Dossier projet

CONCEPTEUR DÉVELOPPEUR D’APPLICATIONS

Adrien COSSON

Table des matières

Abstract

I am working in a company called Cyberauto.

The company was founded in 1998 by Alban Meynet and renamed Art's wave in 2015. Our headquarters are located in Le Mans and we are specialized in search engine optimization.

I have been working in this company as web developer since the beginning of September 2021 and I am in charge of web sites development and web sites maintenance, I'm also responsible for the server efficiency and web sites migrations.

The company provides web sites for real estate agencies and classified ads sites for automobile and housing market.

In this document, I will present a MVP (Minimum Viable Product) made in two months. It aims for developing a website for real estate agencies and people who wants to rent or sell their real estates.

The project is composed of two parts:

The main part is a headless API (Application Protocol Interface) developed using the PHP framework Laravel. This API use MariaDB as database server and OAuth 2 protocol as authentication and authorization solution.

API and database are both containerized separately with docker compose thanks to Laravel package Sail.

The second part, is a lightweight client made with JS Framework Vue.js using tailwind as utility classes CSS framework. The components are developed using API composition and Pinia was used as store system instead of VueX.

Présentation de l’entreprise

Présentation générale de la structure

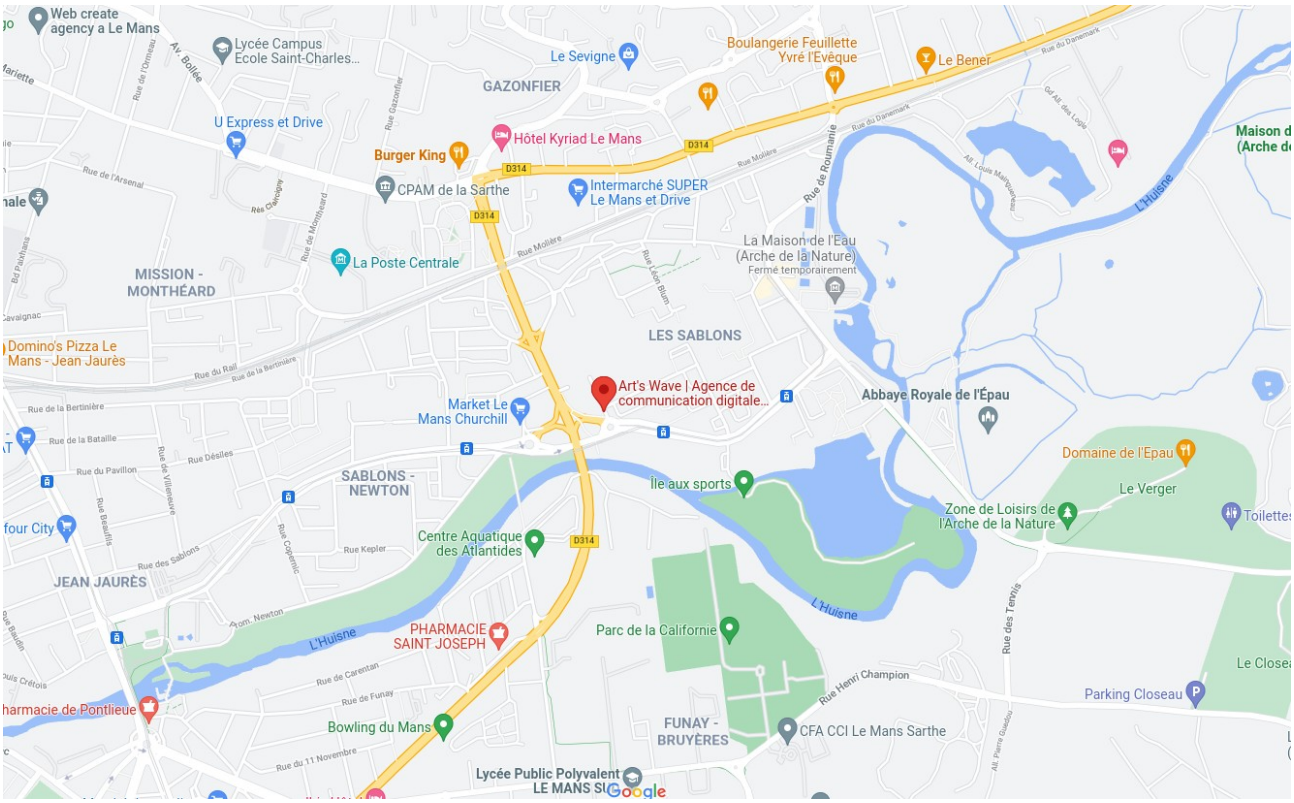
Je travaille dans une entreprise du nom de « CyberAuto », fondée en 1998 par Alban Meynet puis renommée « Art’s Wave » en 2015.

Initialement le domaine d’activité de l’entreprise était les sites de vente de véhicules d’occasions puis la production de sites vitrines destinés à de petites entreprises par exemple : des paysagistes, ostéopathes ou encore des associations.

Actuellement le principal domaine d’activité de l’entreprise est l’immobilier avec plusieurs sites pour des agences immobilières.

ArtsWave s’est récemment spécialisé dans l’optimisation du référencement naturel ou (SEO) et propose à ses clients de les accompagner dans la gestion de leurs comptes « Google my business » et « Google adwords ».

L’entreprise compte en son sein trois employés, le patron Alban Meynet, Théo Uzel en tant qu’intégrateur web en alternance et moi-même en tant que concepteur développeur d’applications. Les locaux se situent dans le quartier des Sablons, dans la ville du Mans.



2 Rue Alcide de Gasperi, 72100 Le Mans

Périmètre et responsabilités de mon poste

Étant seul développeur dans l'entreprise mes missions sont nombreuses :

- Développement et mise en production des nouveaux sites web
- Maintenance des sites existants et migrations vers des technologies plus récentes
- Sauvegardes de tous les projets de l'entreprise ainsi que de leurs bases de données
- Administration du serveur
- Migration des sites vers un nouvel hébergement.
- Gestion et formation de mon collègue

Objectifs personnels

À l'issue de la formation, j'espère dans un premier temps obtenir le titre professionnel de « concepteur développeur d'applications ».

Après quoi je souhaite continuer à me former dans ce domaine, ou, en passant un master Analyse, conception et développement informatiques en alternance à l'université d'Angers ou en passant quelque temps à l'école « 42 » pour à terme, me spécialiser dans une technologie.

À terme, j'aimerais trouver un poste de développeur back-end dans une entreprise de taille moyenne qui utilise des technologies récentes et travailler au sein d'une équipe qualifiée ou me lancer en tant que développeur indépendant.

Résumé du projet

Mon projet de fin d'année est une application web destinée aux particuliers et aux agences immobilières. La plateforme permet de mettre en ligne des biens immobilier destinés à la vente ou à la location.

Le projet est composé de deux parties :

- Une API développée en PHP à l'aide du framework Laravel
- Un client léger développé en JavaScript à l'aide du framework Vue3

Compétences du référentiel couvertes par le projet

Ce projet me permet de couvrir les compétences suivantes :

Activités types	Compétences professionnelles
Concevoir et développer des composants d'interface utilisateur en intégrant les recommandations de sécurité	Maquetter une application
	Développer une interface utilisateur de type desktop
	Développer des composants d'accès aux données
	Développer la partie front-end d'une interface utilisateur web
	Développer la partie back-end d'une interface utilisateur web
Concevoir et développer la persistance des données en intégrant les recommandations de sécurité	Concevoir une base de données
	Mettre en place une base de données
	Développer des composants dans le langage d'une base de données
Concevoir et développer une application multicouche répartie en intégrant les recommandations de sécurité	Concevoir une application
	Développer des composants métier
	Construire une application organisée en couches
	Préparer et exécuter les plans de tests d'une application

Cahier des charges

Présentation du projet

Développement d'une application web à l'intention d'agences immobilières et de particuliers, leurs permettant de gérer la mise en vente et la location de biens immobiliers.

Expression des Besoins

Dans le cadre du développement de son activité dans le secteur de l'immobilier, la société Art's wave à besoin d'une application qui traite et centralise ses clients et leurs catalogues de biens.

L'application devra être capable de fournir des ressources à plusieurs sites internets, par exemple, si une agence immobilière souhaite se servir de l'application comme logiciel métier, elle devra lui permettre de faire développer un site internet pour afficher son catalogue de bien.

Ce projet vise à fournir un PMV (Produit Minimal Viable) à une entreprise spécialisée dans les sites immobiliers, afin de démontrer les avantages d'une application moderne développer à l'aide de différents frameworks.

Compétences métier

Utilisateur

- L'utilisateur doit avoir la possibilité de créer un compte
- Pour mettre un bien en ligne l'utilisateur doit être connecté.
- L'utilisateur inscrit et connecté doit avoir la possibilité d'éditer son profil.

Biens

- Les biens doivent être accessibles de façon publique, c'est-à-dire visible par n'importe quel visiteur du site.
- Il doit être possible d'ajouter, de voir, de modifier et de supprimer un bien.

Design

- L'interface utilisateur doit être faite de façons à s'afficher aussi bien sur des appareils mobile que sur de grands écrans.

Page d'accueil

- Cette page doit donner la possibilité de faire une recherche simple avec un minimum de critères.

Page de liste pour les annonces

- La page de liste doit afficher une liste de biens correspondant aux critères de recherche entrés par l'utilisateur.
- Cette page doit donner la possibilité de faire une recherche avancée.

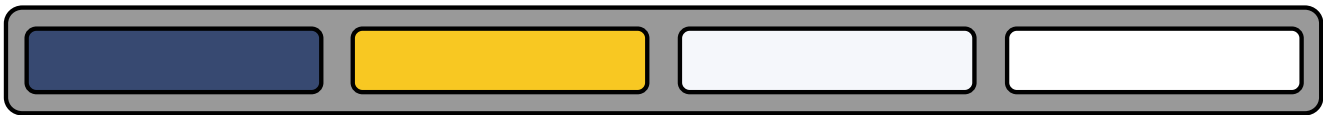
Page de détail annonces

- La page de détail doit afficher les informations concernant un seul bien et doit à minima comporter ces informations :
  - Le titre de l’annonce
  - Le sous-titre
  - La description
  - Le prix de vente
  - Les coordonnées pour joindre le vendeur
  - Les photos du bien

Charte graphique

Le site devra respecter ces couleurs :

#999999



#374971 (Couleur principale) **#F8C822** **#F5F7FB** **#FFFFFF**

Livrable attendu

Ce PMV n’a pas pour objectif de fournir à l’entreprise une application clés en main mais plutôt de lui faire découvrir les avantages à utiliser des technologies modernes. Les fonctionnalités minimums attendu pour la livraison de ce projet sont :

Back-end :

- Inscription / connexion.
- Ajout, affichage, modification et suppression de biens.
- Recherche par ville / département / région

Front-end :

- Affichage de la liste des biens
- Affichage du détail d’un bien
- Inscription / connexion
- Ajout, affichage, modification et suppression de biens.

Gestion de projet

Organisation

Afin d'organiser le développement de mon projet, j'ai fait le choix d'utiliser l'outil de gestion de projet en ligne Trello. Cet outil me permet de lister les taches à réaliser, de voir les tâches en cours ainsi que les tâches terminées.

Je découpe le projet en petites tâches que je réalise les une après les autres, cela me permet de m'organiser et de me motiver.

(Exemple en annexe)

Planning

À partir du 4 avril 2022, je dispose de huit semaines, pour développer ce projet. Afin de m'organiser, je découpe le temps dont je dispose en plusieurs parties :

<table>
<tr><td>Semaine 1</td><td rowspan="2">Deux semaines de conception :
Choix techniques
Swagger
MCD base de données</td></tr>
<tr><td>Semaine 2</td></tr>
<tr><td>Semaine 3</td><td rowspan="3">Trois semaines de développement pour le back-end</td></tr>
<tr><td>Semaine 4</td></tr>
<tr><td>Semaine 5</td></tr>
<tr><td>Semaine 6</td><td rowspan="3">Trois semaines de développement pour le front-end</td></tr>
<tr><td>Semaine 7</td></tr>
<tr><td>Semaine 8</td></tr>
</table>

Versioning et travail collaboratif

Afin de mieux gérer le code, j'utilise l'outil **git cli** afin de le versionner. Ainsi du code supprimé ou remplacé n'est jamais perdu et il est toujours possible d'annuler une modification.

J'utilise **Gitlab**, qui est une plateforme en ligne permettant de stocker et de partager le code. Gitlab permet également de travailler facilement à plusieurs sur un projet.

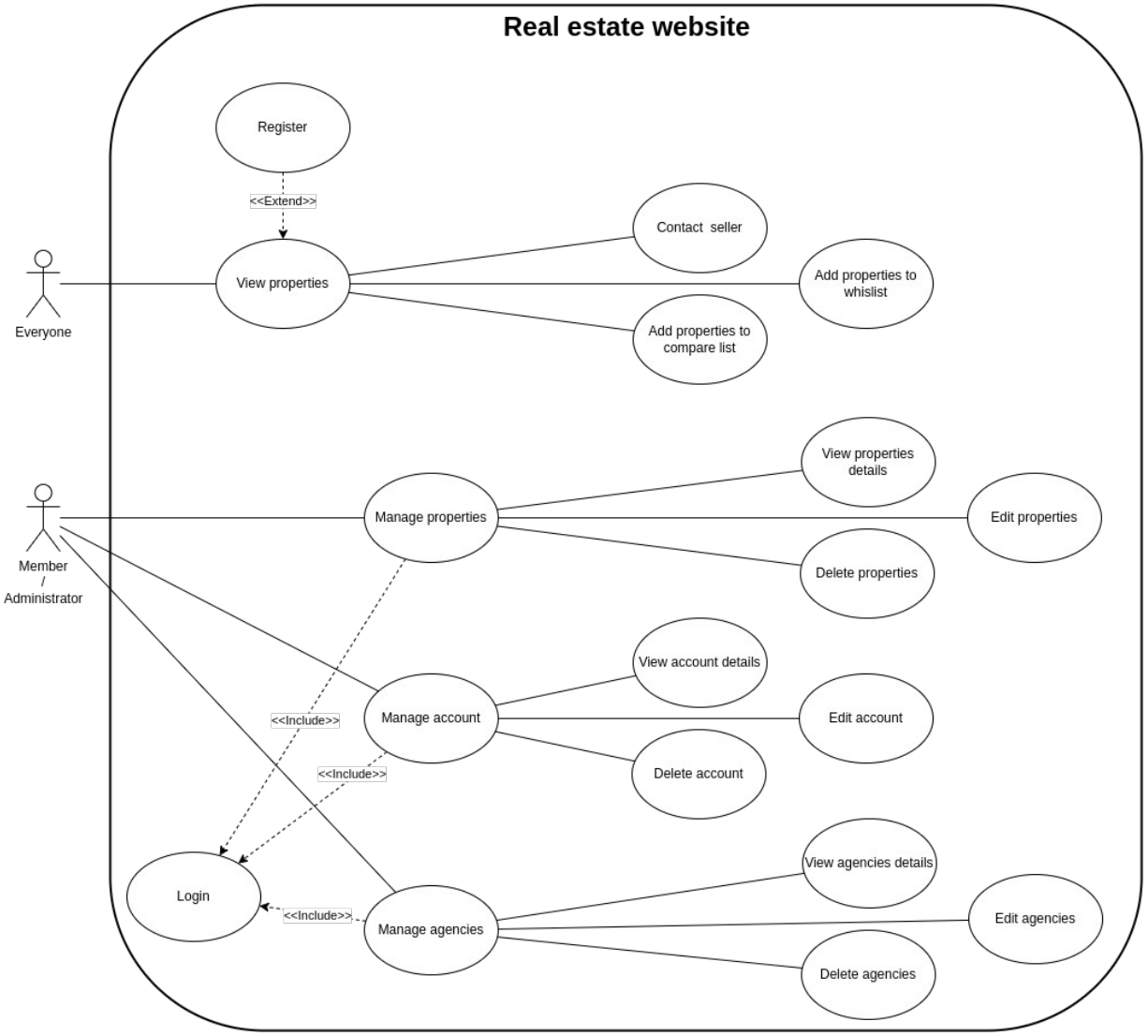
Spécifications fonctionnelles

Cas d’utilisation

Avant de commencer la réalisation du projet je me sers du logiciel draw.io pour schématiser les cas d’utilisation de l’application.

Je représente trois cas différents :

- Visiteur : Le visiteur peut consulter les annonces et contacter les propriétaires, comparer les annonces et les ajouter à sa liste de favoris. Il a aussi la possibilité de s’inscrire.
- Utilisateur : L’utilisateur peut faire la même chose que le visiteur, après connexion il peut gérer son profil, ses annonces et son agence.
- Administrateur : Il peut faire la même chose que l’utilisateur, à la différence qu’il a accès à tous les comptes des utilisateurs inscrit et à leurs annonces.



Connexion

Pour permettre aux utilisateurs de s'inscrire ou de se connecter, un bouton sera présent sur la barre de navigation du site et sera ainsi accessible sur toutes les pages. Au clic sur ce bouton un pop-up s'affiche, avec à l'intérieur deux onglets :

- Un pour s'inscrire avec les champs suivant :
  - Pseudo.
  - Nom / prénom.
  - Adresse e-mail.
  - Mot de passe.
  - Confirmation du mot de passe.
- L'autre pour la connexion avec ces champs :
  - Pseudo / adresse e-mail.
  - Mot de passe.

Barre de navigation

La barre de navigation sera identique sur chacune des pages. Elle sera composée de plusieurs liens de navigation :

- Accueil
- Vente
- Location
- Un bouton pour les annonces favorites et la comparaison
- Un bouton d'inscription/connexion
- Un bouton pour ajouter une annonce

Sur mobile ces liens se trouveront dans le menu burger placé en bas de l'écran.

Page d'accueil

Recherche

Pour une recherche simple et efficace sur la page d'accueil, il sera utilisé trois listes déroulantes :

- Type de bien
- Type de vente
- Ville

Pour le champ « ville », afin de rendre la sélection de la ville plus user friendly il sera possible de faire une recherche textuelle avec autocomplétion.

Dernières annonces

Pour afficher les annonces récentes sur la page d'accueil, je propose un carrousel affichant douze annonces répartis sur trois pages.

Agences les mieux notées

De la même façon, pour afficher les agences les mieux notées je propose un carrousel affichant douze annonces répartis sur trois pages.

Page de liste

Recherche

Pour une recherche avancée sur la page de liste, je propose une barre latérale comportant ces différents champs :

- Type de bien
- Prix max / prix min sous la forme d'un slider
- Secteur
  - Ville
  - Rayon
- Pièces
  - Pièces à vivre
  - Chambres
- Surface
  - Habitable
  - Extérieur
- Proximité sous la forme d'étiquettes à cocher
- Tags sous la forme d'étiquettes à cocher

Sur desktop on peut réduire la barre de recherche.

Back-office

Utilisateur

L'utilisateur inscrit aura accès à un back-office lui permettant :

- d'éditer ses informations,
- d'ajouter des annonces,
- de voir ses annonces favorites

Administrateur

L'administrateur lui aura accès à un « dashboard », un back-office plus complet que celui de l'utilisateur, ce qui lui permettra en autre de gérer :

- les utilisateurs
- les agences
- les annonces les différentes catégories…

Spécifications techniques

Back-end

Afin de pouvoir distribuer les ressources à plusieurs sites internets, j'ai fait le choix d'utiliser une API. Une API est un serveur sur lequel un site internet sans interface graphique permet de gérer des ressources qui ici sont des biens immobiliers.

API REST

Conception et documentation de l'api

Pour concevoir et documenter l'API j'utilise l'application web « Swagger ».

Cet outil me permet de définir les différents endpoints qui seront utilisés, à la fois dans la partie front-end que dans la partie back-end. Cela me permet également de réfléchir à la forme des requêtes dont l'application va avoir besoin.

Swagger est donc un outil qui me permets de réfléchir à la structure de l'API, qu'il s'agisse du nom des routes, des paramètres que prend la requête, de sa méthode ainsi que des différentes réponses qu'elle peut engendrer et ce, sans avoir à coder quoi que ce soit.

Bien réfléchir sont API Swagger présente au moins deux avantages majeurs :

- Premièrement, cela permet d'avoir un modèle sur lequel se baser pour développer son API, car il suffit de reproduire dans son code ce que l'on a déjà défini dans Swagger.
- Deuxièmement, Swagger fait office de documentation et offre la possibilité de tester les requêtes directement sur leurs serveurs. On peut donc les tester avec un logiciel comme Insomnia ou PostMan ou même, directement en code. Cela permet donc de développer la partie front-end d'une application sans que le back-end n'ait encore été développé.

Si nous avions été plusieurs à collaborer sur ce projet, nous aurions pu nous répartir le travail et ainsi développer l'application en parallèle sans que le front-end ne soit dépendant du back-end. Je suis seul à travailler sur le projet et je décide de commencer par la partie back-end.

Inscription et connexion

Pour gérer les accès à l'API, j'ai choisi d'utiliser le protocole **Oauth 2.0**. Cela permettra dans le futur de développer facilement des applications mobiles natives et permettra également aux agences immobilières de faire développer leur site personnel par l'entreprise qu'elle souhaite tout en utilisant l'API comme logiciel métier.

Pour mettre en place le protocole Oauth 2.0, j'utilise le paquet Laravel **Passport**.

Base de données

Afin de stocker les données de l'application j'ai choisis d'utiliser MariaDB principalement, car j'ai l'habitude d'utiliser ce serveur de base de données.

Choix du framework

Pour développer la partie back-end de l'application, je décide d'utiliser le framework PHP Laravel 9, car PHP est un langage que je connais relativement bien et Laravel est un framework que nous avons utilisé au cours de la formation.

J'avais d'abord pensé utilisé le framework lumen qui est un dérivé de « Laravel » et est présenté comme étant plus léger et plus rapide que ce dernier, mais il se trouve que la documentation de Lumen encourage l'utilisation de Laravel. En effets, aux vues des progrès en termes de vitesse, engendrés par les dernières versions de PHP, les développeurs de Lumen recommandent de démarrer un nouveau projet avec Laravel en utilisant le paquet Octane.

Laravel est un framework qui utilise le modèle de conception MVC (Modèle, Vue, Contrôleur) permettant de séparer les responsabilités en découpant l'application en couches :

- Modèle : composant qui gère les données, permet de récupérer les informations dans la base de données, de les organiser pour qu'elles puissent ensuite être traitées par le contrôleur.
- Vue : composant graphique de l'interface qui permet de présenter les données du modèle à l'utilisateur.
- Contrôleur : composant responsable des prises de décision, gère la logique du code, il est l'intermédiaire entre le modèle et la vue.

Cette architecture permet d'avoir une application modulable et rapide, ce qui rend le code plus facile à maintenir.

(Schéma MVC en annexe)

Conteneurisation

La conteneurisation réduit le gaspillage des ressources, car chaque conteneur ne renferme que l'application et les fichiers binaires ou bibliothèques associés. On utilise donc le même système d'exploitation (OS) hôte pour plusieurs conteneurs, au lieu d'installer un OS (et d'en acheter la licence) pour chaque VM invitée. Ce procédé est souvent appelé virtualisation au niveau du système d'exploitation. (schéma en annexe)

Afin d'adopter une approche dite **DevOps** et ainsi faciliter le futur déploiement de l'API, je cherche une solution simple et efficace pour conteneuriser mon application, deux choix s'offre à moi :

- Utiliser docker-compose pour créer et gérer moi-même mes conteneurs.
- Utiliser le paquet Laravel **Sail**.

Pour plus de simplicité je choisis d'utiliser Sail. Ce paquet, fournit par Laravel permet de mettre en place facilement un environnement conteneurisé afin de développer son application. Toutes fois, la documentation de Laravel déconseille de l'utiliser pour le déploiement et recommande d'utiliser une solution plus sécurisée.

J'ai choisi de conteneuriser mon API pour deux raisons :

Premièrement, cela permet de s'affranchir des différences éventuelles entre l'environnement de développement et l'environnement de production, puisque tout ce dont l'application à besoin pour fonctionner, comme les librairies et autres dépendances sont comprises dans le conteneur. Ainsi l'application conteneurisée peut fonctionner quel que soit le système hôte et ses versions logiciels (pour peu que celui-ci supporte la conteneurisation).

Pour la même raison, en cas de travail collaboratif, cela permet aux différents développeurs de travailler avec le système d'exploitation de leur choix.

*Deuxièmement, cela permet d'apporter plus de **scalabilité** et de **résilience** à l'application, car une fois conteneurisée il est possible de la déployer à l'aide d'un orchestrateur de conteneurs comme **Kubernetes** ou **Docker Swarm**. L'intérêt de ces deux technologies est de pouvoir déployer plusieurs instances de l'application sur des serveurs séparés, le chef d'orchestre décide ensuite de l'instance qui prendra en charge la requête. Si jamais une des instances est surchargée ou hors-service, les autres continuent à fonctionner et assurent la continuité des services (schéma en annexe : Load balancing).*

Front-end

Choix du framework

Pour développer la partie front-end de mon application je souhaite utiliser un framework **JavaScript** pour apporter de la rapidité et du dynamisme à mon application. Je retiens trois frameworks différents :

- Vue .js
- Nuxt.js
- Svelte kit
- React

	Vue3	Nuxt.js	Svelte kit	React
Avantages	- Communauté importante	- SSR	- Moins verbeux que Vue3 - DOM classique	- Communauté importante - Application mobile native - SSR
Inconvénients	- SSR non prit en charge nativement	- DOM virtuel Plus complexe que vue.js	- Communauté restreinte	- Documentation pauvre

Le SSR (Server Side Rendering) est un gros plus pour ce genre d'application, car il permet de fournir au navigateur une page web HTML/CSS complète, ce qui permet au site d'être référencé par tous les moteurs de recherche. Sans SSR, peu de moteurs de recherche peuvent indexer une application web JavaScript (Google les indexe).

J'ai choisi d'utiliser le framework JavaScript **Vue3**. J'ai choisi ce framework principalement, car je l'ai étudié en cours.

module bundler

Vue3 utilise de base **webpack** pour compiler les différents modules de l'application, mais je préfère utiliser **vite**. Contrairement à webpack qui compile systématiquement tous les modules de l'application, vite lui ne compile que les modules utilisés par la page courante et permet également de charger les modules à chaud. Cela permet de gagner du temps lors du développement.

Framework CSS

Afin de faciliter le développement du CSS (Cascading Style Sheets), mon attention se porte vers deux frameworks différents :

- Bootstrap, qui est une collection d'outils développer par twitter et pèse environ 300kb. Bootstrap est un ensemble qui contient des codes HTML et CSS, des formulaires, boutons, outils de navigation et autres éléments interactifs, ainsi que des extensions JavaScript en option.
- TailWind.css est un framework de classes utilitaires qui pèse 30kb. Il permet de développer des interfaces de façons « inline » en utilisant uniquement des classes.

Connaissant quelque peu Bootstrap, je décide d'utiliser TailWind pour sa légèreté et surtout pour découvrir et me faire ma propre idée cette techno.

Outils

Pour développer l’application il faudra de préférence utiliser des logiciels opensource et utiliser les logiciels dont le fonctionnement a été étudié au cours de la formation. Le choix a été fait d’utiliser

ces logiciels :

- Xubuntu 22.04 pour le système d’exploitation sur l’ordinateur utilisé pendant le développement.
- Lunacy pour maquetter l’interface utilisateur.
- PHPStorm pour le développement en PHP.
- WebStorm pour le développement html, css et javascript.
- Vue.js pour le développement de l’interface utilisateur.
- Firefox et Google Chrome pour réaliser des tests sur desktop comme sur mobile.
- Draw.io pour schématiser la base de données.
- MariaDB comme serveur de base de données.
- Dbeaver pour accéder à la base de données.
- GitCli et GitLab pour versionner l’application.
- Swagger pour concevoir et documenter l’API.
- Laravel 9 pour développer l’application et organiser le code.

Réalisations

Développement de l’API

Conception et documentation de l’api

Mise en place d’une requête

Pour gérer l’inscription je définis une route « /register » qui utilise la méthode « POST » et qui requière un contenu sous forme de « JSON », ce qui dans swagger donne :

```
/register:
    post:
      tags:
        - developers
      summary: Register
      operationId: register
      description: |
        Route that allows user to create account
      requestBody:
        required: true
        content:
          application/json:
```

Je peux ensuite définir le corps de la requête, il sera composé de quatre attribue :

- username
- email
- password
- password_c

Swagger permet de définir le type du champ ainsi que sa longueur maximale, ce qui pour le nom d’utilisateur donne :

```
properties:
    username:
        description: Username choose by user
        type: string
        format: string
        minimum: 1
        maximum: 50
```

Pour finir, je rends ces champs obligatoires en ajoutant ces quelques lignes :

```
required:
    - username
    - email
    - password
    - password_c
```

Mise en place de la réponse

Après avoir défini la forme et les paramètres de la requête, je définis deux types de réponses.

La première est une réponse au code 201, ce qui correspond à une confirmation de création de ressource. Cette réponse est envoyée par le serveur au cas où le processus de validation et de création de l'utilisateur s'est bien passé. En plus du statut 201, la réponse formatée en JSON est composée des attributs suivant :

- « success » qui est un booléen ayant la valeur « true ».
- User qui est un objet composé de :
  - l'id de l'utilisateur,
  - du username qu'il a choisi,
  - de son type de rôle (par défaut égal à « user »).
- « token » qui correspond à un **bearer token** au format **JWT**

Dans swagger je définis donc le type et la forme de la réponse :

```
responses:
        '201':
          description: account created
          content:
            application/json:
              schema:
                    type: object
```

Ensuite, je définis les attributs, ce qui pour l'utilisateur donne :

```
properties:
    user:
        type: object
        properties:
            id:
                type: integer
                format: integer
                example: 1
            username:
                type: string
                format: string
                example: "JohnyD"
            role:
                type: string
                format: string
                example: "user"
```

Ce qui donne cette réponse :

```
{
  "success": true,
  "user": {
    "id": 1,
    "username": "JohnyD",
    "role": "user"
  },
  "token": "eyJ0eXAiOi […] QiGu4O8BfqttAzY"
}
```

La seconde renvoi un code 400, ce qui correspond à une erreur de validation. Le serveur envoi cette réponse si jamais :

- un des champs requit est manquant
- si le username ou l'adresse mail renseigné existe déjà
- si le mot de passe est trop court ou si la confirmation ne correspond pas au mot de passe renseigné

Comme pour la réponse précédente celle-ci est formatée en JSON et est composée de plusieurs attributs, ces attributs sont :

- success qui est un booléen ayant la valeur « false »
- message qui est une chaîne de caractères contenant un message relatif à l'erreur survenue
- data qui est un objet composé de tableaux contenant les différents messages d'erreur


```
{
  "success": false,
  "message": "erreur de validation",
  "data": {
    "password": [
      [
        "Le mot de passe doit comporter au moins 8 caractères",
        "Le mot de passe est obligatoire",
        "La confirmation du mot de passe ne correspond pas"
      ]
    ],
    "username": [
      [
        "Le nom d'utilisateur est obligatoire",
        "le nom d'utilisateur ne doit pas excéder 50 caractères",
        "Le nom d'utilisateur ':input' existe déjà"
      ]
    ],
    "email": [
      [
        "Veuillez entrer une adresse mail valide",
        "L'adresse mail est obligatoire"
      ]
    ]
  }
}
```


Les messages d'erreurs de cette réponse pourront ensuite être affichés tel-quel dans la partie front-end.

Conception de la base de données

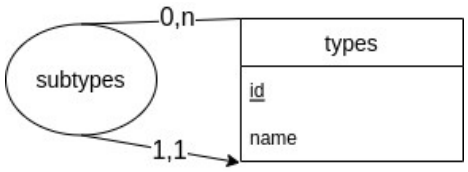
Pour concevoir la base de données, je me sers du logiciel draw.io afin de schématiser les différentes tables et leurs relations (MCD en annexe).

Table récursive

Pour gérer les biens immobiliers, mon application web à besoin de leurs attribuer des types. Mais ces types peuvent avoir des sous types, par exemple :

- Maison
  - Maison de ville
  - Maison de campagne
- Appartement
  - Studio
  - Triplex

Pour répondre à cette problématique, plutôt que de créer une table distincte pour le type et le sous type, j'utilise une table récursive.



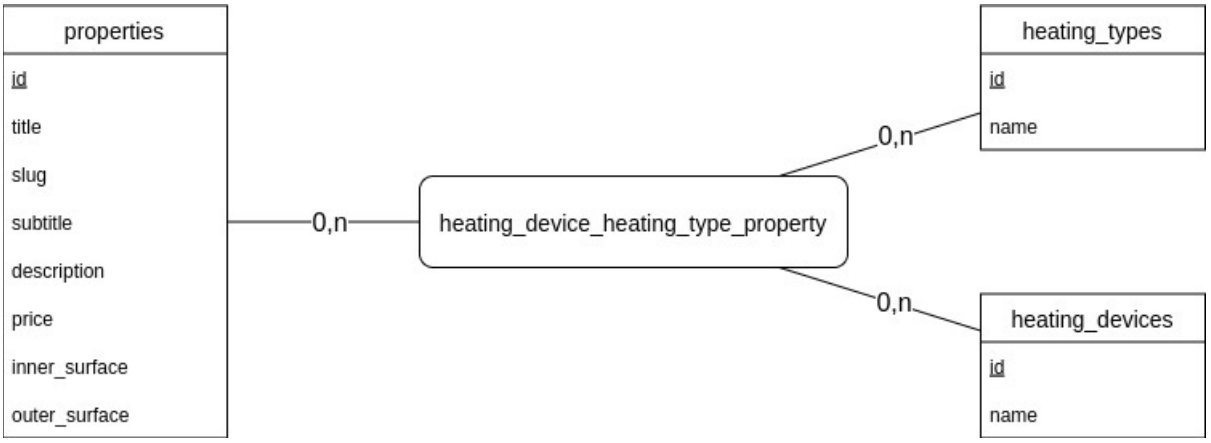
Une table récursive dispose d'une clé étrangère qui référence directement la clé primaire d'un autre champs de la même table. Cela permet d'avoir un nombre infini de sous-type sans avoir besoin de créer d'autres tables.

Table pivot

Afin de gérer les systèmes de chauffage d'un bien immobilier mon application à besoin de deux chose :

- Le type de chauffage (ex : électrique, fioul, gaz, bois…)
- Le type d'appareil de chauffage (ex : au sol, radiateur, poêle…)

Ces informations pourraient être directement stockées dans la table du bien mais dans ce cas, impossible d'assigner plusieurs types de chauffage à un bien. Pour résoudre ce problème, j'utilise une table pivot qui fait le lien entre les trois tables :



Mise en place de la base de données

Pour créer la base de données, Laravel utilise des fichiers de migrations. Chaque fichier correspond à une table et permet de définir :

- le type du champ,
- le nom du champ,
- la valeur par défaut,
- le comportement en cas de suppression.

Je commence par créer mon fichier de migration à l'aide de l'interface de commande « Artisan » en respectant les conventions de nommage proposé par Laravel. En exécutant cette commande, Artisan va créer pour moi un fichier avec tout le **Boilerplate** nécessaire.

```
sail artisan make:migration create_properties_table
```

Champ simple

Je peux ensuite définir mes champs, par exemple pour le champ « subtitle » de la table properties, je le définis ainsi :

- Type du champ : string (Chaîne de caractère).
- Nom du champ : subtitle.
- Maximum de caractères : 50.
- Le champ n'est pas obligatoire.
- Valeur par défaut : chaîne de caractères vide.

```
$table->string('subtitle', 50)->nullable()->default('');
```

Clé étrangère

Pour définir une clé étrangère, je spécifie :

- Le nom du champ qui représente la clé étrangère, ici « user_id ».
- Le nom de la clé primaire dans l'autre tableau, ici « id ».
- Le nom de la table de la clé primaire, ici « users ».
- Le comportement en cas de suppression, ici « cascade ».

```
$table->foreign('user_id')->references('id')->on('users')->onDelete('cascade');
```

Une fois mes fichiers de migrations terminés, je me sers d'Artisan pour créer les tables dans le serveur de base de données :

```
sail artisan migrate
```

(Fichier de migration complet en annexe)

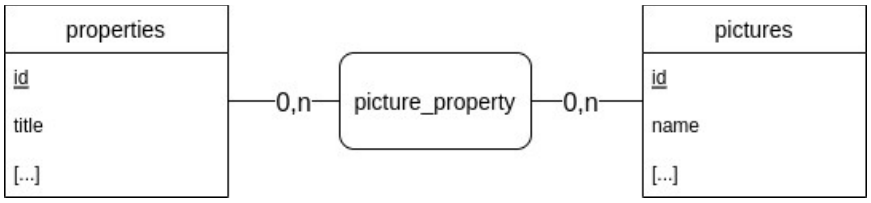
Mise en place des modèles

Les modèles

Les relations

Sur le MCD on peut voir que la table « Properties » entretient des relations avec d'autres tables, par exemple avec la table « Pictures ».

Dans cet exemple, un bien peut être associée à zéro ou plusieurs photos et une photos à zéro ou plusieurs biens. Ces deux tables sont reliées par une relation **Many to many** caractérisée par des cardinalités "0,n" des deux côtés, ce qui implique l'utilisation d'une table pivot.



Développement de l’interface utilisateur

Maquettage

Pour réaliser les maquettes desktop et mobile j’ai choisi d’utiliser le logiciel « Lunacy », principalement pour sa portabilité sur les systèmes unix.

Après installation je crée :

- un nouveau projet
- les différentes pages
- les différents composants, comme :
  - la barre de navigation
  - la barre de recherche de la page d’accueil
  - la barre de recherche de la page de liste
  - la vignette des annonces◦le carrousel des dernières annonces
  - le carrousel des agences les mieux notés

La création de composant me permet d’être plus efficace et de gagner du temps en réutilisant les composants sur différentes pages.

L’absence de contraintes dans le cahier des charges me permet de construire et de faire évoluer les différentes pages comme je le souhaite. Je m’inspire de différents sites immobiliers et de templates pour créer une interface simple, agréable et responsive.

Une fois les maquettes des trois pages principales (accueil, vente et détail) terminées, je crée une maquette de la page d’accueil pour mobile afin de visualiser l’aspect de la barre de navigation et du menu burger.

Les maquettes de toutes les pages non pas été faite, mais je dispose maintenant de suffisamment de matière pour commencer l’intégration.

Le design du dashboard n’étant pas prioritaire il se composera de la même barre de navigation, d’une sidebar et d’un tableau listant les différentes ressources.

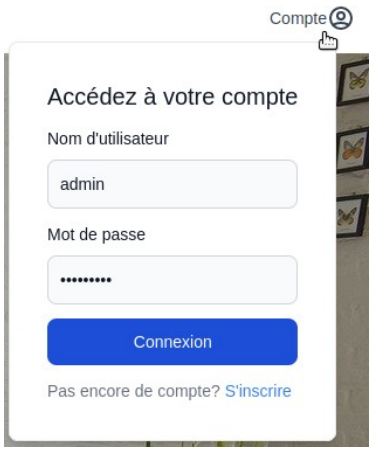
Inscription et connexion à l'API

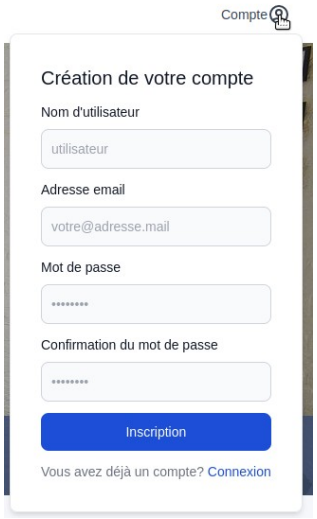
Pour permettre à un utilisateur de créer un compte puis de se connecter à l'API, je crée un **SFC** (Single File Component) nommé « LoginRegister.vue » dans lequel se trouve :

- La logique métier relative à l'inscription et la connexion.
- Le code HTML.
- Le code CSS.

HTML et CSS

Je décide de rester simple et je crée une fenêtre popup qui apparaît au clic sur le bouton « Compte ». Ce popup est composé de deux onglets, un pour la connexion et l'autre pour l'inscription.





Pour réaliser ces deux formulaires, j'ai simplement modifié un composant que j'ai récupéré sur la documentation de flowbite.

Par défaut, le formulaire de connexion s'affiche et il faut cliquer sur « S'inscrire » pour afficher le formulaire d'inscription. Pour rendre cela possible j'ai utilisé l'affichage conditionnel de Vue3 et une variable réactive.

En utilisant l'API de composition je définis un variable « registering » de type booléen ayant pour valeur faux.

```
const registering = ref(false)
```

L'affichage des deux formulaires dépend de la valeur de cette variable.

```
<form v-if="!registering">
        [...]
</form>
<form @submit.prevent="register" v-else class="space-y-3">
        [...]
</form>
```

La valeur de la variable change au clic sur un bouton.

```
<button @click="registering = !registering">Connexion</button>
```

Validation

Avant d'envoyer une requête à l'API, le client effectue une vérification des champs du formulaire. Pour ce faire j'ajoute un « event listener » qui, à la soumission du formulaire va appeler une méthode.

Dans le cas de la connexion, la méthode appelée est « login ».

```
<form @submit.prevent="login" v-if="!registering">
```

Cette méthode est une fonction asynchrone qui effectue des vérifications comme :

- Est-ce que le nom d'utilisateurs a été renseigné ?
- Est-ce que le mot de passe a été renseigné ?
- Est-ce que le mot de passe contient suffisamment de caractères ?

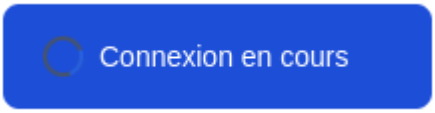
Exemple :

```
if (user.username.length === 0) {
  loginError.username.push('Nom d\'utilisateur obligatoire')
  isValidate = false
}
```

Si la validation réussie, la méthode « login » du **store** « user » est appelée.

```
if (isValidate) {
  const response = await userStore.login(user)
```

En attendant la réponse de l'API, le bouton change pour informer l'utilisateur que quelque chose est en train de se passer et qu'il faut patienter.



Sinon, un message correspondant à l'erreur s'affiche sur le formulaire.

Requête à l'API

Après validation, la méthode « login » du store « user » appelle à son tour une méthode « login » qui se trouve dans un service dédié à la connexion et à l'inscription nommée « auth.service.js ». Ce service s'occupe de faire la requête à l'API en utilisant la bibliothèque **axios**.

Cette méthode « login » est une fonction asynchrone qui renvoie une promesse.

Séparer ainsi le code en couches me permet d'apporter de la souplesse à l'application et de la rendre plus facilement maintenable et modifiable. En effet, si un jour je décide d'utiliser une autre bibliothèque qu'axios, je n'aurais à modifier que le fichier « auth.service.js ».

Si la requête de connexion aboutie, alors la réponse est envoyée au store qui se charge ensuite de stocker dans le stockage local :

- Le nom d'utilisateur.
- Le rôle de l'utilisateur.
- Le jeton d'accès.

Accès aux ressources

Afin d'afficher sur le site les différentes informations provenant de l'API, je mets en place deux « services » que j'appelle « auth-header.js » et « appimmo.service.js ».

auth-header.js

Ce service me permet de récupérer le token présent dans le stockage local pour créer les **headers** nécessaire aux requêtes protégées.

```
return { Authorization: 'Bearer ' + token, 'Content-Type': 'application/json' };
```

appimmo.service.js

Ce service s'occupe uniquement des appels axios et comporte quatre méthodes :

- get,
- store,
- update,
- destroy.

Exemple avec la méthode get :

```
async get(url) {
   try {
      const response = await axios.get(API_URL + url,
         {
            headers: authHeader()
         })

      if (response.status === 200) {
         return response
      }

   } catch (err) {
      console.log(err);
   }
}
```

Chaque une de ces méthodes utilisent les headers du service « auth-header.js ».

Responsive design

Afin de permettre au site de s'afficher aussi bien sur desktop que sur mobile, je ne me sers pas directement de **media queries** mais de classes tailwind.

En effet, grâce à tailwind tout le CSS est assigné aux éléments html de façons **inline**. Il est possible d'assigner une classe à un élément selon un **breakpoint** grâce à un préfixe comme :

- sm : (640px),
- md : (768px),
- lg :(1024px),
- xl : (1280px),
- 2xl : (1536px).

Pour rendre un élément responsive tout en respectant le principe du **mobile first** il suffit alors de lui assigner une classe sans préfixe pour l'affichage mobile, puis une autre classe pour un écran plus grand et encore une autre classe pour un très grand écran.

Par exemple pour la barre de navigation :

```
<nav class="fixed bottom-0 left-0 right-0 md:bottom-auto md:top-0 md:left-0 md:right-0
border-gray-200 px-2 sm:px-4 py-2.5">
```

Ici, les classes « bottom-0, left-0 et right-0 » auront pour effet de placer la barre de navigation en bas de l'écran sur un écran de téléphone, alors que les classes « md:bottom-auto, md :top-0, md:left-0 et md:right-0 » auront pour effet de la placer en haut de l'écran.

Jeu d’essai

Tests

Afin de m’assurer que les différentes fonctionnalités de mon application fonctionnent correctement tout au long du développement, j’ai mis en place des tests unitaires et fonctionnels. Ces tests sont des scripts qui ont pour but de réaliser des actions afin de s’assurer qu’une action ou une fonctionnalité de l’application fonctionne correctement.

Le framework Laravel permet de mettre en place facilement ce genre de tests.

Tests fonctionnels

J’ai mis en place un test qui vise à assurer que l’inscription sur la plateforme est pleinement fonctionnelle. Voici les différents scénarios et résultat que l’application gère :

Scénario	Valeurs saisies	Résultat et réponse
Ajout d’un utilisateur en respectant la forme requise. Le nombre d’utilisateur est compté avant et après.	• Nom d’utilisateur : usertest. • Email : email unique généré automatiquement. • Mot de passe : 123456789 • Confirmation : 123456789	Le test est validé, s’il y a bien un utilisateur en plus dans la base de données après l’exécution du test. • success : true • user: ◦ id ◦ username ◦ role ◦ fullname • Token
Ajout d’un utilisateur sans paramètres.	Aucune valeur.	Le test est validé si le statut de la réponse est égal à 400. • success : false • message : Validation errors, • username: Le nom d’utilisateur est obligatoire • email: Veuillez entrer une adresse mail valide, l’adresse mail est obligatoire. • password : Le mot de passe est obligatoire.
Ajout d’un utilisateur sans nom d’utilisateur.	• Nom d’utilisateur : aucune valeur. • Email : email unique généré automatiquement. • Mot de passe : 123456789. • Confirmation : 123456789.	Le test est validé si le statut de la réponse est égal à 400. • success : false • message : Validation errors, • username: Le nom d’utilisateur est obligatoire

Ajout d'un utilisateur sans mot de passe	• Nom d'utilisateur : usertest. • Email : email unique généré automatiquement. • Mot de passe : aucune valeur. • Confirmation : aucune valeur.	Le test est validé si le statut de la réponse est égal à 400. • success : false • message : Validation errors, • password : Le mot de passe est obligatoire.
Ajout d'un utilisateur avec un mot de passe trop court.	• Nom d'utilisateur : usertest. • Email : email unique généré automatiquement. • Mot de passe : 123. Confirmation : 123.	Le test est validé si le statut de la réponse est égal à 400. • success : false • message : Validation errors, • password : Le mot de passe doit comporter au moins 8 caractères.
Ajout d'un utilisateur sans mot de passe de confirmation.	• Nom d'utilisateur : usertest. • Email : email unique généré automatiquement. • Mot de passe : 123456789. Confirmation : aucune valeur.	Le test est validé si le statut de la réponse est égal à 400. • success : false • message : Validation errors, • password : La confirmation du mot de passe ne correspond pas.
Ajout d'un utilisateur avec un mot de passe de confirmation différent.	• Nom d'utilisateur : usertest. • Email : email unique généré automatiquement. • Mot de passe : 123456789. Confirmation : 987654321.	
Ajout d'un utilisateur avec un nom d'utilisateur déjà existant.	• Nom d'utilisateur : admin. • Email : email unique généré automatiquement. • Mot de passe : 123456789 • Confirmation : 123456789	Le test est validé si le statut de la réponse est égal à 400. • success : false • message : Validation errors, • username: Le nom d'utilisateur « admin » existe déjà.
Ajout d'un utilisateur avec une adresse e-mail déjà existante.	• Nom d'utilisateur : usertest. • Email : admin@admin.fr. • Mot de passe : 123456789 • Confirmation : 123456789	Le test est validé si le statut de la réponse est égal à 400. • success : false • message : Validation errors, • email: L'adresse e-mail « admin@admin.f » existe déjà.

(Exemple de fichier de test et résultat de l'exécution des tests en annexe).

Ce test s'exécute uniquement sur la partie back-end de l'application. Si toutes les étapes sont validées alors cela signifie que l'inscription est pleinement fonctionnelle.

Il sera nécessaire de mettre en place des tests sur la partie front-end en utilisant **Vitest**.

Tests unitaires

Les tests unitaires permettent de s'assurer qu'une petite partie de l'application fonctionne comme on le souhaite. Je m'en sers pour tester les différentes actions liées aux ressources.

Par exemple pour la classe « property » qui correspond à un bien immobilier, j'ai mis en place un test unitaire qui me permet de savoir si les différentes méthodes du contrôleur fonctionnent, comme :

- La création.
- Le listage.
- La visualisation.
- La modification.
- La suppression.

(exemple de résultat de l'exécution des tests en annexe.)

Test de charge

Pour tester les capacités de la base de données, j'utilise les « **factories** » et les « **seeders** » de Laravel.

Les « factories » (usines) sont des classes qui permettent de définir la valeur des différents attribue d'un modèle. Grâce à la bibliothèque « **faker** », il est possible de créer des instances du même modèle avec des valeurs différentes.

J'utilise ces factories dans les seeders. Cela me permet de peupler la base de données avec autant d'entrées que je le souhaite, Ainsi j'ai pu tester l'application avec 3000 biens immobiliers.

Veille

Back-end

Framework

Avant de commencer le développement de l'API, j'ai voulu trouver un framework PHP adapté au développement d'une API. J'ai d'abord trouvé « Lumen » qui est un framework PHP dont les principaux avantages mis en avant sont la légèreté et la rapidité.

Étant basé sur Laravel et aux vues des connaissances dont je dispose avec ce framework je pense tout d'abord que Lumen est tout à fait adapté à mon projet. Cependant, en parcourant la documentation je suis confronté à une recommandation encouragent à commencer de nouveau projet directement avec Laravel.

En effet, cette recommandation met en avant les progrès, en termes de performance réaliser par les dernières versions de PHP et recommande d'utiliser Laravel avec le paquet « Octane ».

Je cherche donc à comparer les trois solutions, Lumen, Laravel et Laravel / Octane. Sur le site « Medium.com », je trouve un article ou l'auteur à effectuer des benchmarks et où il démontre clairement que le couple Laravel / Octane peut servir plus de requêtes à la seconde que les deux autres solutions. (Lien vers l'article dans l'annexe)

Aux vues de ces informations je décide d'utiliser le couple Laravel / Octane.

Front-end

store

Avant de commencer le développement de la partie front-end, je savais qu'il me faudrait mettre en place un système de « store » qui me permettra d'accéder à différentes valeurs et méthodes dans différentes pages ou composants et ce en conservant l'état des variables.

Ayant suivi quelque cours sur Vue3 pendant ma formation, j'ai découvert le concept des stores et vu les bases de VueX, cependant il se trouve que sur la documentation de vue3 les développeurs conseil l'utilisation d'un autre store qui se nomme « pinia ».

Travail de recherche

Ayant fait le choix d'utiliser le framework CSS « Tailwind CSS », j'ai voulu utiliser la bibliothèque JavaScript flowbite, qui met à disposition gratuitement bon nombre de composants utilisant tailwind permettant d'accélérer et faciliter le développement. Cependant je me suis heurté à un mur, les dit composant ne fonctionne qu'à l'intérieur du fichier app.js de vue3. Je me mets donc à la recherche d'une solution pour faire fonctionner flowbite correctement.

Je parcours sur le github de flowbite les différentes « issues » (erreurs) rapportées par les utilisateurs, jusqu'à trouver l'issue « JavaScript doesn't work in Vue3 ». La description du problème correspond à ce que je rencontre, je cherche alors si une solution est proposée par quelqu'un, malheureusement sans succès.

Par contre une des réponses attire mon attention :

« The way the flowbite.bundle.js works is that it looks for the data attributes and then applies event listeners, this means that the JS needs to have an already rendered DOM to make it work.

We are aware of the compatibility issues regarding virtual DOM based libraries and frameworks, we're trying to find solutions.

Please take into account that the other non-interactive components (ie. buttons, alerts, badges) still work regardless of the JS file. »

L'auteur de cette réponse datant du 02/12/2021 m'apprend que le fonctionnement de flowbite, qui se base sur la détection des attributs dans le DOM (Document Object Model) pour ensuite créer des event listeners a besoin d'un DOM déjà rendu, c'est-à-dire un DOM classique. Il précise que les développeurs de flowbite cherche une solution et ajoute que certain composants « non interactifs » comme les boutons, les alertes et les badges fonctionnent très bien. Cette réponse date effectivement de plusieurs moi, mais visiblement aucune solution ou correctif n'a encore été apporté.

Conclusion

Il est évident que des erreurs ont été commises lors du choix des technologies à utiliser, notamment avec le choix du framework front-end Vue3 dont la fonctionnalité de rendu côté serveur (SSR) n'est pas pris en charge nativement et la bibliothèque de composant flowbite, dont le javaScript ne fonctionne pas avec les DOM virtuels. Dans le futur je ferais plus attention qu'en aux choix des technologies à utiliser.

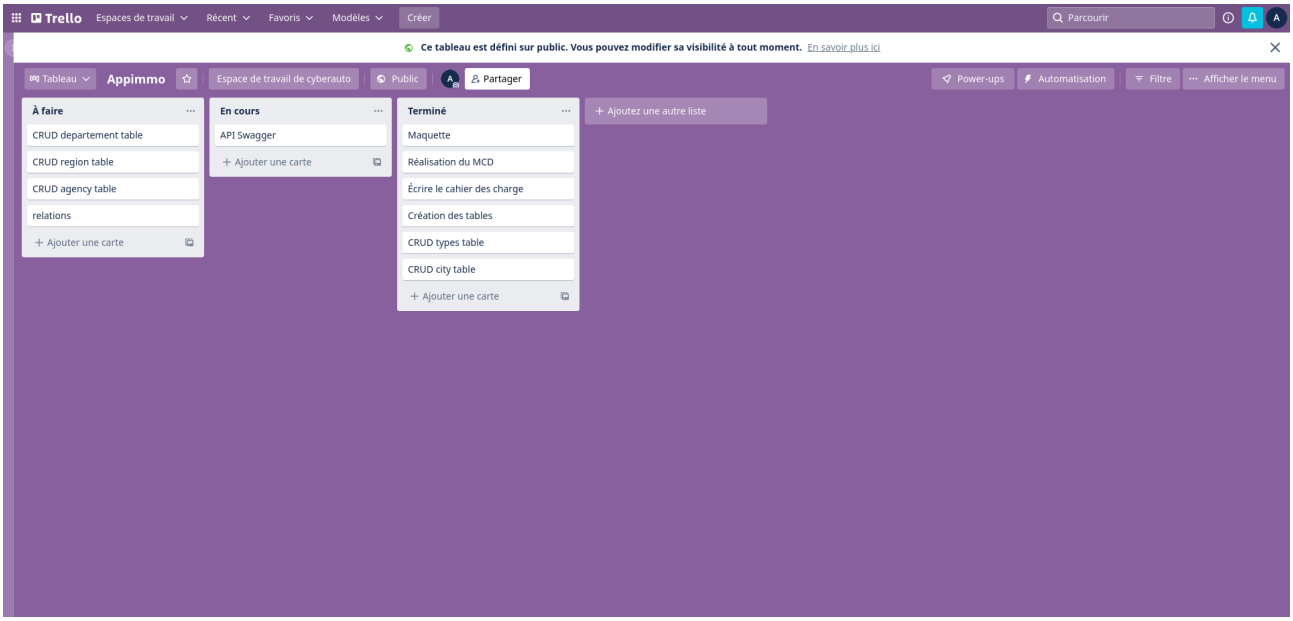
J'ai aussi commis des erreurs lors de la conception de la base de données, notamment avec la gestion des photos. En effet, l'utilisation d'une table pivot entre la table properties et pictures ne me semble plus pertinente. Une relation simple avec l'utilisation d'une clé étrangère dans la table pictures aurait été plus adaptée. Cela me montre à quel point la phase de conception est importante et qu'il ne faut surtout pas la négliger, aux risques de perdre du temps lors du développement.

Ce projet, relativement complexe m'aura permis d'approfondir mes connaissances dans les frameworks Laravel et Vue.js, tant au niveau de la programmation pure avec les langages PHP et JavaScript, qu'au niveau de l'architecture d'une application. J'aurais souhaité pouvoir y consacrer plus de temps, car c'est pour moi, un projet qui aurait pu se dérouler sur toute une année.

Annexes

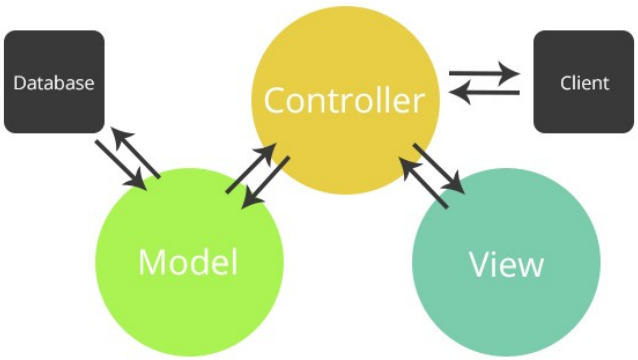
Table des matières

Trello

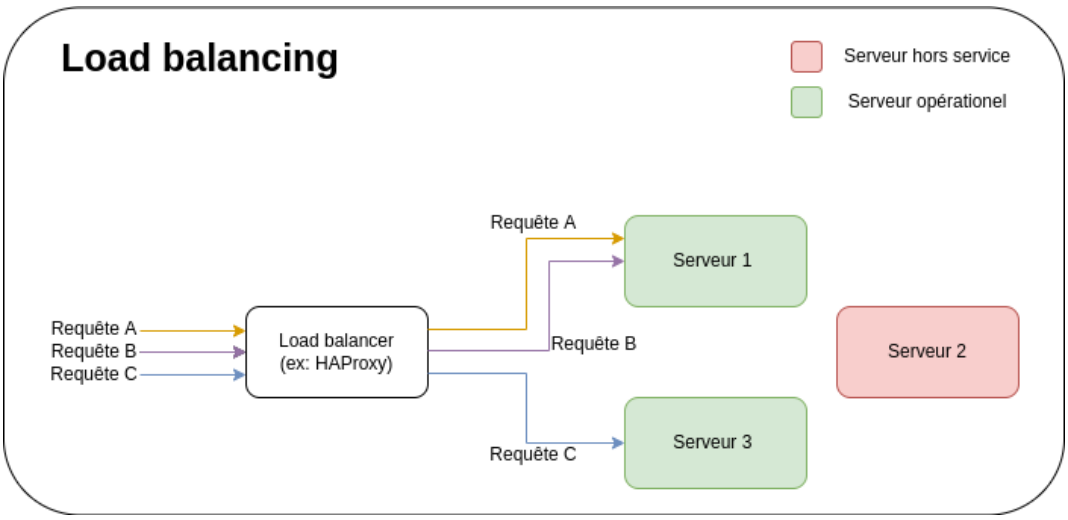


MVC

Modèle d’architecture proposé par le framework PHP Laravel.



Load balancing



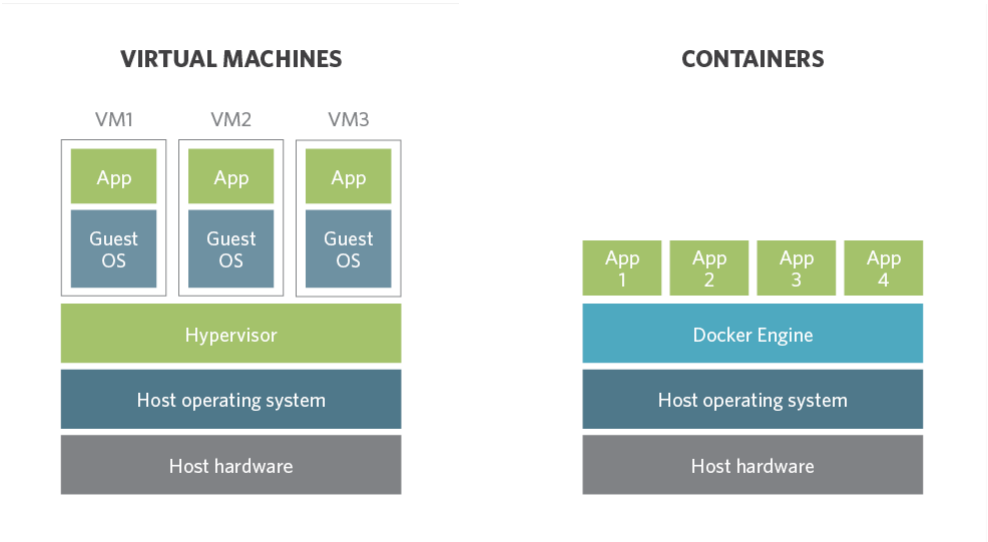
Documentation de l'API REST

Lien vers la documentation Swagger : https://app.swaggerhub.com/apis/C4531/appimmo/1.0.0

Laravel / Octane

Lien vers l'article : https://medium.com/geekculture/speed-up-your-laravel-projects-using-laravel-octane-swoole-server-a39f4a7fa889

Conteneurisation



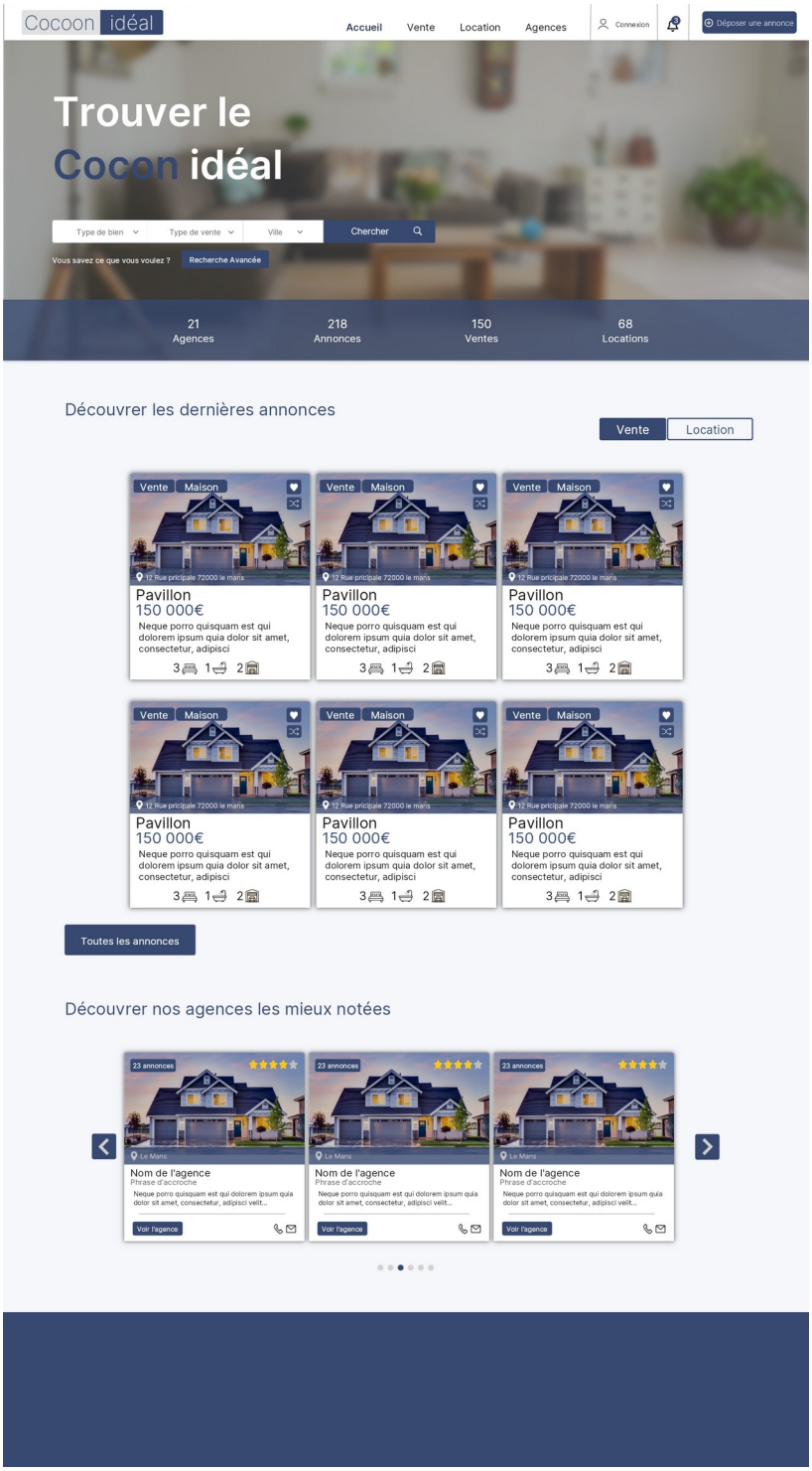
Migration Laravel

Exemple d'un fichier de migration

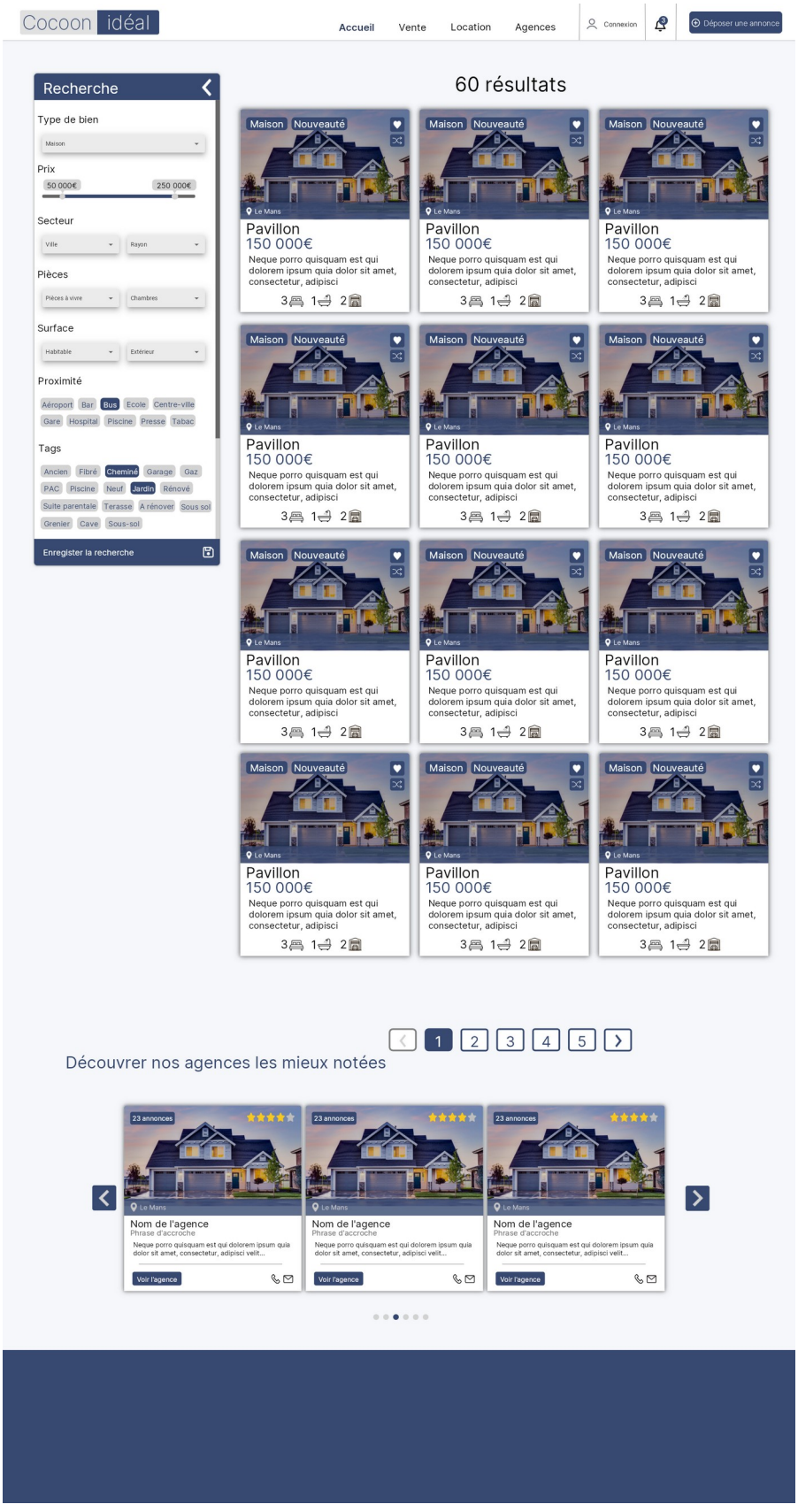
```php
use Illuminate\Database\Migrations\Migration;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Support\Facades\Schema;

return new class extends Migration
{
    public function up()
    {
        Schema::create('properties', function (Blueprint $table) {
            $table->id();
            $table->string('title', 50);
            $table->string('slug', 80)->nullable()->default('');
            $table->string('subtitle', 50)->nullable()->default('');
            $table->text('description')->nullable()->default('');
            $table->string('price')->nullable()->default('');
            $table->string('inner_surface')->nullable()->default(null);
            $table->string('outer_surface')->nullable()->default(null);
            $table->unsignedBigInteger('category_id')->nullable();
            $table->unsignedBigInteger('type_id')->nullable();
            $table->unsignedBigInteger('status_id')->nullable();
            $table->unsignedBigInteger('agency_id')->nullable();
            $table->unsignedBigInteger('user_id');
            $table->unsignedBigInteger('address_id')->nullable();
            $table->timestamps();

            $table->foreign('category_id')->references('id')->on('categories');
            $table->foreign('type_id')->references('id')->on('types');
            $table->foreign('status_id')->references('id')->on('statuses');
            $table->foreign('agency_id')->references('id')->on('agencies');
            $table->foreign('user_id')->references('id')->on('users')->onDelete('cascade');
            $table->foreign('address_id')->references('id')->on('addresses');
        });
    }

    /**
     * Reverse the migrations.
     *
     * @return void
     */
    public function down()
    {
        Schema::dropIfExists('properties');
    }
};
```

Maquettes

Accueil

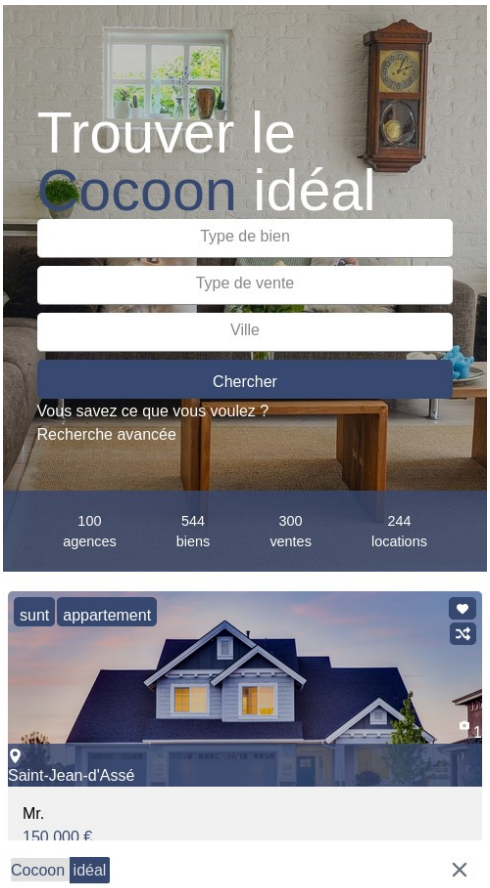


Page de liste



Page de détail

Exemple de maquette mobile



Jeu d’essai

Tests fonctionnels

Résultat du test

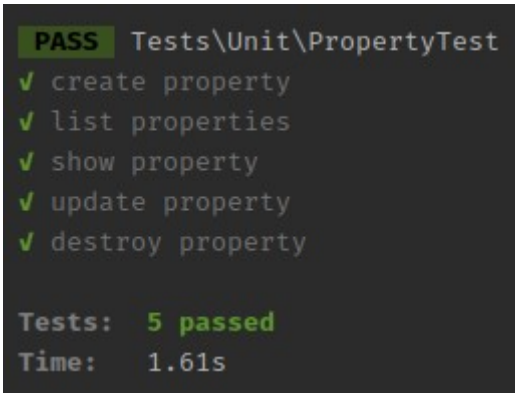
```
 PASS  Tests\Feature\RegistrationTest
✓ registration is valid
✓ registration without parameters
✓ registration without username
✓ registration without email
✓ registration without password
✓ registration without main password
✓ registration without password confirmation
✓ registration with different password confirmation
✓ registration with short password
✓ registration with existing username
✓ registration with existing email

Tests:  11 passed
Time:   1.13s
```

Exemple d’une fonction de test

```php
public function test_registration_with_existing_username()
{
    $faker = Factory::create();
    $email = $faker->unique()->email;
    $response = $this->post('/api/register',
        [
            "username" => "admin",
            "email" => $email,
            "password" => "123456789",
            "password_c" => "12345678",
        ]
    );
    $response
        ->assertStatus(400)
        ->assertJson([
                    'success' => false,
                    'message' => 'Validation errors',
                    'data' => [
                        'password' => ['La confirmation du mot de passe ne correspond pas']
                    ]
                ]);
}
```

Test Unitaire

Résultat du test



Exemple d’une fonction de test

```php
public function test_update_property()
{
    $title = "updated title";
    $this->seed();

    $property = Property::find(1);
    $property->title = $title;
    $property->save();

    $updatedProperty = Property::find(1);
    assertEquals($updatedProperty->title, $title);
}
```

Test de charge

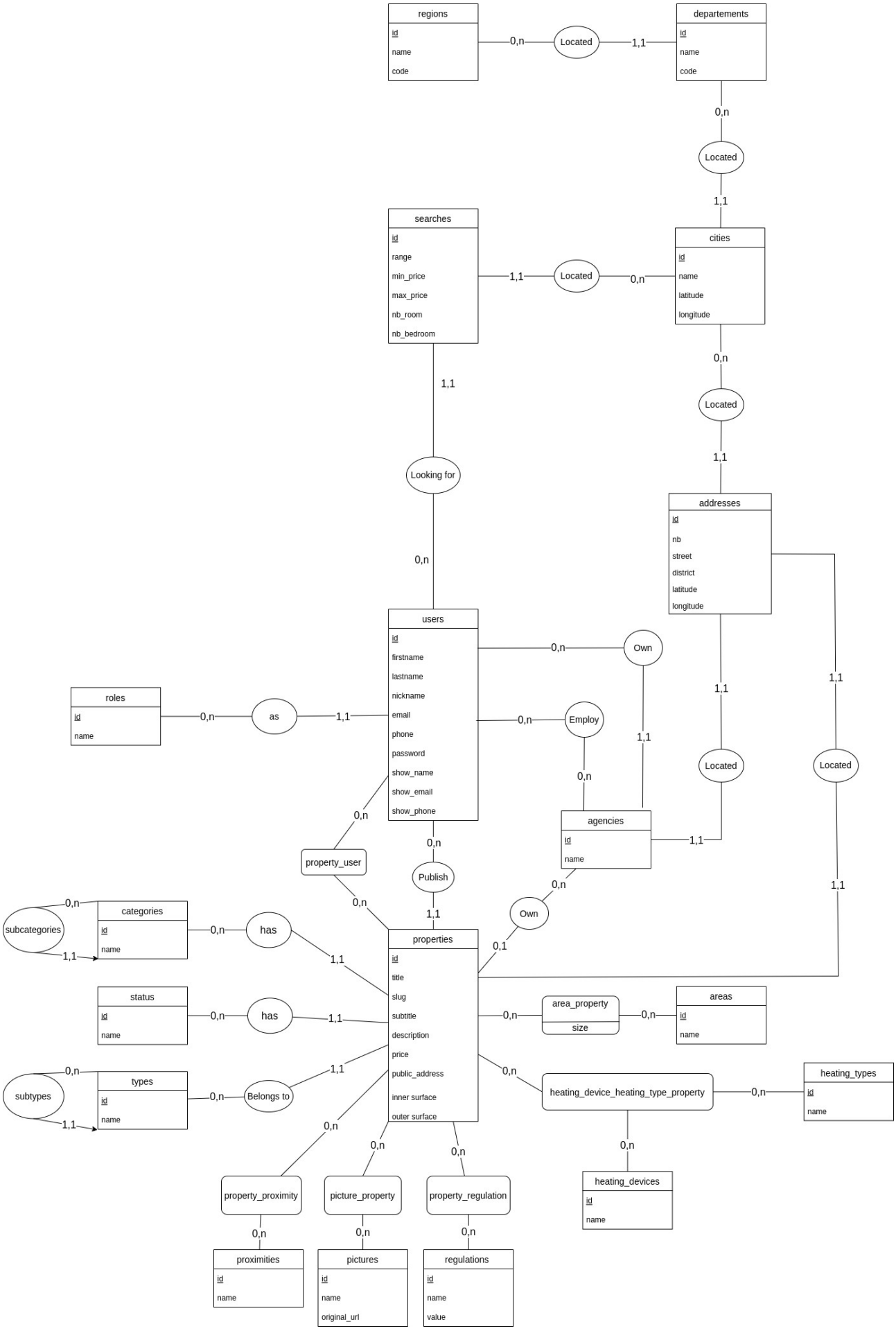
Exemple de factory

```
public function definition()
{
    return [
        'title'=> $this->faker->title(),
        'subtitle'=> $this->faker->title(),
        'description'=> $this->faker->text(),
        'price'=> '150000',
        'inner_surface'=> $this->faker->numberBetween(50,1000),
        'outer_surface'=> $this->faker->numberBetween(50,10000),
        'category_id'=> 1,
        'type_id'=> 1,
        'status_id'=> 1,
        'user_id'=> 1
    ];
}
```

Exemple de seeders

Ligne de code permettant de créer 100 biens en base de données en utilisant une factory.

```
Property::factory(100)->create();
```


regions
id
name
code
0,n
Located
1,1
departements
id
name
code
0,n
Located
1,1
searches
id
range
min_price
max_price
nb_room
nb_bedroom
1,1
Located
0,n
cities
id
name
latitude
longitude
0,n
Located
1,1
1,1
Looking for
0,n
addresses
id
nb
street
district
latitude
longitude
users
id
firstname
lastname
nickname
email
phone
password
show_name
show_email
show_phone
0,n
Own
1,1
1,1
Located
1,1
Located
roles
id
name
0,n
as
1,1
0,n
Employ
0,n
0,n
property_user
0,n
Publish
1,1
agencies
id
name
1,1
0,n
Own
0,1
1,1
subcategories
0,n
1,1
categories
id
name
0,n
has
1,1
properties
id
title
slug
subtitle
description
price
public_address
inner surface
outer surface
status
id
name
0,n
has
1,1
0,n
area_property
size
0,n
areas
id
name
subtypes
0,n
1,1
types
id
name
0,n
Belongs to
1,1
0,n
heating_device_heating_type_property
0,n
heating_types
id
name
0,n
heating_devices
id
name
0,n
0,n
0,n
property_proximity
picture_property
property_regulation
0,n
0,n
0,n
proximities
id
name
pictures
id
name
original_url
regulations
id
name
value