Rapport Projet

Base de Données

Mothieu Ary – Zhan Karim – Mahdjoubi Bilal

Mai 2022

Table des matières

1. Méthode d'analyse du sujet, de spécification et d'implémentation du sujet

a. Analyse du sujet

A la réception du sujet nous avons tout d'abord procédé à une lecture de celui-ci et à une analyse des différents objets évoqué dans celui-ci. Nous avons par la suite réalisé un modèle entité/relation préliminaire.

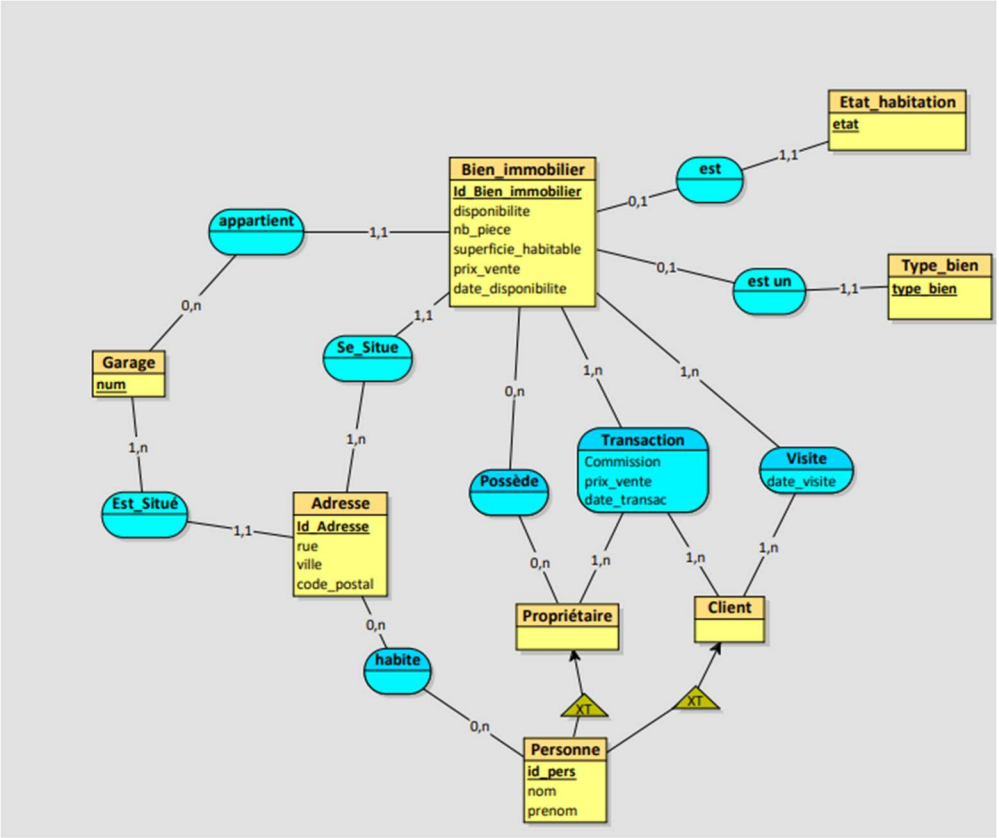


Figure 1. Modèle Entité/Relation préliminaire

Nous avons donc identifié les différentes entités ainsi que leurs attributs qui formeront la base de données et les associations qui lient ces différentes entités.

b. Spécification

Après avoir réalisation le modèle E/R, nous avons réalisé le schéma relationnel de la base de données en appliquant les règles de passage du modèle Entité/Relation au modèle relationnelle.

Bien_Immobilier (int **Id_bien_Immobilier**, booléen disponibilite, int nb_piece, int superficie, int prix_vente, date date_disponibilité, String #etat, String #type_bien, int #id_adresse, int #id_propriétaire)

Etat_habitation (String **etat)**

Type_bien (String **type_bien**)

Adresse (int **id_Adresse**, String rue, String ville, int code_postal)

Garage (int **num**, int #id_bien_immobilier)

Personne (int **id_personne**, String nom, String prénom)

Transaction (int Commission, date date_transac, int #id_client, int #id_propriétaire, int #bien)

Visite (date date_visite, int #id_client, int #bien)

Figure 2. Schéma relationnelle préliminaire de la base de données

On a donc identifié 8 tables issus du modèle E/R ainsi que leur attribut :

- 6 tables issus d'entité : Bien_immobilier, Adresse, Personne, Garage, etat et type_bien
- 2 tables d'association : Transaction, Visite

Nous avons donc grâce à ce schéma relationnelle une idée plus claire de la base de données qui est à créer et qui sera utilisé au cours de notre projet.

Toutefois celle-ci connaitra surement des améliorations afin de la rendre plus simple à utiliser pour l'utilisateur finale.

c. Implémentation

En parallèle de l'analyse du sujet et de sa spécification, nous avons décidé du langage de programmation à utiliser pour réaliser ce projet et nous avons aussi mis en place un outil de suivi de l'avancée du projet.

L'outil de suivi du projet que nous avons choisi d'utiliser est le logiciel de gestion de version Git et notamment sa plateforme web *GitLab*, nous permettant donc de suivre l'avancement du code, des spécifications et de la réalisation ainsi que de suivre la participation de chaque membre au projet. Concernant le langage de programmation utilisé, le premier nous étant venu en tête était le JAVA due à sa facilité a exploité une base de données et nos connaissances riches dans l'utilisation de celui-ci.

Toutefois après mure réflexion, il nous est apparue que la réalisation d'une interface JAVA serait plus compliqué que ce que nous avions prévu. Un interface web nous a semblé la solution la plus adapté au sujet. Nous avons donc choisi d'utilisé *NodeJs* afin de développer le backend de l'application et le Framework de développement Web *React Js* afin de réaliser les interfaces.

2. Description des différentes étapes d'implémentation

a. Phase de formation sur les nouvelles technologies et initialisation du Git

Nous avons donc initialisé un répertoire Git afin de faciliter le suivi des développements. Celui-ci est décomposé en 3 répertoires :

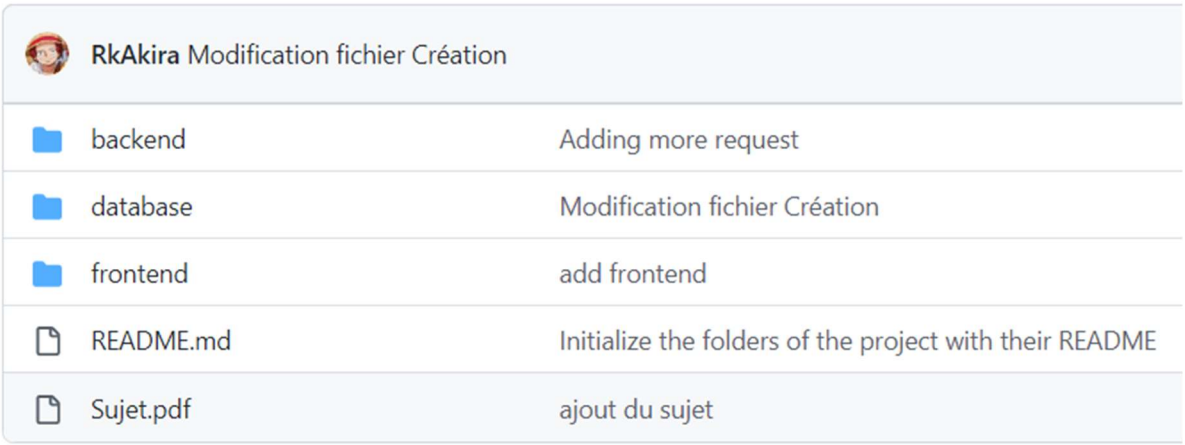


Figure 3. Architecture du répertoire GitHub

- Le répertoire *Backend* : Regroupe les fichiers permettant le développement du backend de l'application (explication du backEnd)
- Le répertoire *Database* : Regroupe les fichiers relatifs à la conception de la base de données, c'est-à-dire les fichiers présentés précédemment.
- Le répertoire *Frontend* : Regroupe les fichiers relatifs aux développements des interfaces utilisateurs de l'application.

Après avoir donc terminer l'organisation du projet, nous avons pu enclencher la phase de développements. Celle-ci démarra par une phase de formation étant donné la nouveauté des langages utilisés (NodeJS, React). Pour ce faire des ressources tels que Youtube ou OpenClassroom nous ont été très utiles afin d'apprendre de manière rapide ainsi que de trouver facilement réponse aux interrogations que nous pouvions avoir.

b. Développement et Implémentation du Backend de l'application

i. Mission du Backend

Prenons un schéma pour mieux comprendre le rôle du backend :

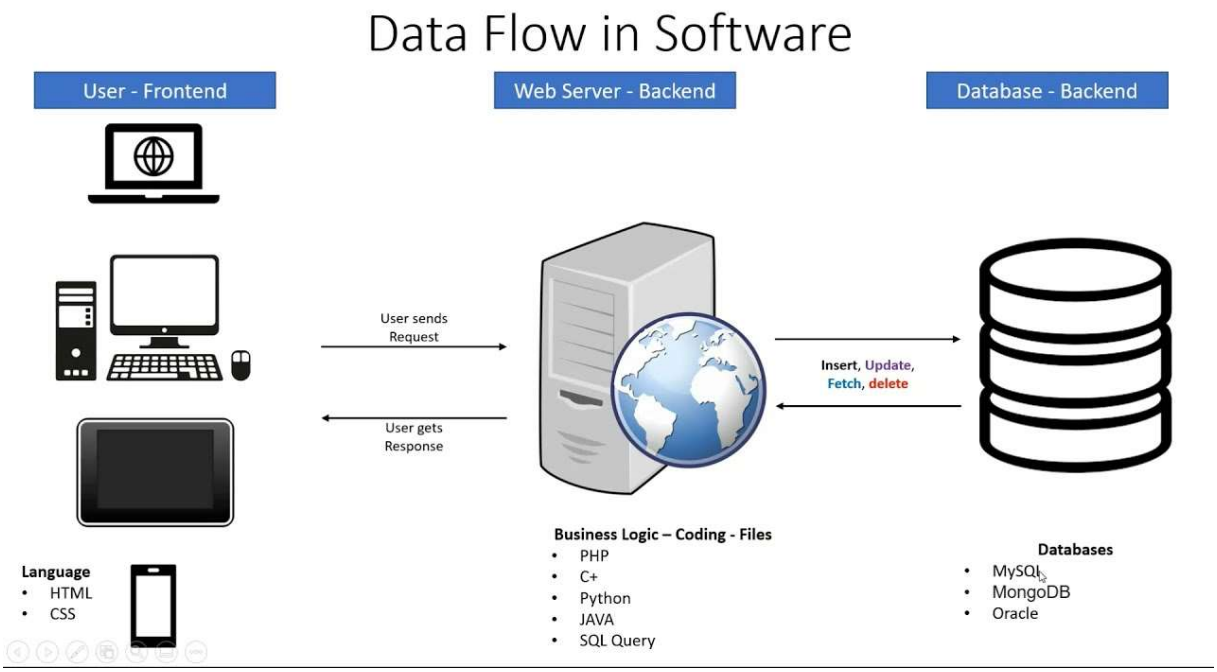


Figure 4. Schéma du rôle du backend

La partie frontend qui va être en contact avec l'user va réaliser des requêtes qui seront envoyés au backend dont la mission va être d'interroger la base de données ou d'y faire des modifications et de retourner des réponses à ces requêtes au frontend.
Le frontend n'a pas le droit d'interroger la base de données directement principalement pour des questions de sécurité permettant d'être sûr que le backend joue un rôle de pare-feu et évite les mauvaises requêtes d'être envoyé à la base de données.

ii. Architecture Routes / Contrôleurs :

Nous avons essayé d'avoir la meilleure architecture possible selon nos connaissances et après quelques heures de recherche on en est venu à la conclusion d'utiliser l'architecture routes / contrôleurs pour le backend.
Cette architecture permet pour chaque type d'appel (ex : appel GET sur les biens immobiliers) de les rediriger sur un contrôleur spécifique qui lui va se charger de réaliser l'action réelle d'interrogation de la base de données.

Exemple :

Redirection des appels vers la bonne route (fichier src/app.js) :

```
app.use('/api/bien_immobilier', bienImmobilierRoutes);
```

Redirection des appels vers le bon contrôleur (fichier routes/bienImmobilierRoutes.js) :

```
router.get('/', bienImmobilierCtrl.getAllBienImmobilier);
router.post('/', bienImmobilierCtrl.createBienImmobilier);
router.get('/id', bienImmobilierCtrl.getBienImmobilierById);
router.put('/id', bienImmobilierCtrl.updateBienImmobilierById);
router.delete('/id', bienImmobilierCtrl.deleteBienImmobilierById);
```

Implémentation du code dans les fonctions contrôleurs qui vont faire les requêtes dans la base (fichier controllers/bienImmobilierControllers.js) :

```
exports.createBienImmobilier = (req, res, next) => {
    disponibilite = req.query.disponibilite;
    nb_piece = req.query.nb_piece;
    superficie = req.query.superficie;
    prix_vente = req.query.prix_vente;
    date_disponibilite = req.query.prix_vente;
    etat = req.query.etat;
    type_bien = req.query.type_bien;
    id_adresse = req.query.id_adresse;
    id_proprietaire = req.query.id_proprietaire;
    if(disponibilite && nb_piece && superficie && prix_vente && date_disponibilite && etat && type_bien && id_adresse && id_proprietaire){
        mysql.query('INSERT INTO Bien_immobilier (disponibilite, nb_piece, superficie, prix_vente, date_disponibilite, etat, type_bien, id_adresse, id_proprietaire) VALUES (?,
            if (!err) {
                res.status(200).json({
                    message: 'Bien immobilier créé !'
                });
            }
            else {
                res.status(500).json(err);
            }
        });
    }
    else {
        res.status(500).json({
            message: 'Veuillez remplir tous les champs !'
        });
    }
}
```

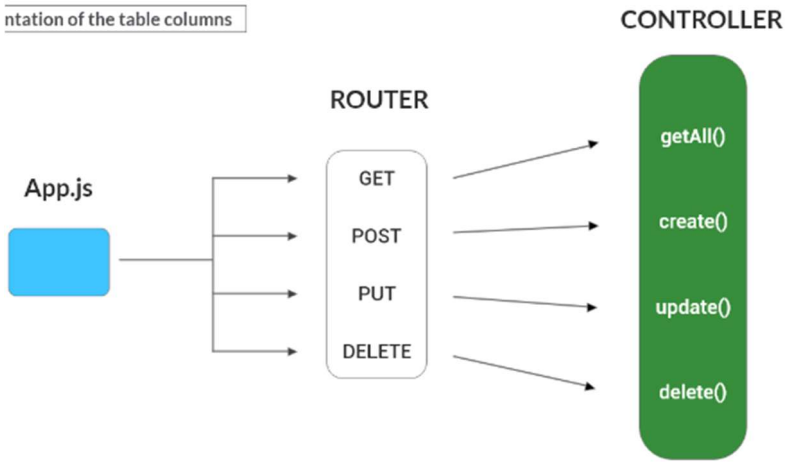
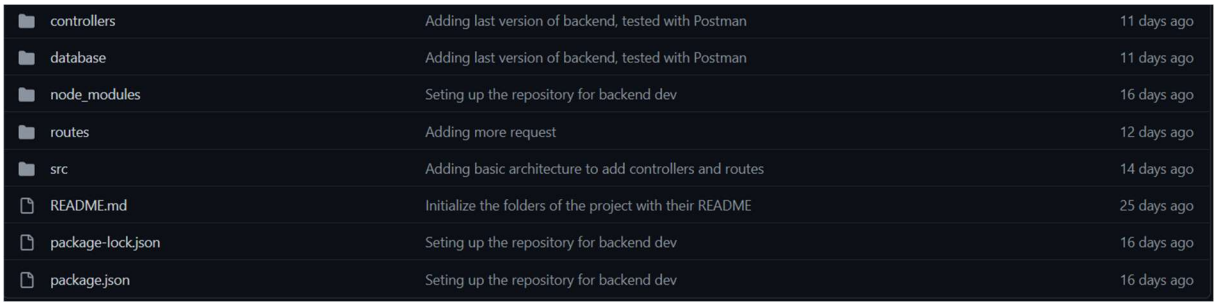


Figure 5. Schéma résumant l'architecture

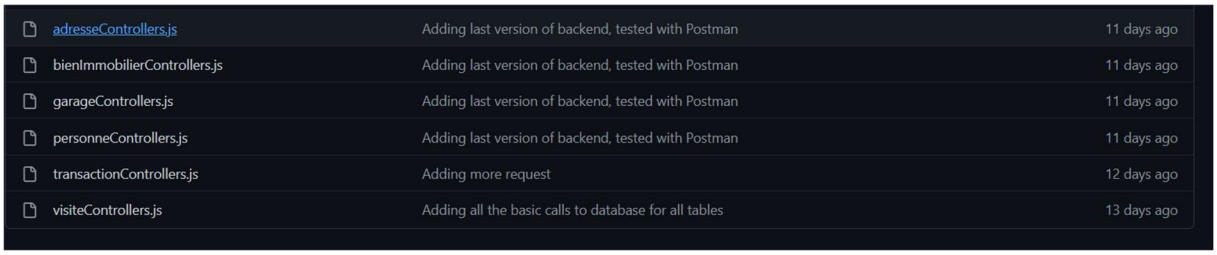
Forces & Faiblesses :
La force de cette architecture nous permet d'avoir une capacité de scalabilité (d'augmenter le nombre de requêtes que notre code peut gérer) sans nous perdre dans notre code avec un seul gros fichier qui gère tout.
Cela nous donne aussi une bonne organisation nous permettant de facilement trouver lorsque l'on debug notre programme ou le code a un problème.
Cette architecture permet cependant une scalabilité moyenne ou l'on peut gérer environ 20 requêtes par contrôleurs mais au-delà ça ne devient plus gérable vu que le fichier devient trop gros et un élément pour un projet plus gros que l'on aurait pu mettre en place aurait été de diviser les fichiers de contrôleurs qui sont actuellement pour chaque table en fichiers pour chaque type de requête (GET, POST) pour encore plus divisé et simplifier le code.

iii. Architecture du Code



Dans cette partie je vais expliquer quelle est la mission de chaque dossier du backend :

- Controllers : Dans ce dossier on va retrouver l'ensemble des controllers classé par TABLE du modèle SQL :



adresseControllers.js	Adding last version of backend, tested with Postman	11 days ago
bienImmobilierControllers.js	Adding last version of backend, tested with Postman	11 days ago
garageControllers.js	Adding last version of backend, tested with Postman	11 days ago
personneControllers.js	Adding last version of backend, tested with Postman	11 days ago
transactionControllers.js	Adding more request	12 days ago
visiteControllers.js	Adding all the basic calls to database for all tables	13 days ago

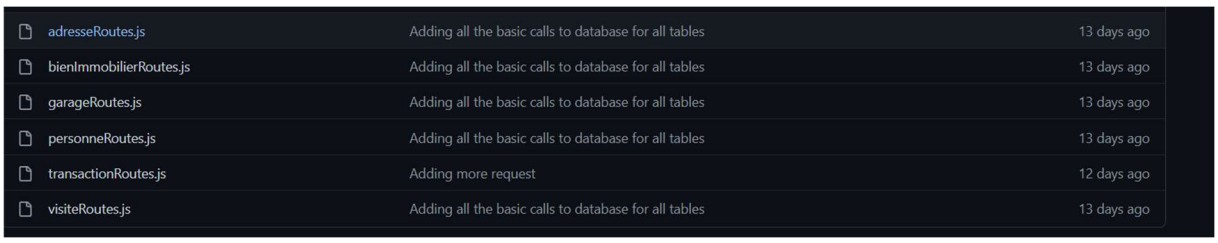
- Database : Dans ce dossier on va retrouver un simple fichier database.js qui va simplement contenir des infos d'authentification pour se connecter à sa base de données MySQL :

```
const mysql = require('mysql2');

module.exports = mysql.createConnection({
    host: 'localhost',
    user: 'root',
    password: 'root',
    database: 'db_project'
});
```

- Nodes_modules : Un dossier généré par notre compileur Javascript qui va simplement contenir toutes nos dépendances qui sont essentielles pour le projet.

- Routes : Le dossier contenant l'ensemble de nos routes qui vont router les appels reçus par le backend vers la bonne fonction dans notre controller.



adresseRoutes.js	Adding all the basic calls to database for all tables	13 days ago
bienImmobilierRoutes.js	Adding all the basic calls to database for all tables	13 days ago
garageRoutes.js	Adding all the basic calls to database for all tables	13 days ago
personneRoutes.js	Adding all the basic calls to database for all tables	13 days ago
transactionRoutes.js	Adding more request	12 days ago
visiteRoutes.js	Adding all the basic calls to database for all tables	13 days ago

- Src : Dossier source de notre projet contenant la définition dans notre code du serveur ainsi que l'application de notre serveur qui sont en quelque sorte les (main) de notre serveur aussi appelés « entrypoint » ou point d'entrée. C'est dans ces fichiers la qu'on décide vers quelle route sont redirigés nos requêtes (qui vont ensuite les rediriger vers les contrôleurs).

```
19  app.use('/api/bien_immobilier', bienImmobilierRoutes);
20  app.use('/api/adresse', adresseRoutes);
21  app.use('/api/garage', garageRoutes);
22  app.use('/api/personne', personneRoutes);
23  app.use('/api/transaction', transactionRoutes);
24  app.use('/api/visite', visiteRoutes);
25
```

- Package.json / package-lock.json : Dossier texte répertoriant l'ensemble des dépendances contenues dans notre dossier nodes_modules.

- README.md : Dossier texte écrit dans le langage Markdown qui décrit le backend du projet.

c. Amélioration de la Base de données de l'application

Durant la phase de développement, nous avons remarqué que le modèle de base de données conceptualisés au préalable n'était pas le plus adapté. Des améliorations ont donc été réalisé et les différents modèles ont été remanié.

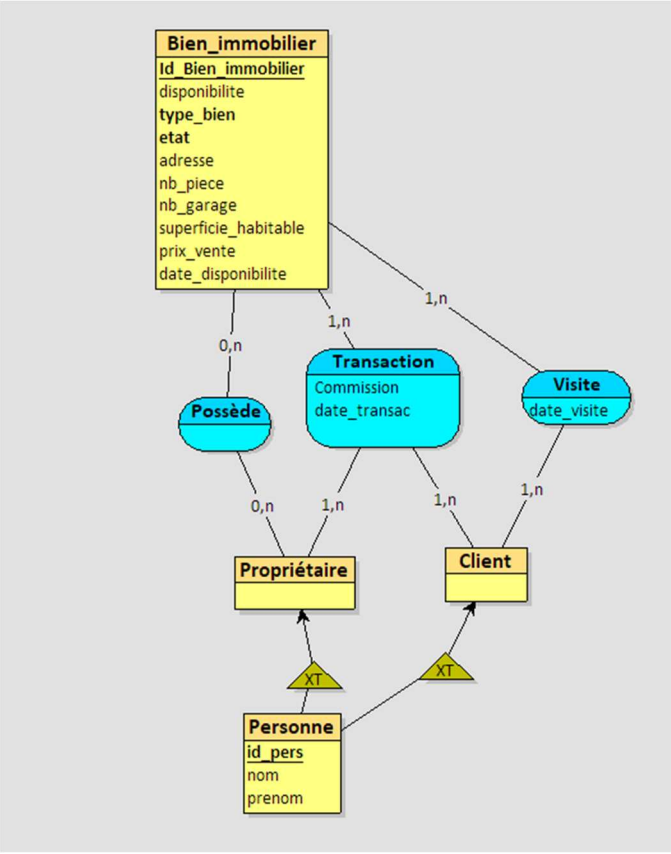


Figure 6. Modèle Entité/Relation Finale

On observe donc que le Modèle E/R a été simplifié :

- Les entités Garage, Adresse, Type_Bien et Etat ont été remplacé par des attributs au sein de la l'entité Bien_immobilier.

- L'attribut prix_vente a été supprimer de l'association Transaction car il est accessible au sein de l'entité Bien_immobilier.

Nous avons donc pu retravailler le schéma relationnel de la base de données puis à partir de celui-ci nous avons pu réaliser les requêtes de création de la base de données.

Bien_Immobilier (int **ID_bien_Immobilier**, booléen disponibilite, int nb_piece, int superficie, int prix_vente, date date_disponibilité, String etat, String type_bien, String adresse, int nb_garage, int #id_propriétaire)

Personne (int **id_personne**, String nom, String prénom)

Transaction (int Commission, date date_transac, int #id_client, int #id_propriétaire, int #bien)

Visite (date date_visite, int #id_client, int #bien)

Figure 7. Schéma relationnel final

```
create table Personne(
               id_personne int PRIMARY KEY NOT NULL auto_increment,
               nom varchar(15),
               prenom varchar(30)
       );

create table Bien_immobilier(id_bien int PRIMARY KEY NOT NULL auto_increment,
                                                    disponibilite boolean,
                                                    nb_piece int,
                                                    superficie int,
                                                    prix_vente int,
                                                    date_disponibilite date,
                                                    etat ENUM ('neuf', 'bon état', 'très bon état', 'mauvais état'),
                                                    type_bien ENUM ('maison', 'appartement'),
                                                    adresse varchar(300),
                                                    id_proprietaire int,
                                                    nb_garage int,
                                                    foreign key(id_proprietaire) references Personne(id_personne) on delete cascade on update cascade
);

create table Transaction(commission int,
                                            date_transac date,
                                            id_client int,
                                            id_propriétaire int,
                                            id_bien int,
                                            constraint pk_transaction primary key(id_client,id_propriétaire, id_bien),
                                            foreign key(id_client) references Personne(id_personne) on delete cascade on update cascade,
                                            foreign key(id_propriétaire) references Personne(id_personne) on delete cascade on update cascade,
                                            foreign key(id_bien) references Bien_immobilier(id_bien) on delete cascade on update cascade,

);
create table Visite(date_visite date,
                                    id_client int,
                                    id_bien int,
                                    constraint pk_visite primary key(id_client,id_bien),
                                    foreign key(id_client) references Personne(id_personne) on delete cascade on update cascade,
                                foreign key(id_bien) references Bien_immobilier(id_bien) on delete cascade on update cascade,
                                );
```

Figure 8. Requête de création de la base de données

d. Développement et implémentation des interfaces de l'application

Création des utilisateurs (Propriétaire/Client) :
On a donc ici un formulaire permettant de créer des nouveaux utilisateurs dans la base de données. Pour cela il suffit de renseigner son nom et prénom.

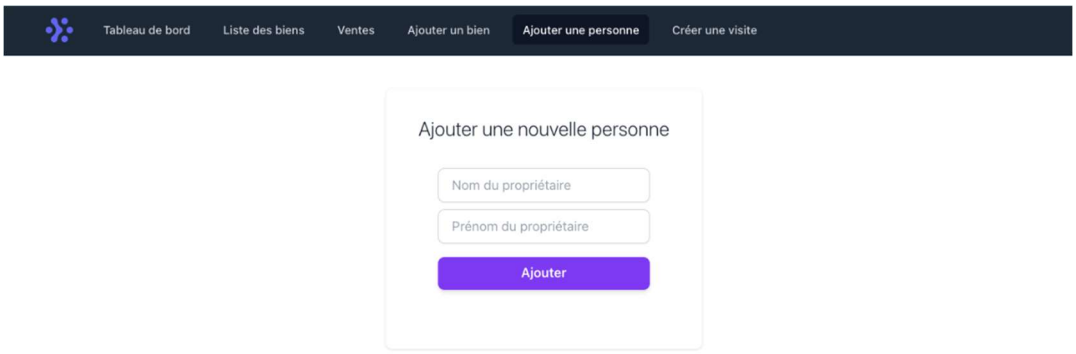


Figure 9. Page d'une personne

Création d'un bien immobilier :
Pour ajouter un nouveau bien immobilier, il faut renseigner le propriétaire, l'adresse, le type de bien, le nombre de pièce, l'état, la date de disponibilité, le prix, le nombre de garage et la superficie.

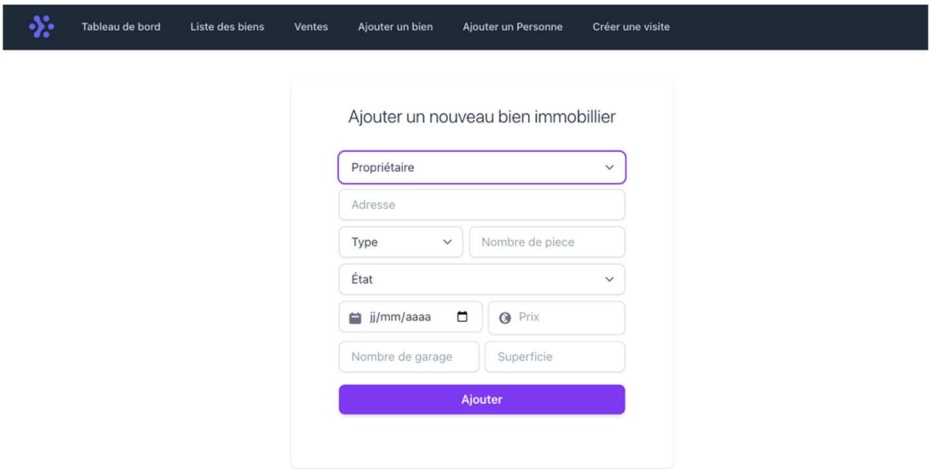


Figure 10. Page de création d'un Bien immobilier

Pour certaines informations, il faut choisir parmi une liste de données. Comme le propriétaire.

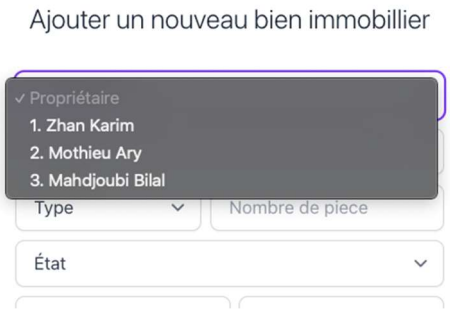


Figure 11. Choix d'un propriétaire

Liste des biens immobilier :

Une fois les des biens ajoutés à la base de données, on aura possibilité de consulter la liste de tous les biens disponibles sur le marché.

En appuyant sur le bouton «Voir» d'un bien, il est ainsi possible de voir toutes les informations liées à celui-ci. S'il est disponible il sera également possible de le vendre. En appuyant sur le bouton vendre.

Figure 12. Liste des bien immobilier

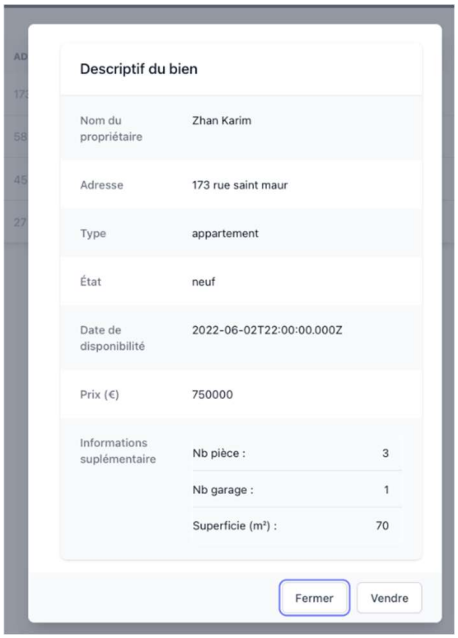


Figure 13. Descriptif d'un Bien

Pour finaliser la vente il est convenu de renseigner le client, en choisissant parmi la liste de tous les clients, la date d'achat ainsi que la commission que va toucher l'agent.

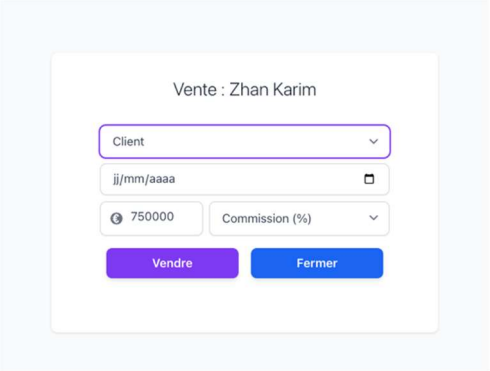


Figure 14. Formulaire de finalisation d'une vente

Liste des ventes :

Une fois le bien vendu, le bien disparait de la liste des biens disponibles.

Néanmoins, il est possible de le retrouver dans la section vente. Ici on va trouver la liste de tous les biens vendus.

Figure 15. Liste des Bien Vendu

Comme pour la liste des biens, il est également possible de visionner ces attributs en cliquant sur « voir ». Ici naturellement le bouton vendre disparait, et la mention « vendu » est spécifiée dans l'entête.

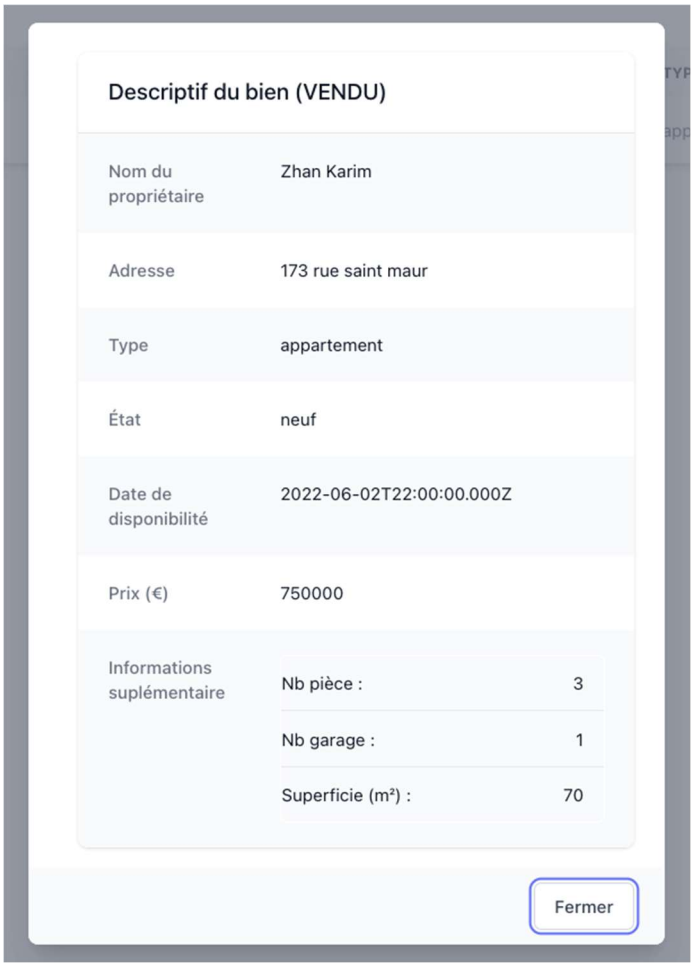


Tableau de bord :

Après avoir effectué des ventes, il va être possible de visionner un petit récapitulatif des activités. On a notamment le nombre total de vente, le montant généré, le montant des commissions, ainsi que la dernière date de vente.

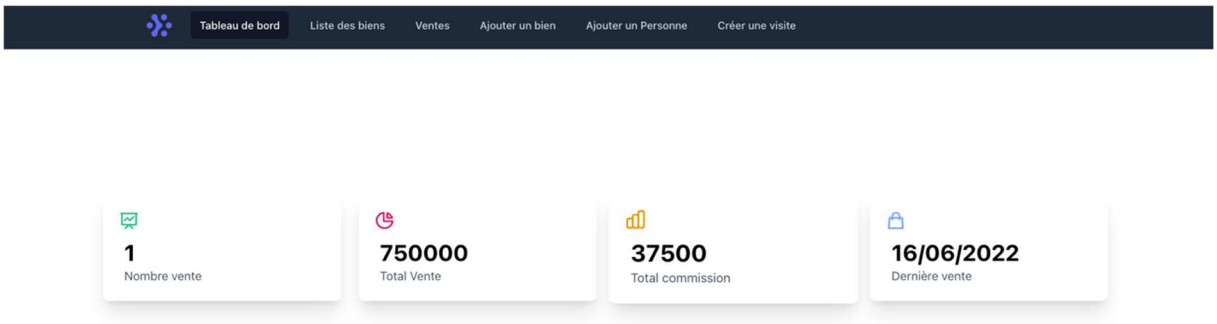


Figure 16. Page de Tableau de Bord

Création de visites :

Avant d'acheter un bien il est tout naturel de la visiter. C'est pourquoi il est possible de créer des visites, en renseigner le bien et le client concerné ainsi que la date à laquelle se déroulera la visite.

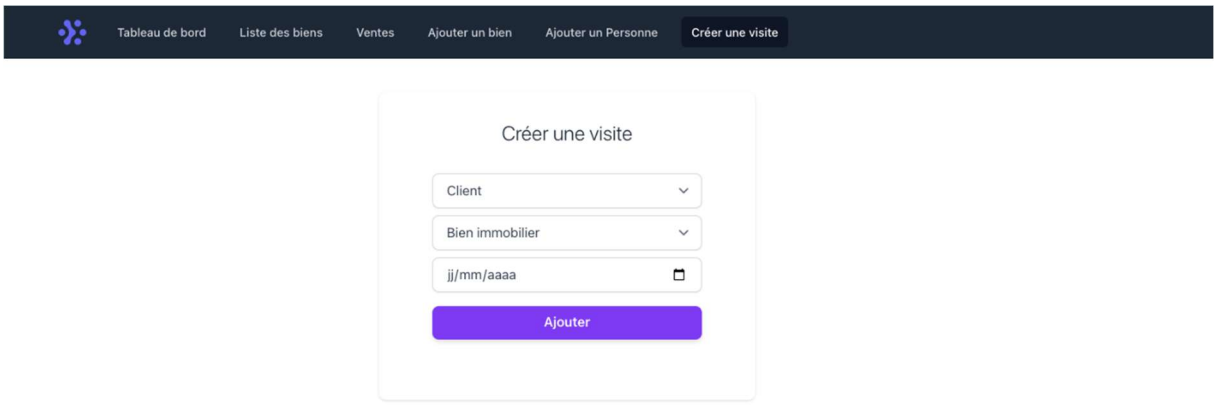


Figure 17. Page de création d'une visite

3. Conclusion

Notre application regroupe toutes les fonctionnalités demandées dans le sujet c'est-à-dire la création et gestion de bien immobilier et de personne ainsi que la gestion des transactions et des visites qui en découle. De plus les interfaces utilisateurs sont simple à prendre en main et visuellement agréable pour l'utilisateur ce qui rajoute donc un plus à l'application.

Néanmoins des améliorations aurait pu être réalisé notamment sur la gestion des adresses ou des garages mais n'ont pas pu être réalisé dû au peu d'informations transmissent sur ces entités. De plus, dû au manque d'informations données par le sujet, la gestion des commissions a été simplifié alors qu'elle aurait pu être, selon nous, étoffé. Une fonctionnalité intéressante que nous aurions pu ajouter serait l'affichage d'un historique de transaction avec différents filtres selon la date, le propriétaire ou le client impliqué par exemple. De même une page de visualisation de la liste des visites selon un client ou selon un bien immobilier aurait été intéressante à développer mais n'a pu être réaliser dû à une contrainte de temps.

Ce projet nous aura donc permis de réaliser une application conséquente mettant en pratique les connaissances et compétences acquises lors du cours de Bases de données tout en nous permettant d'en acquérir de nouvelles via l'apprentissage de NodeJS et React. Nos compétences en travail d'équipe notamment dans la répartition des tâches a elle aussi grandement été amélioré par l'utilisation de GitHub afin de réaliser le suivi du projet.