## Page de garde

**Titre :** UrbanHome – Plateforme de gestion immobilière **Établissement :** [Nom de votre établissement] **Date :** Juin 2025 **Équipe :**

* OUEDRAOGO Saad Moussa : Front‑end
* Nacoulma Wendkouni Ezekiel : Front‑end
* Poda Steve Melchisédek : Back‑end
* Ba Cheick Omar Yobi : Back‑end
* Pitroipa Mohamed : Base de données
* OUEDRAOGO Mohamed Charif : Base de données
* BARRO Rachidatou : Conception UML

## Résumé (Abstract)

Ce rapport présente la conception, la réalisation et le bilan d’**UrbanHome**, une application web de gestion immobilière conçue dans le cadre d’un projet d’équipe de sept développeurs. L’objectif principal était d’offrir une solution MVC sécurisée, intuitive et modulaire pour gérer la publication de biens, la consultation, la messagerie et le suivi des paiements. La méthodologie SCRUM a guidé notre organisation, tandis que MySQL et un langage orienté objet (PHP) ont constitué notre socle technique. Les résultats obtenus montrent une interface responsive et des fonctionnalités robustes, avec des perspectives d’extensions futures (mobile, API REST, optimisation).

## Table des matières

1. Page de garde
2. Résumé
3. Sigles et abréviations
4. Préambule
5. Introduction
6. Chapitre 1 : Méthodologie
7. Chapitre 2 : Analyse et conception
8. Chapitre 3 : Réalisation
9. Chapitre 4 : Bilan et perspectives
10. Conclusion
11. Bibliographie
12. Annexes

## Sigles et abréviations

* MVC : Model‑View‑Controller
* KPI : Key Performance Indicator
* SCRUM : Scrum Agile Framework
* ERD : Entity‑Relationship Diagram
* CSRF : Cross‑Site Request Forgery

## Préambule

La gestion immobilière traditionnelle repose sur de nombreux processus manuels (formulaires papier, déplacements, archivage physique) qui ralentissent les échanges entre bailleurs, agents et clients. L’informatisation de ces tâches permet de centraliser les données, d’automatiser les notifications et d’offrir une traçabilité complète des actions. Dans ce contexte, **UrbanHome** vise à moderniser la gestion des biens immobiliers en proposant une plateforme web évolutive et sécurisée.

## Introduction

La gestion immobilière traditionnelle repose sur des processus manuels longs et fastidieux, nécessitant des déplacements sur site et un traitement papier conséquent. Pour répondre à ces enjeux, notre équipe de sept développeurs a conçu **UrbanHome**, une plateforme web de gestion immobilière basée sur une architecture MVC, accessible à la fois via Internet et en intranet pour les employés de l’agence, les bailleurs et les clients.

**Objectifs du projet**

* Automatiser la publication, la consultation et la gestion des biens immobiliers.
* Garantir la sécurité et la convivialité grâce à l’utilisation de PHPMailer, de tokens CSRF et de redirections HTTPS.
* Offrir un tableau de bord interactif aux bailleurs (KPI, contrats, paiements, messagerie).

**Équipe et répartition des rôles**

| Membre | Rôle |
| --- | --- |
| OUEDRAOGO Saad Moussa | Front‑end |
| Nacoulma Wendkouni Ezekiel | Front‑end |
| Poda Steve Melchisédek | Back‑end |
| Ba Cheick Omar Yobi | Back‑end |
| Pitroipa Mohamed | Base de données |
| OUEDRAOGO Mohamed Charif | Base de données |
| BARRO Rachidatou | Conception UML |

À partir de cette base, le rapport détaillera notre démarche méthodologique, la conception UML, la réalisation technique et le bilan du projet, ainsi que les perspectives d’évolution.

## Chapitre 1 : Méthodologie

### 1.1 Choix de la méthodologie

Nous avons opté pour **SCRUM**, un cadre agile permettant des itérations courtes et des retours fréquents des encadrants. Les sprints duraient deux semaines, chaque fin de sprint donnant lieu à une réunion de revue (démonstration des nouvelles fonctionnalités) et à une rétrospective pour améliorer les processus.

### 1.2 Organisation du projet

* **Outils de gestion :** GitHub Projects (Kanban) pour le suivi des issues, Git pour le versioning.
* **Communication :** Slack pour les échanges quotidiens, réunions Zoom hebdomadaires.
* **Planning :** Un diagramme de Gantt a réparti les grandes phases : analyse (2 semaines), conception (2 semaines), réalisation itérative (8 semaines), tests et validation (2 semaines).

### 1.3 Structure MVC adoptée

Le projet suit le pattern MVC :

* **Model :** classes PHP gérant l’accès à la base de données (DAO, entités).
* **View :** templates Twig pour séparer la logique métier de l’affichage.
* **Controller :** routeurs définis via un front controller (index.php) qui distribue aux contrôleurs spécifiques.

## Chapitre 2 : Analyse et conception

### 2.1 Analyse des besoins fonctionnels et techniques

**Besoins fonctionnels**

1. **Gestion des utilisateurs** : création, authentification, attribution de rôles (administrateur, bailleur, client).
2. **CRUD des biens immobiliers** : ajout, modification, suppression et consultation des fiches de propriété.
3. **Messagerie interne** : envoi et réception de messages entre utilisateurs (bailleurs, clients, administrateurs).
4. **Tableau de bord KPI** : suivi des indicateurs clés (nombre de biens, contrats en cours, paiements en attente).
5. **Gestion des contrats et paiements** : génération de contrats PDF, historique des paiements, alertes automatiques.
6. **Recherche et filtres avancés** : filtrage par localisation, type de bien, fourchette de prix.

**Besoins techniques**

* **Sécurité :**
  + Utilisation de tokens CSRF pour protéger les formulaires.
  + Stockage des mots de passe avec bcrypt.
  + Redirections forcées en HTTPS via .htaccess.
* **Performance :**
  + Pagination des listes.
  + Indexation des colonnes critiques en base MySQL.
* **Extensibilité :**
  + Architecture modulaire MVC.
  + Conventions PSR-4 pour l’autoloading des classes.
* **Interopérabilité :**
  + Utilisation de PHPMailer pour l’envoi de mails.
  + Préparation du système pour une future API REST.

### 2.2 Diagrammes UML

Pour formaliser notre conception, plusieurs diagrammes UML ont été réalisés :

* **Diagramme de cas d’utilisation** : présente les acteurs (client, bailleur, administrateur) et leurs interactions avec le système.
* **Diagramme de classes** : décrit les entités métier (Utilisateur, Bien, Contrat, Message) et leurs relations.
* **Diagramme de séquence** : illustre le processus de création d’un bien et d’envoi d’un message.
* **Diagramme d’activité** : détaille le flux de validation d’un contrat.
* **Diagramme de déploiement** : schématise l’infrastructure serveur, base de données et serveur web.

Les diagrammes sont disponibles en annexe (cf. Annexe 1) et dans le dépôt GitHub du projet.

### 2.3 Modèle de données

Le modèle relationnel suit les règles d’intégrité et de normalisation. Les principales tables sont :

| Table | Clé primaire | Colonnes principales |
| --- | --- | --- |
| Utilisateur | id\_user | nom, prénom, email, mot\_de\_passe, rôle |
| Bien | id\_bien | titre, description, adresse, prix, id\_user |
| Contrat | id\_contrat | date\_début, date\_fin, id\_bien, id\_user, statut |
| Message | id\_message | contenu, date\_envoi, id\_expediteur, id\_dest |

Les relations sont :

* **Utilisateur (1,n) ⇆ (n,1) Bien**
* **Bien (1,n) ⇆ (n,1) Contrat**
* **Utilisateur (1,n) ⇆ (n,1) Message**

La base de données est implémentée en MySQL, avec encodage UTF-8 et moteur InnoDB pour garantir les transactions et les contraintes d’intégrité référentielle.

## Chapitre 3 : Réalisation

### 3.1 Environnement de développement

* **Langage et serveur** : PHP 8.1 exécuté sous Apache 2.4 sur Ubuntu 20.04.
* **Gestion de versions** : Git 2.34, hébergé sur GitHub avec branches feature, develop et main.
* **Base de données** : MySQL 8.0, encodage UTF-8, moteur InnoDB.
* **Templates** : Twig 3 pour la séparation vue/logique.
* **Dépendances** : Composer 2 pour l’autoloading PSR‑4 et les bibliothèques tierces (PHPMailer).
* **IDE et outils** : VSCode avec extensions PHP Intelephense, Docker (conteneurs en local) et Postman pour les tests d’API.

### 3.2 Architecture logicielle

L’application suit l’architecture **MVC** :

* **Front controller (index.php)** : point d’entrée unique, charge l’autoloader et route les requêtes.
* **Routeur** : définit les routes via FastRoute, lie chaque URL à un contrôleur et une action.
* **Contrôleurs** : classes responsables de la logique métier, appellent les modèles et renvoient les vues.
* **Modèles (DAO)** : classes PHP encapsulant l’accès à la base (CRUD, requêtes préparées).
* **Vues (Twig)** : templates avec inheritance, blocs et macros pour un rendu dynamique.

L’arborescence du projet :

/src  
 /Controller  
 /Model  
 /View  
 /Entity  
 /Repository  
/public  
 index.php  
 /assets  
 /css  
 /js  
 /images  
/config  
 routes.php  
 database.php  
/vendor

### 3.3 Implémentation des fonctionnalités principales

#### 3.3.1 Gestion des utilisateurs

* **Inscription & connexion** : formulaire avec validation côté serveur, hash bcrypt des mots de passe.
* **Attribution des rôles** : middleware qui vérifie les permissions avant chaque action.

#### 3.3.2 CRUD des biens immobiliers

* Formulaire de création/édition avec upload d’images multiple.
* Pagination et tri sur liste des biens.
* Gestion des droits : seul le propriétaire ou l’admin peut modifier/supprimer.

#### 3.3.3 Messagerie interne

* Interface en temps réel via polling Ajax (auto-refresh toutes les 5 s).
* Filtrage des conversations par interlocuteur.

#### 3.3.4 Tableau de bord KPI

* Calcul en back-end des indicateurs (total biens, contrats actifs, retards de paiement).
* Affichage en front-end sous forme de cards et graphiques générés avec Chart.js.

#### 3.3.5 Gestion des contrats et paiements

* Génération automatique de PDF via Dompdf.
* Envoi de notifications email aux bailleurs et clients.
* Historique et statut des paiements mis à jour en temps réel.

### 3.4 Captures d’écran de l’application

*(Intégrer ici des captures : page d’accueil, formulaire création de bien, tableau de bord, messagerie.)*

### 3.5 Tests et validation

* **Tests unitaires** : PHPUnit pour les modèles et contrôleurs critiques (couverture ~ 75 %).
* **Tests fonctionnels** : scénarios via Behat pour la connexion, CRUD biens et envoi de message.
* **Rapport de bugs** : 12 issues majeures fermées, 3 en cours (annexe 2).

## Chapitre 4 : Bilan et perspectives

### 4.1 Points forts

* **Modularité et maintenabilité** : architecture MVC claire facilitant l’ajout de nouvelles fonctionnalités.
* **Sécurité accrue** : implémentation de CSRF, hash bcrypt et HTTPS obligatoire.
* **Interface utilisateur** : design responsive avec Twig et Chart.js pour une expérience fluide.
* **Collaboration efficace** : workflow GitHub et méthodologie Scrum ont permis un suivi rigoureux.
* **Automatisation** : génération de contrats PDF et notifications email réduisant les tâches manuelles.

### 4.2 Difficultés rencontrées

* **Familiarisation aux outils** : prise en main de GitHub Projects et FastRoute plus longue que prévu.
* **Tests automatisés** : configuration de PHPUnit et Behat a nécessité un temps de paramétrage important.
* **Gestion des images** : upload multiple et redimensionnement en PHP ont engendré des erreurs de mémoire.
* **Synchronisation temps réel** : polling Ajax a limité la performance; envisager WebSockets.

### 4.3 Perspectives d’évolution

* **API RESTful** : exposer les services via une API sécurisée pour intégration mobile et tiers.
* **Application mobile** : développement d’une application Flutter ou React Native.
* **Notification push** : migration vers WebSockets (Ratchet) pour un chat en temps réel.
* **Optimisations** : mise en cache des requêtes fréquentes et refactoring des requêtes lourdes.
* **Scalabilité** : containerisation complète (Docker Compose / Kubernetes).

## Conclusion

Ce projet **UrbanHome** a permis de concevoir et développer une plateforme web complète de gestion immobilière, répondant aux besoins identifiés et respectant les contraintes techniques et organisationnelles. Grâce à une architecture MVC robuste et une méthodologie Scrum rigoureuse, nous avons livré un prototype fonctionnel mettant l’accent sur la sécurité et l’ergonomie. Les perspectives d’évolution ouvertes (API, mobile, temps réel) fournissent un roadmap clair pour la suite du développement.

## Bibliographie

1. The PHP Group. *PHP Manual*, Version 8.1. Disponible sur <https://www.php.net/manual/en/>.
2. Oracle. *MySQL Reference Manual*, Version 8.0. Disponible sur <https://dev.mysql.com/doc/>.
3. Fabian Potencier. *Twig Documentation*, Version 3. Disponible sur <https://twig.symfony.com/doc/3.x/>.
4. Scott Chacon & Ben Straub. *Pro Git*, Git version 2.34. Disponible sur <https://git-scm.com/book/en/v2>.
5. Jeffery Palermo. *Model-View-Controller*, Pattern description. Disponible sur <https://martinfowler.com/eaaDev/>

## Annexes

**Annexe 1 : Diagrammes UML**

* Diagramme de cas d’utilisation (PDF pages 3–4 du dossier UML).
* Diagramme de classes (PDF page 5).
* Diagramme de séquence création de bien (PDF page 6).
* Diagramme d’activité validation de contrat (PDF page 7).
* Diagramme de déploiement (README.md, section UML).

**Annexe 2 : Scripts SQL**

* Fichier schema.sql définissant la structure des tables.
* Fichier data.sql insérant les données de test.

**Annexe 3 : Extraits de code**

* Contrôleur BienController.php (CRUD des biens).
* Modèle UserRepository.php (connexion et gestion des rôles).

**Annexe 4 : Planning détaillé (Gantt)**

* Planning prévisionnel réalisé avec Excel (section Planning du README.md).

**Annexe 5 : Rapport de bugs**

* GitHub Issues : 15 tickets ouverts sur le dépôt (lien en README.md).