République Algérienne Démocratique et Populaire

Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

Université M'Hamed Bougera - Boumerdès



Faculté des Sciences

Département d'Informatique

Domaine : Mathématiques Informatique

Filière : Informatique

Spécialité : Systèmes d'informatique

Mémoire de fin d'études en vu de l'obtention du

Diplôme de Licence Académique

Thème

Developpement d'une application web pour la location saisonnière des logements

Présenté par :

Gairaa Yakoub

Elmahdi Fraoucene

DAAMOUCHE Yakoub

Mesbahi Abd el aziz

Membres du jury :

Encadreur : Gunadiz Safia

Examinateur : Mokrani Hocine

Table des matieres

Introduction générale 7
Chapitre 1 : Analyse et spécification des besoins 7
Introduction 7
1. Motivation du projet 8
2. Étude des applications existantes 8
2.1 L'application Krello 8
2.2 L'application Kerya 8
2.3 Analyse critique 8
3. Besoins fonctionnels 9
3.1 Fonctionnalités du client 9
3.3 Fonctionnalités du l'administrateur 10
4. Besoins non fonctionnels 10
5. Solutions proposées 10
Conclusion 11
Chapitre 2 : Analyse et spécification des besoins 12
Introduction 12
1. Identification des acteurs 12
2. Diagramme de cas d'utilisation 12
3. Diagramme de séquence 15
4. Diagramme de classe 16
5. Dictionnaire de données 17
Conclusion 21
Chapitre 3 : Implémentation 22
Introduction 22
1. Environnement de travail 22
1.1. Environnement matériel 22
1.2. Environnement technique 22
2. Architecture de l'application 24
3. Description de l'application 24
3.1. Page d'accueil 24
3.2. Recherche et filtrage 25
3.3. Page de détail d'une annonce 26

3.4. gestion de l’annonce publiée 26
3.5. Messagerie en temps réel 27
3.6. Interface administrateur 27

Conclusion 28

Conclusion générale et perspectives 29

Références bibliographiques 30

Liste des figures

Figure 1 : Diagramme de cas d’utilisation (Marketplace cote acheteur) 13
Figure 2 : Diagramme de cas d’utilisation (Vendeurs et administration).....14
Figure 3 : Diagramme de séquence 16
Figure 4 : Diagramme de classe 17
Figure 3.1 : Page d'accueil avec les catégories de logements 25
Figure 3.2 : Interface de recherche avec filtres avancés 25
Figure 3.3 : Page de détail d’un logement avec formulaire de réservation 26
Figure 3.4 : Gestion des l’annonces 27
Figure 3.5 : Interface de messagerie en temps réel 27
Figure 3.6 : Tableau de bord administrateur 28
Figure 3.7 : File d'attente de validation des annonces 28

Remerciements

Nous remercions Allah pour la force et la patience qu'il nous a données afin de réaliser ce travail.

Nous remercions notre encadreur pour ses conseils et son suivi durant ce projet.

Nous remercions également les enseignants du département d'informatique

Résumé

Ce mémoire présente la conception et le développement d'une application web dédiée à la location saisonnière de logements en Algérie. L'objectif principal du projet est de proposer une plateforme permettant aux utilisateurs de publier des annonces, rechercher des logements, effectuer des réservations et communiquer grâce à une messagerie intégrée.

Le travail a commencé par une phase d'analyse des besoins et l'étude de quelques applications existantes afin d'identifier leurs limites et les fonctionnalités nécessaires à notre plateforme. Ensuite, une phase de conception a été réalisée à l'aide de la modélisation UML, notamment les diagrammes de cas d'utilisation, de classes et le modèle logique de données.

La phase d'implémentation a été réalisée avec React.js et Chakra UI pour le frontend, Node.js, Express.js et TypeScript pour le backend, PostgreSQL comme système de gestion de base de données et Socket.io pour la messagerie en temps réel.

Cette expérience nous a permis de mettre en pratique plusieurs connaissances acquises durant notre formation et de mieux comprendre les différentes étapes de développement d'une application web complète.

Mots-clés : application web, location saisonnière, réservation, React.js, Node.js, PostgreSQL, Socket.io.

Abstract

This thesis presents the design and development of a web application dedicated to seasonal accommodation rental in Algeria. The main objective of this project is to provide a platform where users can publish listings, search for accommodations, make reservations, and communicate through an integrated messaging system.

The work started with a requirements analysis phase and the study of existing applications in order to identify their limitations and define the functionalities needed for our platform. Then, a design phase was carried out using UML modeling, including use case diagrams, class diagrams, and the logical data model.

The implementation phase was developed using React.js and Chakra UI for the frontend, Node.js, Express.js, and TypeScript for the backend, PostgreSQL as the database management system, and Socket.io for real-time messaging.

This project allowed us to apply several skills acquired during our studies and to better understand the different stages involved in developing a complete web application.

Liste des abréviations

API	Application Programming Interface
BDD	Base de Données
CSS	Cascading Style Sheets
CRUD	Create, Read, Update, Delete
http	HyperText Transfer Protocol
JWT	JSON Web Token
MLD	Modèle Logique de Données
ORM	Object-Relational Mapping
REST	Representational State Transfer
SQL	Structured Query Language
UML	Unified Modeling Language
UUID	Universally Unique Identifier
VITE	Outil de build frontend rapide basé sur ESModules
WS	WebSocket

Introduction générale :

Avec le développement des applications et des plateformes électroniques, plusieurs services sont devenus plus simples et plus rapides à utiliser au quotidien. Parmi ces services, on trouve la location d'appartements saisonniers, surtout avec l'augmentation des déplacements et des voyages pour le tourisme, le travail ou la participation à différents événements.

C'est dans ce contexte que s'inscrit notre projet, qui consiste à développer une plateforme web dédiée à la location d'appartements saisonniers. Cette plateforme permet aux propriétaires de publier leurs logements et aide les clients à rechercher facilement un logement adapté grâce à plusieurs fonctionnalités facilitant le processus de location de manière simple et organisée.

Chapitre 1 : Analyse et spécification des besoins

Introduction :

Dans ce premier chapitre, nous allons présenter l'analyse de notre projet ainsi que les différents besoins nécessaires à sa réalisation. Nous commencerons par expliquer la motivation du projet, puis nous présenterons quelques applications existantes avec leurs avantages et leurs inconvénients. Ensuite, nous définirons les besoins fonctionnels et non fonctionnels de la plateforme avant de présenter la solution proposée. Enfin, nous terminerons ce chapitre par une conclusion.

1. Motivation du projet :

Le choix de ce projet est né d'un constat concernant la recherche de logements en location en Algérie. La plupart des annonces sont publiées sur les réseaux sociaux, notamment Facebook, ou sur des sites généralistes comme Ouedkniss, ce qui rend la recherche de logements parfois difficile et peu organisée pour les utilisateurs. Par ailleurs, les plateformes existantes regroupent souvent plusieurs services, tels que la vente, l'échange et la location, ce qui complique l'accès rapide aux offres de location saisonnière. Dans cette optique, une plateforme dédiée exclusivement à la location de logements saisonniers a été développée afin de simplifier la recherche, d'améliorer

l'organisation des annonces et de faciliter la communication entre les propriétaires et les clients.

2. Étude des applications existantes :

Avant de concevoir notre plateforme, nous avons analysé quelques applications existantes afin d'identifier leurs points forts et leurs limites.

2.1. L'application Krello :

Krello est une application mobile dédiée à la location d'appartements. Elle propose également des services de vente et d'échange de logements, et permet aux utilisateurs d'accéder à différentes offres immobilières depuis une seule plateforme.

2.2. L'application Kerya :

Kerya est une application mobile spécialisée uniquement dans la location de logements. Elle permet aux utilisateurs de consulter des annonces de location, de rechercher des logements disponibles et de faciliter la mise en relation entre les propriétaires et les clients à travers une interface simple et pratique.

2.3. Analyse critique des applications existantes :

Avantages

- Les applications existantes disposent généralement d'une interface simple qui permet aux utilisateurs de naviguer facilement entre les différentes annonces et d'accéder rapidement aux informations importantes des logements.
- Chaque annonce affiche directement le numéro de téléphone du propriétaire, ce qui permet à l'utilisateur de le contacter facilement et rapidement.
- Les annonces présentent les informations principales des logements, telles que le prix, les photos, la localisation et la description générale, ce qui aide l'utilisateur à se faire une première idée sur le logement.

Inconvénients

- Plusieurs plateformes ne disposent pas d'un système de messagerie intégré, ce qui oblige souvent les clients à utiliser des moyens externes pour contacter les propriétaires.

- Certaines applications ne proposent pas un système d'évaluation fiable, ce qui rend difficile pour les utilisateurs de juger la qualité réelle du logement ou la fiabilité du propriétaire.
- Les annonces sont parfois mal organisées et ne bénéficient pas d'une catégorisation claire, ce qui complique la recherche pour les utilisateurs.
- La présence de plusieurs services dans une même plateforme peut rendre l'utilisation plus complexe pour les personnes qui recherchent uniquement des logements à louer.
- Dans plusieurs cas, les plateformes n'affichent pas d'informations claires sur la disponibilité des logements, ce qui peut générer des confusions lors de la recherche.

3. Besoins fonctionnels

Dans cette section, nous présentons les principales fonctionnalités offertes par la plateforme selon les deux types d'utilisateurs : l'administrateur et l'utilisateur.

3.1 Fonctionnalités du client

- Lors de son inscription, l'utilisateur doit créer un compte utilisateur lui permettant d'accéder aux fonctionnalités de la plateforme.
- Le client peut consulter les différentes annonces disponibles sur la plateforme et accéder aux informations essentielles des logements, telles que le prix, les photos, la localisation et la description.
- La plateforme propose un système de recherche et de filtrage permettant au client de trouver plus rapidement le logement correspondant à ses besoins.
- Le client peut publier des annonces de logements à louer afin de proposer ses biens aux autres utilisateurs de la plateforme.
- Le client peut contacter le propriétaire via un système de messagerie intégré afin d'obtenir des informations supplémentaires sur le logement.
- Le client peut ajouter des logements à une liste de favoris afin de les retrouver facilement par la suite.
- Après la location, le client a la possibilité de laisser une évaluation sur le logement et de partager son avis concernant le service reçu.

3.3 Fonctionnalités de l'administrateur

- Accéder à un tableau de bord affichant des statistiques : nombre total d'annonces, nombre d'annonces vérifiées, nombre d'annonces en attente de validation.
- Consulter la file d'attente des annonces soumises par les utilisateurs et les valider (publier) ou les rejeter.
- Consulter et gérer l'ensemble des annonces et des réservations de la plateforme.

4. Besoins non fonctionnels

Après avoir défini les besoins fonctionnels, nous présentons ci-dessous les contraintes et les qualités que doit respecter le système afin d'assurer une bonne performance et une expérience utilisateur satisfaisante.

- Performance : L'application doit être rapide et répondre aux actions des utilisateurs dans un délai acceptable afin de garantir une utilisation fluide.
- Simplicité : L'application doit être facile à prendre en main. Chaque utilisateur doit pouvoir comprendre et utiliser le système sans difficultés particulières.
- Ergonomie de l'interface : L'interface doit être claire, simple et agréable afin de faciliter la navigation entre les différentes pages de la plateforme.
- Modularité : Le système doit être bien organisé en modules indépendants, ce qui facilite sa modification ou son évolution future.
- Maintenabilité : l'utilisation de TypeScript côté backend améliore la lisibilité du code et facilite la détection des erreurs.
- Fiabilité : Le système doit fonctionner correctement et rester stable, même en cas d'utilisation intensive ou de problèmes techniques mineurs.
- Sécurité : L'application doit protéger les données personnelles des utilisateurs. Ces informations doivent être sécurisées et accessibles uniquement aux personnes autorisées.

5. Solutions proposées

Sur la base de l'analyse des applications existantes et des besoins identifiés, plusieurs améliorations sont proposées pour notre plateforme, détaillées ci-dessous :

- Intégration d’un système d’évaluation permettant aux utilisateurs de noter les logements après leur séjour, ce qui contribue à améliorer la qualité des services et à orienter les futurs clients.
- Classement des logements du mieux noté au moins bien noté selon les évaluations des clients, afin d’aider les utilisateurs à trouver plus facilement les logements les plus appréciés.
- Affichage de la disponibilité des logements en temps réel, afin de permettre aux utilisateurs de savoir immédiatement si un logement est libre ou déjà réservé.
- Classement des logements par type (F2, F3, F4, etc.), ce qui facilite la recherche et permet aux utilisateurs de trouver rapidement le logement correspondant à leurs besoins.

Conclusion

Dans ce chapitre, nous avons présenté l’analyse de notre projet ainsi que la motivation qui nous a conduits à sa réalisation. Nous avons étudié les applications existantes afin d’identifier leurs limites et de mieux comprendre les besoins des utilisateurs.

Nous avons ensuite défini les besoins fonctionnels et non fonctionnels de la plateforme, ainsi que les améliorations que nous proposons d’apporter par rapport aux solutions existantes. Ces éléments constituent la base de notre travail et nous serviront de référence pour les étapes de conception et de développement qui suivront.

Chapitre 2 : Analyse et spécification des besoins

Introduction :

Dans ce chapitre, nous allons aborder la phase de conception de notre système de location d'appartements. Après avoir analysé les besoins fonctionnels et non fonctionnels dans le chapitre précédent, cette étape consiste à transformer ces besoins en une structure plus détaillée et organisée.

Pour cela, nous allons commencer par identifier les différents acteurs du système ainsi que leurs rôles. Ensuite, nous présenterons la modélisation du système à travers les diagrammes UML, notamment le diagramme de cas d'utilisation et le diagramme de classes, afin de mieux représenter les interactions et la structure du système.

Enfin, nous terminerons par la conception de la base de données et des principales interfaces de la plateforme. Ce chapitre permet ainsi de préparer la mise en œuvre du projet en définissant une vision claire et structurée du système.

1. Identification des acteurs :

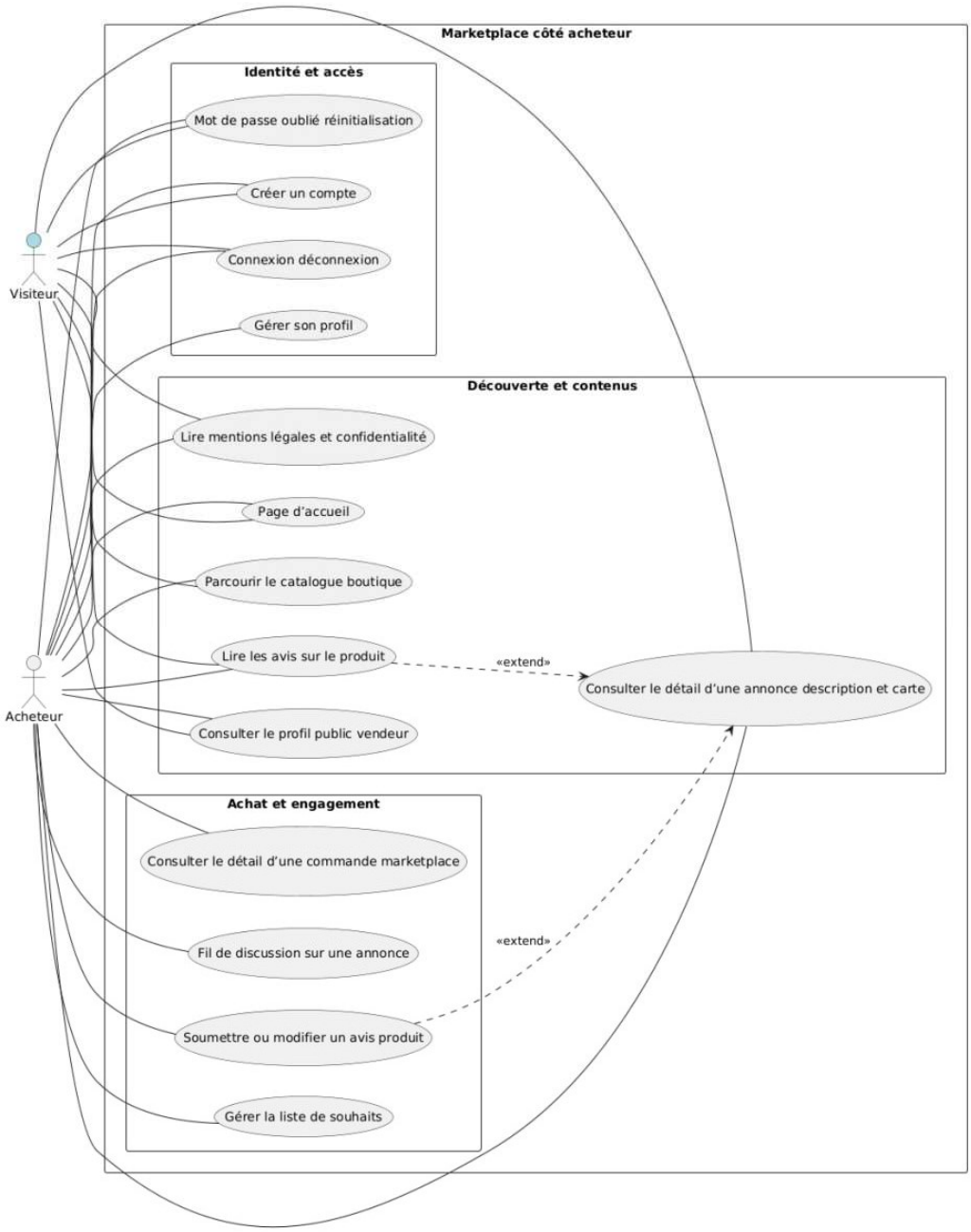
Lors de la phase de conception, nous avons identifié deux principaux acteurs du système :

Acteur	Description
Utilisateur	Compte créé après inscription sur la plateforme. L'utilisateur peut à la fois rechercher et consulter les annonces de logements, ainsi que publier ses propres annonces afin de proposer ses biens à la location. Il peut également effectuer des réservations, utiliser la messagerie et gérer sa liste de favoris.
Administrateur	Compte disposant d'un accès privilégié pour la gestion de la plateforme. Il peut consulter et valider les annonces soumises par les utilisateurs, ainsi que suivre les statistiques générales de l'application..

2. Diagramme de cas d'utilisation :

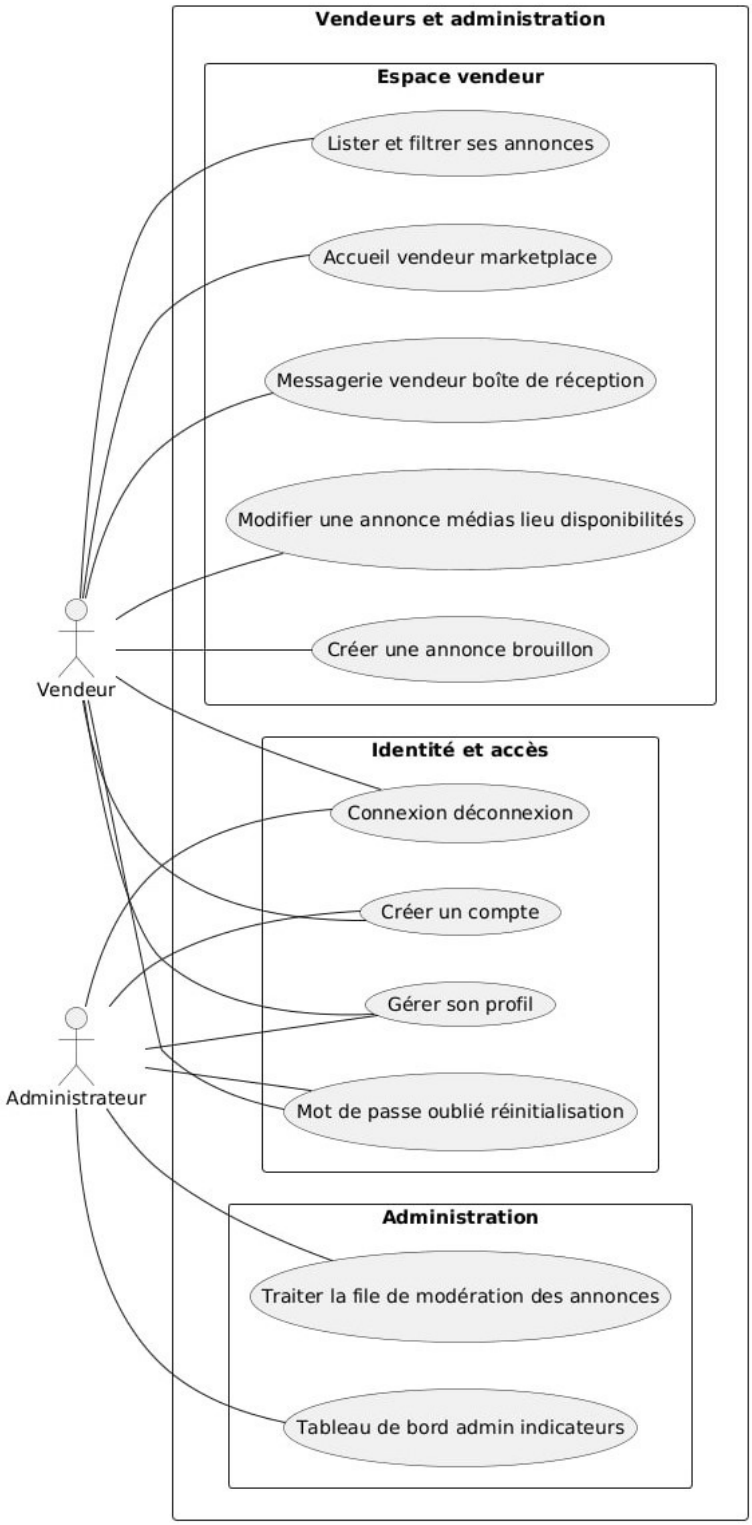
Nous avons réalisé deux diagrammes de cas d'utilisation pour représenter les interactions entre les acteurs et le système.

- Le premier diagramme (Figure 1) représente les fonctionnalités accessibles aux utilisateurs côté acheteur/locataire : gestion du compte, consultation des annonces, recherche, messagerie, évaluations et liste de favoris.



[Figure 1 : Diagramme de cas d'utilisation – côté acheteur/locataire]

- Le deuxième diagramme (Figure 2) représente les fonctionnalités du vendeur/propriétaire et de l'administrateur : création et gestion des annonces, messagerie, modération des annonces et tableau de bord.



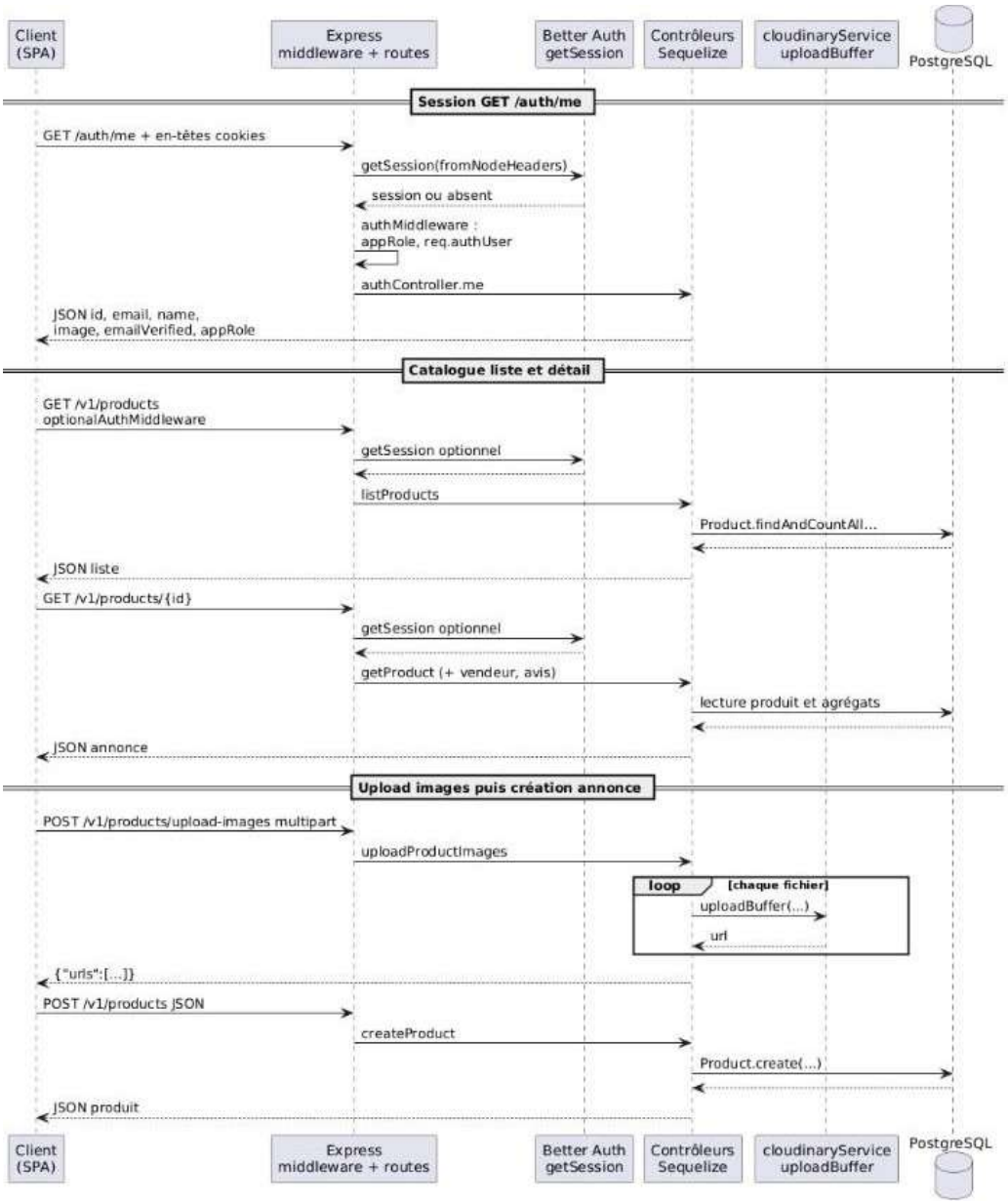
[Figure 2 : Diagramme de cas d'utilisation – vendeurs et administration]

3. Diagramme de séquence :

Nous avons élaboré un diagramme de séquence pour représenter les principaux flux de communication entre les différentes couches de notre application : le client (SPA React), le serveur Express, le système d'authentification Better-Auth, les contrôleurs Sequelize et la base de données PostgreSQL.

Ce diagramme illustre trois scénarios clés que nous avons implémentés :

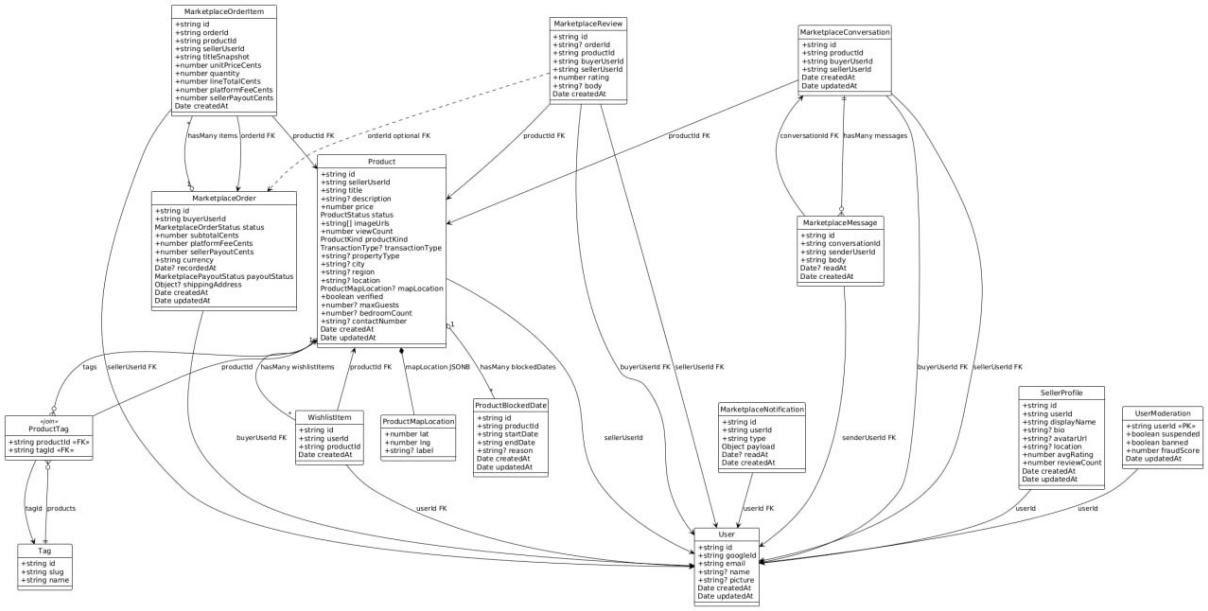
- La récupération de la session utilisateur via GET /auth/me, avec vérification du token et retour des informations de l'utilisateur connecté.
- La consultation du catalogue d'annonces via GET /v1/products, avec pagination et filtrage.
- L'upload d'images et la création d'une annonce, avec stockage sur Cloudinary puis insertion en base de données.



[Figure 3 : Diagramme de séquence]

4. Diagramme de classe :

Le diagramme de classes que nous avons conçu représente la structure des principales entités de notre application et les relations qui existent entre elles. Les entités principales sont : User, Product, SellerProfile, MarketplaceConversation, MarketplaceMessage, MarketplaceReview, MarketplaceOrder, WishlistItem, Tag, et UserModeration



[Figure 4 : Diagramme de classes]

5. Dictionnaire de données :

Le dictionnaire de données ci-dessous décrit les tables principales que nous avons créées dans notre base de données PostgreSQL, avec les champs et leurs types associés.

Table : products (annonces de logements)

Champ	Type	Description
Id	Uuid	Identifiant unique de l'annonce
seller_user_id	varchar(255)	Identifiant du propriétaire
Title	varchar(512)	Titre de l'annonce
Description	Text	Description du logement
Price	bigint	Prix par nuit (en centimes)
Status	varchar(32)	État de l'annonce : draft, pending_review, active, rented, archived
image_urls	jsonb	Liste des URLs des images (stockées sur Cloudinary)

property_type	varchar(64)	Type de logement (Studio, Villa, Appartement, etc.)
City	varchar(128)	Ville du logement
Region	varchar(128)	Wilaya
map_location	Jsonb	Coordonnées GPS pour Google Maps
Verified	Boolean	Annonce vérifiée par l'administrateur
bedroom_count	Integer	Nombre de chambres
max_guests	Integer	Capacité maximale d'accueil
contact_number	varchar(48)	Numéro de contact du propriétaire
created_at	timestamptz	Date de création
updated_at	timestamptz	Date de dernière modification

Table : marketplace_conversations (conversations entre utilisateurs)

Champ	Type	Description
Id	Uuid	Identifiant unique de la conversation
product_id	Uuid	Référence vers l'annonce concernée
buyer_user_id	varchar(255)	Identifiant du locataire
seller_user_id	varchar(255)	Identifiant du propriétaire
created_at	timestamptz	Date de création
updated_at	timestamptz	Date de modification

Table : marketplace_messages (messages échangés)

Champ	Type	Description

Id	uuid	Identifiant unique du message
conversation_id	uuid	Référence vers la conversation
sender_user_id	varchar(255)	Identifiant de l'expéditeur
Body	text	Contenu du message
read_at	timestamptz	Date de lecture (null si non lu)
created_at	timestamptz	Date d'envoi

Table : marketplace_reviews (évaluations des logements)

Champ	Type	Description
Id	uuid	Identifiant unique de l'évaluation
product_id	uuid	Référence vers l'annonce évaluée
buyer_user_id	varchar(255)	Identifiant de l'évaluateur
seller_user_id	varchar(255)	Identifiant du propriétaire
Rating	integer	Note de 1 à 5
Body	text	Commentaire textuel
created_at	timestamptz	Date de l'évaluation

Table : wishlist_items (liste de favoris)

Champ	Type	Description
Id	uuid	Identifiant unique
user_id	varchar(255)	Identifiant de l'utilisateur
product_id	uuid	Référence vers l'annonce ajoutée en favori

created_at	timestamptz	Date d'ajout

Table : seller_profiles (profil des propriétaires)

Champ	Type	Description
Id	uuid	Identifiant unique
user_id	varchar(255)	Référence vers l'utilisateur (UNIQUE)
display_name	varchar(255)	Nom affiché publiquement
Bio	text	Courte description du propriétaire
avg_rating	decimal(4,2)	Moyenne des évaluations reçues
review_count	integer	Nombre d'évaluations reçues
created_at	timestamptz	Date de création du profil

Table : user_moderation (gestion des utilisateurs)

Champ	Type	Description
user_id	varchar(255)	Identifiant de l'utilisateur (clé primaire)
Suspended	boolean	Compte suspendu ou non
Banned	boolean	Compte banni ou non
fraud_score	integer	Score de risque calculé par le système
updated_at	timestamptz	Date de dernière modification

Table : product_blocked_dates (dates d'indisponibilité)

Champ	Type	Description
Id	uuid	Identifiant unique

product_id	uuid	Référence vers l'annonce concernée
start_date	date	Début de la période d'indisponibilité
end_date	date	Fin de la période d'indisponibilité
Reason	varchar(512)	Raison de l'indisponibilité (optionnelle)

Conclusion :

Ce chapitre nous a permis de structurer notre système avant de passer à la phase d'implémentation. La modélisation UML et la conception de la base de données nous ont aidés à avoir une vision claire de l'architecture de l'application. C'est sur cette base que nous avons développé les différentes fonctionnalités présentées dans le chapitre suivant.

Chapitre 3 : Implémentation

Introduction :

Ce chapitre décrit la réalisation concrète de l'application. Nous commencerons par présenter l'environnement de travail et les technologies utilisées, puis nous décrirons les principales interfaces de la plateforme et quelques points d'implémentation clés. Nous terminerons par les difficultés rencontrées et les solutions adoptées.

1. Environnement de travail :

1.1 Environnement matériel :

	Machine 1	Machine 2
Système d'exploitation	Windows 11	Windows 10
Processeur	Intel Core i5 @ 2.40 GHz	Intel Core i3 @ 2.00 GHz
RAM	8 Go	4 Go
Stockage	SSD 256 Go	SSD 128 Go

1.2 Environnement technique :

Pour réaliser notre application, nous avons choisi une stack technique entièrement basée sur JavaScript/TypeScript. Ce choix nous a permis d'utiliser le même langage côté backend et côté frontend, ce qui a facilité le développement et la communication entre les deux parties.

Côté backend :

- Node.js + TypeScript : environnement d'exécution JavaScript côté serveur. TypeScript ajoute le typage statique, ce qui améliore la qualité du code et facilite la détection des erreurs à la compilation.
- Express.js : framework minimaliste pour construire l'API REST. Il gère le routage, les middlewares d'authentification et la validation des requêtes.

- PostgreSQL : base de données relationnelle utilisée pour stocker toutes les données de l'application (annonces, réservations, messages, utilisateurs…).
- Sequelize : ORM qui permet d'interagir avec PostgreSQL via des objets TypeScript plutôt qu'en écrivant du SQL brut. Les modèles Sequelize définissent la structure des tables et leurs associations.
- Better-Auth : bibliothèque d'authentification qui gère les sessions, le hachage des mots de passe, la connexion Google OAuth2 et l'émission des tokens Bearer.
- Socket.io : bibliothèque pour la communication WebSocket utilisée pour la messagerie en temps réel entre utilisateurs.
- Cloudinary + Sharp : Cloudinary est utilisé comme service de stockage et de diffusion des images. Sharp compresse les images avant l'upload pour réduire la taille des fichiers.
- Zod : bibliothèque de validation de schémas TypeScript utilisée pour valider les données reçues dans chaque endpoint de l'API avant traitement.
- Helmet + express-rate-limit : middlewares de sécurité HTTP configurés dans l'application Express.

Côté frontend :

- React.js 18 : bibliothèque JavaScript pour construire des interfaces utilisateur basées sur des composants réutilisables.
- Refine : framework open-source basé sur React qui simplifie la gestion des ressources, la pagination et les opérations CRUD. Il est utilisé notamment pour l'interface d'administration.
- Chakra UI : bibliothèque de composants UI avec un design system cohérent. Elle gère la mise en page, les formulaires, les modales et le responsive design.
- React Router v6 : gestion de la navigation côté client avec des routes protégées selon le rôle de l'utilisateur connecté.
- Socket.io client : permet d'établir une connexion WebSocket depuis le navigateur pour la messagerie temps réel.
- i18next : bibliothèque d'internationalisation intégrée pour la gestion des traductions de l'interface.
- @vis.gl/react-google-maps : intégration de Google Maps pour afficher la localisation des annonces sur une carte interactive.

- framer-motion : utilisé pour quelques animations d'interface sur la page du tableau de bord vendeur.

2. Architecture de l'application :

Notre application suit une architecture client-serveur classique, avec une séparation claire entre le backend (API REST) et le frontend (application React). Voici comment nous avons organisé les différentes couches de notre système :

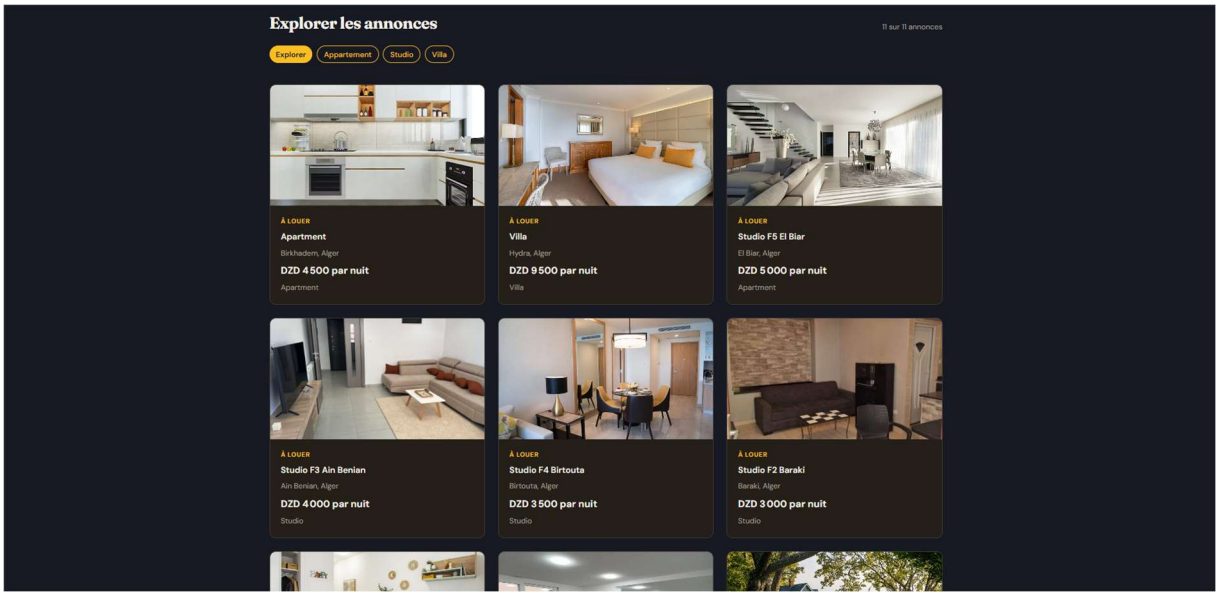
- Le frontend React communique avec le backend via des requêtes HTTP (API REST) pour toutes les opérations CRUD (consultation d'annonces, création d'une annonce, gestion du profil...).
- Pour la messagerie en temps réel, nous avons mis en place une connexion WebSocket entre le frontend et le backend à l'aide de Socket.io. Cela permet d'envoyer et de recevoir des messages sans avoir besoin de rafraîchir la page.
- Côté backend, nous avons structuré le code en plusieurs couches : les routes Express, les middlewares (authentification, validation), les contrôleurs et les modèles Sequelize.
- La base de données PostgreSQL est accessible uniquement depuis le backend, ce qui garantit que les données ne sont jamais exposées directement au client.
- Les images des annonces sont envoyées au serveur, compressées avec Sharp, puis uploadées sur Cloudinary. Seules les URLs des images sont stockées dans la base de données.

3. Description de l'application :

3.1. Page d'accueil :

La page d'accueil est la première interface que nous avons développée côté frontend. Elle affiche la liste des annonces actives (dont le statut est 'active' en base de données), filtrables par type de bien via des onglets de catégories : Explorer, Studio, Villa, Appartement, Duplex, etc. Les valeurs de ces catégories correspondent directement au champ property_type que nous avons défini dans notre base de données.

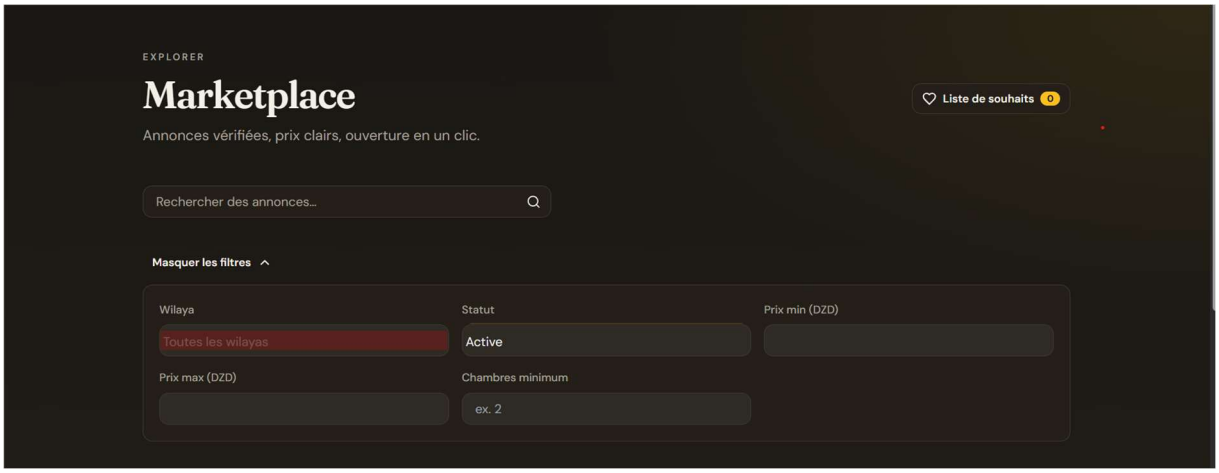
Chaque annonce est présentée sous forme de carte affichant la photo principale, le titre, la ville, le type de bien et le prix par nuit. Nous avons conçu ce composant de manière réutilisable afin de l'utiliser également dans d'autres parties de l'application



[Figure 3-1 : Page d'accueil avec les catégories de logements]

3.2. Recherche et filtrage :

Nous avons développé une page de recherche permettant aux utilisateurs de filtrer les logements selon plusieurs critères : la ville, la wilaya, le type de logement, le statut (actif/loué), le nombre de chambres minimum, le prix minimum et le prix maximum. Ces filtres sont envoyés en paramètres de requête à notre API backend, qui retourne les annonces correspondantes depuis la base de données.

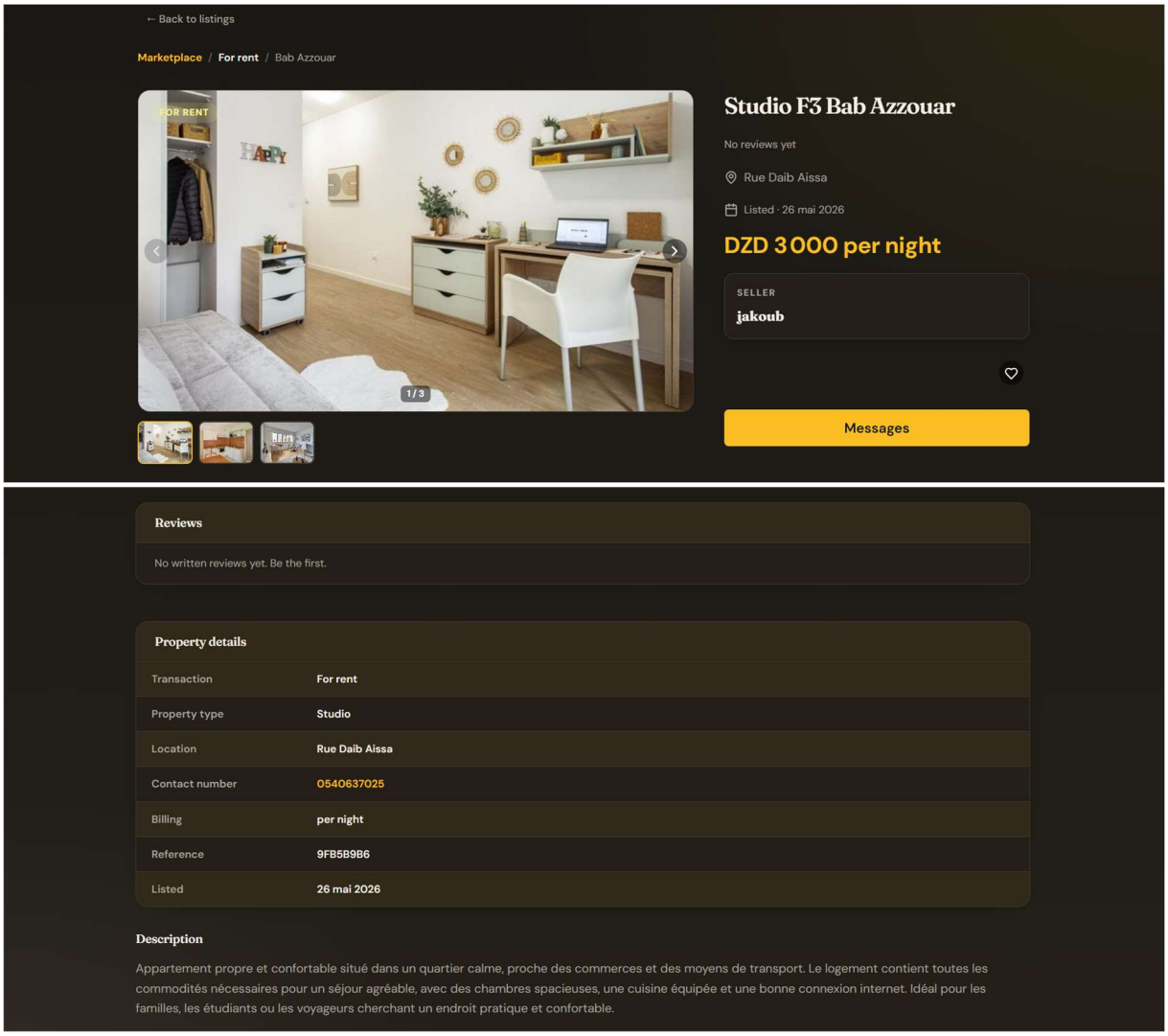


[Figure 3-2 : Interface de recherche avec filtres avancés]

3.3. Page de détail d'une annonce :

Cette page, que nous avons développée en React, affiche toutes les informations détaillées d'un logement : galerie de photos, titre, type de logement, nombre de chambres, capacité d'accueil, description, emplacement

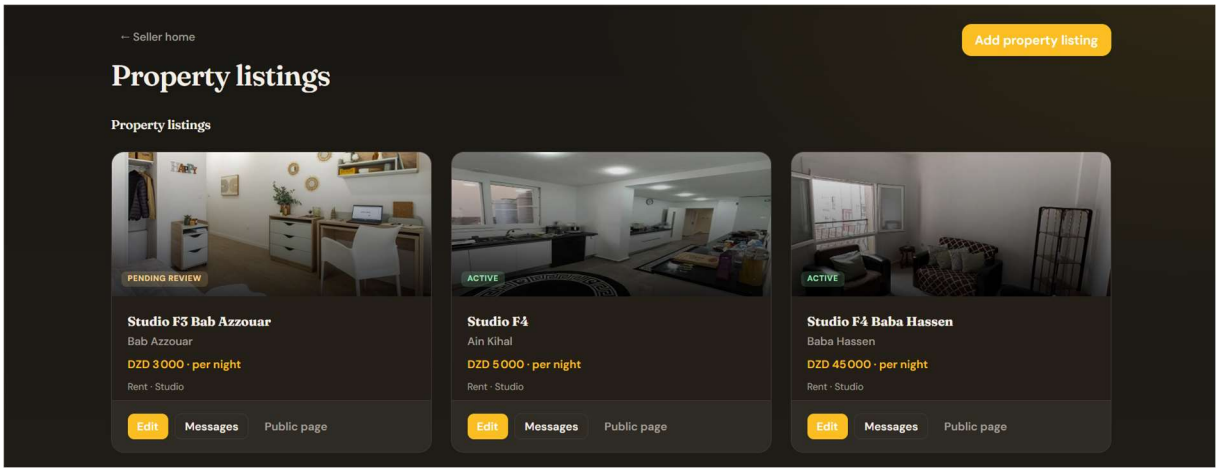
sur une carte Google Maps interactive, évaluations des anciens locataires, et profil du propriétaire. L'utilisateur peut également contacter directement le propriétaire depuis cette page via le bouton de messagerie, qui ouvre la conversation en temps réel



[Figure 3-3 : Page de détail d’un logement avec formulaire de réservation]

3.4. gestion de l’annonce publiée :

Nous avons développé un espace dédié aux propriétaires, depuis lequel ils peuvent consulter la liste de leurs annonces avec leur statut (en attente de validation, active, louée...), modifier une annonce existante, et en créer une nouvelle. La création d'une annonce passe par plusieurs étapes : upload d'images, saisie des informations du logement, et soumission pour validation par l'administrateur.

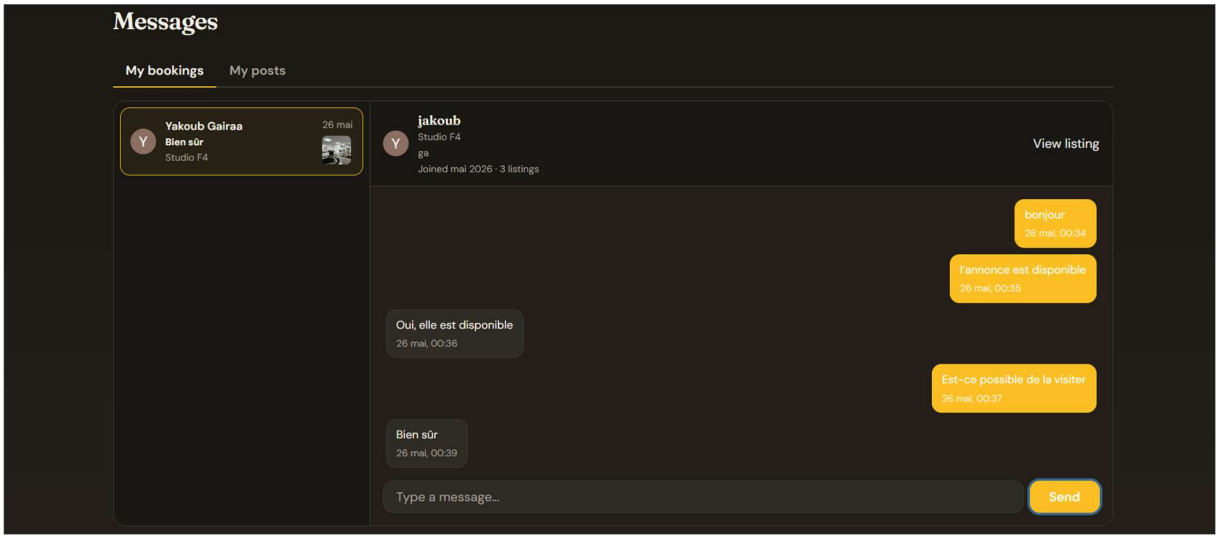


[Figure 3-4 : Gestion des l'annonces]

3.5. Messagerie en temps réel :

La messagerie en temps réel est l'une des fonctionnalités les plus importantes de notre application. Nous l'avons développée en utilisant Socket.io côté serveur et côté client. L'interface affiche la liste des conversations à gauche et les messages de la conversation sélectionnée à droite. Les messages sont envoyés et reçus instantanément, sans rechargement de page.

Côté backend, nous avons géré les événements Socket.io pour l'envoi de messages, la notification des nouvelles conversations, et la mise à jour du statut de lecture des messages. Les messages sont également persistés dans notre base de données PostgreSQL pour garantir leur conservation.

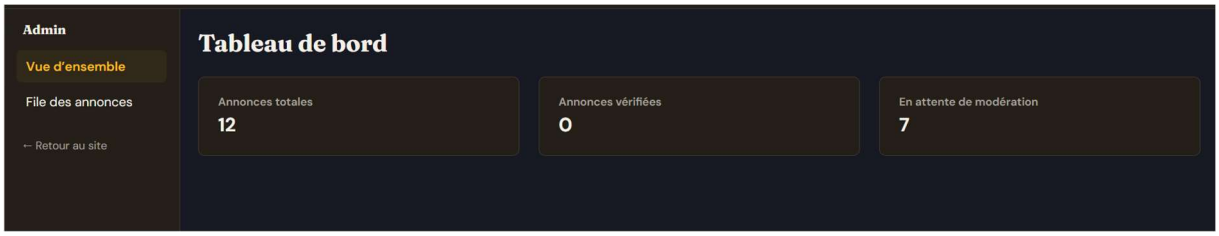


[Figure 3-5 : Interface de messagerie en temps réel]

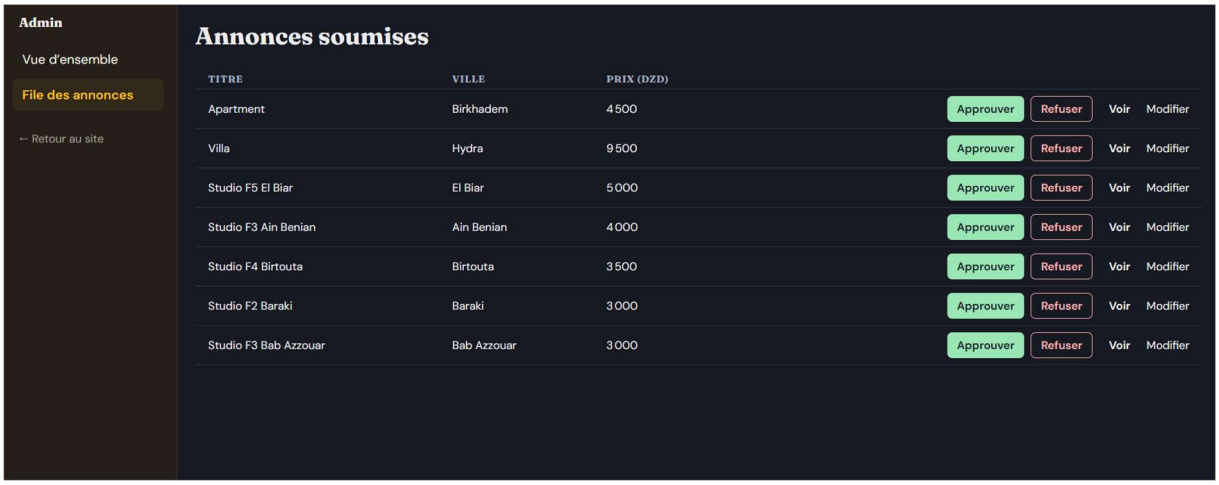
3.6. Interface administrateur :

Nous avons développé une interface d'administration dédiée, accessible uniquement aux utilisateurs ayant le rôle 'administrateur'. Cette interface est construite avec le framework Refine, qui nous a permis de gérer facilement les opérations de modération.
L'interface comprend deux sections principales :

- Le tableau de bord : il affiche des statistiques générales que nous récupérons depuis notre API backend, notamment le nombre total d'annonces, le nombre d'annonces vérifiées et le nombre d'annonces en attente de modération.
- La file d'attente des annonces : l'administrateur peut consulter les annonces soumises par les propriétaires et les approuver ou les rejeter. Lorsqu'une annonce est approuvée, son statut passe de 'pending_review' à 'active' dans notre base de données.



[Figure 3-6 : Tableau de bord administrateur]



[Figure 3-7 : File d'attente de validation des annonces]

Conclusion :

Ce chapitre nous a permis de présenter en détail le travail que nous avons réalisé lors de la phase d'implémentation. Nous avons développé une application web complète, depuis la conception de la base de données PostgreSQL jusqu'à l'interface utilisateur en React.js, en passant par le développement de l'API REST avec Node.js/Express.js et la mise en place de la messagerie en temps réel avec Socket.io. Les technologies que nous

avons utilisées nous ont permis de construire une application fonctionnelle et répondant aux objectifs que nous nous étions fixés au début du projet.

Conclusion générale et perspectives :

Ce projet de fin d'études nous a permis de concevoir et de développer une plateforme web de location saisonnière de logements répondant à des besoins réels. À travers ce travail, nous avons pu mettre en pratique plusieurs connaissances acquises durant notre parcours universitaire, aussi bien dans le développement backend que frontend.

La plateforme réalisée permet aux utilisateurs de publier et gérer des annonces, rechercher des logements, effectuer des réservations et communiquer grâce à une messagerie intégrée. Une interface administrateur a également été mise en place afin de gérer et valider les annonces publiées.

La réalisation de ce projet nous a aussi permis de découvrir et d'utiliser plusieurs technologies modernes comme TypeScript, Node.js, React.js, PostgreSQL et Socket.io. Cette expérience a été très enrichissante et nous a aidés à mieux comprendre le développement d'une application web complète.

Certaines fonctionnalités peuvent encore être améliorées ou ajoutées à l'avenir, comme l'intégration d'un système de paiement en ligne, l'ajout d'un système d'évaluation des logements, l'amélioration des notifications en temps réel ou encore le déploiement de l'application sur un serveur en ligne.

Enfin, ce projet représente une expérience importante dans notre parcours, car il nous a permis de travailler sur une application concrète et de mieux comprendre les différentes étapes de réalisation d'un projet web

Références bibliographiques :

[1] Node.js Foundation. Node.js — Run JavaScript Everywhere. https://nodejs.org/

[2] OpenJS Foundation. Express.js — Fast, unopinionated web framework for Node.js. https://expressjs.com/

[3] Sequelize. Sequelize — Feature-rich ORM for modern TypeScript & JavaScript. https://sequelize.org/

[4] The PostgreSQL Global Development Group. PostgreSQL 16 Documentation. https://www.postgresql.org/docs/

[5] Meta Open Source. React — A JavaScript library for building user interfaces. https://react.dev/

[6] Refine. Refine — Open-source Retool for Enterprise. https://refine.dev/

[7] Chakra UI. Chakra UI — Simple, modular and accessible component library. https://chakra-ui.com/

[8] Cloudinary Inc. Cloudinary — Image and Video Upload, Storage, Optimization and CDN. https://cloudinary.com/

[9] Socket.IO. Socket.IO — Bidirectional and low-latency communication for every platform. https://socket.io/

[10] Better-Auth. Better-Auth — The most comprehensive authentication library for TypeScript. https://better-auth.com/

[11] Zod. Zod — TypeScript-first schema validation with static type inference. https://zod.dev/

[12] OMG. Unified Modeling Language Specification Version 2.5.1. https://www.omg.org/spec/UML/

[13] Microsoft. TypeScript — JavaScript With Syntax For Types. https://www.typescriptlang.org/docs/

[14] Fowler, M. (2002). Patterns of Enterprise Application Architecture. Addison-Wesley Professional.

[15] Wieruch, R. (2020). The Road to React. Self-published.