriétaire.

ADMINISTRER (entre USERS et BIENS) :

La relation avec la multiplicité de type 1 à plusieurs (1 à \*). Un utilisateur (agent) peut venire gérer plusieurs biens.

GERER (entre USERS et Clients) :

La relation avec la multiplicité de type 1 à plusieurs (1..\*). Un utilisateur peut générer plusieurs clients.

AJOUTER (entre USERS et EVENTS) :

La relation avec la multiplicité de type plusieurs à plusieurs (\*..\*). Un utilisateur peut ajouter plusieurs événements, et un événement peut être ajouté par plusieurs utilisateurs

CONTACTER (entre Clients et MESSAGES) :

La relation avec la multiplicité de type 1 à plusieurs (1..\*). Un client peut envoyer plusieurs messages

# Base de données

## Base de donnée V1

### MLD

Ce modèle logique de données (MLD) représente la structure de la base de données pour une application de gestion immobilière. L'objectif de ce modèle est de décrire les relations entre les biens, les utilisateurs, et les événements associés, afin de gérer efficacement l'ensemble des opérations liées aux activités de l'agence immobilière.

**Description générale du modèle :**

Le MLD est composé de quatre principales entités : **biens**, **pièces**, **utilisateurs**, et **événements**. Chacune de ces entités est connectée par des associations qui reflètent la manière dont elles interagissent dans le système.

**Détails des entités :**

1. **Biens** : Cette table représente les propriétés immobilières gérées par l'agence. Chaque bien est identifié par un id\_bien unique et possède des attributs tels que l'adresse (rue, ville, code postal), le type de bien, la surface, le nombre de pièces, le prix, et un statut indiquant s'il est disponible à la location ou à l'achat.
2. **Pièces** : Cette table répertorie les types de pièces présentes dans les biens. Chaque pièce est identifiée par un id\_piece unique et un libelle piece décrivant la nature de la pièce (ex. : salon, chambre).
3. **Utilisateurs** : Les utilisateurs de l'application, comme les agents et les responsables, sont représentés dans cette table. Chaque utilisateur est identifié par un uid et a des attributs tels que le nom (last\_name), le prénom (first\_name), le numéro de téléphone (phone), et le type d'utilisateur (type\_u).
4. **Événements** : Cette table contient des informations sur les événements organisés, tels que des visites de biens ou des rendez-vous. Chaque événement est identifié par un id\_event et possède des détails comme la date (date event), l'heure (heure), le libellé de l'événement (libelle event), et l'adresse (rue\_event, code\_postal\_event, ville\_event).

**Associations entre les entités :**

* **Comporte** : Un bien peut comporter plusieurs pièces. La relation entre biens et pièce est de type 1,n pour les biens et 0,n pour les pièces, indiquant qu'un bien peut avoir plusieurs pièces et qu'une pièce doit appartenir à un bien.
* **Gère** : Un utilisateur peut gérer plusieurs biens, mais chaque bien est géré par un seul utilisateur. Cette relation est représentée par un lien entre user et biens de type 1,n (utilisateur) et 1,1 (bien).
* **Ajoute** : Cette association relie un utilisateur à la création d'événements. Un utilisateur peut ajouter plusieurs événements (relation 0,n), mais chaque événement est lié à un utilisateur unique (1,1).

Une image contenant texte, capture d’écran, diagramme, Police

Description générée automatiquement

Figure 3 : MLD V2

### Mld textuel

users = (uid\_user VARCHAR(50), last\_name VARCHAR(50), first\_name VARCHAR(50), phone VARCHAR(50), type\_user INT);

piece = (id\_piece VARCHAR(50), libelle\_piece VARCHAR(50));

event = (id\_event VARCHAR(50), date\_event DATE, heure\_event TIME, libelle\_event VARCHAR(50), rue\_event VARCHAR(50), code\_postal\_event INT, ville\_event VARCHAR(50), #uid\_user);

biens = (id\_bien VARCHAR(50), rue\_bien VARCHAR(50), ville\_bien VARCHAR(50), cpostal\_bien INT, type\_bien INT, surface\_bien DECIMAL(15,2), nbr\_piece\_bien INT, prix DECIMAL(15,2), louable\_achetable INT, #uid\_user);

comporte = (#id\_bien, #id\_piece);

### Script de creation

CREATE TABLE users(

uid\_user VARCHAR(50),

last\_name VARCHAR(50),

first\_name VARCHAR(50),

phone VARCHAR(50),

type\_user INT,

PRIMARY KEY(uid\_user)

);

CREATE TABLE piece(

id\_piece VARCHAR(50),

libelle\_piece VARCHAR(50),

PRIMARY KEY(id\_piece)

);

CREATE TABLE event(

id\_event VARCHAR(50),

date\_event DATE,

heure\_event TIME,

libelle\_event VARCHAR(50),

rue\_event VARCHAR(50),

code\_postal\_event INT,

ville\_event VARCHAR(50),

uid\_user VARCHAR(50) NOT NULL,

PRIMARY KEY(id\_event),

FOREIGN KEY(uid\_user) REFERENCES users(uid\_user)

);

CREATE TABLE biens(

id\_bien VARCHAR(50),

rue\_bien VARCHAR(50),

ville\_bien VARCHAR(50),

cpostal\_bien INT,

type\_bien INT,

surface\_bien DECIMAL(15,2),

nbr\_piece\_bien INT,

prix DECIMAL(15,2),

louable\_achetable INT,

uid\_user VARCHAR(50) NOT NULL,

PRIMARY KEY(id\_bien),

FOREIGN KEY(uid\_user) REFERENCES users(uid\_user)

);

CREATE TABLE comporte(

id\_bien VARCHAR(50),

id\_piece VARCHAR(50),

PRIMARY KEY(id\_bien, id\_piece),

FOREIGN KEY(id\_bien) REFERENCES biens(id\_bien),

FOREIGN KEY(id\_piece) REFERENCES piece(id\_piece)

);

## Base de données V2

### MLD

Dans la version 2 du Modèle Logique de Données (MLD), plusieurs tables ont été ajoutées pour enrichir le système et améliorer son organisation. Ces nouvelles tables sont PROPRIÉTAIRE, CLIENT, et MESSAGE, chacune ayant des rôles bien définis.

La table PROPRIÉTAIRE contient les attributs suivants : uid, nom, prenom, email, et telephone. Elle permet de gérer les informations personnelles des propriétaires de biens immobiliers, facilitant leur identification et leur contact.

La table CLIENT possède les mêmes attributs que celle des propriétaires : uid, nom, prenom, email, et telephone. Elle est dédiée à la gestion des clients intéressés par l’achat ou la location de biens. La distinction entre propriétaires et clients assure une meilleure organisation des données et une gestion adaptée à chaque type d’acteur.

Enfin, la table MESSAGE contient quatre attributs : uid, description, date, et heure. Elle est utilisée pour gérer les messages échangés dans le cadre du système. Chaque message est associé à un identifiant unique, accompagné de son contenu (description) et de sa date et heure d’envoi, garantissant une traçabilité précise des communications.

Ces ajouts structurent davantage le système, permettant une gestion distincte des rôles (propriétaires et clients) et centralisant les interactions via les messages. Ils rendent le MLD plus clair, fonctionnel, et adapté aux besoins d’une gestion immobilière moderne.

Une image contenant texte, diagramme, capture d’écran, ligne

Description générée automatiquement

Figure 4:MLD V2

### Mld textuel

**users = (uid\_user *VARCHAR(50)***, last\_name *VARCHAR(50)*, first\_name *VARCHAR(50)*, phone *VARCHAR(50)*, type\_user *INT*, login\_id *VARCHAR(255)***);**

**event = (id\_event *VARCHAR(50)***, date\_event *DATE*, heure\_event *TIME*, libelle\_event *VARCHAR(50)*, rue\_event *VARCHAR(50)*, code\_postal\_event *INT*, ville\_event *VARCHAR(50)***);**

**client = (uid *VARCHAR(255)***, nom *VARCHAR(50)*, prenom *VARCHAR(50)*, email *VARCHAR(50)*, telephone *VARCHAR(50), #uid\_user***);**

**Proprietaire = (uid *VARCHAR(255)***, nom *VARCHAR(50)*, prenom *VARCHAR(50)*, email *VARCHAR(50)*, telephone *VARCHAR(50)***);**

**message = (uid *VARCHAR(50)***, description *VARCHAR(50)*, \_date *DATE*, heure *TIME, #uid\_user***);**

**BIENS = (id\_bien *VARCHAR(50)***, rue\_bien *VARCHAR(50)*, ville\_bien *VARCHAR(50)*, cpostal\_bien *INT*, type\_bien *INT*, surface\_bien *DECIMAL(15,2)*, nbr\_piece\_bien *INT*, prix *DECIMAL(15,2)*, louable\_achetable *INT, #uid, #uid\_1, #uid\_user***);**

**ajoute = (*#uid\_user, #id\_event*);**

### Script de creation

CREATE TABLE users(

uid\_user VARCHAR(50),

last\_name VARCHAR(50),

first\_name VARCHAR(50),

phone VARCHAR(50),

type\_user INT,

login\_id VARCHAR(255),

PRIMARY KEY(uid\_user)

);

CREATE TABLE event(

id\_event VARCHAR(50),

date\_event DATE,

heure\_event TIME,

libelle\_event VARCHAR(50),

rue\_event VARCHAR(50),

code\_postal\_event INT,

ville\_event VARCHAR(50),

PRIMARY KEY(id\_event)

);

CREATE TABLE client(

uid VARCHAR(255),

nom VARCHAR(50),

prenom VARCHAR(50),

email VARCHAR(50),

telephone VARCHAR(50),

uid\_user VARCHAR(50) NOT NULL,

PRIMARY KEY(uid),

FOREIGN KEY(uid\_user) REFERENCES users(uid\_user)

);

CREATE TABLE Proprietaire(

uid VARCHAR(255),

nom VARCHAR(50),

prenom VARCHAR(50),

email VARCHAR(50),

telephone VARCHAR(50),

PRIMARY KEY(uid)

);

CREATE TABLE message(

uid VARCHAR(50),

description VARCHAR(50),

\_date DATE,

heure TIME,

uid\_user VARCHAR(50) NOT NULL,

PRIMARY KEY(uid),

FOREIGN KEY(uid\_user) REFERENCES users(uid\_user)

);

CREATE TABLE BIENS(

id\_bien VARCHAR(50),

rue\_bien VARCHAR(50),

ville\_bien VARCHAR(50),

cpostal\_bien INT,

type\_bien INT,

surface\_bien DECIMAL(15,2),

nbr\_piece\_bien INT,

prix DECIMAL(15,2),

louable\_achetable INT,

uid VARCHAR(255) NOT NULL,

uid\_1 VARCHAR(255) NOT NULL,

uid\_user VARCHAR(50) NOT NULL,

PRIMARY KEY(id\_bien),

FOREIGN KEY(uid) REFERENCES client(uid),

FOREIGN KEY(uid\_1) REFERENCES Proprietaire(uid),

FOREIGN KEY(uid\_user) REFERENCES users(uid\_user)

);

CREATE TABLE ajoute(

uid\_user VARCHAR(50),

id\_event VARCHAR(50),

PRIMARY KEY(uid\_user, id\_event),

FOREIGN KEY(uid\_user) REFERENCES users(uid\_user),

FOREIGN KEY(id\_event) REFERENCES event(id\_event)

);

# Solution mise en place

Donc pour la réalisation de se projet nous avons utiliser la langue de programmation python car il y a un simplicité d’utilisasation de plus il possèdent un grand nombre de framework et de bibliothèque

Nous avons utilisé certain bibliothèque pour faire ce projet tel que :

* + PYQt5 pour linterface
  + LDAP3 pour la connexion pour ldap
  + Psycopg2 pour la base de données postegre
  + Private
  + Configparser pour la lecture de .env

Table des illustrations

[Figure 1: diagramme de cas d'utilisation 4](#_Toc183442889)

[Figure 2 : diagramme de classe 5](#_Toc183442890)

[Figure 3 : MLD V2 8](#_Toc183442891)

[Figure 4:MLD V2 11](#_Toc183442892)