



UNIVERSITÉ HASSAN II DE CASABLANCA  
**Faculté des Sciences et Techniques**

Filière Ingénierie  
Data Science et Informatique

---

**PROJET DU SEMESTRE**

---

Intitulé :

**Design and Development of a Cross-Platform Real Estate Application (Web and Mobile)**

Présenté par :

**YOUSSEFI NOUHAILA ET AKARZABI HAYAT**

Encadrant pédagogique :

Pr. Sbai Hanae

Soutenu le 03 Juin 2026

Année Universitaire 2025–2026

## RÉSUMÉ

Ce rapport présente la conception, le développement et la mise en uvre dune plateforme de gestion immobilière permettant de centraliser la gestion des biens, des contrats de location, des paiements et des réclamations. Lapplication vise à faciliter les interactions entre les administrateurs, les locataires, les propriétaires et les visiteurs à travers des interfaces dédiées.

La solution sappuie sur une architecture basée sur **Spring Boot** pour le backend, exposant des **API REST** sécurisées par **Spring Security** et lauthentification **JWT**. La persistance des données est assurée par une base de données **PostgreSQL**, tandis que le stockage des fichiers est pris en charge par **MinIO**. Le frontend est développé en **Angular** pour lapplication web et en **Flutter** pour lapplication mobile orientée locataire.

Ce rapport décrit les choix de conception, limplémentation des principales fonctionnalités et la présentation des résultats à travers des cas dutilisation et des captures décran de lapplication.

**Mots clés :** *Spring Boot, Angular, Flutter, PostgreSQL, MinIO, API REST, JWT, Spring Security, gestion immobilière.*

---

## ABSTRACT

This report presents the design, development, and implementation of a real estate management platform that centralizes the management of properties, rental contracts, payments, and complaints. The application aims to facilitate interactions between administrators, tenants, property owners, and visitors through dedicated user interfaces.

The solution is based on a **Spring Boot** backend exposing secure **REST APIs** using **Spring Security** and **JWT** authentication. Data persistence is ensured by a **PostgreSQL** database, while file storage is handled by **MinIO**. The frontend is developed using **Angular** for the web application and **Flutter** for the tenant-oriented mobile application.

This report describes the system design choices, the implementation of the main features, and the presentation of the results through use cases and application screenshots.

**Keywords :** *Spring Boot, Angular, Flutter, PostgreSQL, MinIO, REST API, JWT, Spring Security, real estate management.*

---

## TABLE DES MATIÈRES

<b>Résumé</b>	<b>2</b>
<b>Abstract</b>	<b>3</b>
<b>1 Contexte général</b>	<b>10</b>
1.1 Cadre général du projet . . . . .	10
1.1.1 Présentation du projet . . . . .	10
1.1.2 Cycle de vie du projet . . . . .	12
1.2 Conclusion . . . . .	13
<b>2 Analyse et Conception</b>	<b>14</b>
2.1 Analyse des besoins . . . . .	14
2.1.1 Besoins fonctionnels . . . . .	14
2.1.2 Besoins non fonctionnels . . . . .	16
2.2 Conception . . . . .	19
2.2.1 Modélisation UML . . . . .	19
2.2.2 Diagramme Use Case . . . . .	19
2.2.3 Diagrammes de séquence . . . . .	22
<b>3 Cadre de développement</b>	<b>25</b>
3.1 Environnement de développement . . . . .	25
3.2 Architecture Spring Adoptée . . . . .	27
3.2.1 Spring Boot . . . . .	27
3.2.2 Architecture en Couches . . . . .	27

3.2.3	Workflow d'une Requête . . . . .	29
3.2.4	Bénéfices . . . . .	29
3.3	Spring Boot REST API . . . . .	29
3.3.1	Principes des services web REST . . . . .	30
3.3.2	Les méthodes fondamentales de HTTP . . . . .	30
3.3.3	HTTP Standard Status Codes . . . . .	32
3.3.4	Usages de Spring Boot avec REST . . . . .	32
3.3.5	Bonnes pratiques de sécurité des API REST . . . . .	32
3.3.6	Bonnes pratiques de développement d'une API REST avec Spring Boot . . . . .	33
3.4	API Documentation . . . . .	33
3.4.1	Outils de build et gestion de dépendances : Gradle . . . . .	34
3.5	Spring Security . . . . .	35
3.5.1	Spring Security Filters Chain . . . . .	35
3.5.2	AuthenticationManager . . . . .	36
3.5.3	AuthenticationProvider . . . . .	36
3.5.4	UserDetailsService . . . . .	36
3.6	JWT avec Spring Security . . . . .	37
3.7	JPA/Hibernate . . . . .	38
3.8	MapStruct . . . . .	39
3.8.1	Project Lombok : réduction du code boilerplate en Java . . . . .	40
3.9	MinIO pour la gestion des fichiers . . . . .	41
3.10	Technologies Front-end . . . . .	41
3.10.1	Angular . . . . .	42
<b>4</b>	<b>Mise en uvre</b>	<b>44</b>
4.1	Présentation des résultats . . . . .	44
4.1.1	Application dadministration . . . . .	44
4.1.2	Espace public . . . . .	50
<b>5</b>	<b>Application Mobile de Gestion Immobilière</b>	<b>56</b>
5.1	Introduction . . . . .	56
5.2	Interface Locataire . . . . .	56
5.2.1	Tableau de Bord Principal . . . . .	56
5.2.2	Module de Paiement . . . . .	58

5.2.3	Gestion des Biens . . . . .	59
5.2.4	Gestion des Demandes de Location . . . . .	61
5.2.5	Gestion des Contrats . . . . .	61
5.2.6	Système de Réclamations . . . . .	62
5.2.7	Notifications et Alertes . . . . .	63
<b>6</b>	<b>Application Mobile de Gestion Immobilière</b>	<b>64</b>
6.1	Introduction . . . . .	64
6.2	Interface Locataire . . . . .	64
6.2.1	Tableau de Bord Principal . . . . .	64
6.2.2	Module de Paiement . . . . .	66
6.2.3	Gestion des Biens . . . . .	67
6.2.4	Gestion des Demandes de Location . . . . .	69
6.2.5	Gestion des Contrats . . . . .	69
6.2.6	Système de Réclamations . . . . .	71
6.2.7	Notifications et Alertes . . . . .	72
6.3	Interface Administrateur . . . . .	73
6.3.1	Tableau de Bord Administratif . . . . .	74
6.3.2	Gestion des Demandes de Location . . . . .	75
6.3.3	Gestion des Biens . . . . .	75
6.3.4	Configuration Générale . . . . .	76
6.3.5	Gestion des Réclamations . . . . .	77
	<b>Conclusion Générale</b>	<b>79</b>

---

## TABLE DES FIGURES

1.1	Cycle de développement Agile Scrum . . . . .	12
2.1	Diagramme de cas d'utilisation de l'application . . . . .	21
2.2	Diagramme de séquence : login JWT . . . . .	22
2.3	Diagramme de séquence : create a Property . . . . .	23
2.4	Diagramme de séquence : Cancel a Contract . . . . .	24
3.1	IDE IntelliJ IDEA . . . . .	25
3.2	Visual Studio Code . . . . .	25
3.3	Docker Desktop . . . . .	25
3.4	Git . . . . .	26
3.5	Github . . . . .	26
3.6	PostgreSQL . . . . .	26
3.7	Spring Framework . . . . .	27
3.8	Spring Boot Layered architecture . . . . .	27
3.9	REST API Model . . . . .	29
3.10	Components of an API Request and API Response . . . . .	30
3.11	Status Code . . . . .	32
3.12	Swagger . . . . .	33
3.13	Gradle build . . . . .	34
3.14	Spring Security Architecture . . . . .	35
3.15	JWT Structure . . . . .	37
3.16	Mapstruct Object Mapping . . . . .	39
3.17	MinIO . . . . .	41

3.18	CSS . . . . .	41
3.19	HTML . . . . .	41
3.20	TypeScript . . . . .	41
3.21	Angular . . . . .	42
4.1	Page login pour Admin . . . . .	44
4.2	dashboard admin avec stats . . . . .	45
4.3	le Crud pour gestion Biens . . . . .	46
4.4	Valider un bien . . . . .	46
4.5	le crud pour gestion des owners . . . . .	47
4.6	traiter la demande de profil Owner . . . . .	47
4.7	le crud pour gestion des Tenants . . . . .	48
4.8	Le crud pour gestion des Contrats . . . . .	48
4.9	Interface de traitement des Reclamations . . . . .	49
4.10	Page d'accueil pour visiteurs . . . . .	50
4.11	Page d'accueil des visiteurs . . . . .	51
4.12	Liste des biens proposes . . . . .	51
4.13	Details du Bien choisi . . . . .	52
4.14	page d'inscription autant que Locataire/tenant . . . . .	53
4.15	Tableau de bord d'un Tenant . . . . .	53
4.16	demandeur le profil Owner . . . . .	54
5.1	Page d'accueil locataire - Vue d'ensemble des statistiques . . . . .	57
5.2	Menu principal avec l'ensemble des fonctionnalités accessibles . . . . .	57
5.3	Interface de sélection du mode de paiement . . . . .	58
5.4	Formulaire de saisie des coordonnées bancaires . . . . .	58
5.5	Interface de recherche avancée avec filtres multicritères . . . . .	59
5.6	Vue détaillée d'un bien - Informations générales . . . . .	60
5.7	Vue détaillée d'un bien - Galerie photos et équipements . . . . .	60
5.8	Tableau de suivi des demandes de location . . . . .	61
5.9	Liste des contrats avec statut et actions . . . . .	62
5.10	Interface de création et suivi des réclamations . . . . .	62
5.11	Centre de notifications avec alertes personnalisées . . . . .	63
6.1	Page d'accueil locataire - Vue d'ensemble des statistiques . . . . .	65
6.2	Menu principal avec l'ensemble des fonctionnalités accessibles . . . . .	65

6.3	Interface de sélection du mode de paiement . . . . .	66
6.4	Formulaire de saisie des coordonnées bancaires . . . . .	66
6.5	Interface de recherche avancée avec filtres multicritères . . . . .	67
6.6	Vue détaillée d'un bien - Informations générales . . . . .	68
6.7	Vue détaillée d'un bien - Galerie photos et équipements . . . . .	68
6.8	Tableau de suivi des demandes de location . . . . .	69
6.9	Liste des contrats avec statut et actions . . . . .	70
6.10	Exemple de quittance pdf . . . . .	70
6.11	Exemple de quittance suite . . . . .	71
6.12	Interface de création et suivi des réclamations . . . . .	72
6.13	Centre de notifications avec alertes personnalisées . . . . .	72
6.14	Page d'accueil admin - Menu principal et accès rapide . . . . .	73
6.15	Page d'accueil admin - Suite du menu et fonctionnalités avancées . . . . .	74
6.16	Tableau de bord administrateur - Métriques et statistiques globales . . . . .	74
6.17	Interface de gestion des demandes de location . . . . .	75
6.18	Interface de gestion du catalogue des biens . . . . .	76
6.19	Interface de configuration générale de la plateforme . . . . .	77
6.20	Interface de supervision et gestion des réclamations . . . . .	77

# CHAPITRE 1

---

## CONTEXTE GÉNÉRAL

### 1.1 Cadre général du projet

#### 1.1.1 Présentation du projet

Dans le cadre du module **JEE / Spring Boot**, il nous a été confié la réalisation d'un projet de développement d'une application **full-stack** combinant une plateforme **web** et une application **mobile**. Ce projet s'inscrit dans une démarche pédagogique visant à mettre en pratique les concepts théoriques étudiés en cours, notamment l'architecture client-serveur, le développement d'API REST, la sécurité applicative et l'intégration de technologies frontend modernes.

L'objectif principal de ce projet est de concevoir et développer un système informatique moderne, scalable et sécurisé, accessible à la fois via une application web et une application mobile. Cette solution vise à répondre à des besoins fonctionnels réels tout en respectant les bonnes pratiques du génie logiciel et du développement orienté services.

Le système développé devait répondre aux exigences suivantes :

- Concevoir un **backend robuste** exposant des services REST permettant la gestion de la logique métier ;
- Assurer une **séparation claire** entre les couches backend, frontend web et frontend mobile ;
- Permettre une interaction fluide entre le backend et les clients web et mobile ;

- Garantir la **sécurité des échanges** grâce à un mécanisme d'authentification et d'autorisation ;
- Offrir une interface web moderne et réactive à l'aide du framework **Angular** ;
- Proposer une application mobile multiplateforme performante développée avec **Flutter**.

#### Cahier des charges technique :

Le projet repose sur une architecture **full-stack** organisée autour des technologies suivantes :

- **Backend** : développé en **Spring Boot**, il implémente la logique métier, la gestion des utilisateurs, la sécurité applicative via **Spring Security**, et expose des **API REST** consommées par les applications frontend.
- **Frontend Web** : réalisé avec **Angular**, il permet une navigation fluide, une gestion dynamique des composants et une communication efficace avec le backend via des services HTTP.
- **Frontend Mobile** : développé en **Flutter**, il offre une application mobile multiplateforme (Android / iOS) avec une interface utilisateur cohérente et performante.
- **Base de données** : utilisation d'une base de données relationnelle pour la persistance des données et la gestion des entités du système.
- **Sécurité** : mise en place d'un système d'authentification et d'autorisation basé sur des rôles, garantissant un accès sécurisé aux ressources.

Ce projet a été réalisé selon une approche structurée et itérative, permettant de assurer une montée en compétences progressive en développement backend et frontend, ainsi qu'une compréhension approfondie des interactions entre les différentes couches d'une application moderne.

### 1.1.2 Cycle de vie du projet

Le cycle de vie d'un projet logiciel correspond à l'ensemble des phases par lesquelles il passe, depuis l'expression des besoins jusqu'à la livraison finale. Afin de garantir une meilleure flexibilité, une adaptation continue aux besoins et une amélioration progressive du produit, nous avons adopté une méthodologie **Agile**, plus précisément le cadre **Scrum**.

La méthodologie Agile repose sur un développement itératif et incrémental, permettant de livrer régulièrement des fonctionnalités opérationnelles. Le projet a ainsi été découpé en plusieurs **sprints**, chacun ayant des objectifs clairement définis.

Les principales étapes du cycle de vie de notre projet sont :

- Analyse et formalisation des besoins fonctionnels et techniques ;
- Découpage du projet en **user stories** et priorisation du **backlog produit** ;
- Planification des sprints ;
- Conception et développement itératif des fonctionnalités ;
- Tests et validation continue à la fin de chaque sprint ;
- Intégration progressive des différentes composantes (backend, web et mobile).

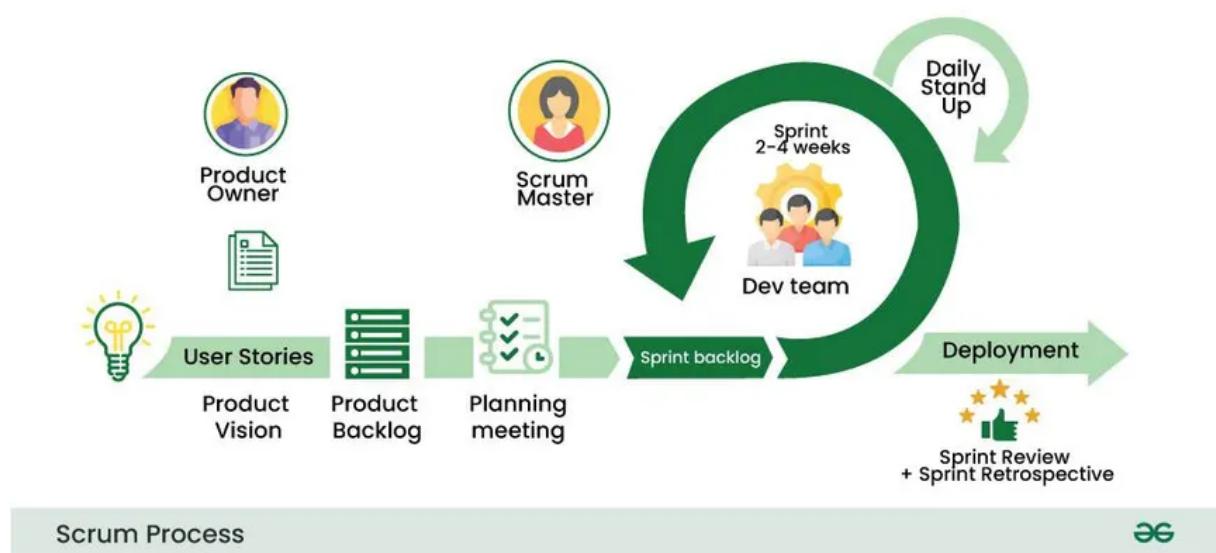


FIGURE 1.1 – Cycle de développement Agile Scrum

Cette approche nous a permis de mieux gérer la complexité du projet, d'améliorer la collaboration au sein de l'équipe et de assurer une évolution continue de la solution développée tout au long du cycle de développement.

## **1.2 Conclusion**

Dans ce chapitre, nous avons présenté le cadre général du projet, ses objectifs, son périmètre ainsi que les technologies utilisées. Nous avons également décrit la méthodologie de développement adoptée, basée sur l'approche Agile Scrum. L'analyse détaillée des besoins et la conception de l'architecture du système feront l'objet du chapitre suivant.

## CHAPITRE 2

### ANALYSE ET CONCEPTION

## 2.1 Analyse des besoins

Ce chapitre a pour objectif d'identifier les besoins du système à travers une analyse fonctionnelle et non fonctionnelle, puis de proposer une conception logique basée sur des diagrammes UML. Cette étape permet de structurer le projet et de préparer efficacement la phase de développement.

### 2.1.1 Besoins fonctionnels

Les besoins fonctionnels décrivent l'ensemble des services et fonctionnalités que le système doit offrir aux différents utilisateurs. Ils ont été identifiés à partir de l'analyse du domaine métier de la gestion immobilière et des échanges au sein de l'équipe projet. L'application vise à digitaliser et centraliser la gestion des biens immobiliers, des contrats de location, des paiements et des réclamations, tout en assurant une séparation claire des responsabilités selon les profils utilisateurs.

Le système distingue quatre profils principaux : **Administrateur**, **Visiteur**, **Locataire** et **Propriétaire**. Chaque profil dispose de droits et de fonctionnalités spécifiques.

#### 1. Administrateur

L'administrateur est le gestionnaire principal de l'application. Il dispose d'un accès complet à l'ensemble des fonctionnalités du système via l'interface web. Ses besoins fonctionnels

sont les suivants :

- Gérer les utilisateurs du système (création, modification, activation, désactivation et attribution des rôles).
- Gérer les biens immobiliers : consultation, ajout, modification et suppression.
- Valider ou refuser les biens proposés par les propriétaires avant leur publication sur la plateforme.
- Gérer les contrats de location (création, consultation, suivi du statut, résiliation).
- Traiter les demandes de location envoyées par les locataires.
- Gérer les demandes de résiliation de contrats.
- Consulter et suivre les paiements des loyers effectués par les locataires.
- Gérer les réclamations déposées par les locataires (consultation, traitement, mise à jour du statut).
- Accéder à un tableau de bord global offrant une vue synthétique sur les biens, contrats, paiements et réclamations.

## 2. Visiteur

Le visiteur est un utilisateur non authentifié. Il dispose d'un accès limité à l'application, lui permettant uniquement de consulter les informations publiques. Ses besoins fonctionnels sont :

- Consulter la liste des biens immobiliers disponibles.
- Visualiser les détails d'un bien (description, localisation, images, prix du loyer, etc.).
- Créer un compte afin d'accéder aux fonctionnalités avancées de l'application.

## 3. Locataire

Le locataire est un utilisateur authentifié disposant d'un compte. Il interagit principalement avec l'application mobile, tout en pouvant accéder à certaines fonctionnalités via le web. Ses besoins fonctionnels sont :

- S'authentifier à l'application et gérer son profil.
- Consulter les biens immobiliers disponibles.
- Envoyer une demande de location pour un bien sélectionné.
- Consulter l'historique de ses demandes de location.

- Consulter ses contrats de location en cours et passés.
- Effectuer le paiement du loyer via lapplication mobile.
- Télécharger la quittance de paiement après validation du règlement.
- Déposer une réclamation liée à un contrat (problème technique, maintenance, etc.).
- Consulter lhistorique de ses réclamations et leur état de traitement.
- Recevoir des notifications concernant les échéances de paiement, les contrats proches de lexpiration ou létat de ses demandes.
- Accéder à un tableau de bord personnel récapitulant ses contrats, paiements et réclamations.

## 4. Propriétaire

Le propriétaire est un utilisateur authentifié souhaitant proposer des biens immobiliers à la location. Laccès à ce profil nécessite une demande préalable. Ses besoins fonctionnels sont :

- Envoyer une demande pour obtenir le rôle de propriétaire.
- Proposer un nouveau bien immobilier à la publication.
- Consulter létat de validation de ses biens par ladministrateur.
- Consulter la liste de ses biens publiés.
- Consulter les contrats de location associés à ses biens.
- Suivre les paiements de loyers reçus.
- Accéder à un tableau de bord récapitulatif de ses biens, contrats et revenus locatifs.

## Conclusion

Cette répartition fonctionnelle permet dassurer une gestion claire, sécurisée et efficace du système immobilier. La distinction des rôles garantit une séparation des responsabilités et facilite lévolution future de lapplication, tout en offrant une expérience utilisateur adaptée aussi bien sur le web que sur mobile.

### 2.1.2 Besoins non fonctionnels

En complément des besoins fonctionnels, le système doit satisfaire un ensemble de besoins non fonctionnels garantissant sa qualité, sa fiabilité et sa pérennité. Ces exigences

concernent principalement la sécurité, la performance, la disponibilité, l'ergonomie ainsi que les aspects techniques liés à l'architecture choisie.

## Sécurité

La sécurité constitue un élément central de l'application, compte tenu de la sensibilité des données manipulées (informations personnelles, contrats, paiements, documents).

- L'accès aux fonctionnalités doit être sécurisé par un mécanisme d'authentification basé sur **JWT (JSON Web Token)**.
- La gestion des autorisations doit être assurée par un système de rôles clairement définis (administrateur, locataire, propriétaire).
- Les API REST doivent être protégées à l'aide de **Spring Security**, empêchant tout accès non autorisé.
- Les données sensibles (informations utilisateurs, contrats, paiements) doivent être protégées contre les accès frauduleux.
- Les fichiers et documents (images des biens, contrats, quittances) doivent être stockés de manière sécurisée via le système de stockage objet **MinIO**.

## Performance

L'application doit offrir des temps de réponse rapides et une expérience fluide aussi bien sur le web que sur le mobile.

- Les requêtes vers l'API doivent être optimisées afin de réduire le temps de chargement des pages et des tableaux de bord.
- Le système doit être capable de gérer simultanément plusieurs utilisateurs sans dégradation notable des performances.
- Le chargement des images des biens immobiliers doit être optimisé pour éviter une surcharge inutile du réseau.

## Disponibilité et fiabilité

Le système doit être accessible de manière continue et fiable.

- L'application doit être disponible 24h/24 et 7j/7.

- Les données doivent être persistées de manière fiable dans une base de données **PostgreSQL**.
- Les services doivent être déployés dans des conteneurs **Docker** afin de garantir la stabilité de l'environnement d'exécution.
- Le système doit être tolérant aux erreurs et capable de gérer les exceptions sans interruption critique du service.

## Ergonomie et expérience utilisateur

L'interface utilisateur doit être simple, intuitive et adaptée aux différents profils.

- L'interface web développée avec **Angular** doit proposer une navigation claire, des tableaux de bord lisibles et des formulaires ergonomiques.
- L'application mobile développée avec **Flutter** doit être orientée principalement vers les besoins du locataire, avec une navigation fluide et adaptée aux écrans mobiles.
- Les actions importantes (paiement, soumission de demande, dépôt de réclamation) doivent être accompagnées de messages de confirmation clairs.

## Maintenabilité et évolutivité

L'architecture du système doit faciliter la maintenance et l'évolution future de l'application.

- Le backend doit suivre une architecture basée sur les **API REST**, assurant une séparation claire entre la logique métier et les interfaces clientes.
- Le code source doit être structuré, modulaire et documenté afin de faciliter les futures évolutions.
- L'architecture doit permettre l'ajout ultérieur de nouvelles fonctionnalités (notifications avancées, statistiques, gestion multi-agences, etc.).

## Portabilité

- Le recours à **Docker** permet de déployer l'application sur différents environnements sans dépendance au système d'exploitation.
- L'application doit être compatible avec les navigateurs web modernes et les systèmes Android et iOS pour la partie mobile.

## Conclusion

Le respect de ces besoins non fonctionnels permet d'assurer un système sécurisé, performant et évolutif, répondant aux exigences d'une application de gestion immobilière moderne, accessible aussi bien via le web que via le mobile.

## 2.2 Conception

### 2.2.1 Modélisation UML

Le UML (Unified Modeling Language) est un langage de modélisation graphique standardisé, destiné à décrire, concevoir et documenter les systèmes logiciels. Issu de la fusion de plusieurs méthodes orientées objet (Booch, OMT, OOSE), UML a été adopté comme norme par l'Object Management Group (OMG) en 1997, puis par l'ISO en 2005.

Le **UML** (Unified Modeling Language) joue un rôle fondamental dans la maîtrise de la complexité des systèmes. Il offre une **langue visuelle commune** permettant de :

- Spécifier clairement les besoins fonctionnels et non fonctionnels ;
- Visualiser la structure statique (entités, classes) et le comportement dynamique (flux, séquences) du système ;
- Construire une conception robuste guidant efficacement les phases de développement ultérieures.

Dans ce chapitre, plusieurs **diagrammes UML** seront présentés pour formaliser la conception de l'application : *cas d'utilisation*, *classes* et éventuellement *séquences*. Ces modèles visuels permettront de décrire de manière précise l'interaction entre les utilisateurs et le système, ainsi que la structure interne de l'application.

### 2.2.2 Diagramme Use Case

Le diagramme de cas d'utilisation permet de représenter de manière globale les interactions entre les différents acteurs du système et les fonctionnalités offertes par l'application. Il constitue un outil fondamental dans l'analyse fonctionnelle, en mettant en évidence les responsabilités de chaque profil utilisateur ainsi que le périmètre fonctionnel du système.

Dans le cadre de notre application de gestion immobilière, plusieurs acteurs ont été identifiés, chacun disposant de droits et de fonctionnalités spécifiques :

- **Administrateur (Agent)** : acteur central du système, responsable de la gestion globale de lapplication. Il supervise la création et la validation des biens immobiliers, la gestion des contrats de location, lannulation des contrats, lassignation des tickets de maintenance, ainsi que la gestion des utilisateurs et des paiements.
- **Locataire (Tenant)** : utilisateur disposant dun compte lui permettant de consulter les biens disponibles, soumettre des demandes de location, effectuer des paiements de loyer ou de caution, consulter ses contrats, générer des quittances et créer des réclamations de maintenance.
- **Propriétaire (Owner)** : utilisateur pouvant consulter les biens lui appartenant, suivre les contrats associés ainsi que les paiements reçus.
- **Technicien (Tech)** : acteur chargé du traitement des tickets de maintenance qui lui sont assignés, depuis la prise en charge jusquà la clôture des interventions.
- **Fournisseur de paiement (PSP)** : acteur externe intervenant dans le processus de paiement électronique via la gestion des webhooks et la confirmation des transactions.

Le diagramme de cas dutilisation ci-dessous synthétise lensemble des fonctionnalités principales du système, réparties par domaines fonctionnels tels que la gestion des biens et des locations, les paiements, la maintenance et les notifications.

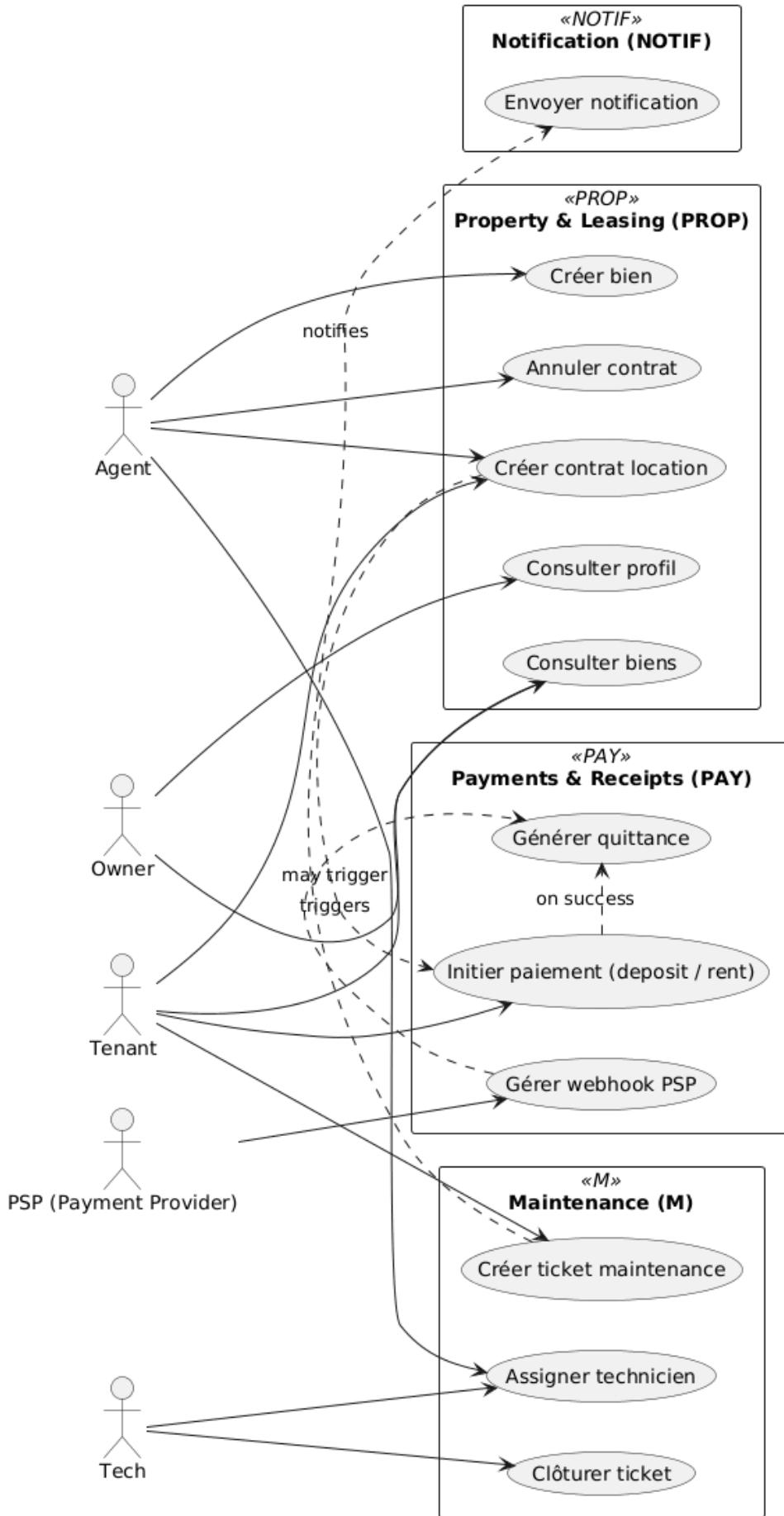


FIGURE 2.1 – Diagramme de cas d'utilisation de l'application

## 2.2.3 Diagrammes de séquence

### 2.2.3.1 Authentification

Ce diagramme décrypte le flux d'authentification : des identifiants soumis par le client à la génération des tokens JWT, en passant par la validation sécurisée via Spring Security et l'adaptation des rôles. Une vue d'ensemble des interactions critiques entre couches métier et sécurité.

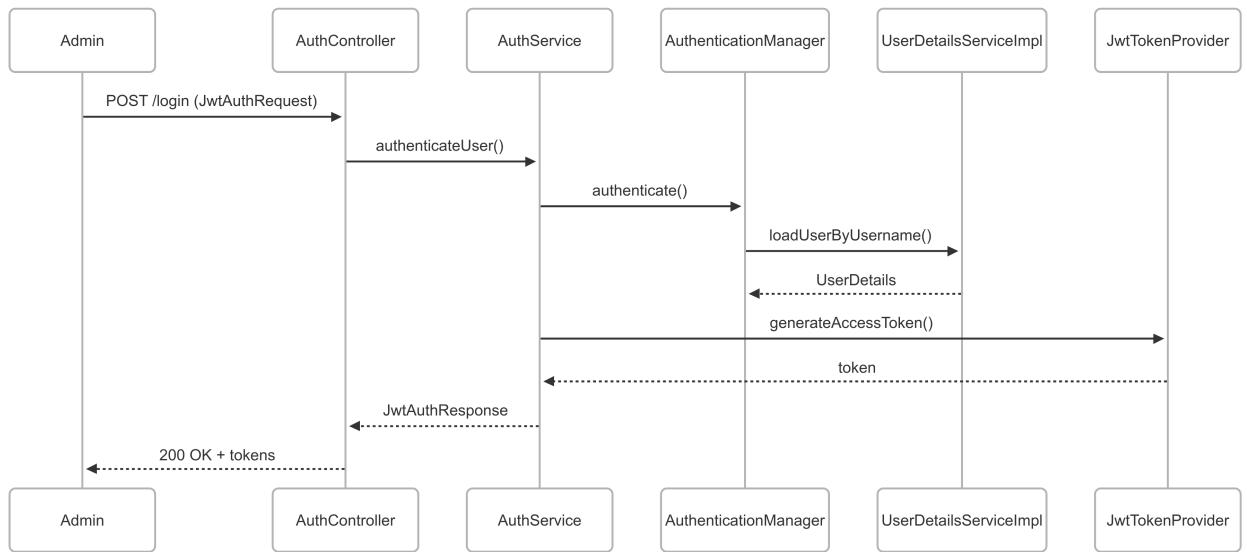


FIGURE 2.2 – Diagramme de séquence : login JWT

### 2.2.3.2 Crédation dun bien immobilier

Ce diagramme de séquence illustre le processus de création dun bien immobilier au sein du système. Linitiation de cette action se fait par ladministrateur via linterface frontend, qui transmet la requête au backend à travers lAPI Gateway.

Le service de gestion des biens est alors chargé de valider les informations fournies, notamment lexistence du propriétaire associé au bien ainsi que la conformité des données saisies. Une fois les validations effectuées avec succès, le bien est enregistré dans la base de données et un événement métier est généré afin de notifier les autres composants du système.

Le mécanisme doutbox est utilisé pour garantir la fiabilité de la communication événementielle, permettant la publication de lévénement de création vers le broker de messages. Le service de notification est ensuite déclenché afin dinformer les parties concernées de la création effective du bien.

Ce diagramme met en évidence la séparation des responsabilités entre les différentes couches du système, ainsi que l'utilisation d'une architecture orientée événements pour assurer la cohérence et la scalabilité de l'application.

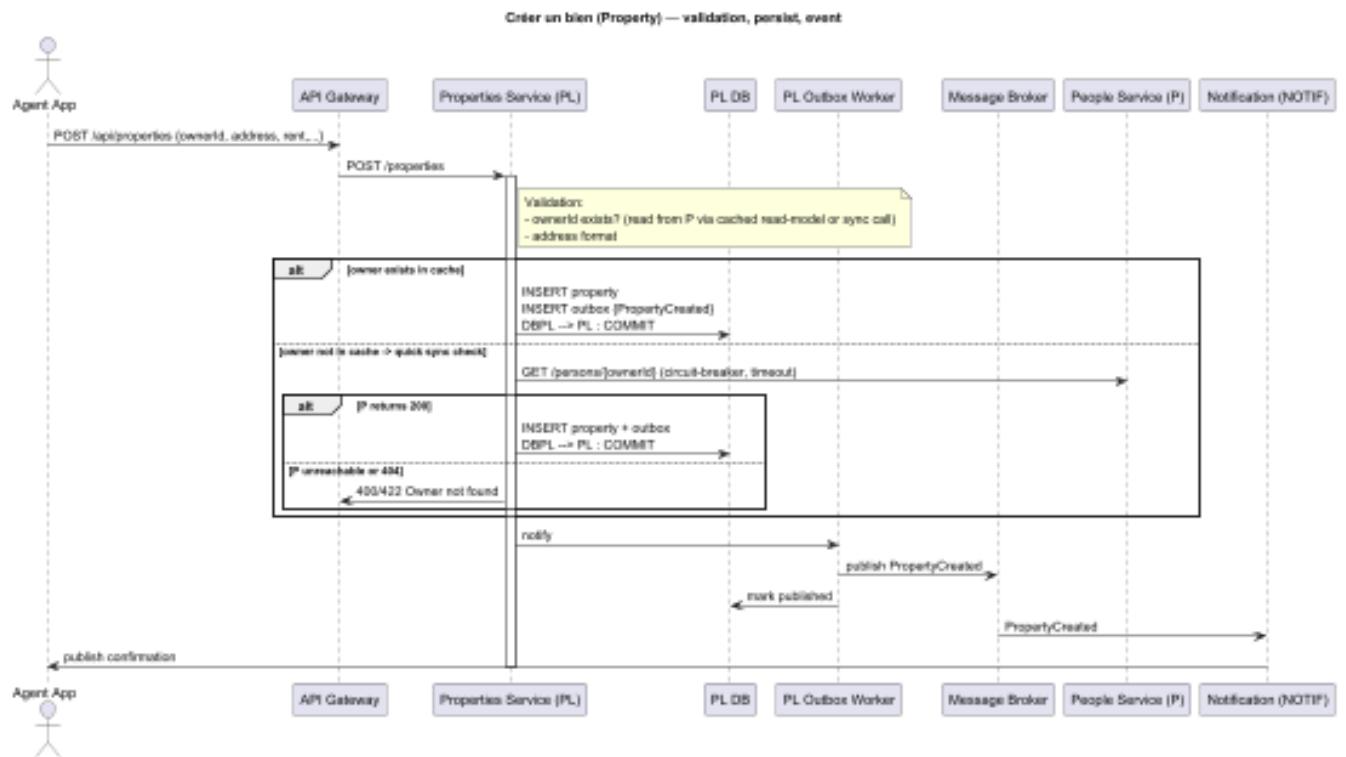


FIGURE 2.3 – Diagramme de séquence : create a Property

### 2.2.3.3 Résiliation dun contrat de location

Le diagramme de séquence de résiliation dun contrat décrit le déroulement complet de lannulation dun contrat de location, initiée soit par le locataire, soit par ladministrateur. La demande est envoyée au backend via lAPI Gateway et traitée par le service de gestion des biens et des contrats.

Le système commence par vérifier la validité de la demande, notamment les dates de résiliation et létat courant du contrat. Une fois les contrôles effectués, le contrat est mis à jour avec létat *résilié* et le bien concerné est automatiquement rendu disponible à la location.

Un événement de résiliation est ensuite enregistré et publié à travers le mécanisme doutbox, garantissant une diffusion fiable vers les autres services. Le service de notification est alors sollicité pour informer lutilisateur de la prise en compte de la résiliation.

Ce diagramme met en évidence la gestion transactionnelle des données, lautomatisation des mises à jour liées à létat des biens, ainsi que lintégration des notifications pour assurer un suivi clair et transparent des opérations effectuées.

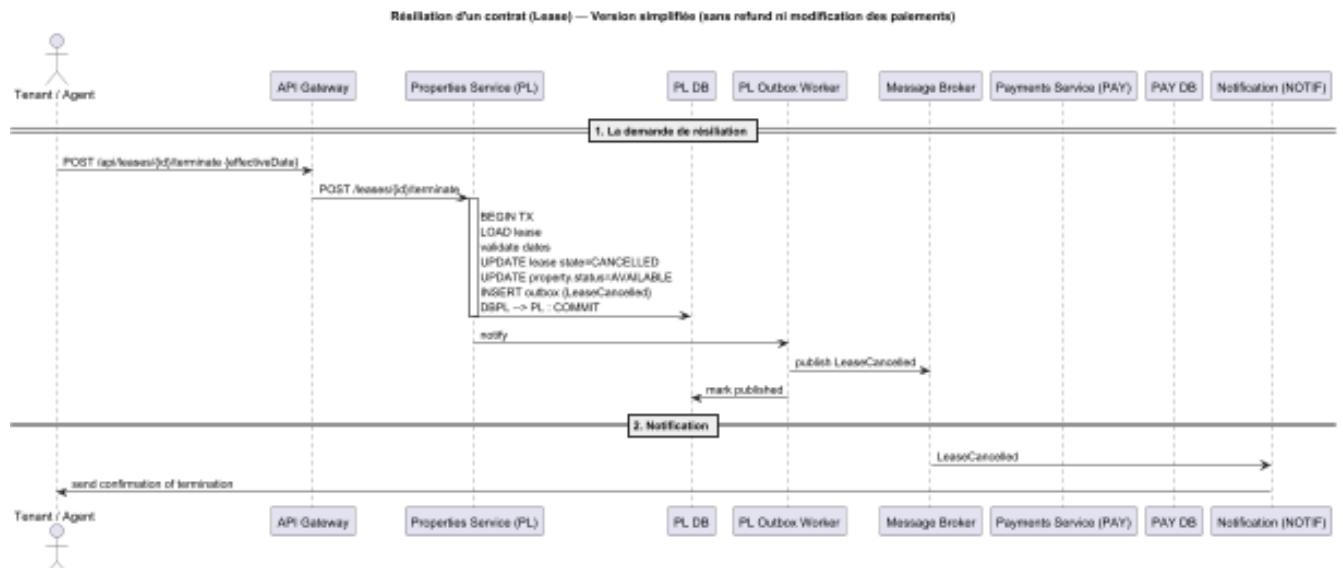


FIGURE 2.4 – Diagramme de séquence : Cancel a Contract

## Conclusion

Lanalyse et la conception ont permis de définir clairement les attentes du système et son organisation interne. Dans le chapitre suivant, nous décrirons le cadre technique et les outils utilisés pour développer la plateforme.

# CHAPITRE 3

## CADRE DE DÉVELOPPEMENT

Ce chapitre présente l'environnement technique, les choix technologiques et les outils utilisés lors de la réalisation du projet. Il met en lumière les raisons ayant motivé ces choix, afin de garantir un développement cohérent, sécurisé et maintenable.

### 3.1 Environnement de développement

**IntelliJ IDEA** est un IDE Java professionnel, renommé pour son assistance au codage, sa navigation intelligente et son refactoring avancé. Il prend en charge de multiples langages (Java, Kotlin, SQL) et facilite les opérations de build, test et debug grâce à son intégration avec Git et Docker. **Visual Studio Code** a été utilisé pour le développement du frontend Angular, bénéficiant d'extensions dédiées TypeScript, HTML et CSS. Enfin, **Docker Desktop** a permis de déployer localement les conteneurs PostgreSQL et MinIO, garantissant un environnement isolé et reproductible.



FIGURE 3.1 – IDE IntelliJ IDEA



FIGURE 3.2 – Visual Studio Code

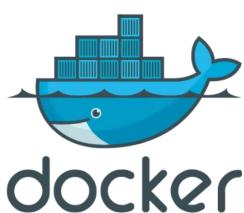


FIGURE 3.3 – Docker Desktop

## Contrôle de version

**Git** et **GitHub** ont été les piliers du versioning, collaboration et gestion des branches, facilitant les revues de code et l'historisation des modifications.



FIGURE 3.4 – Git

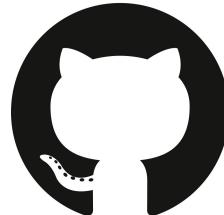


FIGURE 3.5 – Github

## Base de données et stockage

La base de données relationnelle **PostgreSQL** a été utilisée pour assurer la persistance des données métier de l'application. Elle a été déployée au sein d'un conteneur **Docker**, permettant de reproduire un environnement proche de la production et de faciliter le déploiement, la configuration et la portabilité du système.

PostgreSQL a été choisie pour sa robustesse, sa conformité aux standards SQL, ainsi que pour ses performances et sa fiabilité dans la gestion des transactions et de l'intégrité des données.



FIGURE 3.6 – PostgreSQL

## 3.2 Architecture Spring Adoptée

De nombreux *frameworks* de développement d'applications sont disponibles, mais l'un des plus populaires lorsque l'on travaille avec le langage Java est le **framework Spring**. Spring offre une grande souplesse et une capacité de gestion des applications remarquable.

### 3.2.1 Spring Boot



FIGURE 3.7 – Spring Framework

Spring Boot est un *module* de Spring Framework. Il est utilisé afin de construire rapidement des applications Spring autonomes, prêtes pour la production. Spring Boot s'appuie sur Spring Framework et adopte une **architecture en couches** (layered architecture) afin d'organiser le code de manière modulaire. Chaque couche communique avec la couche immédiatement en dessous ou au dessus d'elle.

### 3.2.2 Architecture en Couches

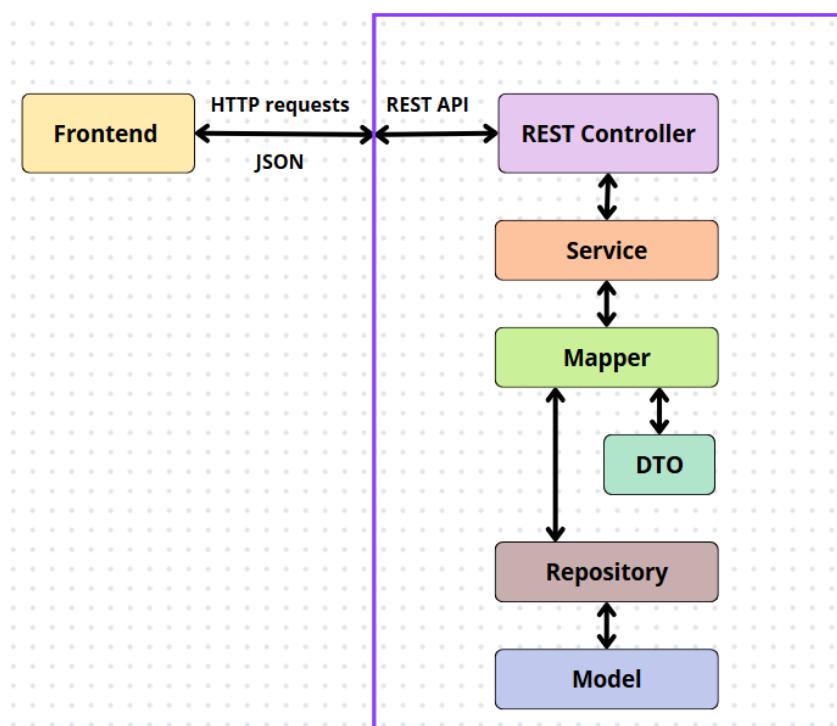


FIGURE 3.8 – Spring Boot Layered architecture

### **3.2.2.1 Couche Contrôleur (Controller Layer)**

Cette couche est responsable de la gestion des *reqûetes* entrantes. Les contrôleurs reçoivent les requests, les traitent, puis invoquent le service afin de récupérer ou manipuler des données. Ils sont généralement annotés avec `@Controller` ou `@RestController`.

### **3.2.2.2 Couche Service (Service Layer)**

Elle encapsule la *logique métier* de lapplication et sert dintermédiaire entre le contrôleur et la couche daccès aux données. Les services vont appliquer toutes les opérations de lentreprise afin dassurer la cohérence des cas dutilisation. Ils sont généralement annotés avec `@Service`.

### **3.2.2.3 Couche Repository/DAO (Repository Layer)**

Sa fonction est dinteragir avec la base de données ou avec une source de données externe. Elle consiste en des interfaces ou des classes définissant des méthodes CRUD (Créer, Lire, Mettre à jour, Supprimer). Ces repositories sont généralement annotés avec `@Repository`.

### **3.2.2.4 Couche Entité (Entity Layer)**

Les entités *représentent le modèle de données de lapplication* et sont généralement associées à des tables de la base de données. Ces classes contiennent des champs, généralement annotés avec `@Entity`, `@Column`, etc.

### **3.2.2.5 DTO (Data Transfer Object)**

Les DTO permettent de véhiculer des données entre les couches sans exposer directement le modèle de données de lentité. Ils permettent de structurer et de formater les données en fonction des besoins de la représentation.

### **3.2.2.6 Couche de Sécurité (Security Layer)**

Elle gère laauthentification, lautorisation, et la protection contre certaines failles de sécurité. *Spring Security* est généralement utilisée afin de rendre lapplication plus robuste.

### **3.2.2.7 Couche de Configuration (Configuration Layer)**

Elle regroupe toutes les classes de configuration de lapplication. Ces classes sont généralement annotées avec `@Configuration`, `@Bean` ou définies dans des fichiers de configuration

XML.

### 3.2.3 Workflow d'une Requête

- Un client envoie une *requête HTTP* à l'application.
- La couche *Controller* la prend en charge et invoque le service approprié.
- La couche *Service* applique la logique métier et fait appel, si besoin, à la couche *Repository* afin d'accéder ou de manipuler les données.
- La couche *Repository* exécute les opérations CRUD sur la base de données.
- La couche *Service* retourne le résultat, généralement sous la forme d'un DTO.
- La couche *Controller* renvoie ce résultat en tant que *réponse HTTP* au client.

### 3.2.4 Bénéfices

- **Modularité** : chaque couche a une responsabilité définie, ce qui facilite la maintenance.
- **Séparation des responsabilités** : la représentation, la logique métier et l'accès aux données sont découplés.
- **Testabilité** : chaque couche peut être testée de manière indépendante.
- **Scalabilité** : les couches peuvent être dimensionnées et déployées de manière indépendante en fonction des besoins de l'application.

## 3.3 Spring Boot REST API

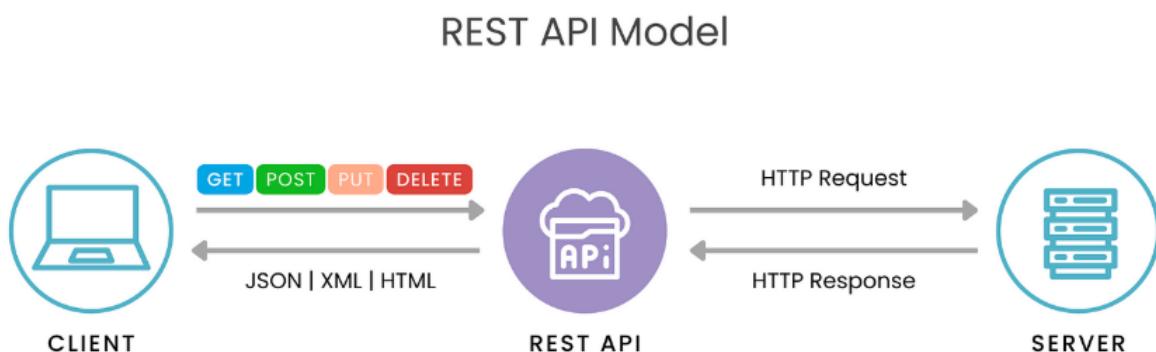


FIGURE 3.9 – REST API Model

REST (Representational State Transfer) a été développé par **Roy Thomas Fielding** afin d'utiliser de manière efficace le **protocole HTTP**. REST est donc une architecture, pas

un standard, designed to expose resources through a uniform and lightweight interface. C'est le style d'API le plus répandu aujourd'hui.

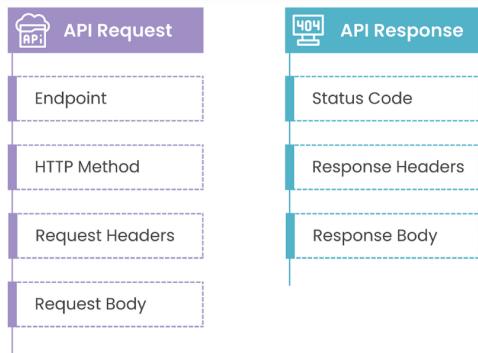


FIGURE 3.10 – Components of an API Request and API Response

### 3.3.1 Principes des services web REST

Les principaux principes sont :

- **Resource Identification through URI** : Chaque ressource est identifiée par une URL unique.
- **Uniform Interface** : Un jeu d'opérations fixe (GET, POST, PUT, DELETE) est exposé afin d'interagir avec les données.
- **Self-descriptive messages** : La représentation des données est indépendante de leur représentation (JSON, XML).
- **Stateful Interaction through hyperlinks** : Chaque appel est *stateless*, sans contexte de la demande précédente.

### 3.3.2 Les méthodes fondamentales de HTTP

REST s'aligne sur le jeu de méthodes du protocole **HTTP** :

#### 3.3.2.1 GET

Récupère une ressource.

```
@GetMapping("/user/{userId}")  
public ResponseEntity<Object> getUser(@PathVariable int userId) {  
    UserEntity user = userService.getUser(userId);  
    return new ResponseEntity<>(user, HttpStatus.OK);  
}
```

**Exemple :** GET /employees retourne toutes les données des employés.

### 3.3.2.2 POST

Crée une nouvelle ressource.

```
@PostMapping("/user")
public ResponseEntity<Object> addUser(@RequestBody UserEntity user) {
    userService.saveOrUpdate(user);
    return new ResponseEntity<>("User is created successfully",
        HttpStatus.CREATED);
}
```

**Exemple :** POST /employees crée un nouvel employé.

### 3.3.2.3 PUT

Met à jour une ressource existante.

```
@PutMapping("/user/{userId}")
public ResponseEntity<Object> updateUser(@PathVariable int userId,
    @RequestBody UserEntity user) {
    userService.saveOrUpdate(user);
    return new ResponseEntity<>("User is updated successfully", HttpStatus.OK);
}
```

**Exemple :** PUT /concourss/{id} modifie les détails d'un concours.

### 3.3.2.4 DELETE

Supprime une ressource.

```
@DeleteMapping("/user/{userId}")
public ResponseEntity<Object> deleteUser(@PathVariable int userId) {
    userService.deleteUser(userId);
    return new ResponseEntity<>("User is deleted successfully", HttpStatus.OK);
}
```

**Exemple :** DELETE /concourss/{id} supprime un concours particulier.

### 3.3.3 HTTP Standard Status Codes

Les principaux codes de statut sont :

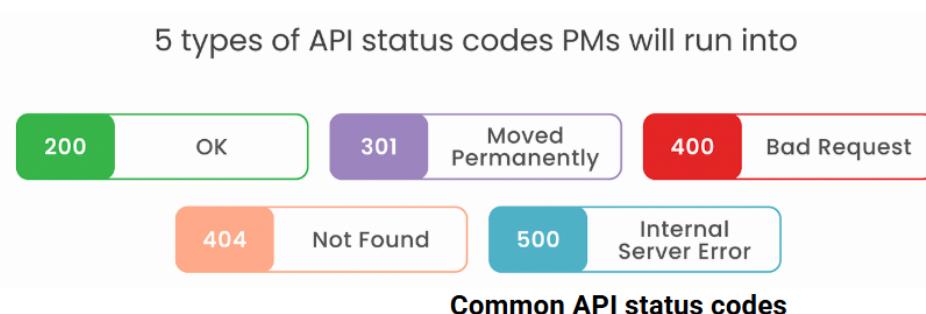


FIGURE 3.11 – Status Code

- **200 : Success**
- **201 : Created**
- **401 : Unauthorized**
- **404 : Not Found**
- **500 : Internal Server Error**

### 3.3.4 Usages de Spring Boot avec REST

Les services REST sont :

- **Complètement sans état (Stateless)**
- **Évolutif et léger**
- Naturellement cacheable avec GET
- Conçu idéalement pour exposer le back-end à différents clients (Web, Mobile).

### 3.3.5 Bonnes pratiques de sécurité des API REST

Pour assurer la sécurité de l'API :

- **Authentication and Authorization** : Spring Security + JWT.
- **Rate Limiting** : Empêcher certains abus.
- **Input Validation** : Validation des données entrantes contre les injection attacks.
- **Enforce HTTPS** : Communication chiffrée afin d'éviter l'espionnage.

### 3.3.6 Bonnes pratiques de développement d'une API REST avec Spring Boot

- **Project Structure** : Adopter le *package-by-feature* afin de rendre le code plus modulaire.
- **API Versioning** : Inclure le numéro de version dans l'URL (/api/v1).
- **Security** : Utilisation de Spring Security avec JWT.
- **Performance** : Caching, compression des données, et optimisation des méthodes d'accès.

## 3.4 API Documentation

Afin de documenter et tester l'API REST développée dans ce projet, l'outil **Swagger**, reposant sur la spécification **OpenAPI**, a été intégré via la dépendance `springdoc-openapi`. Swagger est une suite d'outils open-source conçus pour **décrire**, **visualiser**, **tester** et **générer** des API de manière interactive et standardisée.



FIGURE 3.12 – Swagger

Techniquement, `springdoc-openapi` scanne les contrôleurs Spring à l'exécution, interprète leurs annotations (`@RestController`, `@GetMapping`, etc.), et produit automatiquement un fichier OpenAPI (JSON ou YAML) servi par Swagger UI. Il offre aussi une interface Swagger UI hébergée au chemin `/swagger-ui.html`, qui a facilité les tests de toutes les routes pendant le développement.

En résumé, Swagger (OpenAPI) rend la documentation de l'API **interactif**, **automatique**, **vivante** et **professionnelle**, tout en renforçant la qualité, la fiabilité et la maintenabilité du projet.

### 3.4.1 Outils de build et gestion de dépendances : Gradle

Le projet utilise **Gradle**, un système d’automatisation de build open source largement adopté dans les écosystèmes Java, Android et Kotlin. Gradle se distingue par l’utilisation d’un **DSL** (Domain-Specific Language) basé sur Groovy ou Kotlin, offrant une **flexibilité** et une **expressivité** bien supérieures aux formats XML traditionnels comme ceux de Maven.



FIGURE 3.13 – Gradle build

Gradle repose sur un **graphe acyclique dirigé (DAG)** pour gérer l’ordre d’exécution des tâches, ce qui permet des **builds incrémentaux** et **l'autocache** des artefacts, réduisant significativement le temps de compilation lors des cycles successifs. De plus, le **Gradle Daemon**, processus JVM persistant, accélère les builds en évitant la surcharge du démarrage d’une machine virtuelle Java à chaque exécution.

Un autre atout majeur est la gestion des dépendances intégrée, permettant de déclarer facilement des bibliothèques via Maven Central ou d’autres dépôts, sans gestion manuelle des fichiers JAR. L'utilisation du **Gradle Wrapper** garantit que tous les membres de l'équipe utilisent la même version de Gradle, assurant une **reproductibilité** parfaite du build dans tous les environnements.

Enfin, Gradle est hautement **extensible** grâce à un large écosystème de plugins (pour Spring Boot, Docker, tests, documentation), ainsi que des capacités **d'automatisation avancées** et de **types de tâches personnalisées**, ce qui s'est avéré crucial pour structurer et industrialiser notre chaîne de build.

## 3.5 Spring Security

Spring est un écosystème Java largement reconnu et utilisé, qui regroupe plusieurs frameworks, parmi lesquels Spring Security. Ce dernier est un cadre puissant et personnalisable dédié à l'authentification et à l'autorisation des applications Spring, et constitue la référence de facto pour sécuriser les applications basées sur Spring. Ainsi, pour implémenter une solution d'authentification par jeton JWT (JSON Web Token), il est naturel de s'appuyer sur Spring Security.

Spring Security est le framework de référence de Spring destiné à assurer l'authentification et l'autorisation des applications. Son architecture s'organise en plusieurs couches spécialisées, ce qui le rend particulièrement flexible et robuste.

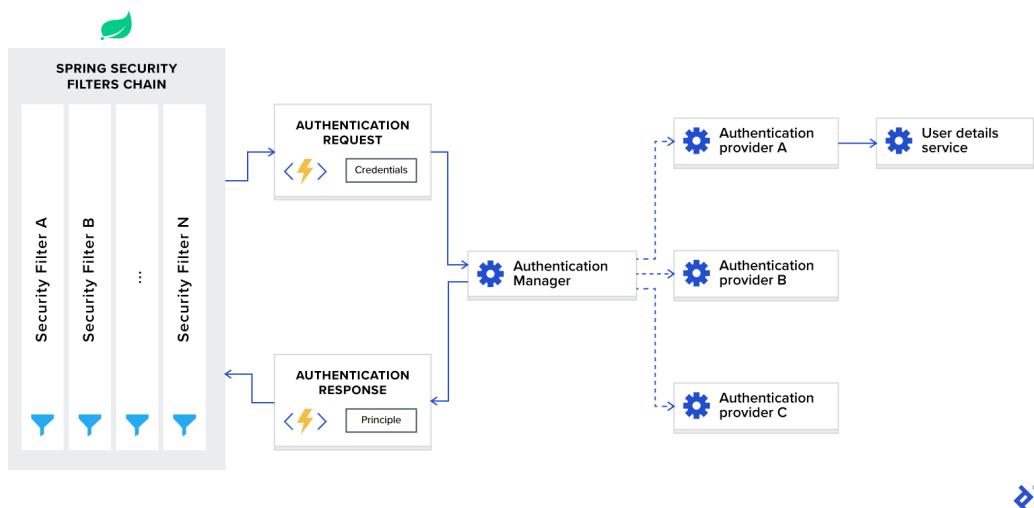


FIGURE 3.14 – Spring Security Architecture

### 3.5.1 Spring Security Filters Chain

Lorsque Spring Security est ajouté à l'application, une **chaîne de filtres** (filter chain) est automatiquement enregistrée afin d'intercepter toutes les requêtes entrantes. Cette chaîne est composée de divers filtres, chacun ayant une fonction bien définie. Par exemple :

- vérifier si l'URL demandée est publique en se basant sur la configuration ;
- en cas d'authentification par session, assurer que l'utilisateur est déjà authentifié ;
- lancer le contrôle d'autorisation afin de décider si l'utilisateur a le droit de effectuer l'action demandée.

Les filtres Spring Security sont placés en priorité la plus basse afin d'être invoqués en premier. Si l'on a besoin d'en ajouter d'autres avant, on peut régler leur ordre avec la

propriété :

```
spring.security.filter.order = 10
```

Ainsi, jusqu'à 10 filtres custom peuvent s'intercaler avant les filtres Spring Security.

### 3.5.2 AuthenticationManager

L'`AuthenticationManager` joue le rôle de coordinateur de l'autentification. Lorsqu'une demande d'autentification est faite, il la redirige vers le `AuthenticationProvider` approprié en fonction du cas d'utilisation.

### 3.5.3 AuthenticationProvider

L'`AuthenticationProvider` est chargé d'authentifier l'utilisateur. Il expose deux méthodes principales :

- `authenticate` : vérifie les identifiant fournis dans la demande d'autentification ;
- `supports` : détermine si le `AuthenticationProvider` prend en charge le type d'autentification demandé.

Dans la plupart des cas, le `DaoAuthenticationProvider` est utilisé. Ce dernier s'appuie sur le `UserDetailsService` afin de récupérer les détails de l'utilisateur en base de données.

### 3.5.4 UserDetailsService

Le `UserDetailsService` est une interface fondamentale de Spring Security. Elle consiste en une unique méthode :

```
loadUserByUsername(username);
```

Elle a pour mission de retrouver l'utilisateur (et ses rôles/autorités) en base de données en utilisant le nom d'utilisateur. Ainsi, le `DaoAuthenticationProvider` peut se servir afin de valider le mot de passe et d'autentifier l'utilisateur.

## 3.6 JWT avec Spring Security

Un JSON Web Token (JWT) est un format compact et autonome permettant de transmettre de manière sécurisée des informations entre deux parties sous forme d'objet JSON. Il garantit l'intégrité et l'authenticité des données sans nécessiter de gestion de session côté serveur, car le jeton est signé numériquement à l'aide d'une clé secrète partagée (HMAC) ou d'une paire de clés publique/privée (RSA ou ECDSA). Cette caractéristique rend les JWT particulièrement adaptés aux environnements stateless et aux applications modernes telles que les API REST ou les systèmes distribués.

La structure d'un JWT comprend trois parties codées en Base64Url et concaténées :

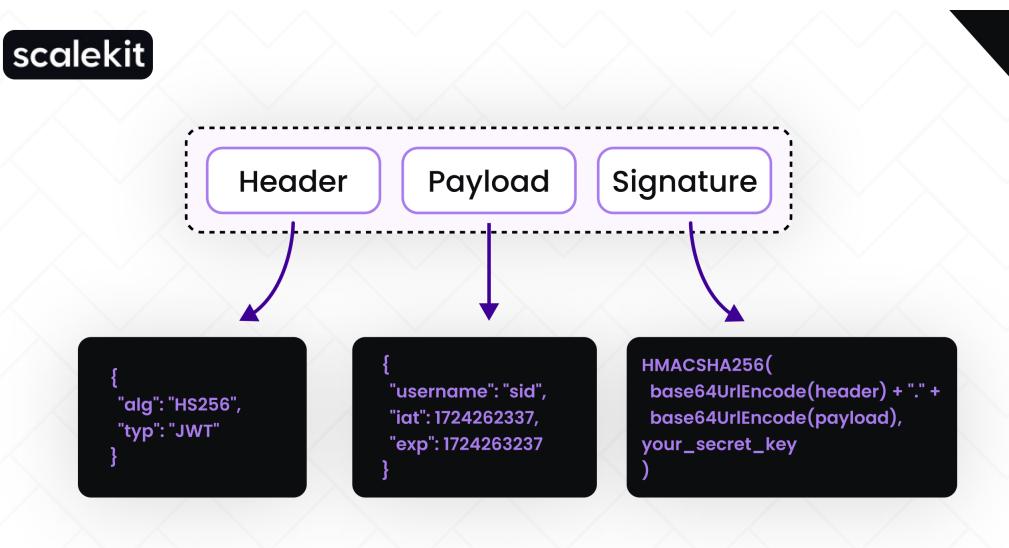


FIGURE 3.15 – JWT Structure

- **Header** : spécifie le type de jeton (JWT) et l'algorithme de signature (par exemple HMAC SHA256).
- **Payload** : contient les claims, c'est-à-dire les informations sur l'utilisateur ou d'autres données. Ces claims peuvent être enregistrés, publics ou privés.
- **Signature** : assure l'intégrité du jeton en combinant le header, le payload, une clé secrète et l'algorithme de signature, empêchant toute falsification.

L'utilisation de JWT permet un système d'authentification *stateless*, où chaque requête inclut le jeton JWT dans la tête **Authorization**. Le serveur vérifie la validité et la signature du jeton, sans stocker l'état de session côté serveur. Cela améliore la scalabilité et la réactivité, particulièrement dans les architectures distribuées. Cependant, il est conseillé

de stocker les JWT dans des cookies HTTPOnly et Secure pour limiter les risques de vol de jeton, plutôt que dans le stockage local du navigateur.

Enfin, Spring Security facilite la mise en place d'une gestion fine des accès grâce à une autorisation basée sur les rôles (`ROLE_USER_ADMIN`, `ROLE_AUTHOR_ADMIN`, `ROLE_BOOK_ADMIN`), appliqués aux différentes routes REST, soit via une configuration par URL, soit par annotations.

Ainsi, l'intégration de JWT avec Spring Security dans mon backend assure une authentification sécurisée, efficace et adaptée aux applications modernes, tout en offrant un contrôle précis des droits d'accès utilisateurs.

## 3.7 JPA/Hibernate

La **Java Persistence API** (JPA), récemment renommée *Jakarta Persistence*, est une spécification standard de persistance d'objets relationnels en Java, définie notamment par le langage JPQL et l'API `EntityManager`.

Elle permet de mapper des classes Java (POJOs) vers des tables de bases de données, tout en offrant portabilité et abstraction face aux différents fournisseurs de bases de données.

**Hibernate ORM** est l'implémentation la plus répandue de JPA, développée par Red Hat depuis 2001. Ce framework open-source propose :

- le mappage automatique entre les entités Java et les tables SQL, via annotations ou fichiers XML ;
- un cache de premier et de second niveau pour optimiser les accès à la base ;
- le langage HQL (Hibernate Query Language), similaire à SQL mais orienté objets ;
- la gestion des transactions, du lazy loading et des relations *un* à plusieurs, *plusieurs* à plusieurs, etc.

En utilisant **Spring Data JPA**, on bénéficie d'une intégration fluide entre Spring et JPA/Hibernate, avec l'automatisation des répertoires (repositories CRUD) et une réduction significative du code pompe au niveau de l'accès aux données. Ce duo (JPA + Hibernate) permet d'augmenter la productivité, de maintenir une base de code propre et portable, et de garantir des performances optimisées via la mise en cache et la gestion simplifiée des entités.

## 3.8 MapStruct

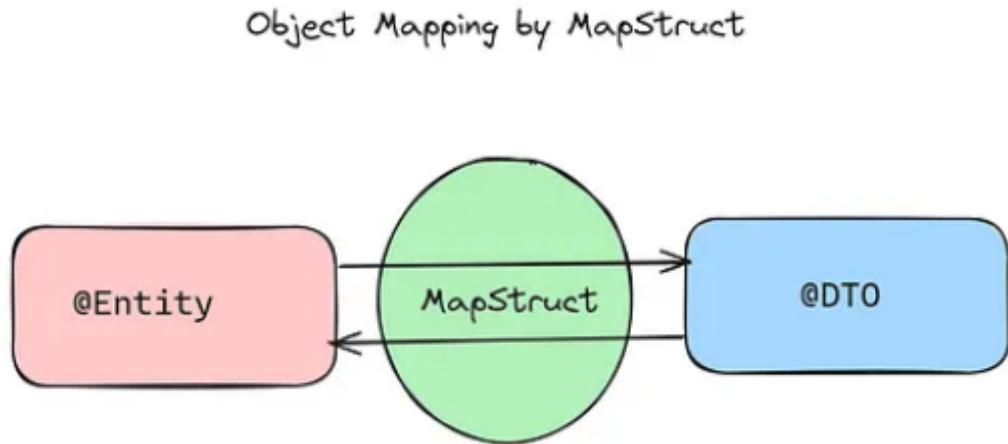


FIGURE 3.16 – Mapstruct Object Mapping

MapStruct est une bibliothèque Java spécialisée dans le *object mapping*, c'est-à-dire la conversion entre objets de types différents (par exemple, entités JPA et DTO), courante dans les architectures multicouches. Elle se distingue par sa génération de code de mapping **à la compilation** via des annotations (`@Mapper`, `@Mapping`), garantissant des implémentations efficaces, typées et lisibles, sans surcharge à lexécution.

### Avantages

- **Performance** : le code généré appelle directement les getters/setters, évitant la réflexion, et sexécute au moins aussi vite que du code manuel.
- **Sécurité au typage** : les erreurs de mapping sont détectées dès la compilation, avec retour d'erreur immédiat.
- **Réduction du code boilerplate** : on se concentre sur l'interface de mapping, MapStruct génère l'implémentation automatiquement.
- **Lisibilité et maintenabilité** : les annotations offrent une vision claire du mapping, tandis que le code généré reste simple à comprendre et à déboguer.

**Points de vigilance** Bien que globalement facile à démarrer, MapStruct nécessite des annotations explicites pour chaque interface de mapping et peut demander un certain temps d'adaptation pour des cas complexes.

**Conclusion** Dans les applications Spring Boot métiers, l'utilisation de *MapStruct* permet de générer des mappings typés, performants et sûrs, tout en minimisant la quantité

de code manuel requis. Comparable au code écrit à la main en termes de performance, il offre en plus une sécurité accrue et une meilleure maintenabilité.

### 3.8.1 Project Lombok : réduction du code boilerplate en Java

Project Lombok est une bibliothèque Java qui utilise le traitement d'annotations à la compilation pour générer automatiquement du code répétitif (getters, setters, constructeurs, `equals/hashCode`, etc.), via l'injection dans l'AST du compilateur, sans surcoût à l'exécution.

#### Fonctionnalités clés

- `@Getter`, `@Setter`, `@ToString`, `@EqualsAndHashCode`: génération automatique de méthodes communes.
- `@NoArgsConstructor`, `@AllArgsConstructor`, `@RequiredArgsConstructor`: constructeurs générés selon les besoins.
- `@Data`: annotation tout-en-un regroupant getters, setters, `toString`, ... .
- `@Builder`, `@Value`, `val/var`: support du pattern builder, immutabilité et inférence locale.

#### Avantages

- **Réduction du code répétitif**, gain de lisibilité, notamment pour les POJO/DTO/entités.
- **Pas de surcharge runtime**: le code est injecté à la compilation, aucun reflet n'est utilisé à l'exécution.
- **Productivité accrue**, car on se concentre sur la logique métier, non sur le code de support.

### 3.9 MinIO pour la gestion des fichiers



FIGURE 3.17 – MinIO

Minio est un serveur de stockage d'objets distribué à hautes performances, conçu pour une infrastructure de cloud privée à grande échelle. Minio agrège des volumes persistantes en stockage d'objets distribué en utilisant des API REST Amazon S3.

Il est idéal pour stocker des données non structurées comme des photos, des vidéos, des fichiers journaux, des sauvegardes, des machines virtuelles et des images de conteneur.

### 3.10 Technologies Front-end



FIGURE 3.18 – CSS



FIGURE 3.19 – HTML



FIGURE 3.20 – TypeScript

#### HTML

HTML (HyperText Markup Language) est le langage de balisage standard utilisé pour structurer et organiser le contenu d'une page Web, incluant textes, images, liens et formulaires. Il définit la sémantique du contenu, permettant une accessibilité optimale, une indexation par les moteurs de recherche et une compatibilité universelle via les navigateurs.

## CSS

CSS (Cascading Style Sheets) est un langage de feuille de style qui sépare la présentation du contenu (HTML). Il permet de contrôler l'apparence visuelle : mise en page, couleurs, polices, alignements et responsive design. Grâce à l'héritage et à la cascade, les styles peuvent être définis globalement et réutilisés dans toute l'application.

## TypeScript

TypeScript est un superset typé de JavaScript qui compile vers du JavaScript standard. Il introduit le typage statique optionnel, les interfaces, les classes et les generics, renforçant la sécurité des types et la maintenabilité du code. Son adoption améliore l'autocomplétion, la détection d'erreurs à la compilation et la robustesse des projets, en particulier dans les environnements JavaScript de grande ampleur.

### 3.10.1 Angular

Angular est un framework open-source développé par Google, écrit en TypeScript, et conçu pour créer des applications web monopage robustes, modulaires et scalables : contentReference[oaicite :1]index=1. Il propose une architecture par composants, un routage, l'injection de dépendances, des formulaires avancés, et des outils CLI pour faciliter le développement.



FIGURE 3.21 – Angular

**Mise à jour majeure : version 18 (22 mai 2024)** Cette version introduit plusieurs améliorations notables :

- **Détection de modifications "zoneless"** en mode expérimental, basée sur des Signaux, pour améliorer les performances et réduire la taille des bundles.
- **Orchestration des routes et redirections** via des fonctions, plus flexible et adaptée aux scénarios dynamiques.
- **Routage par composants autonomes** et autres raffinements sur les fonctionnalités de v17 (views différables, flux de contrôle intégré, Material 3), désormais matures.

- **Améliorations SSR** (prérendu, hydratation, event replay) et meilleure compatibilité avec Angular CDK/Material.
- **Support de TypeScript 5.4** (précédemment 5.3) et optimisation du CLI, de la compilation (Ivy, esbuild), réduisant le temps de build et la taille des bundles.

Angular 18 constitue une mise à jour stratégique offrant des gains de performance (détection zoneless), une plus grande flexibilité (routing fonctionnel), de meilleures capacités pour le SSR, et un alignement sur les dernières versions de TypeScript idéal pour renforcer l'expressivité et l'évolutivité des interfaces dans les applications web modernes.

## Conclusion

Cette panoplie d'outils reflète une architecture moderne et professionnelle, garantissant performance, maintenabilité, sécurité et qualité de développement logiciel. Après avoir posé les bases techniques du projet, nous allons dans le chapitre suivant décrire concrètement la mise en œuvre de la plateforme, en illustrant les principales interfaces développées et les fonctionnalités implémentées.

# CHAPITRE 4

## MISE EN UVRE

### 4.1 Présentation des résultats

Cette section présente les principales interfaces développées pour la plateforme de gestion immobilière. Elle met en évidence les fonctionnalités offertes à chaque type d'utilisateur à travers les interfaces web réalisées avec Angular.

#### 4.1.1 Application dadministration

##### 4.1.1.1 Page login

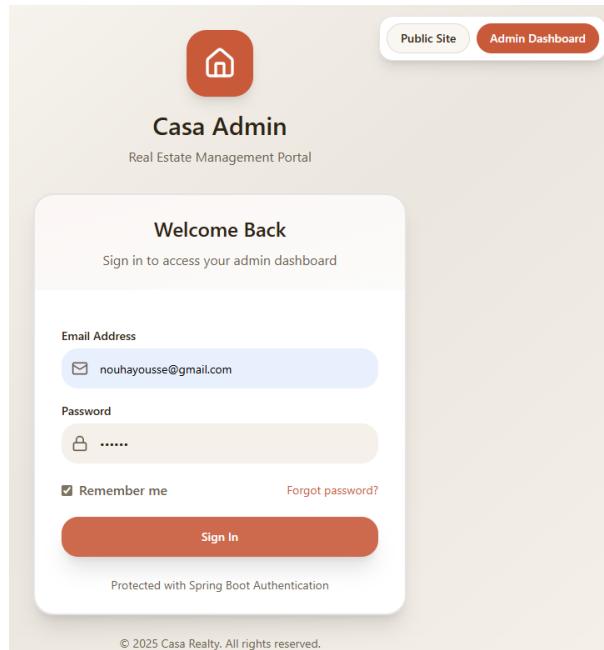


FIGURE 4.1 – Page login pour Admin

Cette page permet aux agents immobiliers et administrateurs d'accéder à l'application interne de gestion. Elle propose un formulaire d'authentification sécurisé reposant sur la saisie des identifiants utilisateur. L'accès aux fonctionnalités de gestion est protégé par un mécanisme d'authentification basé sur JWT, garantissant que seules les personnes autorisées peuvent accéder au tableau de bord administratif.

#### 4.1.1.2 Tableau de bord administrateur



FIGURE 4.2 – dashboard admin avec stats

Le tableau de bord administrateur offre une vue globale et synthétique de l'activité de la plateforme. Il présente des indicateurs clés tels que le nombre total de biens, les locations actives, les validations en attente, ainsi que le nombre de propriétaires et de locataires. Des cartes statistiques et des graphiques facilitent le suivi de l'état du système et aident l'administrateur à prendre des décisions rapidement.

#### 4.1.1.3 Gestion des biens immobiliers

The screenshot shows the 'Agent Dashboard' interface for managing a real estate portfolio. On the left is a sidebar with navigation links: Dashboard, Properties (highlighted in orange), Validations, Owners, Tenants, Contracts, Complaints, Requests, Profile, and Settings. The main area displays a search bar, a status filter dropdown set to 'Validated', and a button for 'More Filters'. Below this is a section titled 'All Properties (8)' with the sub-instruction 'Manage property listings and details'. A table lists eight properties with columns for Property, Location, Type, Owner, Price, Status, and Action. Each property row includes a small icon, the property name, location, type, owner's name, price, status (e.g., 'Validated', 'Pending', 'Rejected'), and a series of circular icons for different actions.

FIGURE 4.3 – le Crud pour gestion Biens

Cette interface permet à l'administrateur de gérer l'ensemble des biens immobiliers proposés sur la plateforme. Elle affiche la liste des biens selon leur statut (validé, en attente ou rejeté) et offre des actions de consultation, validation ou refus. Cette étape garantit que seuls les biens conformes aux règles de la plateforme sont publiés sur l'application publique.

The screenshot shows the validation screen for a property named 'Cozy Studio'. The sidebar on the left is identical to Figure 4.3. The main area shows the property details: 'Cozy Studio', 'Valencia, Spain', 'Owner: Luis Sanchez', and 'Submitted: 2025-01-15'. A 'Pending Review' button is visible. Below this, there are two sections: 'Property Information' and 'Description'. 'Property Information' contains cards for Type (Studio), Price (€750/mo), Bedrooms (1), Bathrooms (1), Area (35 m<sup>2</sup>), and Images (3 photos). The 'Description' section contains a text block: 'Beautiful studio in the heart of Valencia with modern amenities.' At the bottom, there is a 'Validation Notes (Optional)' field containing the message 'Les papiers demandés ne sont pas fournis ! et certains sont invalides ou expirés.' and two buttons: 'Approve Property' and 'Reject Property'.

FIGURE 4.4 – Valider un bien

#### 4.1.1.4 Gestion des propriétaires

Owner	Email	Phone	Properties	Status	Actions
Carlos Rodriguez	carlos@email.com	+34 612 345 678	3 properties	active	<button>View Details</button>
Ana Martinez	ana@email.com	+34 623 456 789	2 properties	active	<button>View Details</button>
Sofia Lopez	sofia@email.com	+34 634 567 890	1 properties	active	<button>View Details</button>
Miguel Torres	miguel@email.com	+34 645 678 901	4 properties	active	<button>View Details</button>
Isabel Ruiz	isabel@email.com	+34 656 789 012	2 properties	inactive	<button>View Details</button>

FIGURE 4.5 – le crud pour gestion des owners

La page de gestion des propriétaires permet à l'administrateur de consulter les profils des propriétaires, d'accéder à leurs biens publiés et de suivre leur activité. Elle facilite également le traitement des demandes de passage au rôle de propriétaire soumises par les utilisateurs.

2
Pending owner requests

**Roberto Silva**

Email: roberto@email.com  
Phone: +34 678 901 234  
Properties to list: 2  
Request date: 2025-01-22

Approve as Owner
 Reject Request

**Marina Vega**

Email: marina@email.com  
Phone: +34 689 012 345  
Properties to list: 1  
Request date: 2025-01-21

Approve as Owner
 Reject Request

FIGURE 4.6 – traiter la demande de profil Owner

#### 4.1.1.5 Gestion des locataires

Tenant	Email	Property	Tenant Since	Status	Actions
Elena Garcia	elena@email.com	Mediterranean Villa	2024-06-15	active	<button>View Details</button>
David Fernandez	david@email.com	Modern Apartment	2024-08-20	active	<button>View Details</button>
Laura Gomez	laura@email.com	Historic Townhouse	2024-03-10	active	<button>View Details</button>
Pablo Herrera	pablo@email.com	Luxury Penthouse	2024-11-05	active	<button>View Details</button>
Lucia Moreno	lucia@email.com	Garden Cottage	2025-01-10	pending	<button>View Details</button>

FIGURE 4.7 – le crud pour gestion des Tenants

Cette interface permet de visualiser et gérer les comptes des locataires inscrits sur la plateforme. Ladministrateur peut consulter leur historique de contrats, leurs demandes de location ainsi que les réclamations associées, assurant ainsi un suivi complet du cycle locatif.

#### 4.1.1.6 Gestion des contrats

Property	Tenant	Owner	Start Date	End Date	Status	Actions
Mediterranean Villa	Elena Garcia	Carlos Rodriguez	2024-06-15	2025-06-14	active	<button>View</button>
Modern Apartment	David Fernandez	Ana Martinez	2024-08-20	2025-08-19	active	<button>View</button>
Historic Townhouse	Laura Gomez	Sofia Lopez	2024-03-10	2025-03-09	active	<button>View</button>
Luxury Penthouse	Pablo Herrera	Carmen Diaz	2024-11-05	2025-11-04	active	<button>View</button>
Garden Cottage	Lucia Moreno	Juan Morales	2025-01-10	2026-01-09	pending	<button>View</button>

FIGURE 4.8 – Le crud pour gestion des Contrats

La page de gestion des contrats centralise l'ensemble des contrats de location. Elle permet de consulter les contrats actifs, expirés ou résiliés, ainsi que de suivre les informations liées aux biens, aux locataires et aux propriétaires. Cette fonctionnalité assure une gestion cohérente et structurée des relations contractuelles.

#### 4.1.1.7 Gestion des reclamations

The screenshot displays the 'Agent Dashboard' interface for managing real estate portfolios. The left sidebar, titled 'Casa Admin Agent Portal', contains links for Dashboard, Properties, Validations, Owners, Tenants, Contracts, Complaints (which is currently selected and highlighted in orange), and Requests. The right side features a summary of ticket counts: Open (1), In Progress (1), and Resolved (1). Below this, two specific complaints are listed:

- Heating not working**: Property: Modern Apartment • Submitted by: David Fernandez • 2025-01-20. Status: high priority, open. Response / Notes: "we will contact you about this ASAP". Buttons: Mark as Resolved, Update Status.
- Leaking faucet in bathroom**: Property: Garden Cottage • Submitted by: Lucia Moreno • 2025-01-18. Status: medium priority, in progress. Response / Notes: "Add response or update notes...". Buttons: Mark as Resolved, Update Status.

FIGURE 4.9 – Interface de traitement des Reclamations

Cette page est dédiée au traitement des réclamations soumises par les locataires. L'administrateur peut consulter les tickets de maintenance, suivre leur état, et coordonner leur prise en charge jusqu'à leur clôture, garantissant ainsi la qualité des services proposés.

## 4.1.2 Espace public

### 4.1.2.1 Page d'accueil

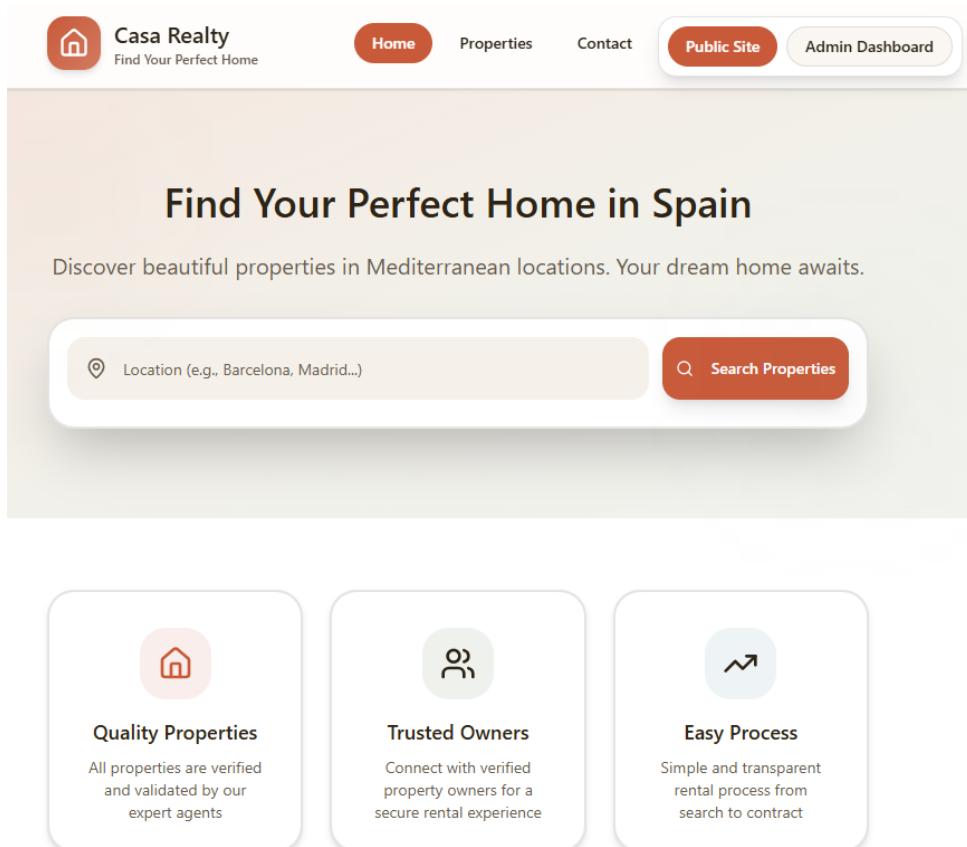


FIGURE 4.10 – Page d'accueil pour visiteurs

La page d'accueil constitue le point d'entrée principal de l'application publique. Elle met en avant les biens disponibles à la location à travers une présentation visuelle attractive. Son design chaleureux et convivial vise à instaurer un climat de confiance et à faciliter la navigation pour les visiteurs.

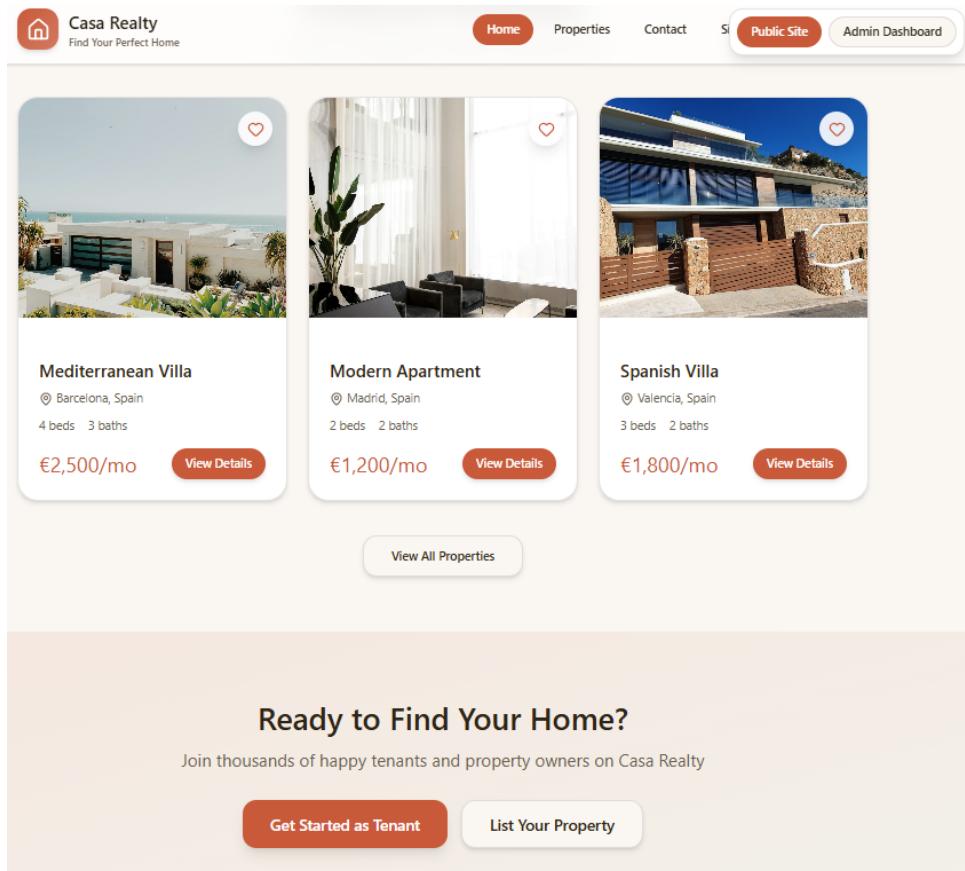


FIGURE 4.11 – Page d'accueil des visiteurs

#### 4.1.2.2 Liste des biens immobiliers

The screenshot shows a search results page on the Casa Realty website. On the left side, there is a sidebar with a "Filters" section containing dropdown menus for Location (All locations), Property Type (All types), Price Range (€500 - €3000), and Bedrooms (Any). A "Apply Filters" button is also present. To the right of the filters is a search bar with the placeholder "Search properties..." and a dropdown menu set to "Newest First". Below the search bar, there are four property cards displayed in a grid:

- Mediterranean Villa** located in Barcelona. It has 4 beds, 3 baths, and is a House. It is listed at €2500/mo. A "View Details" button is present.
- Modern Apartment** located in Madrid. It has 2 beds, 2 baths, and is an Apartment. It is listed at €1200/mo. A "View Details" button is present.
- Spanish Villa** located in Valencia. It has 3 beds, 2 baths, and is a House. It is listed at €1800/mo. A "View Details" button is present.
- Cozy Studio** located in Valencia. It has 1 bed, 1 bath, and is a Studio. It is listed at €750/mo. A "View Details" button is present.

FIGURE 4.12 – Liste des biens proposées

Cette page permet aux visiteurs de consulter l'ensemble des biens disponibles. Des filtres sont proposés afin d'affiner la recherche selon différents critères tels que le prix, le type de bien ou la localisation, améliorant ainsi l'expérience utilisateur.

#### 4.1.2.3 Détails d'un bien

FIGURE 4.13 – Details du Bien choisi

La page de détail présente les informations complètes d'un bien immobilier, incluant une galerie d'images, une description détaillée, le prix et la localisation. Un bouton de demande de location permet à l'utilisateur d'initier une procédure, sous réserve d'être authentifié en tant que locataire.

#### 4.1.2.4 Inscription et authentification

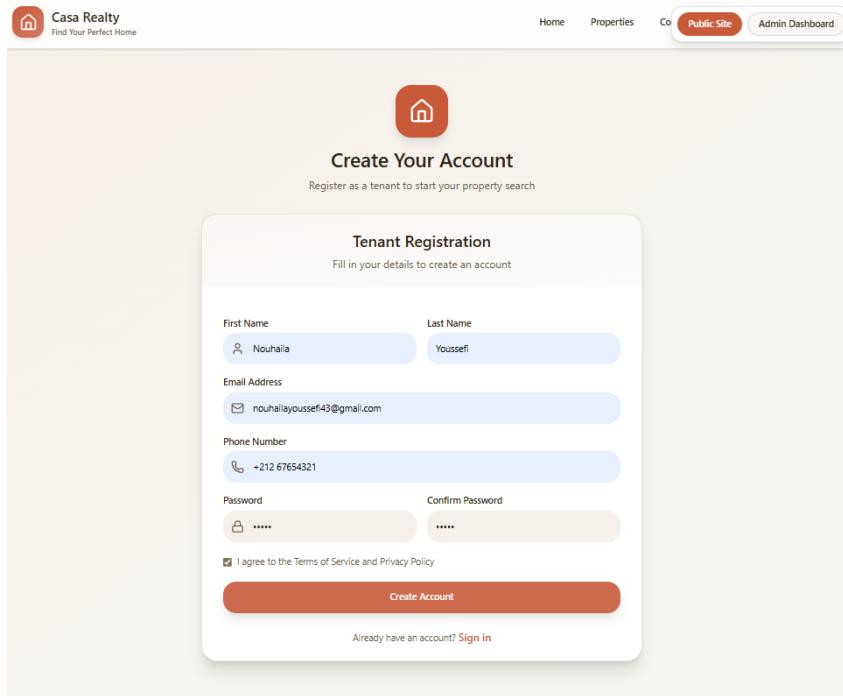


FIGURE 4.14 – page d’inscription autant que Locataire/tenant

Ces pages permettent aux utilisateurs de créer un compte locataire ou de sauthentifier. Linscription est nécessaire pour accéder aux fonctionnalités avancées, telles que lenvoi de demandes de location ou le suivi des contrats.

#### 4.1.2.5 Tableau de bord locataire

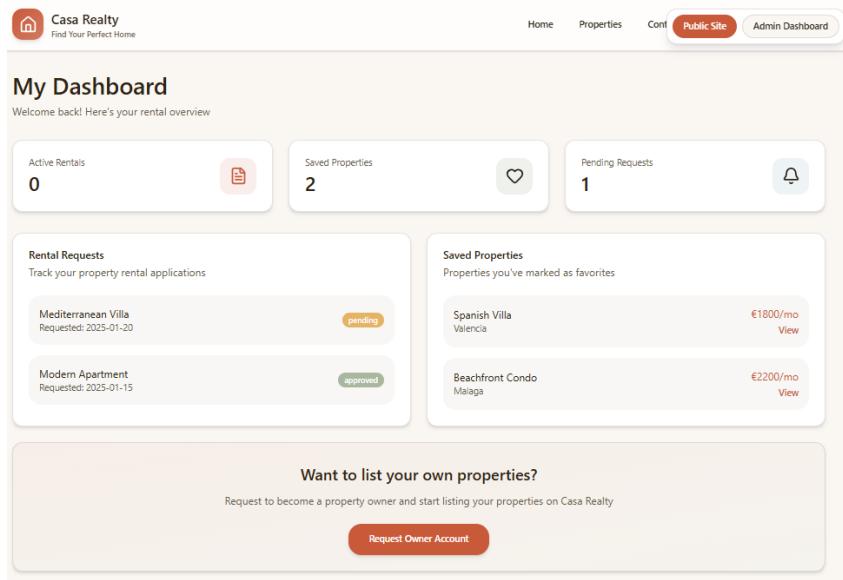


FIGURE 4.15 – Tableau de bord d’un Tenant

Le tableau de bord locataire offre une vue personnalisée des activités de l'utilisateur. Il permet de consulter l'historique des demandes de location, les contrats actifs, les réclamations ainsi que les notifications liées aux paiements et aux échéances.

#### 4.1.2.6 Demande de profil propriétaire

The screenshot shows a web page titled 'Become a Property Owner' under the 'Public Site' tab. The page has a header with the Casa Realty logo and navigation links for Home, Properties, Contact, Public Site (which is active), and Admin Dashboard. Below the header is a large orange button with a house icon. The main content area has a title 'Become a Property Owner' and a subtitle 'List your properties on Casa Realty and reach thousands of potential tenants'. A 'Owner Request Form' section contains fields for Full Name (salim berrada), Email Address (salim@gmail.com), Phone Number (+212 7553299), Number of Properties to List (3), and Additional Information (a text area with placeholder text). A 'Submit Owner Request' button is at the bottom.

FIGURE 4.16 – demander le profil Owner

Cette page permet à un utilisateur inscrit de soumettre une demande afin d'obtenir le rôle de propriétaire au sein de la plateforme. À travers un formulaire dédié, l'utilisateur fournit les informations nécessaires justifiant sa demande. Celle-ci est ensuite transmise à l'administrateur pour validation. Ce mécanisme garantit un contrôle rigoureux des rôles et des droits d'accès, tout en assurant que la publication des biens immobiliers se fait sous supervision administrative avant leur mise à disposition sur l'application publique.

## **Conclusion**

Les interfaces développées répondent aux besoins fonctionnels définis lors de la phase d'analyse. Elles offrent une séparation claire entre l'application d'administration et l'application publique, tout en garantissant une expérience utilisateur fluide, cohérente et sécurisée.

## CHAPITRE 5

---

### APPLICATION MOBILE DE GESTION IMMOBILIÈRE

#### 5.1 Introduction

L'application mobile de gestion immobilière a été développée avec Flutter pour offrir une solution complète et moderne aux différents acteurs du marché immobilier. Cette application permet de gérer l'ensemble du processus de location, depuis la recherche de biens jusqu'au suivi des paiements et des réclamations. Trois types d'utilisateurs sont pris en charge : les locataires, les propriétaires et les administrateurs. Chaque profil dispose d'une interface adaptée à ses besoins spécifiques.

#### 5.2 Interface Locataire

##### 5.2.1 Tableau de Bord Principal

L'écran d'accueil du locataire présente un tableau de bord complet avec les statistiques personnelles et un accès rapide aux principales fonctionnalités. Cette interface offre une vue d'ensemble instantanée sur la situation locative de l'utilisateur.

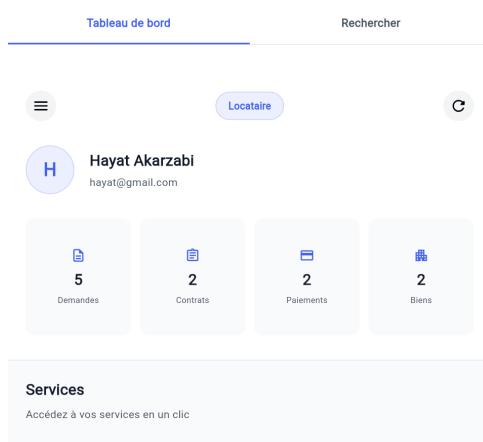


FIGURE 5.1 – Page d'accueil locataire - Vue d'ensemble des statistiques

La première partie de l'accueil affiche des indicateurs clés sous forme de cartes statistiques, permettant au locataire d'avoir une vision rapide de sa situation : nombre de contrats actifs, paiements en attente, et autres métriques importantes.

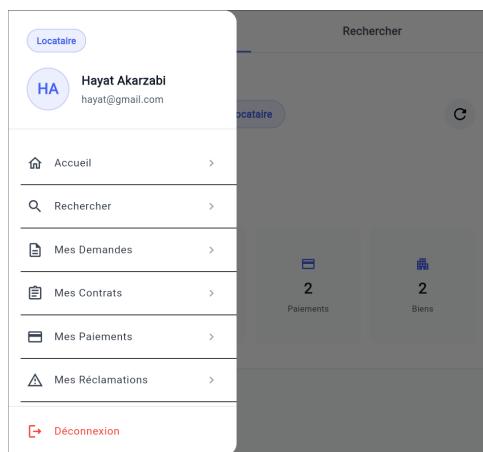


FIGURE 5.2 – Menu principal avec l'ensemble des fonctionnalités accessibles

La seconde partie présente un menu organisé en grille avec toutes les fonctionnalités principales, offrant une navigation intuitive vers les différents modules de l'application.

#### Statistiques affichées :

- **Nombre de contrats actifs** : Contrats de location en cours
- **Nombre de paiements à effectuer** : Loyers et charges en attente
- **Nombre de biens en location** : Biens actuellement loués
- **Nombre de demandes de location envoyées** : Demandes en cours de traitement
- **Montant total des loyers** : Somme des loyers mensuels

#### Boutons d'accès rapide :

1. **Paiements** : Module de paiement en ligne sécurisé
2. **Biens** : Consultation et recherche de biens disponibles
3. **Demandes** : Gestion des demandes de location
4. **Contrats** : Visualisation et gestion des contrats
5. **Réclamations** : Signalement et suivi des problèmes
6. **Catalogue** : Exploration complète du parc immobilier

### 5.2.2 Module de Paiement

Interface permettant aux locataires d'effectuer leurs paiements de loyer en ligne de manière sécurisée, avec génération automatique de quittances.

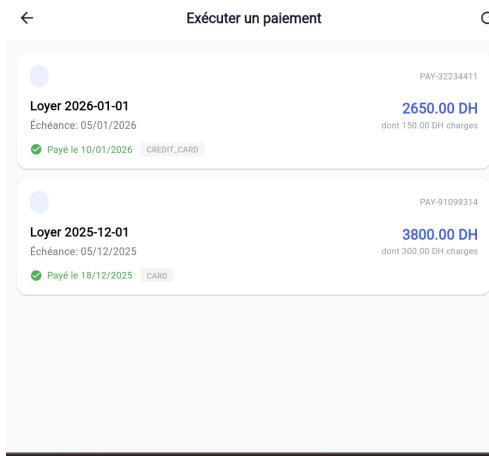


FIGURE 5.3 – Interface de sélection du mode de paiement

Cette écran permet de choisir entre différentes méthodes de paiement (carte bancaire, virement, etc.) avec une présentation claire des options disponibles et des frais éventuels.

The screenshot shows a mobile application interface titled "Paiement sécurisé". It displays a payment summary and a card information input field:

**Résumé du paiement**

Contrat:	CTR-TEST-2026-001
Adresse:	Adresse non disponible
Propriétaire:	Propriétaire
Période:	Janvier 2026
Loyer mensuel:	2500.00 DH
Charges:	150.00 DH
<b>TOTAL À PAYER</b>	
<b>2650.00 DH</b>	

**Informations de paiement**  
Veuillez saisir les informations de votre carte bancaire

Numéro de carte

FIGURE 5.4 – Formulaire de saisie des coordonnées bancaires

Interface sécurisée pour la saisie des informations de carte bancaire, avec validation en temps réel et indicateurs visuels de sécurité.

#### Fonctionnalités :

- **Exécuter un paiement** : Processus simplifié en quelques clics
- **Paiement en ligne sécurisé** : Cryptage SSL et conformité PCI DSS
- **Génération automatique de quittances PDF** : Reçus téléchargeables instantanément
- **Paiement par carte** : Support des principales cartes bancaires
- **Téléchargement des reçus** : Archivage numérique des justificatifs

#### 5.2.3 Gestion des Biens

Module permettant aux locataires de rechercher et consulter les biens disponibles à la location, avec des outils de recherche avancés.

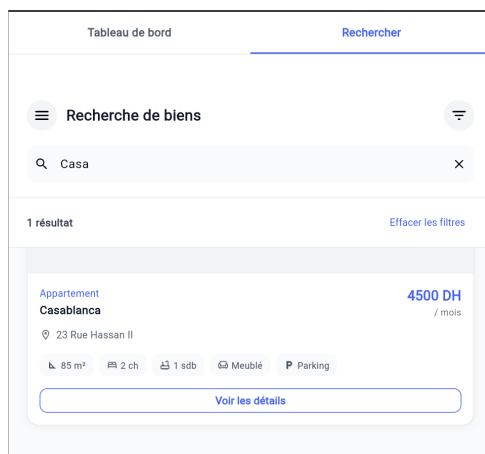


FIGURE 5.5 – Interface de recherche avancée avec filtres multicritères

Ecran de recherche permettant de filtrer les biens par localisation (ville, GPS), prix, superficie, équipements, et autres critères spécifiques pour une recherche ciblée.

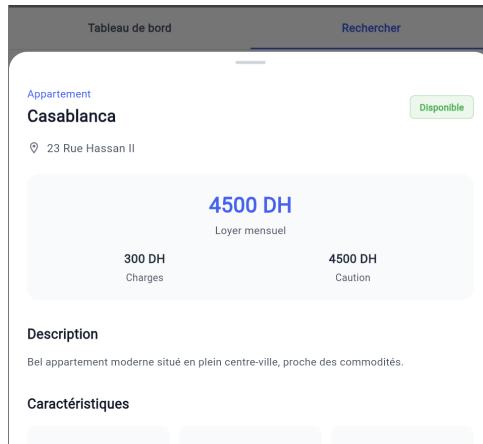


FIGURE 5.6 – Vue détaillée d'un bien - Informations générales

Première partie de la fiche détaillée présentant les informations essentielles : localisation, superficie, prix, et caractéristiques principales du bien.

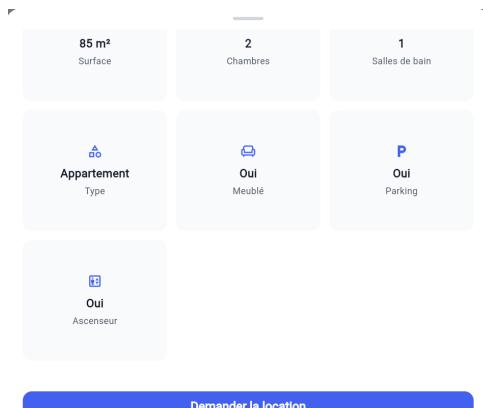


FIGURE 5.7 – Vue détaillée d'un bien - Galerie photos et équipements

Seconde partie de la fiche avec galerie photos interactive et liste complète des équipements disponibles, permettant une inspection virtuelle du bien.

#### Fonctionnalités :

- **Recherche multicritère** : Filtrage par ville, prix, surface, type de bien
- **Filtres avancés** : Options pour biens meublés, équipements spécifiques, accessibilité
- **Galerie photos** : Visualisation haute qualité avec zoom
- **Détails complets** : Toutes les informations techniques et légales
- **Demande de location** : Bouton direct pour initier une demande

### 5.2.4 Gestion des Demandes de Location

Interface permettant de suivre les demandes de location envoyées, avec un système de statut transparent.

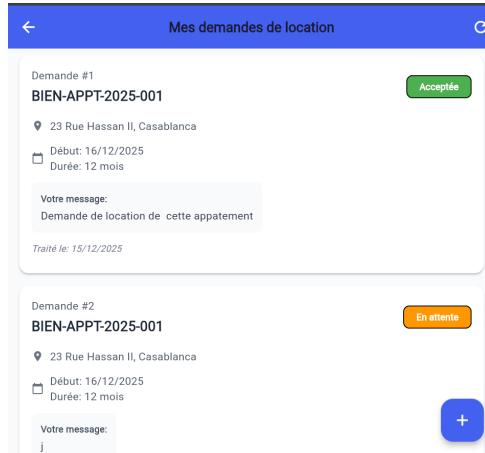


FIGURE 5.8 – Tableau de suivi des demandes de location

Ecran listant toutes les demandes envoyées avec leur état actuel, dates d'envoi, et actions possibles. Chaque demande peut être suivie individuellement.

#### États des demandes :

- **En attente** : Demande soumise et en cours d'examen par le propriétaire
- **Acceptée** : Demande approuvée, possibilité de signer le contrat
- **Refusée** : Demande rejetée avec indication du motif
- **Contrat généré** : Phase finale avec contrat prêt à être signé

### 5.2.5 Gestion des Contrats

Visualisation et gestion des contrats de location, avec accès aux documents et informations légales.

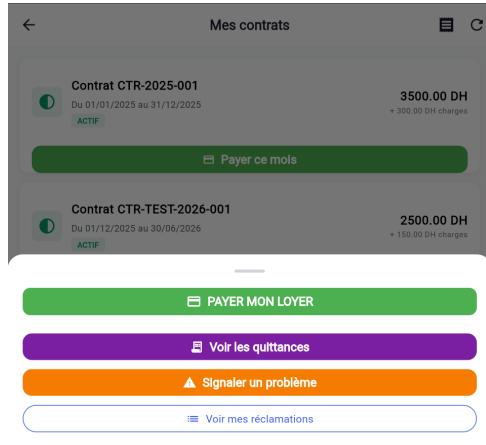


FIGURE 5.9 – Liste des contrats avec statut et actions

Interface présentant tous les contrats du locataire sous forme de cartes organisées, avec mise en évidence visuelle de leur état et échéances importantes.

#### Informations affichées :

- **Référence du contrat** : Identifiant unique pour suivi administratif
- **Dates de début et fin** : Période de validité du contrat
- **Montant du loyer** : Loyer mensuel et charges associées
- **Charges incluses** : Détail des charges comprises ou supplémentaires
- **État du contrat** : Visualisation claire (actif, terminé, résilié)
- **Téléchargement PDF** : Accès aux documents contractuels officiels

#### 5.2.6 Système de Réclamations

Module permettant de signaler des problèmes liés au logement et de suivre leur résolution en temps réel.

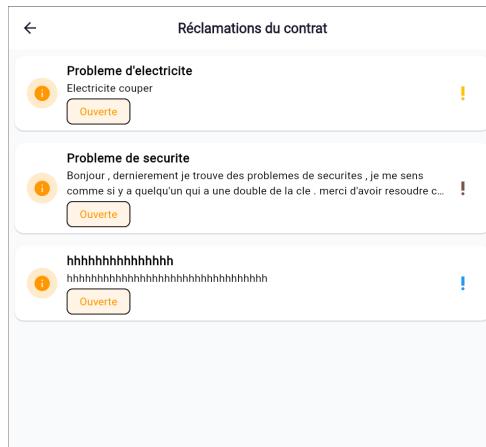


FIGURE 5.10 – Interface de création et suivi des réclamations

Ecran dédié à la gestion des réclamations, permettant de créer de nouveaux signalements, suivre leur progression, et communiquer avec le gestionnaire.

#### Fonctionnalités :

- **Création de réclamation** : Formulaire structuré avec catégories prédéfinies
- **Upload de photos** : Ajout de preuves visuelles pour documentation
- **Suivi d'état** : Progression visible (ouverte, en cours, résolue)
- **Communication intégrée** : Messagerie avec le propriétaire/gestionnaire
- **Historique complet** : Archives des réclamations précédentes

#### 5.2.7 Notifications et Alertes

Système de notifications intégré pour informer le locataire des événements importants.

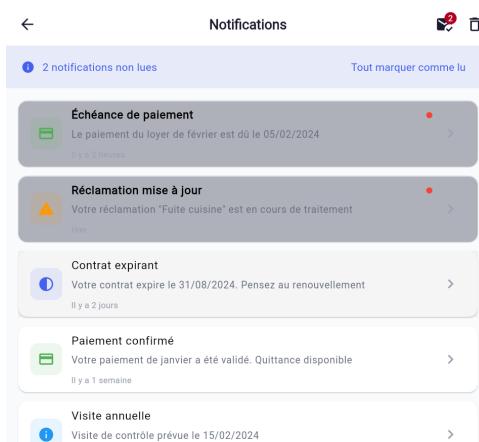


FIGURE 5.11 – Centre de notifications avec alertes personnalisées

Centre de notifications regroupant toutes les alertes importantes : échéances de paiement, réponses aux demandes, mises à jour de réclamations, et communications des propriétaires. Les notifications peuvent être filtrées par type et priorité.

## CHAPITRE 6

---

### APPLICATION MOBILE DE GESTION IMMOBILIÈRE

#### 6.1 Introduction

L'application mobile de gestion immobilière a été développée avec Flutter pour offrir une solution complète et moderne aux différents acteurs du marché immobilier. Cette application permet de gérer l'ensemble du processus de location, depuis la recherche de biens jusqu'au suivi des paiements et des réclamations. Trois types d'utilisateurs sont pris en charge : les locataires, les propriétaires et les administrateurs. Chaque profil dispose d'une interface adaptée à ses besoins spécifiques.

#### 6.2 Interface Locataire

##### 6.2.1 Tableau de Bord Principal

L'écran d'accueil du locataire présente un tableau de bord complet avec les statistiques personnelles et un accès rapide aux principales fonctionnalités. Cette interface offre une vue d'ensemble instantanée sur la situation locative de l'utilisateur.

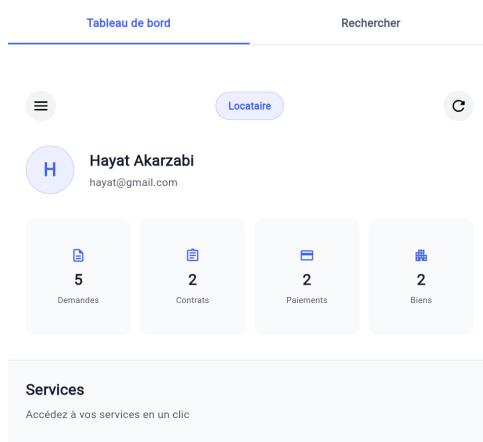


FIGURE 6.1 – Page d'accueil locataire - Vue d'ensemble des statistiques

La première partie de l'accueil affiche des indicateurs clés sous forme de cartes statistiques, permettant au locataire d'avoir une vision rapide de sa situation : nombre de contrats actifs, paiements en attente, et autres métriques importantes.

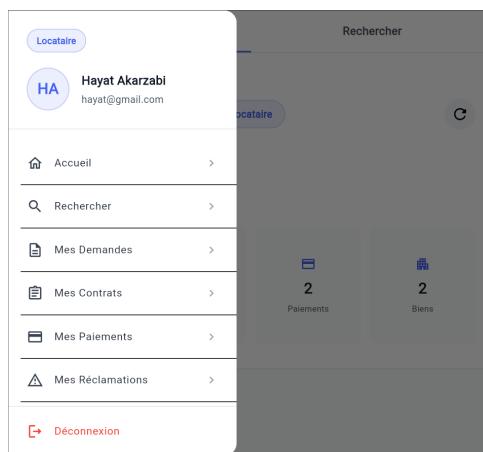


FIGURE 6.2 – Menu principal avec l'ensemble des fonctionnalités accessibles

La seconde partie présente un menu organisé en grille avec toutes les fonctionnalités principales, offrant une navigation intuitive vers les différents modules de l'application.

#### Statistiques affichées :

- **Nombre de contrats actifs** : Contrats de location en cours
- **Nombre de paiements à effectuer** : Loyers et charges en attente
- **Nombre de biens en location** : Biens actuellement loués
- **Nombre de demandes de location envoyées** : Demandes en cours de traitement
- **Montant total des loyers** : Somme des loyers mensuels

#### Boutons d'accès rapide :

1. **Paiements** : Module de paiement en ligne sécurisé
2. **Biens** : Consultation et recherche de biens disponibles
3. **Demandes** : Gestion des demandes de location
4. **Contrats** : Visualisation et gestion des contrats
5. **Réclamations** : Signalement et suivi des problèmes
6. **Catalogue** : Exploration complète du parc immobilier

### 6.2.2 Module de Paiement

Interface permettant aux locataires d'effectuer leurs paiements de loyer en ligne de manière sécurisée, avec génération automatique de quittances.

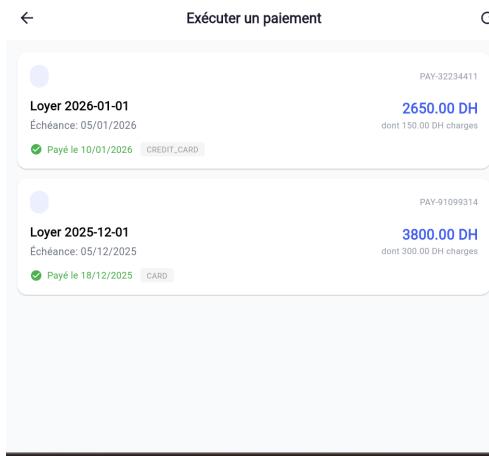


FIGURE 6.3 – Interface de sélection du mode de paiement

Cette écran permet de choisir entre différentes méthodes de paiement (carte bancaire, virement, etc.) avec une présentation claire des options disponibles et des frais éventuels.

The screenshot shows a mobile application interface titled "Paiement sécurisé". It displays a payment summary and a card information input field:

**Résumé du paiement**

Contrat:	CTR-TEST-2026-001
Adresse:	Adresse non disponible
Propriétaire:	Propriétaire
Période:	Janvier 2026
Loyer mensuel:	2500.00 DH
Charges:	150.00 DH
<b>TOTAL À PAYER</b> 2650.00 DH	

**Informations de paiement**  
Veuillez saisir les informations de votre carte bancaire

Numéro de carte

FIGURE 6.4 – Formulaire de saisie des coordonnées bancaires

Interface sécurisée pour la saisie des informations de carte bancaire, avec validation en temps réel et indicateurs visuels de sécurité.

#### Fonctionnalités :

- **Exécuter un paiement** : Processus simplifié en quelques clics
- **Paiement en ligne sécurisé** : Cryptage SSL et conformité PCI DSS
- **Génération automatique de quittances PDF** : Reçus téléchargeables instantanément
- **Paiement par carte** : Support des principales cartes bancaires
- **Téléchargement des reçus** : Archivage numérique des justificatifs

#### 6.2.3 Gestion des Biens

Module permettant aux locataires de rechercher et consulter les biens disponibles à la location, avec des outils de recherche avancés.

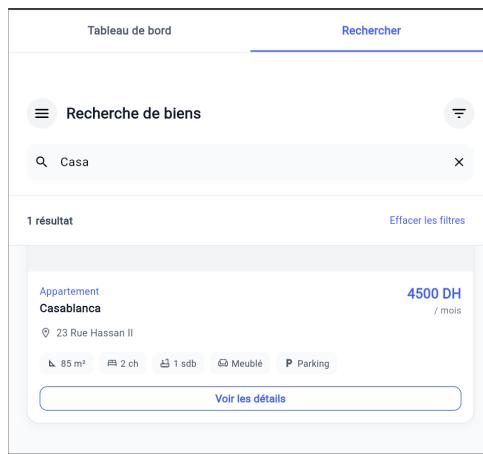


FIGURE 6.5 – Interface de recherche avancée avec filtres multicritères

Ecran de recherche permettant de filtrer les biens par localisation (ville, GPS), prix, superficie, équipements, et autres critères spécifiques pour une recherche ciblée.

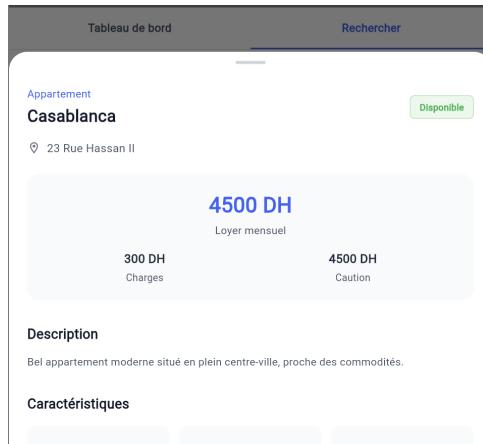


FIGURE 6.6 – Vue détaillée d'un bien - Informations générales

Première partie de la fiche détaillée présentant les informations essentielles : localisation, superficie, prix, et caractéristiques principales du bien.

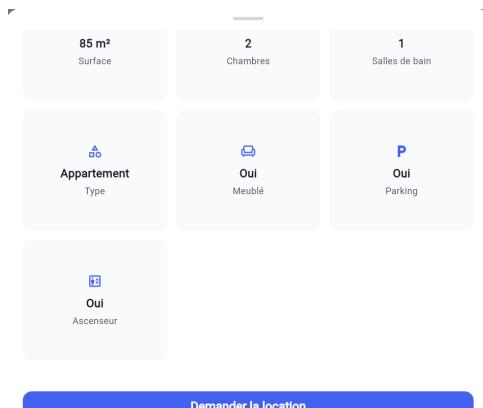


FIGURE 6.7 – Vue détaillée d'un bien - Galerie photos et équipements

Seconde partie de la fiche avec galerie photos interactive et liste complète des équipements disponibles, permettant une inspection virtuelle du bien.

#### Fonctionnalités :

- **Recherche multicritère** : Filtrage par ville, prix, surface, type de bien
- **Filtres avancés** : Options pour biens meublés, équipements spécifiques, accessibilité
- **Galerie photos** : Visualisation haute qualité avec zoom
- **Détails complets** : Toutes les informations techniques et légales
- **Demande de location** : Bouton direct pour initier une demande

#### 6.2.4 Gestion des Demandes de Location

Interface permettant de suivre les demandes de location envoyées, avec un système de statut transparent.

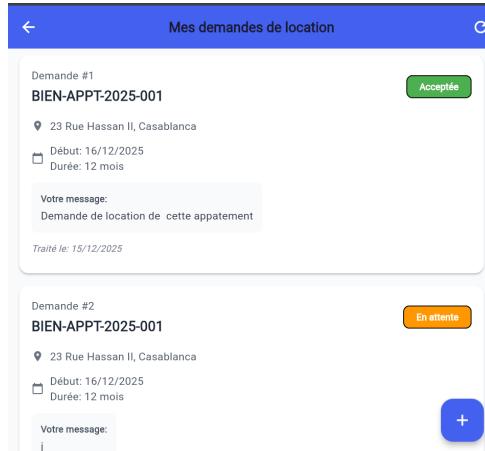


FIGURE 6.8 – Tableau de suivi des demandes de location

Ecran listant toutes les demandes envoyées avec leur état actuel, dates d'envoi, et actions possibles. Chaque demande peut être suivie individuellement.

##### États des demandes :

- **En attente** : Demande soumise et en cours d'examen par le propriétaire
- **Acceptée** : Demande approuvée, possibilité de signer le contrat
- **Refusée** : Demande rejetée avec indication du motif
- **Contrat généré** : Phase finale avec contrat prêt à être signé

#### 6.2.5 Gestion des Contrats

Visualisation et gestion des contrats de location, avec accès aux documents et informations légales.

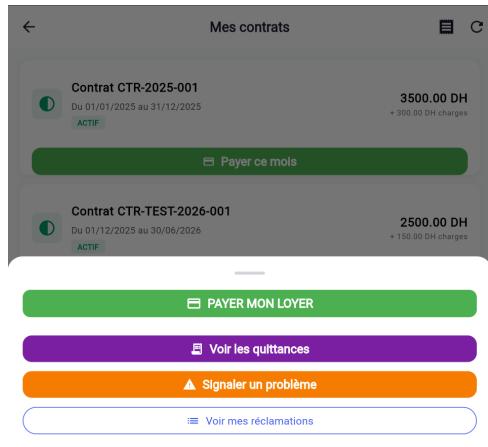


FIGURE 6.9 – Liste des contrats avec statut et actions

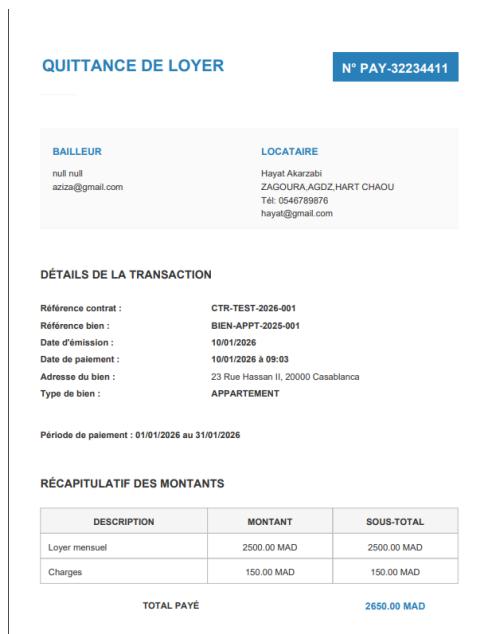


FIGURE 6.10 – Exemple de quittance pdf

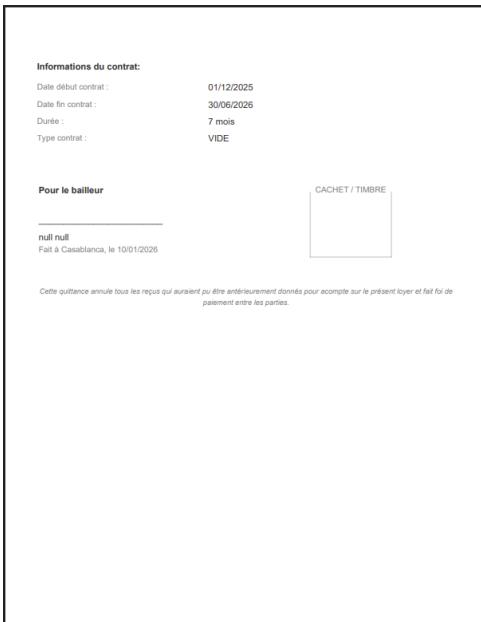


FIGURE 6.11 – Exemple de quittance suite

Interface présentant tous les contrats du locataire sous forme de cartes organisées, avec mise en évidence visuelle de leur état et échéances importantes.

#### Informations affichées :

- **Référence du contrat** : Identifiant unique pour suivi administratif
- **Dates de début et fin** : Période de validité du contrat
- **Montant du loyer** : Loyer mensuel et charges associées
- **Charges incluses** : Détail des charges comprises ou supplémentaires
- **État du contrat** : Visualisation claire (actif, terminé, résilié)
- **Téléchargement PDF** : Accès aux documents contractuels officiels

#### 6.2.6 Système de Réclamations

Module permettant de signaler des problèmes liés au logement et de suivre leur résolution en temps réel.

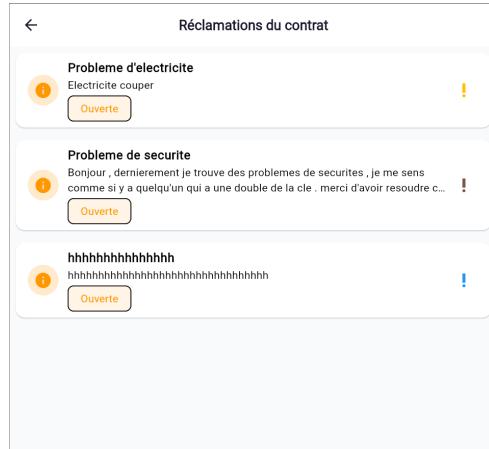


FIGURE 6.12 – Interface de création et suivi des réclamations

Ecran dédié à la gestion des réclamations, permettant de créer de nouveaux signalements, suivre leur progression, et communiquer avec le gestionnaire.

### Fonctionnalités :

- **Création de réclamation** : Formulaire structuré avec catégories prédéfinies
  - **Upload de photos** : Ajout de preuves visuelles pour documentation
  - **Suivi d'état** : Progression visible (ouverte, en cours, résolue)
  - **Communication intégrée** : Messagerie avec le propriétaire/gestionnaire
  - **Historique complet** : Archives des réclamations précédentes

### 6.2.7 Notifications et Alertes

Système de notifications intégré pour informer le locataire des événements importants.

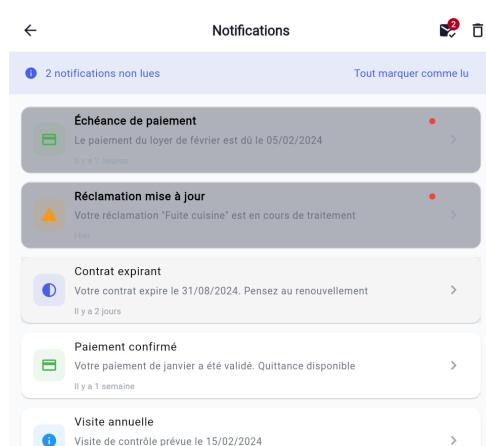


FIGURE 6.13 – Centre de notifications avec alertes personnalisées

Centre de notifications regroupant toutes les alertes importantes : échéances de paiement, réponses aux demandes, mises à jour de réclamations, et communications des propriétaires. Les notifications peuvent être filtrées par type et priorité.

## 6.3 Interface Administrateur

L'interface administrateur constitue le panneau de contrôle central de la plateforme, accessible uniquement aux utilisateurs ayant des priviléges d'administration. Cette interface permet la supervision complète de toutes les activités, la gestion des utilisateurs, des biens, des contrats et des transactions financières. L'administrateur a ainsi une vision globale et détaillée de l'ensemble du système.

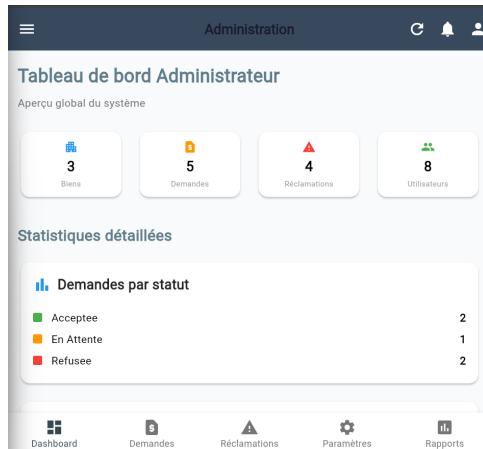


FIGURE 6.14 – Page d'accueil admin - Menu principal et accès rapide

La première partie de l'accueil administrateur présente le menu principal organisé de manière hiérarchique, offrant un accès direct à toutes les sections de gestion. L'interface est conçue pour une navigation intuitive entre les différentes fonctions administratives.

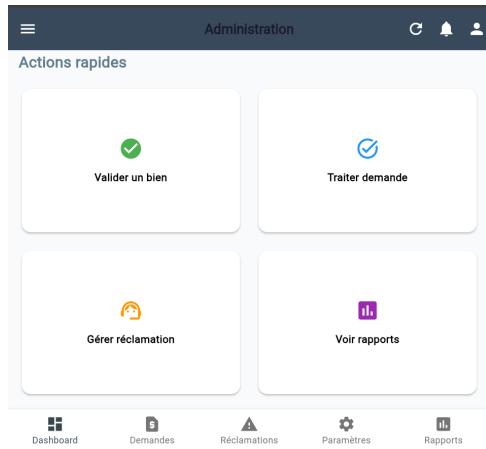


FIGURE 6.15 – Page d'accueil admin - Suite du menu et fonctionnalités avancées

La seconde partie de l'accueil complète le menu principal avec des fonctionnalités administratives avancées, incluant les paramètres système, les outils de reporting et les configurations générales de la plateforme.

### 6.3.1 Tableau de Bord Administratif

Dashboard complet offrant une vue d'ensemble sur toutes les activités de la plateforme, avec des indicateurs clés en temps réel et des outils d'analyse approfondie.

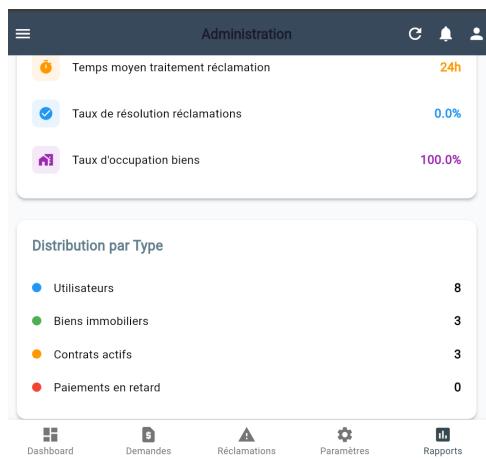


FIGURE 6.16 – Tableau de bord administrateur - Métriques et statistiques globales

Ce tableau de bord présente une synthèse complète des activités de l'agence, affichant :

- **Nombre total de biens** : Parc immobilier géré par l'agence
- **Utilisateurs actifs** : Répartition entre locataires et propriétaires
- **Réclamations en cours** : Signalements nécessitant une intervention
- **Demandes de location** : Requêtes en attente de traitement

- **Paiements en retard** : Suivi des impayés et échéances dépassées
- **Taux d'occupation** : Pourcentage de biens actuellement loués

### 6.3.2 Gestion des Demandes de Location

Interface permettant de valider ou refuser les demandes de location, avec un processus de décision structuré et traçable.

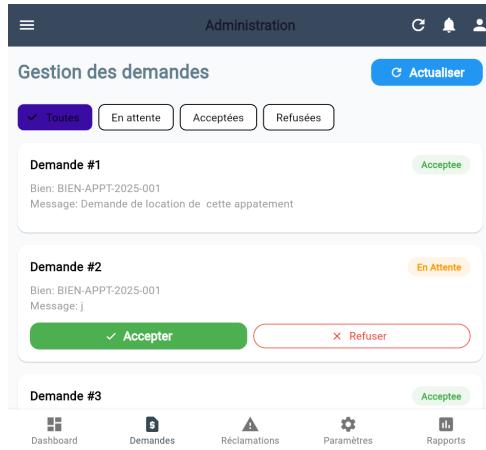


FIGURE 6.17 – Interface de gestion des demandes de location

Cette interface permet à l'administrateur d'examiner chaque demande en détail, de consulter les informations du demandeur, du bien concerné, et de prendre une décision éclairée avec possibilité d'ajouter des commentaires justificatifs.

#### Fonctionnalités :

- **Liste filtrée des demandes** : Tri par date, bien, statut et priorité
- **Consultation détaillée** : Accès complet au profil du demandeur et aux informations du bien
- **Boutons d'acceptation/refus** : Décision rapide avec validation en un clic
- **Commentaire de décision** : Justification obligatoire pour le suivi administratif
- **Notification automatique** : Information immédiate envoyée au demandeur

### 6.3.3 Gestion des Biens

Administration complète du catalogue des biens, permettant l'ajout, la modification et la suppression de propriétés dans le système.

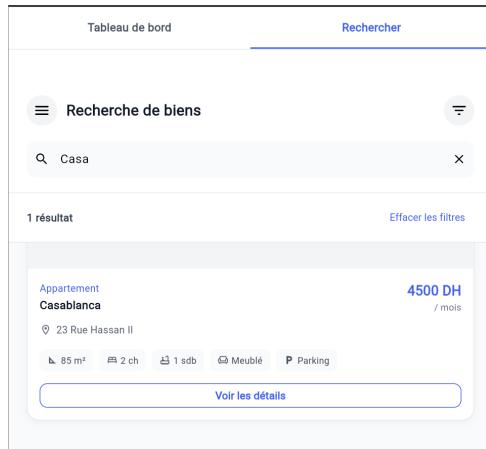


FIGURE 6.18 – Interface de gestion du catalogue des biens

L’interface de gestion des biens permet à l’administrateur de rechercher, filtrer et modifier l’ensemble des propriétés enregistrées dans le système, avec un contrôle total sur chaque élément du catalogue immobilier.

#### Actions possibles :

- **Ajout de nouveaux biens** : Formulaire complet avec toutes les caractéristiques
- **Modification des informations** : Mise à jour des données existantes
- **Suppression de biens** : Retrait définitif du catalogue (avec archives)
- **Activation/désactivation** : Gestion de la visibilité des biens
- **Gestion des photos** : Upload et organisation de la galerie d’images
- **Définition des prix** : Paramétrage des loyers, charges et garanties

#### 6.3.4 Configuration Générale

Paramètres système et configurations globales de la plateforme pour personnaliser son fonctionnement et son apparence.

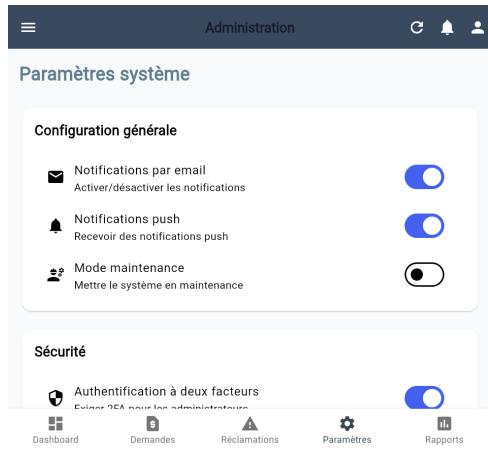


FIGURE 6.19 – Interface de configuration générale de la plateforme

Cet écran permet à l'administrateur de configurer les paramètres globaux du système, incluant :

- **Paramètres de notification** : Configuration des alertes et messages automatiques
- **Options de paiement** : Gestion des méthodes de paiement acceptées
- **Paramètres de sécurité** : Configuration des politiques d'accès et d'authentification
- **Personnalisation** : Adaptation du thème et de l'apparence de l'application
- **Paramètres de facturation** : Configuration des tarifs et frais de service

### 6.3.5 Gestion des Réclamations

Administration complète du système de réclamations, permettant le suivi, l'attribution et la résolution des signalements.

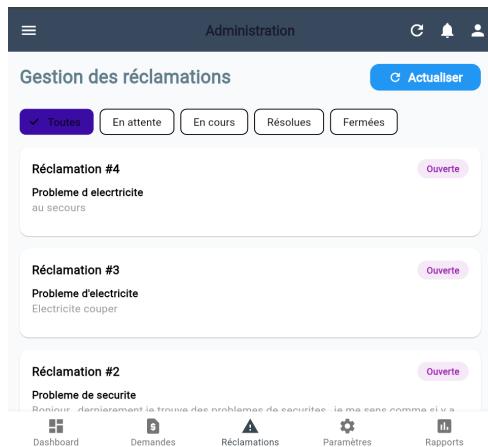


FIGURE 6.20 – Interface de supervision et gestion des réclamations

Plateforme centralisée pour la gestion de toutes les réclamations, offrant des outils avancés de tri, d'attribution et de suivi pour assurer une résolution efficace des problèmes signalés.

#### **Actions disponibles :**

- **Vue d'ensemble des réclamations** : Tableau de bord avec statistiques et tendances
- **Suivi de la résolution** : Monitoring en temps réel de l'avancement
- **Communication centralisée** : Interface de messagerie intégrée avec toutes les parties
- **Clôture et archivage** : Finalisation des réclamations résolues avec documentation

## CONCLUSION GÉNÉRALE

Ce projet a constitué une expérience particulièrement enrichissante, nous permettant de mettre en pratique les compétences acquises tout au long de notre formation à travers la réalisation d'un système complet de gestion immobilière. Il s'inscrit dans un contexte réel où la digitalisation des processus liés à la location, à la gestion des biens et aux relations entre locataires, propriétaires et administrateurs devient un enjeu majeur.

Tout au long de ce travail, nous avons adopté une démarche méthodologique rigoureuse, allant de l'analyse des besoins fonctionnels et non fonctionnels à la conception UML, jusqu'à l'implémentation et la mise en production d'une solution **full-stack**. Le backend a été développé en **Spring Boot**, exposant des **API REST sécurisées** grâce à **Spring Security** et à l'authentification par **JWT**, tandis que les interfaces utilisateur ont été réalisées à travers deux applications distinctes : une application **web en Angular** et une application **mobile en Flutter**.

La plateforme permet à l'administrateur de gérer l'ensemble du système (utilisateurs, biens, contrats, paiements, réclamations et validations), tout en offrant aux locataires et aux propriétaires des espaces dédiés adaptés à leurs besoins respectifs. Cette séparation claire des rôles et des responsabilités améliore à la fois la sécurité, l'ergonomie et la maintenabilité du système.

L'utilisation de **PostgreSQL** pour la persistance des données, couplée à **MinIO** pour le stockage des fichiers (images des biens, contrats, quittances), le tout déployé dans un environnement **Dockerisé**, a permis de se rapprocher des conditions d'un environnement professionnel réel. Ce projet nous a également permis de consolider nos compétences en intégration continue, en structuration d'architectures modernes et en conception d'applications.

cations orientées services.

Au-delà des aspects techniques, ce projet a renforcé notre capacité à travailler de manière collaborative, à documenter efficacement chaque étape du développement, et à concevoir une solution robuste répondant à des besoins métier concrets.

## Perspectives

Bien que la solution développée réponde pleinement aux objectifs initiaux du projet, plusieurs axes d'amélioration et dévolution peuvent être envisagés afin de renforcer la robustesse, la scalabilité et l'adaptabilité à grande échelle.

- **Migration vers une architecture microservices** : le découpage du système en microservices indépendants (gestion des utilisateurs, gestion des biens, paiements, notifications, réclamations, etc.) permettrait d'améliorer la maintenabilité, la résilience et la scalabilité de l'application.
- **Communication asynchrone entre services** : l'intégration d'un système de messagerie tel que **Apache Kafka** ou **RabbitMQ** permettrait de gérer efficacement les événements métier (paiement effectué, contrat créé, résiliation, notification envoyée) via une approche événementielle.
- **Implémentation de Webhooks** : l'utilisation de webhooks, notamment pour les services de paiement, offrirait une meilleure synchronisation avec des systèmes externes et une gestion plus fiable des confirmations de transactions.
- **Scalabilité et haute disponibilité** : la mise en place d'un **load balancer** (NGINX, Traefik) combinée à des stratégies de déploiement avancées (containers orchestrés via Kubernetes) permettrait de supporter une montée en charge importante.
- **Observabilité et monitoring** : l'ajout d'outils de monitoring et de traçabilité (Prometheus, Grafana, ELK Stack) offrirait une meilleure visibilité sur les performances et l'état du système en production.
- **Renforcement de la sécurité** : amélioration de la gestion des rôles et permissions, rotation des clés JWT, gestion des secrets via des coffres sécurisés (Vault), et audit des accès.

Ces perspectives ouvrent la voie à une évolution vers une plateforme immobilière moderne, hautement scalable et conforme aux standards actuels de l'architecture logicielle distribuée.

---

## BIBLIOGRAPHIE

- [1] GeeksforGeeks, “Spring boot - architecture.”
- [2] P. Walpita, “Software architecture patterns layered architecture.”
- [3] Indeed, “What are the 5 primary layers in software architecture ?”
- [4] R. Fadatare, “Spring boot architecture : Controller, service, repository, database.”
- [5] TechTarget, “The pros and cons of a layered architecture pattern.”
- [6] Swagger, “Openapi specification.”
- [7] O. Initiative, “Openapi initiative.”
- [8] O. Initiative, “Openapi-specification.”
- [9] S. Overflow, “What is the point of using openapi at all ?.”
- [10] A. Toolkit, “Unlocking the advantages of openapi (swagger).”
- [11] GeeksforGeeks, “Jwt authentication with spring security using mysql database,” 2025.
- [12] D. Code, “Spring security 6 with spring boot 3 and jwt tutorial,” 2024.
- [13] Medium, “Spring security jwt authentication & authorization,” 2023.
- [14] I. Gordadze, “Spring security jwt tutorial.”
- [15] Curity, “Securing a spring boot api with jwts.”

- [16] M. Gechev, “Angular v18 is now available!,” 2024.
- [17] A. Sharma, “Zoneless change detection in angular 18 : Boost performance,” *Syncfusion Blog*, 2024.
- [18] A. Team, “Angular routing and navigation,” 2024.
- [19] A. Team, “Angular material - what’s new,” 2024.
- [20] T. Team, “Typescript 5.4 release notes,” 2024.