

# LaJungle Immobilier

---

Interface d'administration d'une agence immobilière réalisée avec les technologies du web

- [Origine du nom](#)
- [Répartition des rôles](#)
- [Accès au site](#)
- [Environnement technique](#)
- [Dépendances node.js](#)
- [Process de développement Frontend](#)
  - [1. Faire une maquette de site Web qui affiche le modèle de données de la base de données.](#)
  - [2. Analyser la maquette et la diviser en plusieurs petits composants](#)
  - [3. Transformer ces composants sous forme d'une hiérarchie](#)
- [Structure des fichiers frontend](#)
- [Process de développement Backend](#)
- [Table Transaction :](#)
- [Table Visite :](#)
- [Table Logement :](#)
- [Table Garage :](#)
- [Table Garage :](#)
- [etatHabitation](#)
- [typeLogement](#)
- [Bilan de projet](#)
  - [Ce qui a été fait](#)
  - [Perspectives d'amélioration](#)

Membres de l'équipe :

Laurie BRAL

Xing CHEN

Mélanie DANG

Antoine DESPRÉS

Groupe : L3-APP-LSI 1

05/06/2022

## Origine du nom

---

Le nom « LaJungle » est tout droit inspiré du nom du réseau d'agences immobilières « Laforêt » créé en 1991. Lors de la lecture du sujet, nous nous sommes en effet souvenus de la bande sonore de [leur publicité](#) que nous entendions à la télévision lorsque nous étions plus jeunes. Nous avons également envisagé le nom « Gros-Caillou Immobilier » en référence au quartier éponyme du VII<sup>e</sup> arrondissement de Paris, mais LaJungle a obtenu notre préférence.

## Répartition des rôles

---

Laurie BRAL - développement Frontend  
CHEN Xing - développement Frontend  
Mélanie Dang - développement Backend  
Antoine DESPRÉS - développement Backend

## Accès au site

---

Nous avons profité de ce projet pour prendre en main le déploiement d'un site web sur serveur. Ainsi, nous vous proposons d'accéder au site depuis l'adresse <http://lajungle.antoinedespres.fr/>. En cas d'indisponibilité du site, nous vous invitons à suivre les étapes de configuration

## Environnement technique

---

### Frontend

- React 18 - bibliothèque frontend publiée par Meta (Facebook) en 2013. Celle-ci permet de réaliser l'interface utilisateur facilement et rapidement, en développant des composants réutilisables à travers le site.

### Backend

- Looping (logiciel permettant la création du modèle relationnel)
- MySQL (SGBD servant à la création de tables, requêtes, et du stockage de la base de données)
- NodeJs
- Express

Nous avons choisi React comme bibliothèque pour créer l'interface utilisateur (UI) en composants. Un component est un unité de code. Cela nous permet de rendre l'UI plus

maintenable.

Côté design, on a choisi la biliothèque React Material UI pour réaliser l'UI rapidement.

React permet de créer une SPA (Single Page Application) qui génère du HTML en JavaScript côté navigateur contrairement à une application traditionnel où le HTML est généré côté serveur avec par exemple le langage de programmation PHP.

Pour que la SPA communique avec le serveur, on doit réaliser une API côté serveur qui envoie des données en format JSON.

## Dépendances node.js

---

### Frontend

Les dépendances sont installées et configurées avec l'outil d'installation de Meta [Create React App](#).

Nom	Description	Version
react	Fonctionnalités pour définir un composant React, il doit être installé avec <code>react-dom</code>	18.1.0
react-dom	Rendre les composants sur le DOM(Document Object Model) HTML	18.1.0
<code>@mui/material</code> , <code>@mui/lab</code> <code>@emotion/styled</code> <code>@emotion/react</code>	bibliothèque Material UI de Google	5.8.2
sass	Utiliser le préprocesseur Sass afin d'écrire du CSS de manière plus pratique	1.51.0
typescript	Language qui ajoute des syntaxes pour typer JavaScript	4.6.4
react-icons	Utiliser les icônes sous forme de composants React. Les icônes sont répertoriées sur <a href="#">React icons</a>	4.3.1
@types/react	Ajouter les définitions de types des méthodes de React. La dépendance doit être installée avec <code>typescript</code> .	18.0.9
@types/react-dom	Ajouter les définitions de types des méthodes de <code>react-dom</code> .	18.0.3
@types/node	Ajouter les définitions de types des méthodes de Node.js	16.11.33

## Backend


Nom	Description	Version
cors	Middleware pour permettre le <a href="#">CORS</a> entre le backend et le frontend	2.8.5
<a href="#">express</a>	Framework JavaScript permettant la conception d'APIs	4.18.1
express-session	Gestion de la connexion des utilisateurs, extension d'express	1.17.3
mysql2	Permet la connexion à la base de données, supporte les requêtes préparées	2.2.5
node-fetch	Permet l'usage de l'API Fetch dans Node.js	2.6.7

## Process de développement Frontend

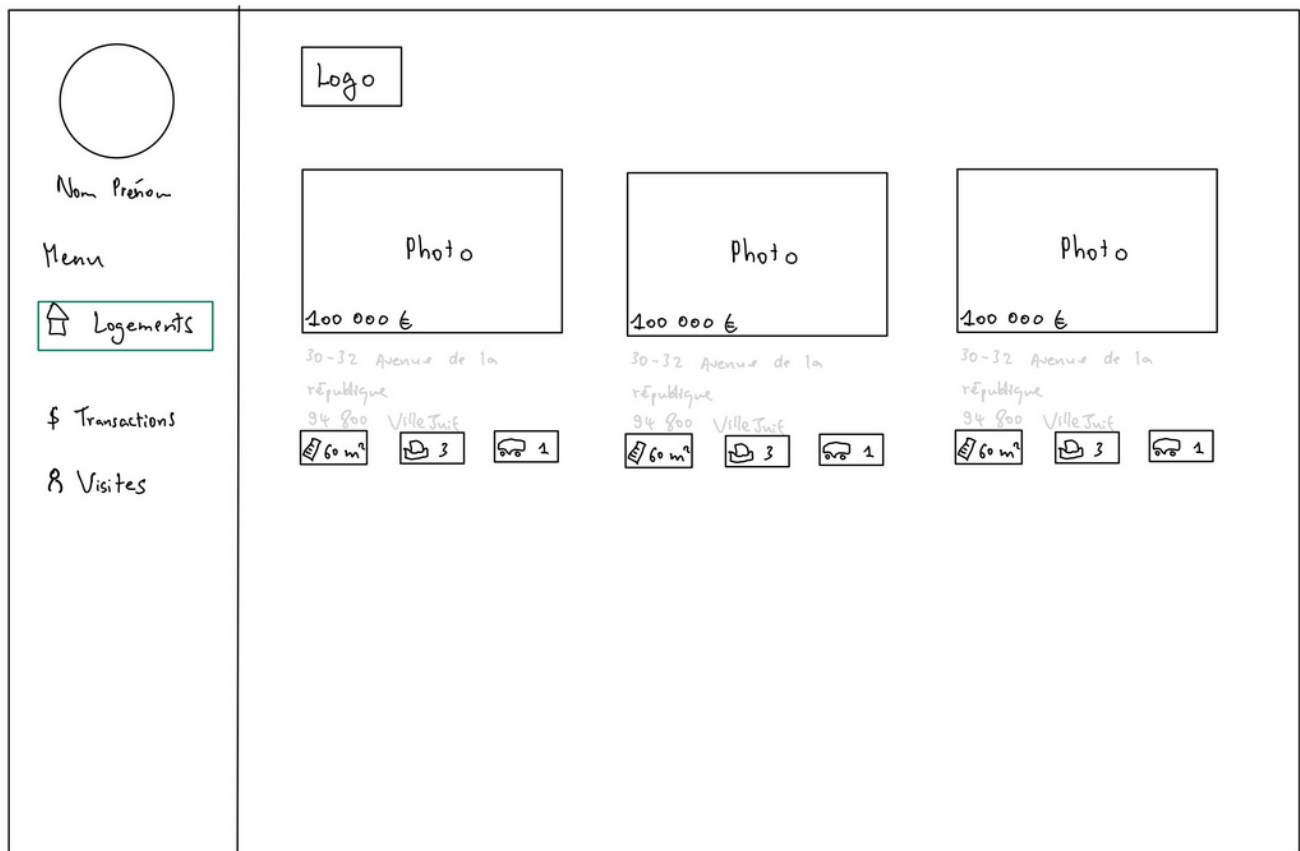
La réalisation d'une application React consiste en 3 étapes :

### 1. Faire une maquette

Choix de la palette de couleur

Couleurs	Code couleur	Représentation
Couleur primaire	Vert jungle #29AB87	

Nous avons simplifié la maquette suivante pour l'exemple dans le rapport:



source : [134 Shades of Green Color With Names, Hex, RGB, CMYK Codes](#)

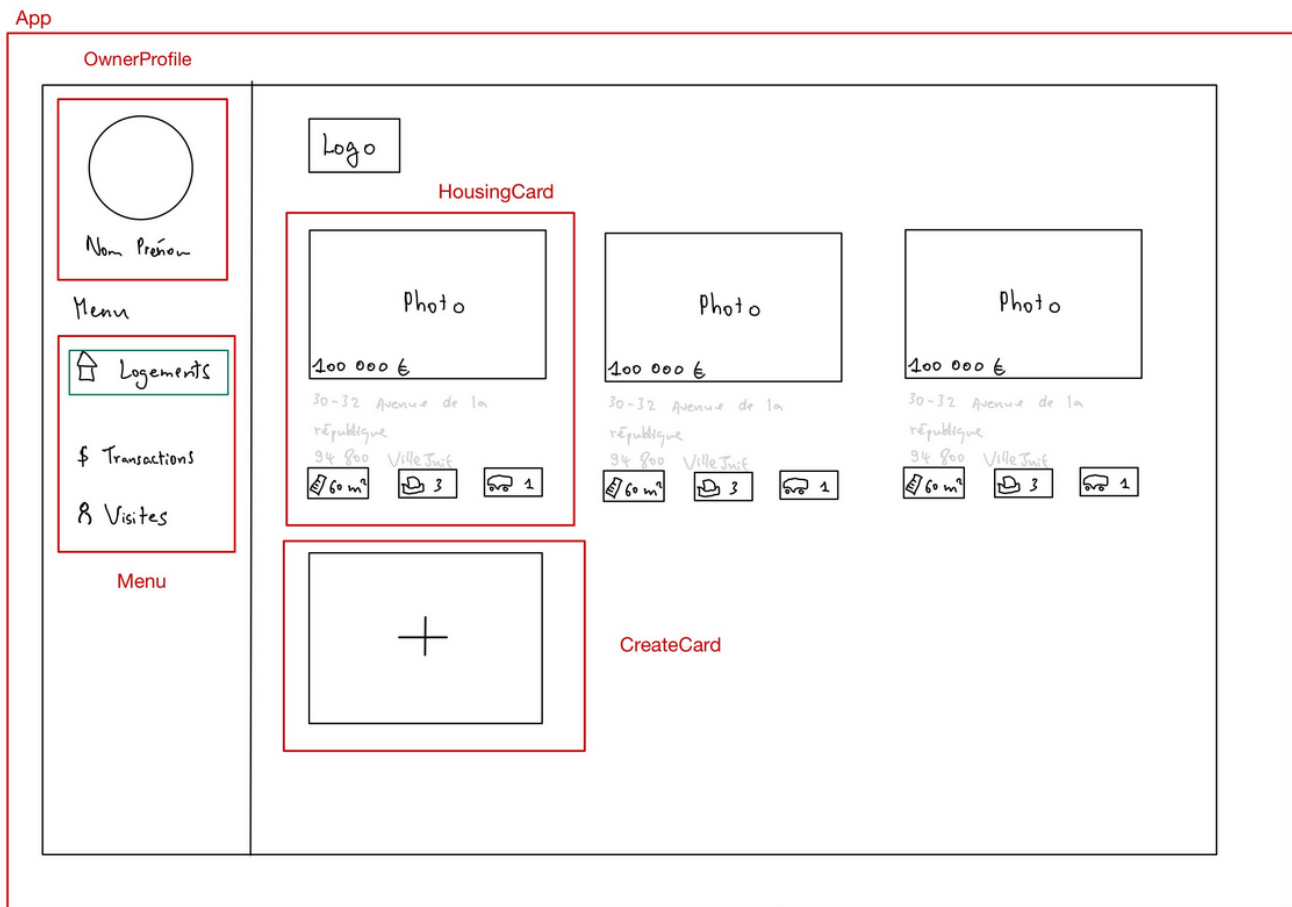
Inspiration maquette :

[Chameleon Logo - Flaticon](#)

[Real estate - Dribble](#)

## 2. Analyser la maquette et la diviser en plusieurs petits composants

Diviser et identifier chaque partie de l'UI. Donner un nom de composant à chaque partie divisé :



`App` est le composant racine qui contient tous les composants de l'application.

`HousingList` n'est pas présent sur l'illustration ci-dessus, mais il contient la liste de `HousingCard` ainsi que `CreateCard`.

### 3. Transformer ces composants sous forme d'une hiérarchie

Les composants Reacts s'organisent sous forme d'un arbre. `App` est le composant racine qui contient tous les composants de l'application.

`HomePage` représente la page d'accueil et il contient les **composants enfants** `Logo`, `OwnerProfile`, `Menu`, `HousingCard`. `App` est le **composant parent**.

- `App`
  - `OwnerProfile`
  - `Menu`
  - `Logo`
  - `HousingList`
    - `HousingCard`

## Structure des fichiers backend

---

Tout le code source est situé dans le répertoire `src/`. Sa structure est la suivante :

<<<<<< HEAD

- `controllers/` : Chaque fichier JS correspond à une table de données. Un contrôleur fait appel au modèle qui lui est associé pour récupérer les résultats d'une requête en passant les paramètres voulus. Le contrôleur renvoie ensuite les résultats à la partie *front* au format JSON.
- `helpers/` : le fichier `dbconnect.js` à l'intérieur du dossier gère la connexion à la base de données `mySQL` et crée les tables au démarrage de l'API si elles n'existent pas. La configuration `mySQL` (identifiant, mot de passe) se trouve dans le fichier `src/config.json`.
- `middlewares/` : Le fichier `auth.js` à l'intérieur du dossier gère l'authentification via JSON Web Token (JWT) et vérifie que l'utilisateur est autorisé à accéder aux pages du site.

=====

- `config.json` : Ce fichier contient un exemple de configuration permettant d'utiliser l'API. Il faut donc créer un fichier `config.json` dans le même répertoire et le compléter avec les informations nécessaires, notamment le nom d'utilisateur et le mot de passe permettant de se connecter à `mySQL`.
- `controllers/` : Chaque fichier JS correspond à une table de données. Un contrôleur fait appel au modèle qui lui est associé pour récupérer les résultats d'une requête en passant les paramètres voulus. Le contrôleur renvoie ensuite les résultats à la partie *front* au format JSON.
- `helpers/` : le fichier `dbconnect.js` à l'intérieur du dossier gère la connexion à la base de données `mySQL` et crée les tables au démarrage de l'API si elles n'existent pas. La configuration `mySQL` (identifiant, mot de passe) se trouve dans le fichier `src/config.json`.

| | | | | | | 0943edd7c7cc556e251572f54730f0b0ed29ff75

- `models/` : Chaque fichier JS à l'intérieur du dossier correspond à une table de la base de données. Ils contiennent les différentes requêtes SQL possibles, reçoivent les paramètres envoyés par le contrôleur et lui renvoient le résultat de la requête.  
Par exemple, on trouvera dans le fichier `logementDAO.js` les requêtes SQL permettant d'ajouter, de consulter, de mettre à jour ou encore de supprimer un logement.



- `routes/` : Contient les différentes routes de l'API, par exemple : `/api/exemple` . Utilise la dépendance `express-router`.
- `strategies/` : Permet l'authentification de l'utilisateur via son compte Microsoft.

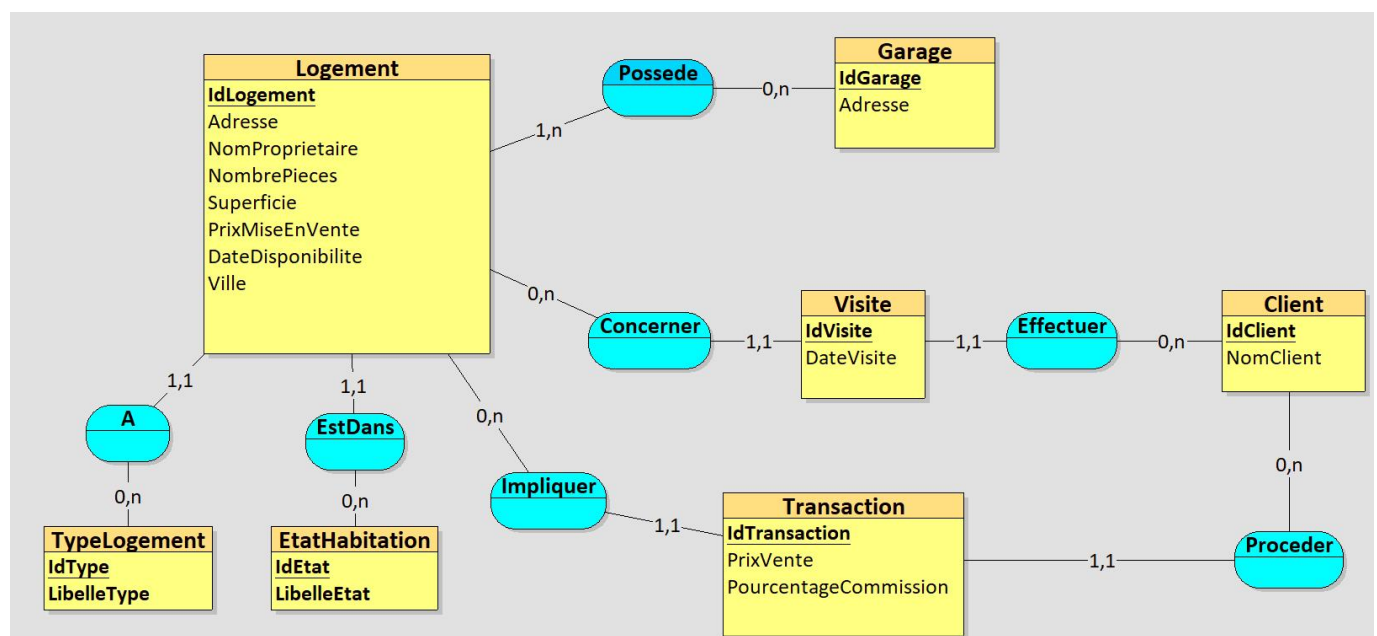
## Structure des fichiers frontend

Tout le code source est situé dans le répertoire `src/` . Sa structure est la suivante :

- `assets` - fichiers statiques comme les images, logo, vidéos, etc.
- `components` - des composants globaux et réutilisables partout dans l'application
- `styles` - les fichiers de styles en `.scss`
- `views` - une vue est une page de l'application. Ce répertoire contient toutes les vues de l'application et ainsi les composants utilisés dans une vue spécifique. Par exemple le component `HousingCard` est spécifique à la route `/index` qui est la page d'accueil.

## Process de développement Backend

### Modèle relationnel



### Choix du modèle relationnel

- On considère qu'un logement possède 0 ou plusieurs garages, et qu'un garage est possédé par 1 ou plusieurs logements (dans le cas d'un parking commun à toute une résidence, par exemple). Dans un souci de clarté, on a ajouté un attribut "Adresse" à la table garage.

- Une visite concerne 1 logement, et un logement peut être visité plusieurs fois.
- Un client peut effectuer plusieurs visites, et une visite est faite par un seul client.
- Un client peut procéder à 0 ou plusieurs transactions, et une transaction ne concerne qu'un client. Une transaction implique un seul logement à la fois, et un logement est impliqué dans 0 ou plusieurs transactions.

Nous avons ajouté une table utilisateur avec les attributs "NomUtilisateur" et "MotDePasse", qui permet à une personne de créer son compte et son mot de passe afin d'accéder au site.

**Script SQL** Nous avons rédigé deux scripts SQL afin de construire une base de données en local et ainsi tester notre application.

- Le script permettant d'ajouter les tables nécessaires : [Script de création des tables](#)
- Le script permettant d'insérer des données dans les tables : [Script d'insertion de valeurs](#)

### Requête pour chaque table

- Pour chaque table, nous avons codé des fonctions asynchrones en js afin de répondre aux besoins de l'application. Chaque table contient au moins les fonctions get, create, delete.
- Pour la fonction asynchrone update, qui permet de modifier les informations d'une ligne, nous avons utilisé un string dynamique qui prend seulement en compte les paramètres modifiés.

Nous avons ajouté une table utilisateur avec les attributs "NomUtilisateur" et "MotDePasse", qui permet à une personne de créer son compte et son mot de passe afin d'accéder au site.

**API Routes** Pour chaque requête, un code de statut de réponse HTTP est envoyé afin de vérifier si la requête s'est bien réalisée ou non.

Code HTTP	Description
200	La requête s'est bien déroulée
400	Erreur dans la requête (exemple : ID de l'objet non défini)
500	Erreur du serveur

Voici les API routes pour les requêtes que nous avons utilisé dans notre projet :

## Table Transaction :

<<<<<< HEAD

<http://localhost:5000/api/transaction/create> :

===== Route <http://localhost:5000/api/transaction/create> :

0943edd7c7cc556e251572f54730f0b0ed29ff75 Permet de créer une transaction. Prend paramètres : | Paramètres | |----  
-----| | IdTransaction | | PrixVente | |  
PourcentageCommission | | IdClient | | IdLogement |

<http://localhost:5000/api/transaction/get> :

<<<<<< HEAD Prend en paramètre l'ID de la transaction souhaitée. Permet de retourner les informations d'une transaction spécifique (IdTransaction, PrixVente PourcentageCommissionIdLogement, IdClient). ===== Route <http://localhost:5000/api/transaction/get> : Prend en paramètre l'id de la transaction souhaitée. Permet de retourner les informations d'une transaction spécifique (IdTransaction, PrixVente PourcentageCommissionIdLogement, IdClient).

0943edd7c7cc556e251572f54730f0b0ed29ff75

Exemple de réponse JSON :

```
{
  <<<<<< HEAD
    "PrixVente": 135.0,
    "PourcentageCommission": 4.2,
    "IdClient": 1,
    "IdLogement": 3
}
```

<http://localhost:5000/api/transaction/getAll> :

Permet de retourner la liste de toutes les transactions de la base de données.

<http://localhost:5000/api/transaction/update> :

**Prend en paramètre l'ID de la transaction souhaitée. Permet de modifier les informations d'une transaction, en fonction des paramètres envoyés.**

```
"data": {
  "idTransaction": 1,
  "prixVente": 135,
  "pourcentageCommission": "4.20",
  "idLogement": 3,
  "idClient": 1
},
"error": null
```

```
}
```

Route <http://localhost:5000/api/transaction/getAll> :  
Permet de retourner la liste de toutes les transactions de la base de données.

```
```json
{
  "data": [
    {
      "idVisite": 1,
      "dateHeureVisite": "2022-11-29T13:15:00.000Z",
      "idLogement": 1,
      "idClient": 4,
      "adresse": "9 Rue du Chat qui Pêche",
      "nomClient": "Mélanie Dang"
    },
    {
      "idVisite": 2,
      "dateHeureVisite": "2022-10-11T08:30:00.000Z",
      "idLogement": 2,
      "idClient": 2,
      "adresse": "25 Rue Brisemiche",
      "nomClient": "Xing Chen"
    }
  ],
  "error": null
}
```

Route <http://localhost:5000/api/transaction/update> : Prend en paramètre l'id de la transaction souhaitée. Permet de modifier les informations d'une transaction, en fonction des paramètres envoyés. Route <http://localhost:5000/api/transaction/remove> : Permet de supprimer une transaction spécifique de la base de données.

||| 0943edd7c7cc556e251572f54730f0b0ed29ff75

## <http://localhost:5000/api/transaction/remove> :

Permet de supprimer une transaction spécifique de la base de données.

Paramètres
IdTransaction

## Table Visite :

Route <http://localhost:5000/api/visite/create> :

Permet de créer une visite. | Paramètres | |-----| | dateHeureVisite | | idLogement | | idClient |

## <http://localhost:5000/api/visite/get> :

Route <http://localhost:5000/api/visite/get> :

Permet de retourner les informations d'une transaction spécifique (IdVisite, DateHeureVisite IdLogement, IdClient).

Paramètres
id

Exemple de réponse JSON :

```
{
  "data": {
    "idVisite": 1,
    "idLogement": 1,
    "idClient": 4
  },
  "error": null
}
```

<<<<<< HEAD

## <http://localhost:5000/api/visite/getAll> :

===== Route <http://localhost:5000/api/visite/getAll> :

0943edd7c7cc556e251572f54730f0b0ed29ff75 Permet de retourner la liste de toutes les transactions de la base de données.

```
{
  "data": [
    {
      "idVisite": 1,
      "dateHeureVisite": "2022-11-29T13:15:00.000Z",
      "idLogement": 1,
      "idClient": 4,
      "adresse": "9 Rue du Chat qui Pêche",
      "nomClient": "Mélanie Dang"
    },
    {
      "idVisite": 2,
      "dateHeureVisite": "2022-10-11T08:30:00.000Z",
      "idLogement": 2,
      "idClient": 2,
      "adresse": "25 Rue Brisemiche",
      "nomClient": "Xing Chen"
    }
  ],
  "error": null
}
```

Route <http://localhost:5000/api/visite/update> :

Permet de modifier les informations d'une visite, en fonction des paramètres envoyés.

Route <http://localhost:5000/api/visite/remove> :

Permet de supprimer une visite spécifique de la base de données.

Paramètres
IdVisite

## Table Logement :

Route <http://localhost:5000/api/logement/create> : Permet de créer un logement.

Paramètres
adresse
descriptionLogement
nomProprietaire
idType
nombrePieces
superficie
idEtat
prixMiseEnVente
dateDisponibilite
codePostal
ville

Route <http://localhost:5000/api/logement/get> : Permet de retourner les informations d'un logement spécifique (IdLogement, Adresse, DescriptionLogement, NomProprietaire, IdType, NombrePieces, Superficie, IdEtat, PrixMiseEnVente, DateDisponibilite, CodePostal, Ville).

Paramètres
id

Exemple de réponse JSON :

```

{
  "data": {
    "id": 1,
    "adresse": "9 Rue du Chat qui Pêche",
    "description": "Charmant appartement dans une ruelle parisienne",
    "nomProprietaire": "Charles Potté",
    "typeLogement": {
      "label": "Appartement",
      "value": "appartement"
    },
    "nombrePieces": 2,
    "superficie": "36.40",
    "etatHabitation": {
      "label": "Bon",
      "value": "bon"
    },
    "prixMiseEnVente": 350000,
    "dateDisponibilite": "2022-06-22T22:00:00.000Z",
    "codePostal": "75005",
    "ville": "Paris",
    "nbGarages": 1
  },
  "error": null
}

```

Route <http://localhost:5000/api/logement/getAll> : Permet de retourner la liste de tous les logements de la base de données. Route <http://localhost:5000/api/logement/update> : Permet de modifier les informations d'un logement, en fonction des paramètres envoyés. Route <http://localhost:5000/api/logement/remove> : Permet de supprimer un logement spécifique de la base de données.

Paramètres
id

## Table Garage :

Route <http://localhost:5000/api/garage/get> : Permet de retourner les informations d'un garage spécifique (IdGarage, Adresse, IdLogement).

Paramètres
id



Exemple de réponse JSON :

```
{
  "data": {
    "idGarage": 1,
    "adresse": "106 Rue des 2 Boules"
  },
  "error": null
}
```

Route <http://localhost:5000/api/garage/getAll> : Permet de retourner la liste de tous les garages de la base de données.

```
{
  "data": [
    {
      "idGarage": 1,
      "adresse": "106 Rue des 2 Boules"
    },
    {
      "idGarage": 2,
      "adresse": "7 Rue des Boulets"
    },
    {
      "idGarage": 3,
      "adresse": "14 Rue Gros"
    },
    {
      "idGarage": 4,
      "adresse": "8 Rue des femmes Fraîches"
    }
  ],
  "error": null
}
```

Route <http://localhost:5000/api/garage/update> : Permet de modifier les informations d'un garage, en fonction des paramètres envoyés. Route <http://localhost:5000/api/garage/remove> : Permet de supprimer un garage spécifique de la base de données.

Paramètres
id

## Table Garage :

Route <http://localhost:5000/api/garage/get> : Permet de retourner les informations d'un garage spécifique (IdGarage, Adresse, IdLogement).

Paramètres
id

Exemple de réponse JSON :

Route <http://localhost:5000/api/garage/getAll> : Permet de retourner la liste de tous les garages de la base de données.

Route <http://localhost:5000/api/garage/update> : Permet de modifier les informations d'un garage, en fonction des paramètres envoyés. Route <http://localhost:5000/api/garage/remove> : Permet de supprimer un garage spécifique de la base de données.

Paramètres
id

## etatHabitation

Route <http://localhost:5000/api/etatHabitation/getList> : Permet de récupérer la liste de tous les états d'habitations.

```
{
  "data": [
    {
      "label": "Bon",
      "value": "bon"
    },
    {
      "label": "Mauvais",
      "value": "mauvais"
    },
    {
      "label": "Neuf",
      "value": "neuf"
    },
    {
      "label": "Très bon",
      "value": "tres_bon"
    }
  ],
  "error": null
}
```

## typeLogement

---

Route <http://localhost:5000/api/typeLogement/getList> : Permet de récupérer la liste de tous les types de logement.

```
{
  "data": [
    {
      "label": "Appartement",
      "value": "appartement"
    },
    {
      "label": "Maison",
      "value": "maison"
    }
  ],
  "error": null
}
```

## Bilan de projet

---

### Ce qui a été fait

Durant ce projet, nous avons pu pratiquer ce que nous avons appris en cours de bases de données et proposer une interface graphique, rendant ainsi l'application utilisable auprès du grand public sous la forme d'un site web. Cela a nécessité une solide coordination entre l'équipe front-end et l'équipe back-end afin de s'assurer de la cohérence des formats de données utilisées. Nous avons eu l'occasion de créer une API permettant au front-end d'aller récupérer les données de la base en spécifiant les paramètres lui permettant d'obtenir les informations dont il a besoin. Tout cela nous a permis de partager nos connaissances sur différents fronts (React, API, déploiement) et gagner en compétences. Nous sommes fiers du travail que nous avons pu réaliser ensemble.

## **Perspectives d'amélioration**

Avec plus de temps à notre disposition, nous aurions pu réaliser plus de fonctions portant notamment sur la gestion des garages et concevoir l'interface pour supprimer des transactions ou des visites. Nous aurions également pu créer une page dédiée aux statistiques afin d'obtenir, entre autres, le prix moyen des logements, le prix moyen des transactions, le total des commissions à l'aide des fonctions AVG et SUM de SQL.

## **Commentaire constructif**

Nous pensons qu'il pourrait être judicieux de lier ce projet à celui de développement web. En effet, ceux ayant opté pour la création d'un site web lors du projet de bases de données devront recommencer de zéro en développement web. Un projet commun nous permettrait de bénéficier d'un temps plus long pour proposer plus de fonctionnalités sans effectuer des tâches en double.