Appartex

Un écosystème logiciel d'une régie pour propriétaires indépendants et leurs locataires

Travail de Bachelor

Kesigan Thavarajasingam

Février 2021

Supervisé par :

Prof. Dr. Jacques Pasquier-Rocha
and
Pascal Gremaud
Software Engineering Group

UNIVERSITÉ DE FRIBOURG
UNIVERSITÄT FREIBURG

Groupe Génie Logiciel
Département d'Informatique
Université de Fribourg (Suisse)

Software Engineering Group

Remerciements

Je souhaiterais remercier le Professeur Pasquier et Pascal Gremaud pour leur engagement dans ce projet, leurs conseils ainsi que le cadre de travail qu'ils m'ont fournis. Je souhaiterais également remercier la famille Bosson pour leur soutien et leur relecture.

Abstract

Il est de plus en plus courant que des particuliers achètent des appartements pour ensuite les louer afin de s'assurer un revenu supplémentaire ou une garantie de retraite. Cependant, beaucoup de propriétaires délèguent la gestion de leurs biens à des régies.
Ce travail de bachelor a pour but de développer une application client-serveur, nommée Appartex, qui se veut comme une alternative viable à une régie. Appartex permet aux propriétaires de gérer leurs appartements ainsi que de communiquer avec leurs locataires via une messagerie. La conception de cette application a été faite grâce à une étude des besoins et de leurs formulations à travers des cas d'utilisation. Les aspects théoriques de la partie serveur ont été présentés et mis en pratique. L'application a été développée selon les principes REST en utilisant les langages ReactJS, Node.js, Koa et MongoDB.

Table des matières

Liste des figures

Liste des tableaux

Liste des codes source

1 Introduction

1.1. Motivations et objectifs

1.1.1. Mise en situation

De nos jours, il est habituel de voir des particuliers acheter des appartements pour ensuite les louer. De cette manière, ils arrivent à se garantir un revenu supplémentaire notamment lors de la retraite. Cependant, un bon nombre de propriétaires n'ont pas les compétences dans ce domaine et ont tendance à déléguer la responsabilité à des régies.

Même si cela est une pratique courante, elle implique plusieurs conséquences qui ne sont pas négligeables :

- Les coûts engendrés par la régie diminuent le rendement qui devient actuellement de plus en plus faible.
- Le propriétaire ne connaît pas l'implication de la gérance dans la recherche de locataires. De plus, l'entretien d'une bonne relation avec ceux-ci ne leur est pas primordiale.
- Le propriétaire perd une bonne partie du contrôle de la gestion des appartements. Il manque de connaissances et la délégation à la régie ne lui permet pas de s'améliorer.
- La régie peut manquer de transparence vis à vis de la comptabilité qui peut être imprécise, en ajoutant ou omettant certains frais.
- Le propriétaire, n'étant pas impliqué dans la gestion de ses appartements ou immeubles, possède très peu de communication avec ses locataires.
- Les gérances, ayant beaucoup de dossiers, peuvent laisser passer des négligences. Il est très facile de payer des prestations ou des frais incorrects. Etant locataire moi-même, il m'est déjà arrivé de payer pour des services qui n'existaient plus tel que la télévision câblée. Comme les locataires, le propriétaire indépendant n'est pas à l'abri de certaines pratiques malhonnêtes.

- Les gérances, pour des soucis de mandats, favorisent les grands clients au dépend des petits propriétaires indépendants. La gestion d'un groupe d'immeuble est plus profitable que plusieurs petits appartements indépendants. Les petits propriétaires ont moins d'influence sur les gérances que les grands clients.
- Le propriétaire perd sa marge de manoeuvre. Il ne peut effectuer des arrangements avec les locataires sans qu'une tierce personne le sache. La présence d'un intermédiaire peut être très handicapant dans certaines situations.

Toutes ces raisons non-exhaustives peuvent pousser les propriétaires à trouver une alternative à la gestion de leurs appartements.

1.1.2. Pourquoi ce projet ?

Lors d'une expérience professionnelle précédente, j'ai travaillé avec un propriétaire indépendant qui souhaitait quitter sa régie pour les raisons mentionnées ci-dessus. Le projet a été développé sur Excel et le propriétaire remplissait des tableaux dynamiques via des formulaires programmés en VBA. Cela lui permettait de gérer sa comptabilité. Il obtenait alors une certaine indépendance qu'il avait perdu auparavant. Même si le projet était en accord avec les demandes du propriétaire, il ne m'apportait pas pleine satisfaction pour deux raisons principales :

- il n'était pas possible d'effectuer des tâches poussées car les limites de VBA étaient rapidement atteintes.
- une fonction indispensable à mon sens était impossible à implémenter : la communication avec les locataires via un chat ou messagerie.

J'ai donc voulu pallier à ces manques en développant une application web full stack de gestion pour propriétaires indépendants.

Ce projet s'inscrit parfaitement dans le cadre de mon travail de bachelor. Il s'agit de développer un écosystème composé d'un frontend, développé en ReactJS, et d'un backend, avec MongoDB, Koa.js et Node.js. Tout cela en respectant les principes d'un RESTful API qui seront expliqués plus tard. Ce projet cible les propriétaires, qui possèdent un ou plusieurs appartements, et se veut comme une alternative aux régies.

1.2. Organisation

Le rapport sera divisé en quatre parties :

La première partie traite des besoins fonctionnels et non-fonctionnels du propriétaire et des locataires. Elle décrit les différents diagrammes de cas d'utilisation.

La deuxième partie explique le fonctionnement du serveur et de sa base de données. Les divers sujets sont traités de manière théorique puis mis en pratique.

La troisième partie présente les fonctionnalités offertes par l'interface client. Ce chapitre comporte une brève explication sur l'interface. Un scénario sera expliqué pour chacun des acteurs.

Finalement, le travail se terminera par une conclusion qui résume le travail et énumère toutes les difficultés rencontrées ainsi que sa future évolution.

2

Les besoins du propriétaire et des locataires

Ce chapitre traite des différents besoins, des cas d'utilisation et de leurs fonctionnements.

2.1. Les besoins

L'application web s'adresse à des bailleurs et à leurs locataires. Il s'agit donc d'identifier les besoins des propriétaires dans un premier temps puis ceux des locataires. Finalement, nous nous intéresserons aux besoins de l'application web.

Les besoins du propriétaire
Le propriétaire doit être en mesure de gérer ses immeubles, s'il en possède, et ses appartements. Il doit notamment être en mesure de gérer ses locataires, leurs contrats de bail, leurs factures ainsi que leurs documents personnels. De plus, il doit avoir la possibilité de distinguer les frais à sa charge et ceux liés aux locataires. Il est également important pour lui de pouvoir créer des tâches dans

l'application web. Ces tâches sont sous la forme de rendez-vous ou d'annotations dans un agenda. Elles doivent pouvoir être inscrites suite à une demande d'un locataire (via message) ou par le propriétaire lui-même. Le bailleur doit pouvoir communiquer avec les locataires et vis-versa. Un système de messagerie permettra de répondre à ce besoin.

Les besoins du locataire
Le locataire doit pouvoir accéder à ses contrats, à ses factures, à ses tâches ainsi que ses documents personnels. Il doit pouvoir modifier ses données personnelles.

Les besoins de l'application web
Un besoin important est que les données inscrites dans l'application web puissent être exportées vers Excel. Ainsi le propriétaire pourra garder une traçabilité sur excel et les imprimer pour les transmettre au locataire.

Le propriétaire est le seul à pouvoir entreprendre des actions sur les divers documents. Le locataire ne peut que visualiser les documents le concernant.

L'application web disposera d'un système de catalogue sur la page d'accueil afin de mettre en avant les appartements à louer.

2.2. Cas d'utilisation

Les diagrammes de cas d'utilisation décrivent ce qu'un système fait depuis le point de vue d'un observateur externe. Ils mettent l'accent sur les actions du système plutôt que sur leur fonctionnement. Les diagrammes de cas d'utilisation sont composés[15] :

- de cas d'utilisation représentant les fonctionnalités disponibles.
- d'un système. Il est défini par un rectangle qui contient les cas d'utilisation. Dans notre cas, les cas d'utilisation ayant une interaction commune sont regroupés sous le même système.
- d'acteurs désignant les personnes qui intéragissent avec le système.
- de liens de communication représentés par des traits noirs. Ils montrent que l'acteur participe à un cas d'utilisation.

Tous les cas d'utilisation doivent être reliés à au moins un acteur.

La relation entre les différents cas d'utilisation peuvent être :

- **générale**, c'est à dire qu'un cas d'utilisation peut être décomposé en d'autres plus spécifiques. Cela signifie que les actions plus spécifiques pointent vers l'action générale.
- **inclusive**, c'est à dire qu'un cas d'utilisation inclut le comportement d'un autre cas d'utilisation. Il en est dépendant.
- **extensible**, c'est à dire qu'un cas d'utilisation peut en appeler un autre lors de son exécution.

Les diagrammes de cas d'utilisation sont utilisés afin de définir les différentes fonctions du système. Ils sont également une excellente manière de pouvoir communiquer avec des entreprises ou personnes qui ne font pas partis du domaine informatique. En effet, les diagrammes sont simples et représentatifs.

Les tableaux explicatifs et les scénarios des différents cas d'utilisations sont disponibles en annexe.

2.2.1. S'inscrire et s'authentifier

La figure 2.1 représente le cas d'utilisation d'un utilisateur non authentifié. L'application web s'adresse à deux types d'utilisateurs : le propriétaire et le locataire. Il est donc indispensable de s'inscrire et de s'authentifier car sans cela, il est impossible d'utiliser l'application et de distinguer les utilisateurs. L'application web peut être utilisée par plusieurs propriétaires.

Le propriétaire peut s'inscrire et s'authentifier. Ce n'est pas le cas du locataire qui peut effectuer son inscription qu'à travers le propriétaire. Lorsque le propriétaire crée un locataire, celui-ci lui crée des identifiants qui lui seront transmis afin de permettre son authentification.

Si un locataire décide de s'inscrire directement sur le site, il sera considéré par le système comme un propriétaire et ne pourra pas voir ses documents.

Toutes les données créées par le propriétaire sont liées à son identifiant. Il est le seul à pouvoir accéder à ses données et celles de ses locataires. Il ne peut pas voir celles des autres propriétaires.

D'autres moyens d'authentification des locataires existent sur le marché tel que le système de code. Le propriétaire fournit un code à tous ses locataires qui l'entrent lors de leur inscription afin d'accéder à un groupe. Ce système comporte un problème majeur : le compte du locataire ne peut pas être désactivé s'il quitte l'appartement. De plus, le propriétaire doit attendre que le locataire ait rejoint son groupe avant de pouvoir lui créer des documents. Ce qui implique une certaine perte de temps. Ces raisons incitent à ne pas utiliser ce système pour ce projet.

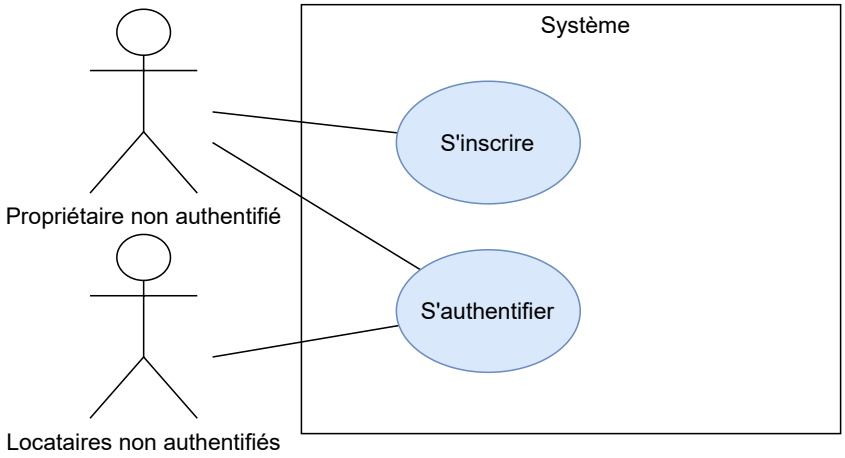


Figure 2.1. – Cas d'utilisation pour un utilisateur non authentifié

Une fois que les utilisateurs sont authentifiés, nous pouvons poursuivre avec les cas d'utilisation suivants.

2.3. Cas d'utilisation du propriétaire

2.3.1. Gestion des immeubles

Les propriétaires peuvent posséder un ou plusieurs immeubles. Il faut différencier un propriétaire qui possède un immeuble avec tous les appartements compris et un propriétaire qui possède des appartements dans différents immeubles. Ce cas d'utilisation est important pour les propriétaires

qui possède un immeuble avec tous les appartements. Cependant, pour ceux qui ne possèdent pas d'immeubles, ce cas d'utilisation est optionnel. En effet, l'application n'en est pas dépendante pour fonctionner correctement.

Le propriétaire peut créer un immeuble, le modifier et le supprimer. Pour chaque immeuble, il doit avoir la possibilité de voir :

- le nombre d'appartements que l'immeuble possède
- le nombre d'appartements occupés

Ces deux éléments sont visibles via des compteurs.
Ces compteurs s'incrémentent et se décrémentent dynamiquement en fonction des actions effectuées lors de la gestion des appartements et des contrats.

Pour chaque immeuble, les propriétaires verront les locataires qui y habitent.

La création d'un immeuble nécessite simplement une adresse. L'intérêt de passer par le cas d'utilisation de gestion d'immeuble est d'éviter la redondance d'adresses dans la base de données.

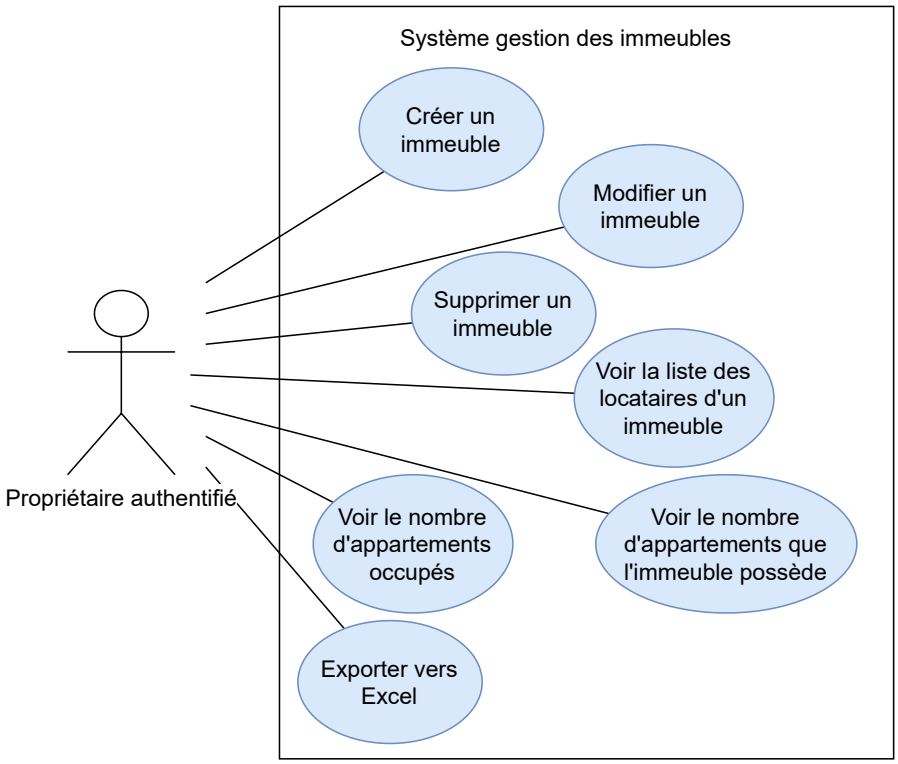


Figure 2.2. – Cas d'utilisation pour la gestion des immeubles

2.3.2. Gestion des appartements

Deux cas de figures se présentent pour la gestion des appartements :

- Premièrement, le propriétaire possède un immeuble et tous ses appartements.
- Deuxièmement, le propriétaire peut avoir un appartement isolé dans un immeuble qui ne lui appartient pas.

Le propriétaire aura la possibilité de choisir entre ces deux options. Comme pour la gestion des immeubles, il peut modifier les appartements et les supprimer.

Si le propriétaire crée un appartement isolé, il devra fournir une adresse et la taille de l'appartement (nombre de pièces). Dans le cas contraire, il devra choisir l'immeuble via un menu déroulant et entrer la taille de l'appartement. Cette action entraîne l'incrémentation du compteur d'appartement de l'immeuble.

La suppression d'un appartement lié à un immeuble entraîne la décrémentation du compteur des appartements de l'immeuble. De plus, le contrat lié à cet appartement est également supprimé.

Le propriétaire peut visualiser les images (photos de l'appartement) qu'il aura ajoutées.

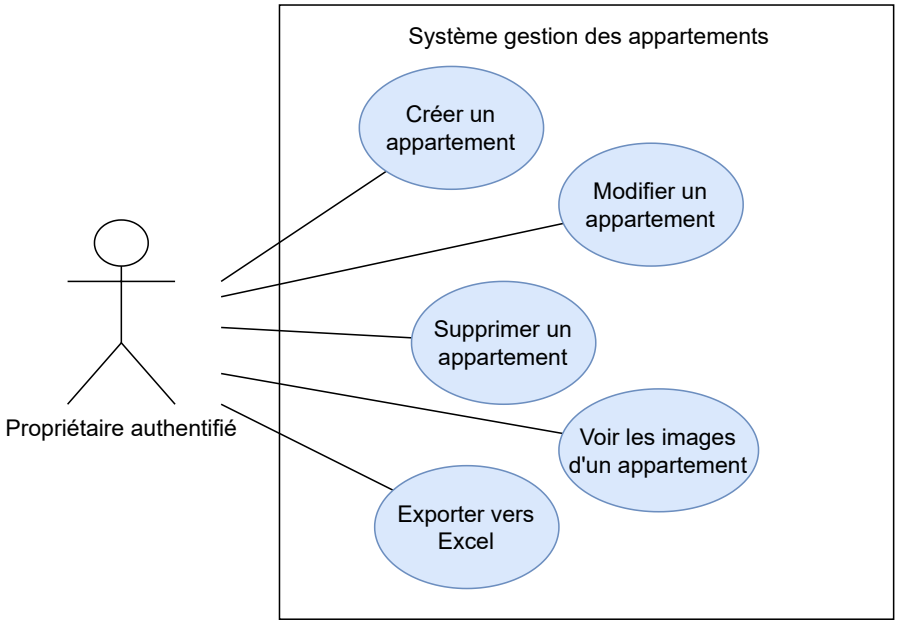


Figure 2.3. – Cas d'utilisation pour la gestion des appartements

2.3.3. Gestion des locataires

La gestion des locataires a deux buts :

- lier les divers documents à chaque locataire.
- permettre aux locataires de se connecter et de voir leurs documents.

Le propriétaire est en mesure de créer, modifier et supprimer des locataires. Il peut également désactiver le compte du locataire, ce qui empêche toutes futures connexions. Cette option peut être utilisée dans le cas où le propriétaire souhaite conserver les documents.

Si nécessaire, il peut réactiver le compte. Un compte désactivé ne peut plus être utilisé ni par le locataire ni par le propriétaire.

Les données demandées pour la création d'un locataire sont un nom, un prénom, un email, un mot de passe (que le locataire devra changer) et une date de naissance. Le locataire pourra éditer ses données via sa page personnelle.

Le propriétaire a la possibilité d'accéder à tous les documents de n'importe quels locataires, tandis que le locataire ne peut accéder qu'à ses documents.

La suppression d'un locataire entraîne la suppression de tous ses contrats, ses factures et leurs historiques. Les appartements occupés par le locataire deviennent libres et sont de nouveau louables.

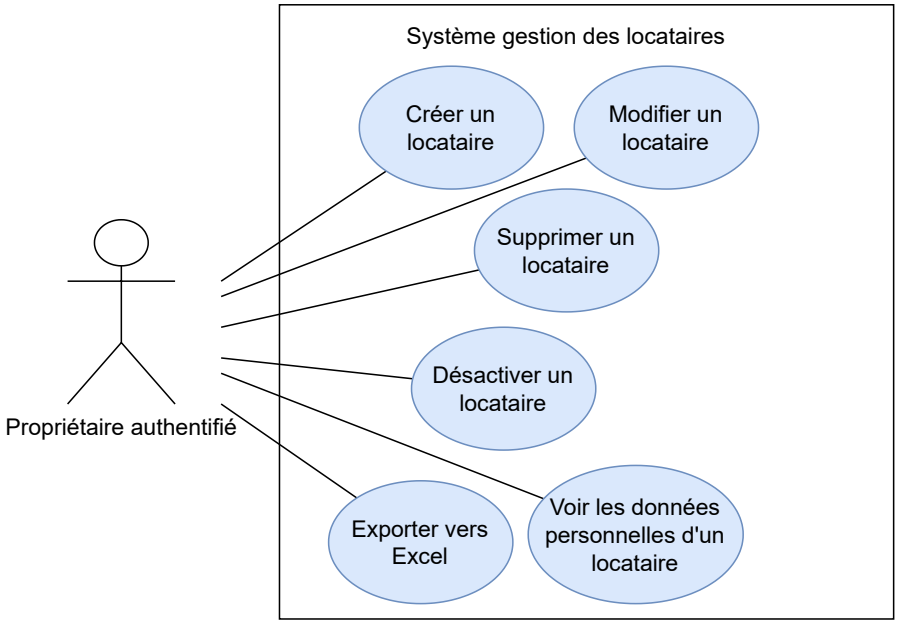


Figure 2.4. – Cas d'utilisation pour la gestion des locataires

2.3.4. Gestion des contrats

Le propriétaire peut créer, modifier, supprimer et archiver un contrat. Un locataire est lié à un propriétaire à travers un contrat. Le contrat possède :

- l'identifiant du locataire
- l'identifiant de l'appartement
- le loyer
- les charges
- les informations supplémentaires
- le statut

Lorsqu'un contrat est créé, l'appartement correspondant change son statut en "Occupé". Il n'est donc plus disponible dans la liste des appartements louables. Si l'appartement en question appartient à un immeuble, le compteur d'appartements "occupés" de l'immeuble s'incrémente.

Lorsqu'un contrat n'est plus actif, ce qui arrive généralement lorsqu'un locataire quitte l'appartement, le propriétaire n'aura qu'à archiver le contrat. Cette action libère l'appartement qui passe en statut "Libre".

Si l'appartement est lié à un immeuble, le compteur d'appartements "occupés" de l'immeuble se décrémente. L'appartement devient de nouveau louable. La suppression d'un contrat actif entraîne les mêmes conséquences que l'archivage d'un contrat.

Le propriétaire a également la possibilité d'ajouter des documents, tel que les fiches de salaire ou les pièces d'identité, au contrat.

Le locataire peut voir ses contrats depuis sa page d'accueil.

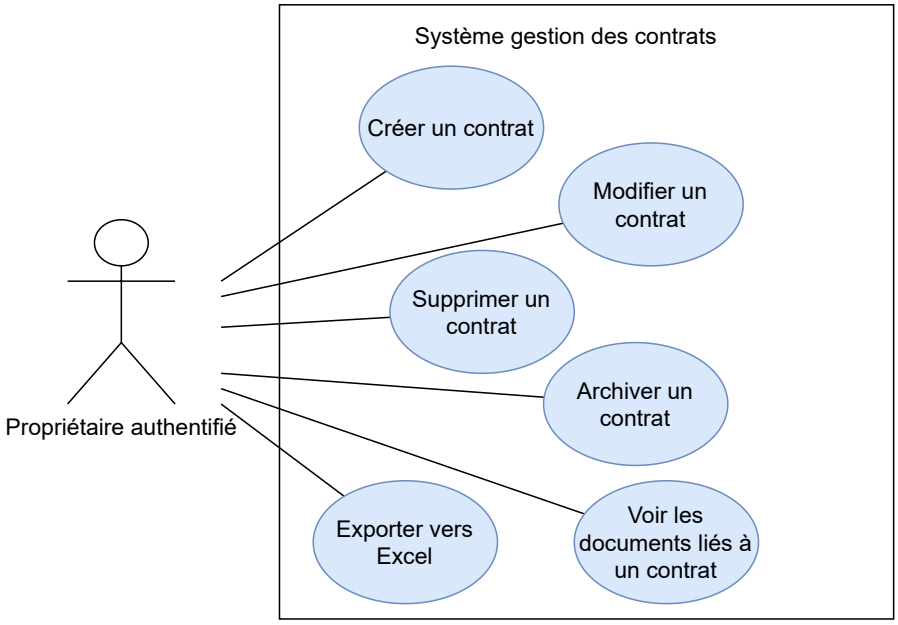


Figure 2.5. – Cas d'utilisation pour la gestion des contrats

2.3.5. Gestion des factures

Le propriétaire peut créer, modifier et supprimer des factures. La création d'une facture nécessite :

- le nom du locataire
- le statut de la facture
- une référence (facultative)
- un montant
- une raison
- une date d'échéance

Le propriétaire doit pouvoir mettre à jour le statut et l'échéance de la facture en fonction des actions du locataire. Par exemple, si le locataire n'a toujours pas payé sa facture, le propriétaire changera le statut en "premier rappel" et prolongera l'échéance. Les modifications effectuées aux factures sont visualisables via un historique de facture. Le propriétaire peut également ajouter des documents aux factures si nécessaire.

La suppression d'une facture entraine également la suppression de toute l'historique de celle-ci.

Le locataire peut accéder à ses factures depuis sa page d'accueil.

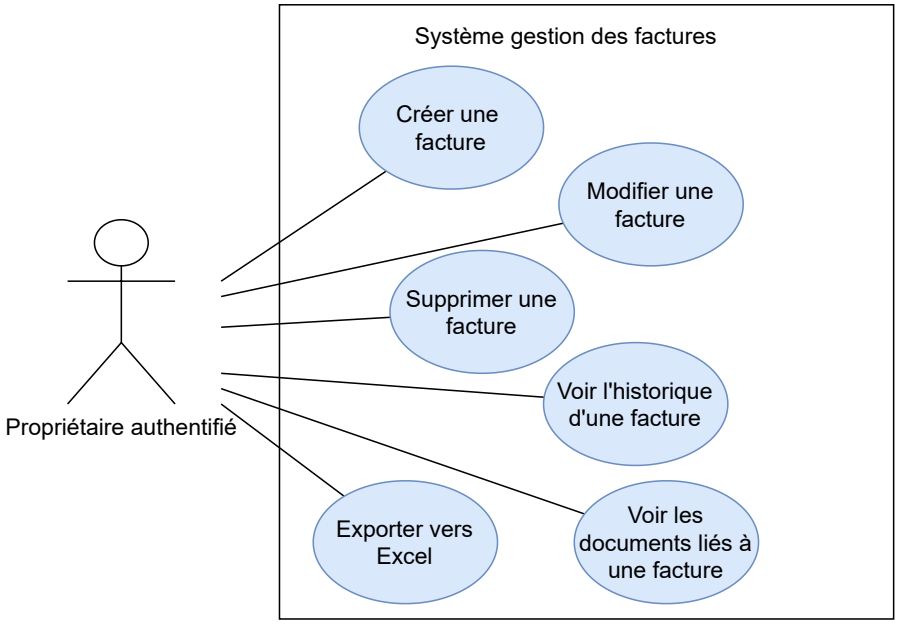


Figure 2.6. – Cas d'utilisation pour la gestion des factures

2.3.6. Gestion des messages

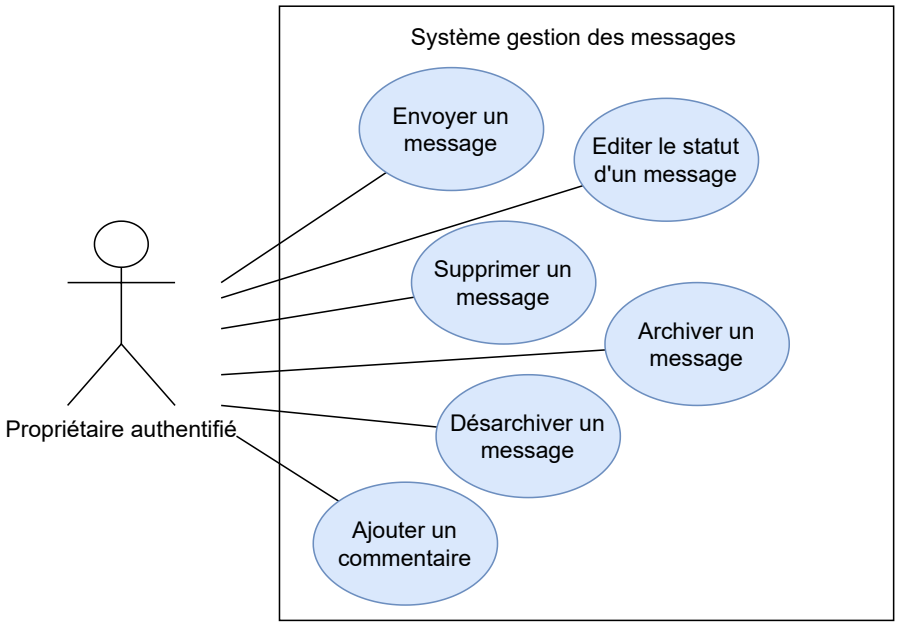


Figure 2.7. – Cas d'utilisation pour la gestion des messages

La gestion des messages est la partie intéractive de l'application web. Elle permet aux locataires de communiquer avec le propriétaire, que ce soit pour des services ou des problèmes. De plus, elle permet au propriétaire de passer des annonces aux locataires.

Le propriétaire peut envoyer un message à un locataire ou un message groupé à plusieurs locataires. Le(s) locataire(s) recevant le message peuvent intéragir en le commentant. Le message et les com-

mentaires sont visibles par l'expéditeur et le(s) destinataire(s). Le contenu du message ne peut pas être modifié une fois envoyé.

Le propriétaire est la seule personne qui a la possibilité de supprimer un message, de l'archiver et de le désarchiver. A partir d'un message, il est possible de créer une tâche (voir point 2.2.8) et de voir l'avancement de son statut.

2.3.7. Gestion des tâches

Une tâche est une action que le propriétaire devra entreprendre vis à vis du locataire. Cette tâche peut découler de deux manières :

- le propriétaire décide d'effectuer une tâche.
- le locataire demande à son propriétaire d'effectuer une tâche via la messagerie.

Un exemple permet d'illustrer le premier point. Le propriétaire décide de repeindre le bâtiment qu'il possède. Dans ce cas, il créera une tâche contenant un titre, une brève description , un statut (qui pourra être modifié par la suite), une date de début et une date de fin. Au fil de l'avancée, le propriétaire mettra à jour les données de la tâche si nécessaire.

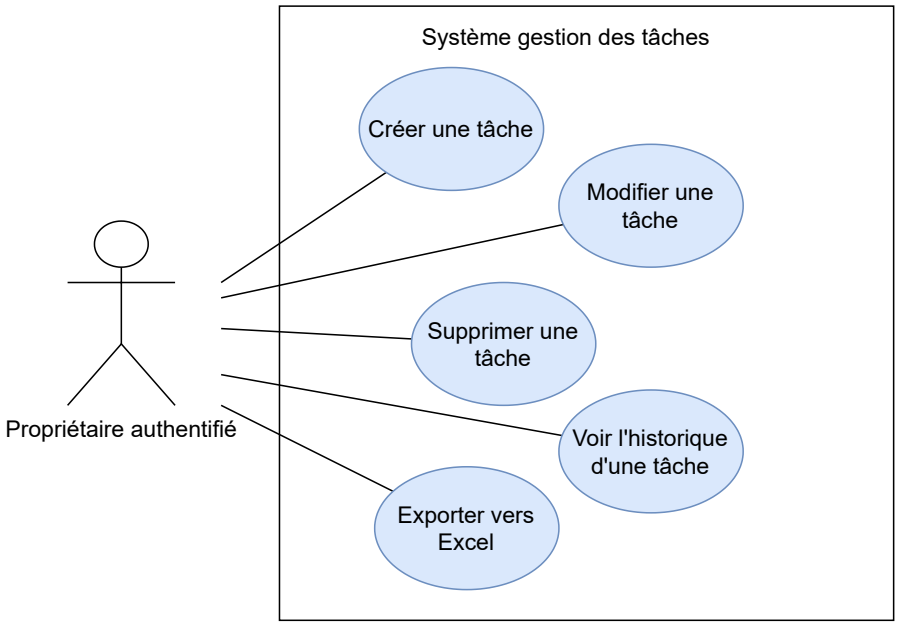


Figure 2.8. – Cas d'utilisation pour la gestion des tâches

Dans le deuxième point, le locataire demande à son propriétaire d'effectuer une tâche via la messagerie. La tâche créée contient alors l'identifiant du message en question. Le propriétaire aura une trace de l'échange et l'action ne pourra pas être répudiée.

La création d'une tâche liée à un message fait intervenir la figure 2.9 qui représente le diagramme d'activité d'une tâche. Ce diagramme est un exemple de cycle de vie d'une tâche, de sa création à sa résolution. Le diagramme n'est pas fixe et les évènements mentionnés n'interviennent pas forcément ou ne se déroule pas dans l'ordre indiqué. Le diagramme commence dès la création de la tâche.

La figure 2.9 se déroule comme ceci :

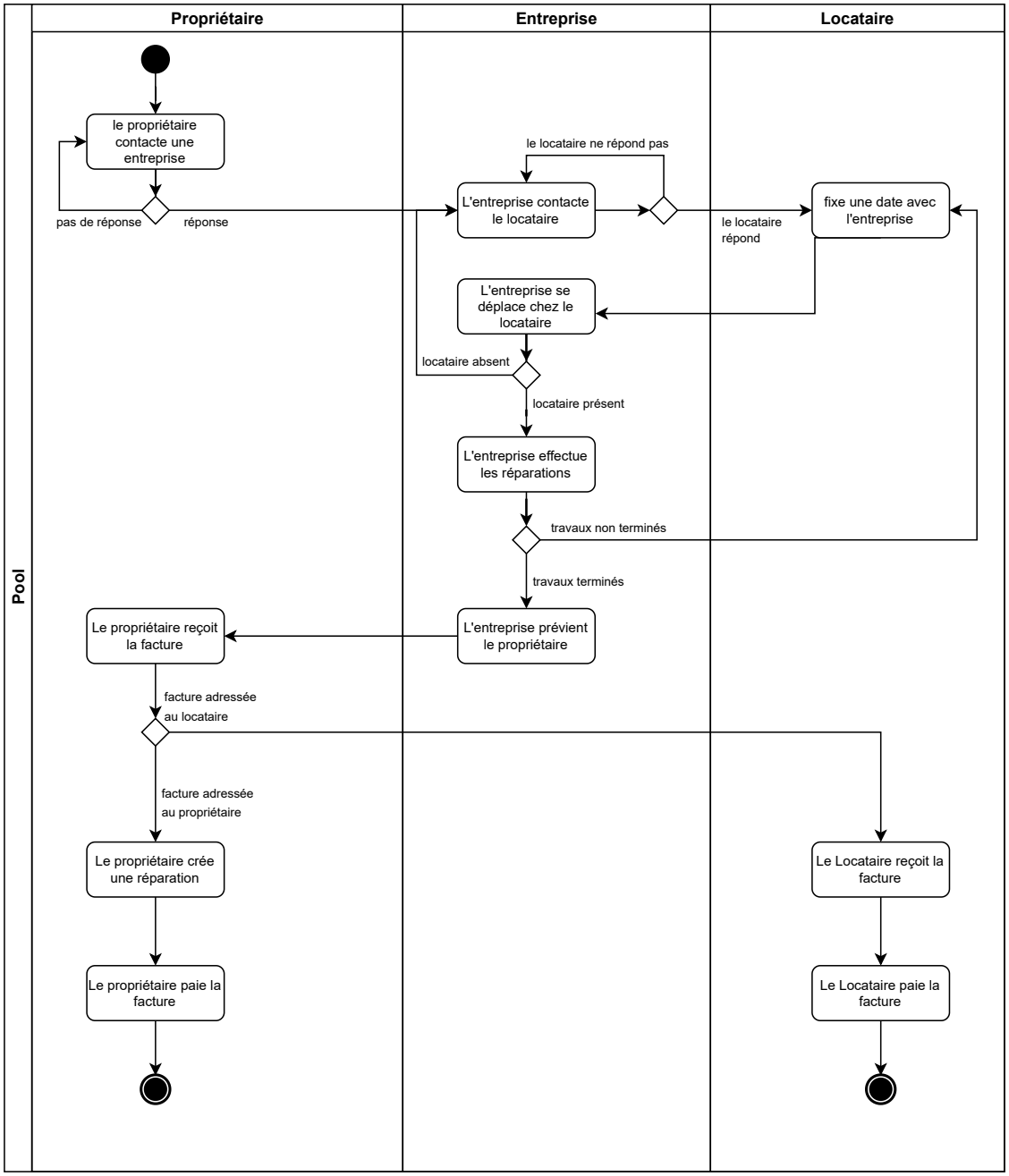


Figure 2.9. – Diagramme d'activité - cycle de vie d'une tâche

1. Le propriétaire contacte une entreprise et lui fait part des problèmes rencontrés par son locataire.
2. L'entreprise contacte le locataire et fixe un rendez-vous.
3. L'entreprise se déplace chez le locataire afin d'effectuer les travaux.
4. Si les travaux nécessitent plus de temps que prévu, l'entreprise fixe un nouveau rendez-vous avec le locataire.
5. Une fois les travaux terminés, l'entreprise envoie la facture au propriétaire.
6. Le propriétaire juge alors si les frais sont à sa charge ou à ceux du locataire.

7. Si les charges lui reviennent (par exemple, le plafond s'effrite dû à l'usure), le propriétaire crée une réparation. Le système de gestion des réparations est expliqué au point 2.3.8.
8. Si le propriétaire juge que les frais sont à la charge du locataire, il lui enverra la facture.

Le propriétaire sera systématiquement mis au courant de l'avancement de la tâche afin qu'il puisse effectuer les modifications nécessaires. Il a également la possibilité de voir l'historique d'une tâche. Le locataire peut voir les tâches qui le concernent depuis sa page d'accueil.

Tous les messages ne nécessitent pas la création d'une tâche. Ce sera au propriétaire de juger au cas par cas.

2.3.8. Gestion des réparations

Les réparations sont les frais liés au propriétaire qui interviennent à la fin d'une tâche. Une tâche est nécessaire pour créer une réparation.

Une réparation peut être modifiée et supprimée par le propriétaire. Il peut également ajouter des documents si nécessaire.

L'utilisation de la gestion des réparations permet de faciliter la comptabilité. En effet, cela permet de séparer les frais en deux catégories : les factures usuelles (loyers, charges) et les réparations (frais extraordinaires liés à des dégâts).

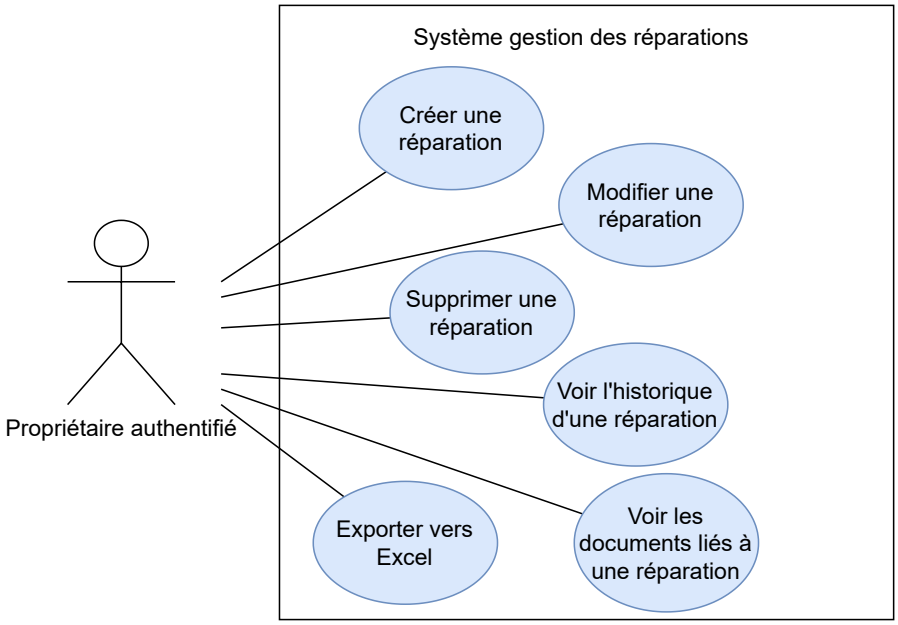


Figure 2.10. – Cas d'utilisation pour la gestion des réparations

2.3.9. Gestion des statuts

La gestion des statuts permet de gérer les différents statuts des factures, des réparations et des tâches. Ils sont gérés de manière dynamique et permettent d'établir un historique. Par exemple, nous pouvons modifier un statut "Premier rappel" en "Deuxième rappel".

Cependant, certains statuts sont statiques. Par exemple, un appartement peut soit être "Libre" soit "Occupé". Il n'y a donc aucune raison de les gérer de manière dynamique. C'est pourquoi ces statuts-là sont gérés depuis l'interface client et non pas par un cas d'utilisation de gestion de statut.

Les différents propriétaires ne traitent pas les factures de la même façon ni avec les mêmes statuts. C'est pourquoi, il existe un système de gestion de statut afin que les propriétaires puissent créer les statuts dont ils ont besoin. Par exemple : A envoyer, A traiter, En attente.

Les statuts peuvent être créer, modifier et supprimer. Ils apparaissent dans les différents filtres disponibles dans la partie client.

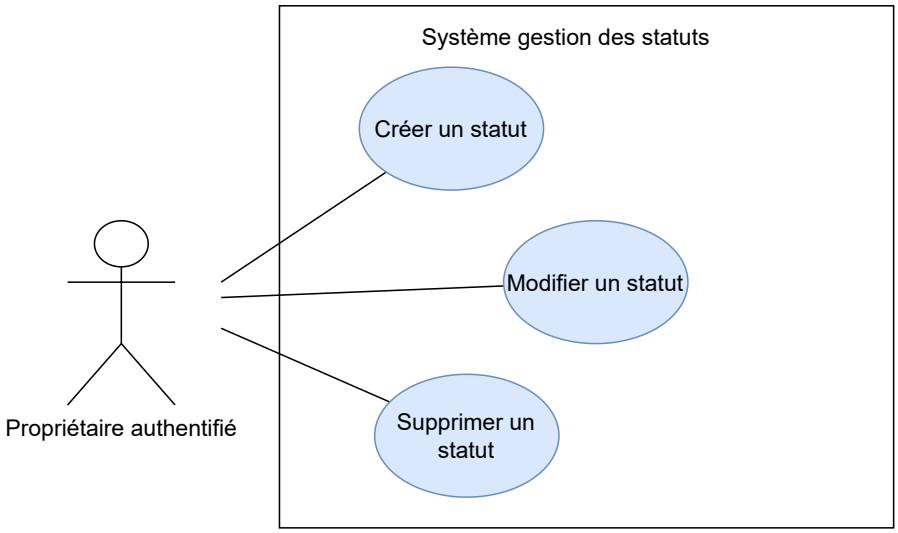


Figure 2.11. – Cas d'utilisation pour la gestion des statuts

2.3.10. Gestion des documents

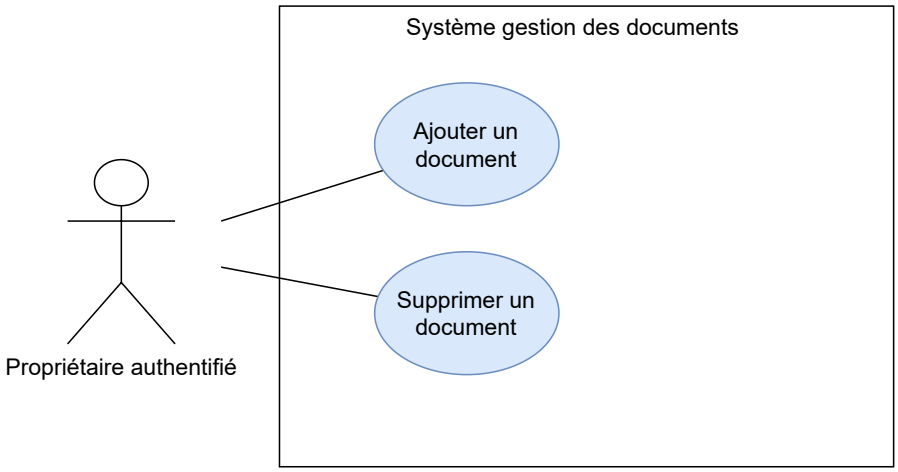


Figure 2.12. – Cas d'utilisation pour la gestion des documents

Afin de maintenir un suivi et une traçabilité, le propriétaire peut ajouter des documents à :

- ses appartements

- ses locataires
- ses contrats
- ses factures
- ses réparations

L'ajout et la suppression des documents ont un cas d'utilisation indépendant afin d'éviter de surcharger les différents systèmes. Les documents peuvent également être visualisés ou téléchargés depuis ce système. Le propriétaire peut ajouter un ou plusieurs documents à la fois. L'ajout des documents peut être une plus-value pour le propriétaire qui peut ainsi garder les preuves des documents directement dans l'application.

2.4. Cas d'utilisation des locataires

Tous les cas d'utilisation présentés jusqu'à maintenant (exceptée la messagerie) ne sont disponibles que pour le propriétaire authentifié. Néanmoins l'application s'adressent aussi aux locataires. C'est pourquoi ils possèdent également des cas d'utilisation mais de type visualisation.

Les locataires peuvent voir leurs factures personnelles, leurs contrats personnels, leurs tâches prévues ainsi que leurs documents personnels. Ils peuvent également mettre à jour leurs données.

Tout comme le propriétaire, les locataires ont la possibilité d'envoyer des messages et de les commenter. Les messages se transmettent uniquement au propriétaire. L'application n'étant pas une messagerie privée, la fonction de discussion entre locataires n'a pas été implémentée.

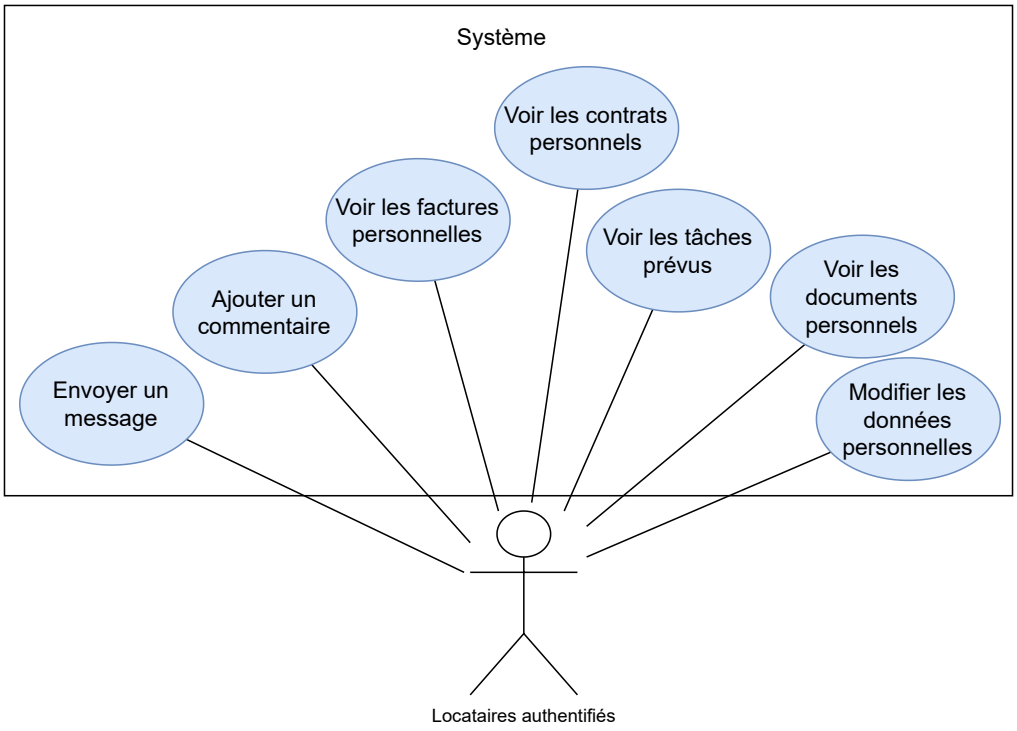


Figure 2.13. – Cas d'utilisation des locataires

Nous avons vu tous les cas d'utilisation : ceux du propriétaire et ceux des locataires. L'étude des cas d'utilisation a permis de mettre en valeur leur fonctionnement complexe et a permis de décomposer

le système en sous-système de gestion. Il est ainsi plus facile de construire son application web et de le présenter. Les cas d'utilisation nous auront permis de :

- déterminer les fonctionnalités de chaque système de gestion.
- expliquer et présenter plus facilement aux personnes externes.
- prévoir plus facilement des tests afin de contrôler le fonctionnement.

3
Serveur et base de données

Le chapitre parle de la composition du serveur ainsi que de la base de données. Les divers aspects seront expliqués de manière théorique.

3.1. Fonctionnement

La structure idéale d'une application dite full stack repose sur deux critères :

- Le client doit être séparé du serveur. Les deux doivent être indépendants l'un de l'autre.
- Le client doit communiquer avec le serveur en effectuant des requêtes. Le serveur envoie une réponse à la requête à traiter.

Divers langages permettent de développer un backend, constitué d'un serveur et d'une base de données. Dans le cas de ce projet, le serveur et la base de données ont été élaborés grâce à :

- Node.js : un compilateur permettant d'exécuter du javascript côté serveur.
- Koa : un framework orienté web pour Node.js
- MongoDB : une base de données de type non-relationnelle.

Le serveur utilise une architecture dite REST (point 3.4) et la communication client-serveur se fait par des endpoints (point 3.5). La majorité des endpoints requièrent des droits d'accès. Ils sont gérés en combinant un middleware (point 3.5.1) avec un token d'authentification (point 3.5.2). Le tout est documenté en utilisant Swagger (point 3.5.3).

3.2. Node.js

Nodejs est un environnement de développement javascript orienté réseau [10]. Il est un excellent outil pour le développement des applications web interactives. Il permet d'effectuer des requêtes en continu contrairement à d'autres langages fonctionnant sous un système I/O. Grâce à son fonctionnement dit asynchrone, les requêtes n'ont pas besoin d'être exécutées dans l'ordre d'arrivée. Il est possible de traiter des grandes quantités de données très facilement. De plus, Node.js dispose également d'une grande communauté qui participe à son existence et à son évolution. Nous pouvons trouver divers outils mis à disposition tels que les librairies, forum en ligne, etc. Finalement, Nodejs est facile et rapide à apprendre et à mettre en place.

3.3. Koa

Koa est un framework orienté web facilitant le développement d'application web [6] avec Node.js. De structure minimaliste, il est facile et rapide à apprendre. Il permet de gérer facilement les différentes erreurs à travers des try/catch.

3.4. L'architecture REST

Un API (application programming interface) est un ensemble de classes, de fonctions, de constantes qui permettent à un système (client) d'utiliser ses services. Un exemple d'API couramment utilisé est celui de Google qui permet à des applications tierces d'utiliser les fonctionnalités de Google Maps ou Calendar.

REST (Representational state transfer) est une architecture logicielle qui définit un ensemble de règle à respecter lors du développement d'une API (offrant des services Web). Les données et les fonctionnalités sont considérées comme des ressources et sont accessibles via des requêtes effectuées sur des liens appelés endpoints. Les communications client-serveur se font via un RESTful API qui utilisent le protocole Web, typiquement https.

Un RESTful API utilise cinq types de requête (aussi appelé verbe http) fourni par l'interface http[3] :

- GET : permet de cibler une ressource et d'en retourner une représentation.
- POST : permet de créer une ressource.
- PUT : permet de mettre à jour une ressource, en modifiant les données ou en créant les champs manquants.
- PATCH : permet de mettre partiellement à jour une ressource.
- DELETE : permet de supprimer une ressource.

Son développement doit suivre six contraintes[5] :

1. **Client-serveur.** Le client et le serveur doivent être indépendants l'un de l'autre. Cela permet une meilleure évolutivité des différents composants.
2. **Sans état.** Les communications doivent se faire sans connaissance de l'état. Cela signifie que les données concernant la session d'utilisateur doivent être seulement stockées côté client. Les modifications devront se faire seulement dans cette partie. Les requêtes sont également

indépendantes les unes des autres. Le client fournit toutes les informations nécessaires pour que le serveur puisse répondre correctement aux requêtes.

3. **Mise en cache.** Les diverses données provenant du serveur peuvent être mises en cache et réutiliser par le client lorsqu'une requête du même type se présente à nouveau. La mise en cache doit être utile au client afin d'avoir des réponses à jour.
4. **En couche.** Afin de faciliter l'évolution et de renforcer la sécurité de l'API, l'architecture REST peut être composée de plusieurs couches indépendantes les uns des autres. Le client ne sait pas à quelle couche du serveur il s'adresse, ni ce qui se passe pour l'obtention de la réponse.
5. **Interface uniforme.** La méthode de communication doit être la même pour toutes les ressources. Les ressource doivent être identifiables, via des URI, représentables et auto-descriptives. L'accès doit se faire via des requêtes (GET, POST, PUT, PATCH ou DELETE) et le serveur retourne une réponse contenant un body et un header.
6. **Code à la demande.** Le serveur doit pouvoir transférer du code exécutable au client, ce qui lui permet d'implémenter moins de fonctions par défaut et améliorer son extensibilité. Cette contrainte est facultative.

L'utilisation d'une architecture REST permet donc un couplage faible entre le client et le serveur et facilite leurs maintenances ainsi que leurs évolutions. De plus, l'utilisation est également facilitée par le respect d'une structure uniforme.

La structure REST n'est pas obligatoire lors du développement d'un API, mais elle permet d'établir quelques bonnes pratiques à suivre.

3.5. Endpoints

Les endpoints sont les points de communication utilisés par le client ou par une interface afin de communiquer avec le serveur et la base de données. Le client effectue des requêtes de type GET, POST, PUT, PATCH ou DELETE et reçoit une réponse de la part du serveur. Les endpoints spécifient où se trouvent les ressources et comment y accéder. Ils permettent également de retourner uniquement la ressource demandée (suivant les droits d'accès) et garantissent une meilleure protection des ressources. Finalement, ils permettent de tester le bon fonctionnement du serveur grâce à l'utilisation d'une documentation tel que Swagger (point 3.5.3).

Figure 3.1. – Structure d'un endpoint

Un endpoint peut être décomposé en plusieurs fragments :

- **Un protocole (http ou https).** Il s'agit d'un ensemble de règles qui doivent être respectées afin d'effectuer un type de communication particulier. La communication s'effectue via le port 80 (http) ou 443 (https).

- **www.** Le world wide web contient des documents textes, images, vidéos etc. Les divers documents sont identifiés grâce à un nom unique. Le contenu d'un document peut pointer vers un autre.
- **appartex-server.heroku-app.com.** Il s'agit du nom de domaine. Il permet au serveur DNS de retourner l'adresse IP correspondant et amène également une meilleure mémorisation par l'utilisateur.
- **building/5fd32687e9ad210015711337.** Il s'agit du chemin d'accès vers une ressource. Dans notre exemple, le endpoint vise une ressource précise se trouvant dans la collection "building".

Il est également possible d'ajouter des paramètres qui peuvent servir de filtre. Ils peuvent être ajoutés à la fin de l'url sous la forme " ?paramètre=value". L'utilité des paramètres dépend de l'implémentation du serveur.

Les différents cas d'utilisation vu au chapitre 2 sont représentés par des endpoints. Le client effectue des requêtes au serveur grâce à ces endpoints et le serveur retourne une réponse et un code de réponse http.

Les codes de réponses sont numérotés de la façon suivante[4] :

- **1xx** indique que le serveur transmet une information. (Par exemple : le code 101 signifie que le changement de protocole a été accepté.)
- **2xx** indique que la requête a pu être traitée avec succès.
- **3xx** indique une redirection. Le client doit effectuer des actions supplémentaires pour traiter la requête.(par exemple : le document souhaité a été déplacé.)
- **4xx** indique que l'erreur provient du client (par exemple : les droits d'accès sont refusés).
- **5xx** indique que l'erreur provient du serveur (par exemple : connexion impossible avec le serveur).

3.5.1. Les middlewares

Un middleware est un logiciel tiers qui facilite la communication entre deux logiciels informatiques. Il peut servir à rajouter des fonctions supplémentaires redondantes ou, comme dans notre cas, à effectuer une vérification afin de donner l'accès à un endpoint. Il n'est pas spécifique au serveur et peut très bien être utilisé dans la partie client.

Le serveur possède :

- un middleware vérifiant l'existence et la validité du token.
- un middleware vérifiant si le token appartient au propriétaire.
- un middleware vérifiant si le token appartient au propriétaire ou si le locataire identifié a le droit d'accéder à la ressource demandée.

Les endpoints sont accessibles par les propriétaires et les locataires. Il est donc important de savoir qui a accès à quoi et d'offrir une certaine sécurité contre les actions malintentionnées.

```
module.exports = async (ctx, next) => {
  if (!ctx.request.jwt) ctx.throw(403, "No token.");
  const { role } = ctx.request.jwt;

  if (role !== "Admin") ctx.throw(403, "Access Forbidden");

  await next();
};
```

Listing 3.1 – Middleware vérifiant l'existence d'un token et sa validité pour un endpoint propriétaire

3.5.2. Jeton d'authentification

Un token ou jeton d'authentification est une méthode d'authentification considérée comme forte. Il est généré par le serveur et contient les informations nécessaires à la gestion des droits d'accès.

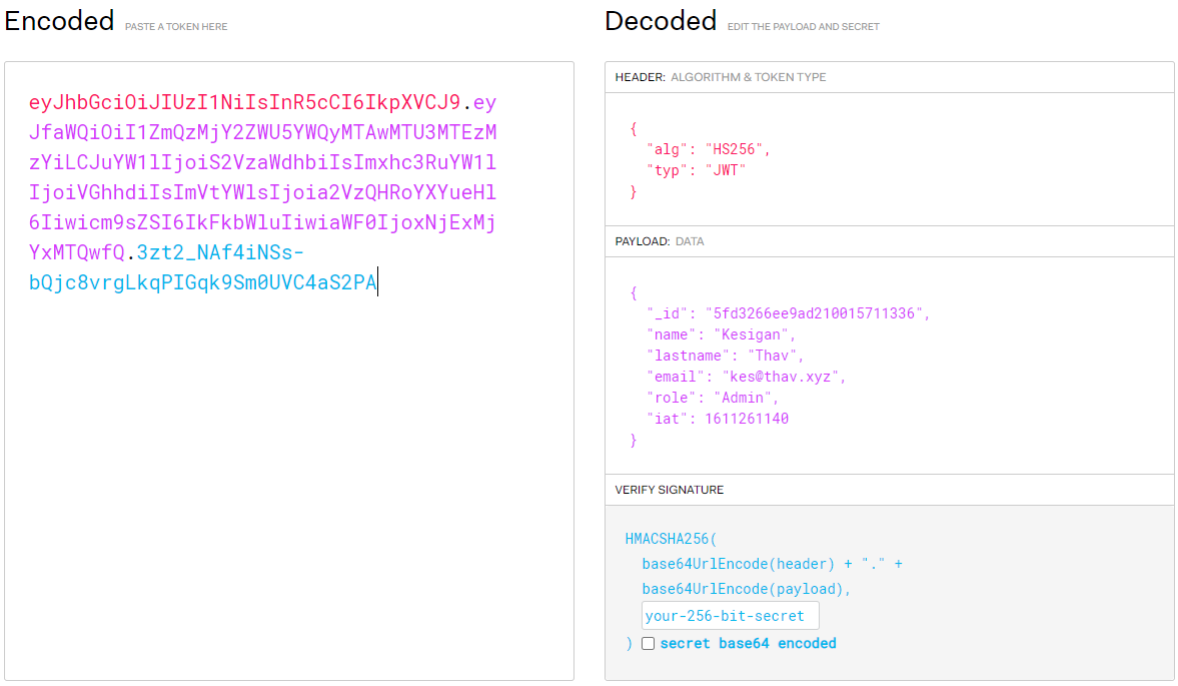


Figure 3.2. – Déchiffrement d'un token - `https://jwt.io/`

Le token peut être décomposé en trois parties[2] :

1. **un header (entête).** Il contient les informations sur l'algorithme utilisé pour générer la signature et le type de token.
2. **un payload.** Il contient les informations utiles à la gestion des droits d'accès. Les données contenues dans cette partie sont choisies par le développeur.
3. **une signature.** La signature permet de vérifier l'authenticité du token. Il est généré en hashant le header avec le payload et une clé secrète.

Lors de chaque requête nécessitant un droit d'accès, le token est envoyé dans l'entête de la requête sous la forme **Authorization : Bearer <token>**. Il est ensuite vérifié par le middleware. Le Bearer indique le type d'autorisation utilisée.

3.5.3. Swagger

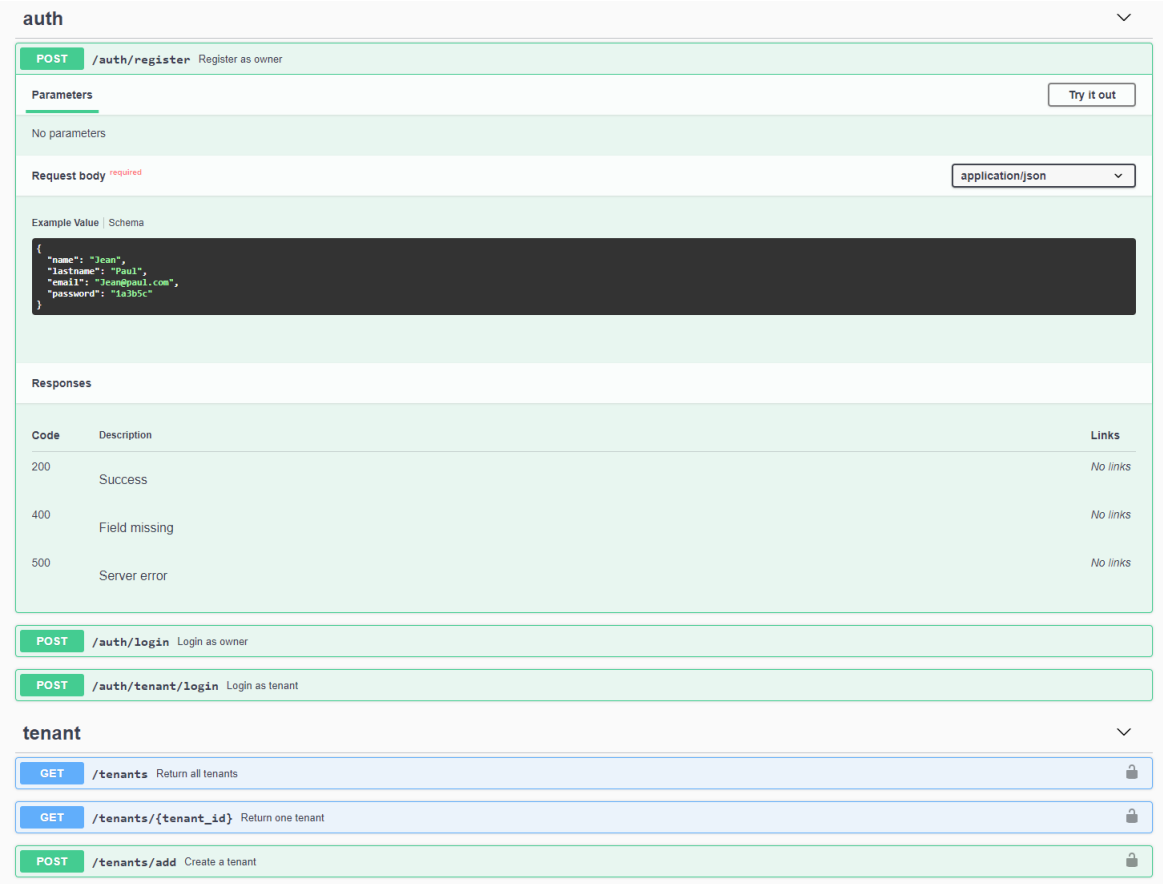


Figure 3.3. – Documentation swagger

Swagger est un outil permettant la documentation des interfaces REST [1]. Il permet également de les tester. La documentation des endpoints peut contenir un ou plusieurs éléments tels que :

- Les type de requêtes qui seront effectués
- Le chemin d'accès à la ressource
- **un paramètre.** Il s'agit généralement de cibler une ressource précise grâce à un identifiant. Par exemple, le endpoint se terminant par "$/tenants/\{tenant\_id\}$" a besoin de l'identifiant du locataire pour être utilisé. Il est souvent employé pour des requêtes de type GET, PUT ou DELETE
- **le corps de requête.** Il précise la structure que doit adopter les données qui seront envoyées vers le serveur. Les requêtes de type POST ou PUT l'utilisent.
- **une réponse.** Il documente les différentes réponses que l'utilisateur peut avoir en employant le endpoint. Les types de réponses dépendent de l'implémentation.

- **un cadenas**. Il indique que l'utilisation de l'endpoint nécessite un droit d'accès. En général, les droits d'accès sont gérés grâce à l'utilisation d'un token d'authentification généré par le serveur (JWT par exemple). Ce token contient les détails sur l'utilisateur.

Certains endpoints requièrent des informations spécifiques fournies par l'entête de la requête (header) :

- **Authorization.** Il contient les informations permettant d'identifier l'utilisateur qui effectue la requête. Il est principalement utile pour les requêtes nécessitant un droit d'accès. Dans notre cas, le serveur génère un token d'authentification qui doit être stocké. A chaque requête où il est nécessité, son existence et sa validation sont vérifiées par le serveur en le déchiffrant.
- **Content-Type.** Il définit le format des données envoyées au serveur et le format de la réponse attendue. Dans notre cas, le format des données par défaut est "application/json" et celui des documents "multipart/form-data". Il est important de le préciser pour éviter les erreurs.

La documentation est disponible ici : `https://appartex-server.herokuapp.com/doc`

3.6. MongoDB

Mongodb est une base de données de type non-relationnelle. Elle est open source et gratuite[8]. Une base de données dite relationnelle stocke les données sous forme de tableaux possédant des relations entre elles. Contrairement à cela, MongoDB crée des collections et y stocke les données sous forme de document JSON. Les documents sont indépendants entre eux et n'ont pas besoin de structure pré-définie, ce qui permet de gérer des structures complexes. La lecture et l'écriture des données sont plus rapides. MongoDB est également simple d'utilisation et flexible. C'est la base de données généralement utilisée avec Node.js.

3.6.1. Structure de la base de données

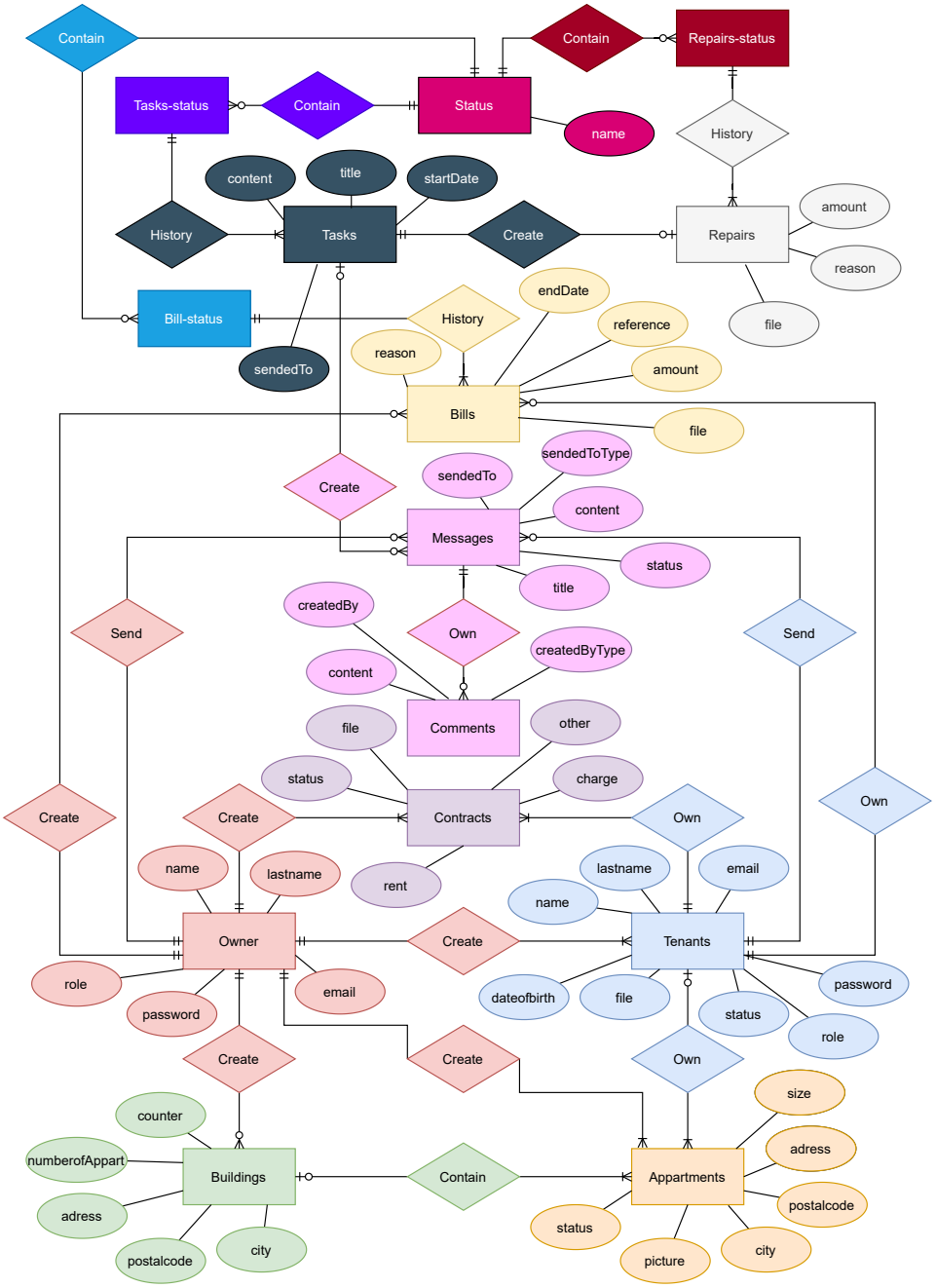


Figure 3.4. – Diagramme entité-relation

La structure et l'organisation de la base de données sont représentées grâce à la figure 3.4. appelée diagramme entité-relation.

Diagramme entité-relation

Un diagramme entité-relation[15] est un modèle conceptuel qui permet d'exprimer les diverses relations existantes entre les entités, représentées avec des rectangles. Les associations entre les entités sont symbolisées par des losanges et les attributs des entités par des ellipses. Les formes aux extré-

mités des entités décrivent le nombre de relations. La notation utilisée est celle de Crow's foot. Les relations se lisent comme ceci :

-  se lit 0 à 1
-  se lit strictement 1.
-  se lit 0 à plusieurs
-  se lit 1 à plusieurs

En tenant compte de ces infomations, la relation propriétaire-factures se lit "chaque propriétaire crée zéro ou plusieurs factures. Chaque facture est créée par un seul propriétaire".

Le modèle entité-relation a l'avantage d'être clair et précis. De plus, lors du développement de notre base de données, nous pouvons nous y référer facilement.

Bien que MongoDB est une base de données non-relationnelle et ne nécessite pas de structure prédéfinie, il est quand même judicieux de définir des modèles. Cela permet de structurer les données qui seront utilisées par le client.

L'entité principale de notre base de données est le propriétaire (Owner). Toutes les entités possèdent l'identifiant du propriétaire afin de les distinguer des autres propriétaires. La structure du diagramme est la suivante :

- Le propriétaire a la possibilité de créer un ou plusieurs locataires (Tenants). Un locataire appartient à un seul propriétaire.
- Le propriétaire est lié aux locataires via des contrats et inversément (Contracts). Un locataire possède un ou plusieurs contrats mais un contrat n'appartient qu'à un seul locataire.
- Le locataire possède un ou plusieurs appartements (Appartments).
- Les appartements peuvent faire partis d'un immeuble, mais ce n'est pas une obligation. Un immeuble doit posséder au moins un appartement.
- Le propriétaire et les locataires peuvent communiquer entre eux par le biais de messages.
- Les messages peuvent posséder des commentaires, s'il y a eu une discussion.
- Les locataires ont des frais vis à vis du propriétaire, que ce soit des loyers, charges ou autres. Tout ceci est représenté par l'entité facture (Bills). Une facture appartient qu'à un seul locataire.
- A chaque création et modification d'une facture, un document est créé afin de garder une trace et d'établir un historique (Bill-status). Ce document possède l'identifiant de la facture ainsi que l'identifiant du statut qui lui a été attribué.
- Un message peut résulter sur la création d'une tâche mais ce n'est pas toujours le cas. Une tâche peut également être créée sans l'intervention d'un message.
- A chaque création et modification d'une tâche, un document apparaît afin de garder une trace et d'établir un historique (Task-status). Ce document possède l'identifiant de la tâche ainsi que l'identifiant du statut qui lui a été attribué.
- Si les frais engendrés par une tâche reviennent au propriétaire, celui-ci créera une réparation.
- A chaque création et modification d'une réparation, un document est créé afin de garder une trace et d'établir un historique (Repair-status). Ce document possède l'identifiant de la réparation ainsi que l'identifiant du statut qui lui a été attribué.

Afin de maintenir une certaine cohérence et une relation entre les diverses entités, la structure des données est prédéfinie grâce à des modèles via la librairie "Mongoose" (disponible en annexe). La validité des données est également vérifiée par le client et le serveur (grâce à la librairie "Joi") avant la création des documents.

```
const Joi = require("joi");
Joi.objectId = require("joi-objectid")(Joi);

const contractSchema = Joi.object({
  charge: Joi.number().min(0).precision(2).required(),
  rent: Joi.number().min(0).precision(2).required(),
  tenant: Joi.objectId().required(),
  appartmentid: Joi.objectId().required(),
  other: Joi.string().allow("", null),
  status: Joi.string(),
});

module.exports = {
  contractSchema,
};
```

Listing 3.2 – Code de validation pour la création d'un contrat en utilisant Joi

Toutes les données, exceptées le mot de passe, sont stockées en brut dans la base de données. Le mot de passe est stocké sous forme de hash pour apporter une protection.
Grâce à tous ces outils présentés, le backend va se construire selon les caractéristiques présentées à chaque point.

4
Client

Ce chapitre présente le structure du client. Nous y verrons la présentation des pages principales, un scénario pour le propriétaire et un scénario pour le locataire.

4.1. Interface

Le client a été développé grâce à ReactJS[12] qui est une librairie javascript créée par Facebook. Elle permet le développement d'applications à page unique (single page application). La structure est orienté javascript et possède des technologies permettant un rendu plus rapide. En effet, lors du chargement de l'application, React crée un DOM virtuel et charge les différents composants. Lors d'un changement de page par exemple, il modifie que le composant qui a besoin d'être mis à jour, ce qui évite le rechargement complet de la page. De plus, React utilise une approche par composant ce qui permet la réutilisabilité.

4.1.1. Structure du client

Cette section fait référence à la figure 4.1. Elle présente la structure du frontend simplifiée.

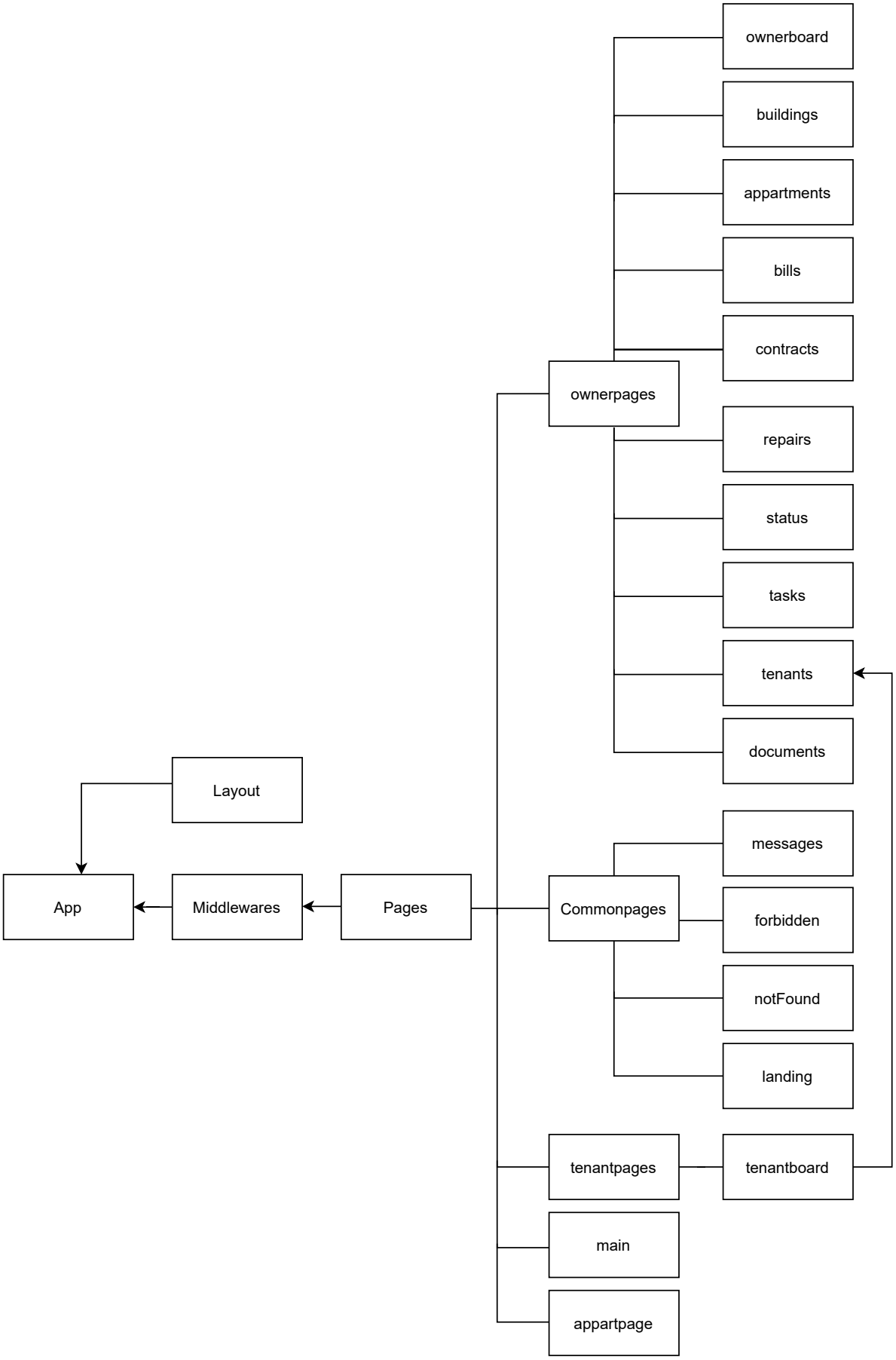


Figure 4.1. – Structure du frontend simplifiée

L'interface client adopte une structure hiérarchique. Le composant principal est App. Il est la racine du projet et contient tous les chemins d'accès aux autres composants. Il stocke les données reçues depuis le serveur et les distribue aux composants qui en ont besoin.

Pour fonctionner correctement, il fait appel à deux autres composants :

- **Layout** qui permet d'ajouter une barre de navigation commune à toutes les pages
- **Middlewares** qui permettent de gérer les accès.

Les trois dossiers principaux qui détiennent les composants sont :

- Ownerpages pour les composants du propriétaire
- Tenantpages pour les composants des locataires
- Commonpages pour les composants en commun

Le composant main est la page d'accueil principale de l'application web (figure 4.2). Le composant appartpage est la page de présentation des appartements (figure 4.3). Ces deux composants seront présentés à la section "Appartex".

4.1.2. Fonctionnement du client

Cette section présente le fonctionnement de la figure 4.1.

Le fontionnement de ces différents composants se déroule de la manière suivante :

- Lorsqu'un utilisateur souhaite accéder à une page, le composant App appelle le composant de la page et le passe à travers un middleware. Le middleware vérifie deux choses :
  - Est-ce que l'utilisateur possède un token d'authentification ?
  - Est-ce que l'utilisateur a les droits pour accéder à cette page ?
- Si le middleware confirme ces deux points, il retourne alors le composant qui peut être affiché à l'utilisateur. Le composant pourra également communiquer avec le composant App afin d'effectuer les requêtes vers le serveur.
  - Si ce n'est pas le cas, c'est le composant "Forbidden" qui sera retourné.
  - Si la page n'existe pas, c'est le composant "notFound" qui sera retourné.

Le design a été réalisé grâce à Material-ui [9], un framework CSS créé par Google.

Cette structure permet de gérer les différentes pages. L'utilisateur ne pourra accéder qu'aux pages dont il possède les droits.

4.2. Appartex

Les concepts vus au chapitre 2 et 3 ont été mis en pratique grâce au développement d'une application web. Cette application se nomme **Appartex**. Elle a pour but de remplacer une régie en fournissant tous les outils pour une gestion d'appartements et de locataires.

L'application est disponible via ce lien : `https://appartex.firebaseapp.com/`

L'interface client est composé de deux parties : La partie catalogue d'appartements et la partie gestion en elle-même.

La partie gestion se divise elle aussi en deux sections : une pour le propriétaire et une pour le locataire. Les droits d'accès sont gérés grâce à des middlewares. Ils vérifient à chaque changement d'URL si la personne est authentifiée et si elle a les droits d'accéder à la page. Si un utilisateur non authentifié essaie d'accéder à une page avec authentification, il sera systématiquement redirigé vers la page d'accueil représentée à la figure 4.4. Si au contraire, la personne est authentifiée mais ne possède pas les droits nécessaires pour la visualisation de la page, une page d'erreur s'affichera.

Lorsque l'utilisateur tape l'url d'Appartex dans le navigateur, il arrive sur la page présentée par la figure 4.2. Cette page est un catalogue de tous les appartements louables existant dans le système. Elle fait office de vitrine et contient les informations sur les divers appartements de tous les propriétaires inscrits dans le système. Cela facilite la tâche des propriétaires dans leur recherche d'un locataire.

Les images affichées sont celles ajoutées par le propriétaire. Ce cas d'utilisation a été traîté au point 2.3.2.

Les appartements peuvent être filtrés par ville et par taille.

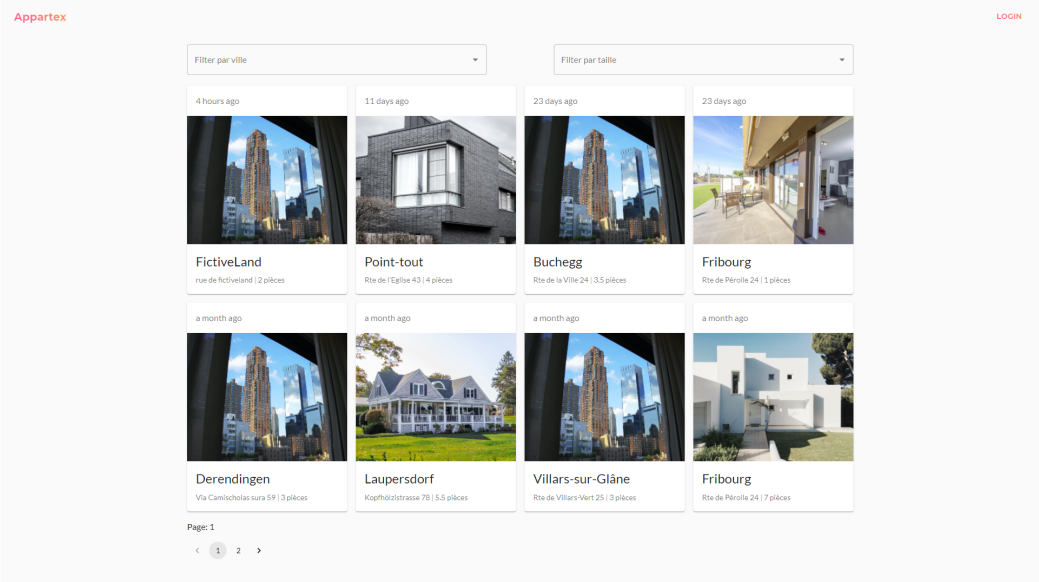


Figure 4.2. – Catalogue des appartements louables

En cliquant sur un appartement, l'utilisateur est redirigé vers une autre page contenant plus de détails ainsi qu'une fonction pour contacter le propriétaire. Il a également la possiblité de voir plus d'images de l'appartement. La fonction de contact n'a pas été implémentée.

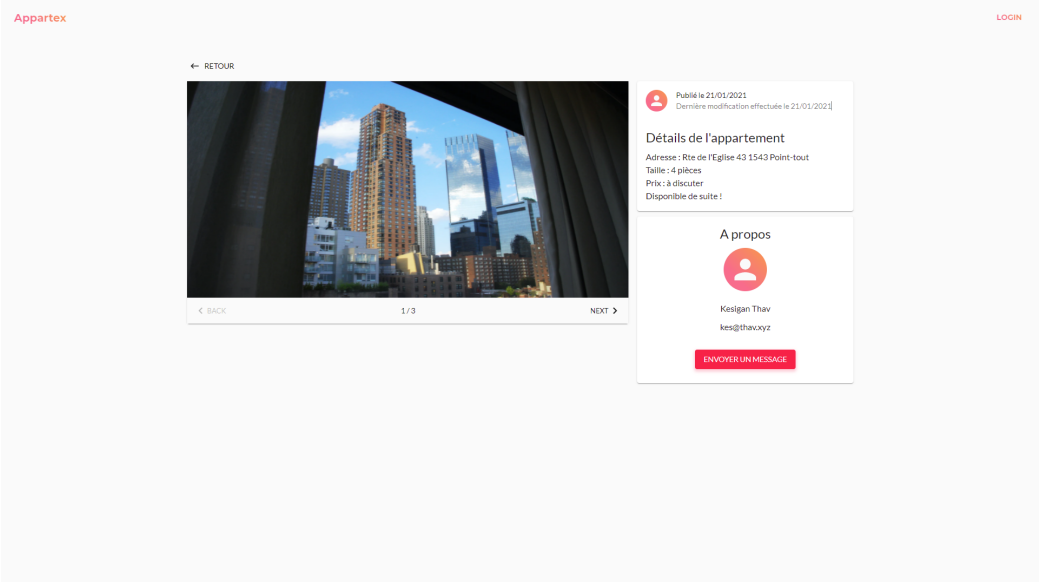


Figure 4.3. – Page contenant les informations sur l'appartement souhaité

Le bouton "LOGIN" présent en haut à droite permet d'accéder à l'application web. La figure 4.4 est la page d'inscription et d'authentification. Elle permet aux propriétaires de s'inscrire ou de s'authentifier. Les locataires peuvent s'authentifier avec les identifiants fournis par leurs propriétaires.

Figure 4.4. – Page d'inscription et d'authentification

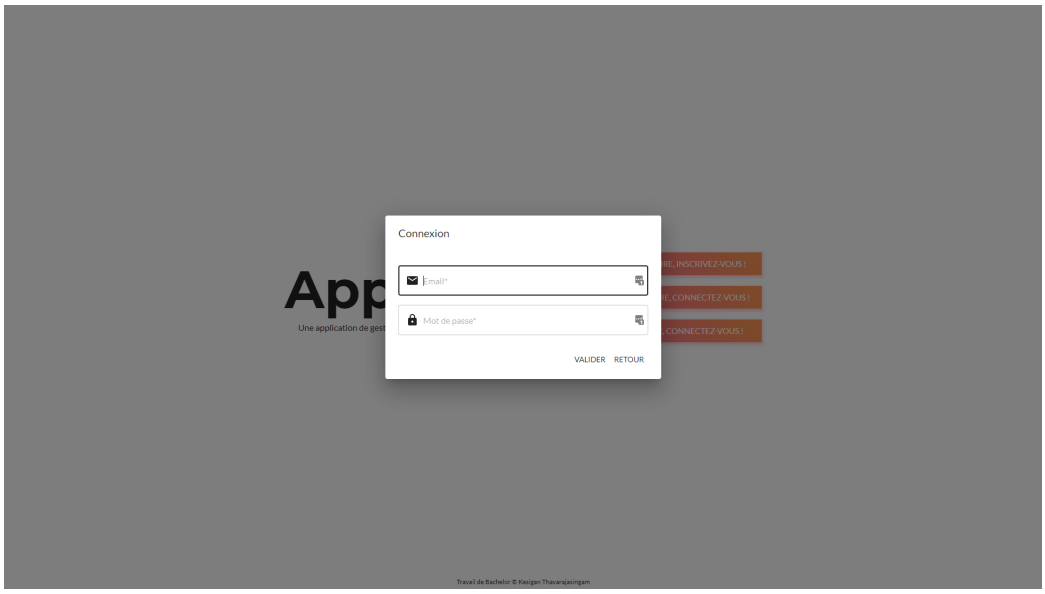


Figure 4.5. – Boîte de dialogue permettant l'authentification

4.3. Scénario

Afin d'éviter la redondance, un seul scénario sera présenté pour le propriétaire dans ce rapport. Tous les autres cas d'utilisation de gestion ont un fonctionnement similaire.

Pour tester les scénarios, les identifiants suivants peuvent être utilisés :

- Propriétaire → email : kes@thav.xyz / mot de passe : 123456
- Locataire → email : dimitri@boscova.xyz / mot de passe : 123456

4.3.1. Scénario propriétaire

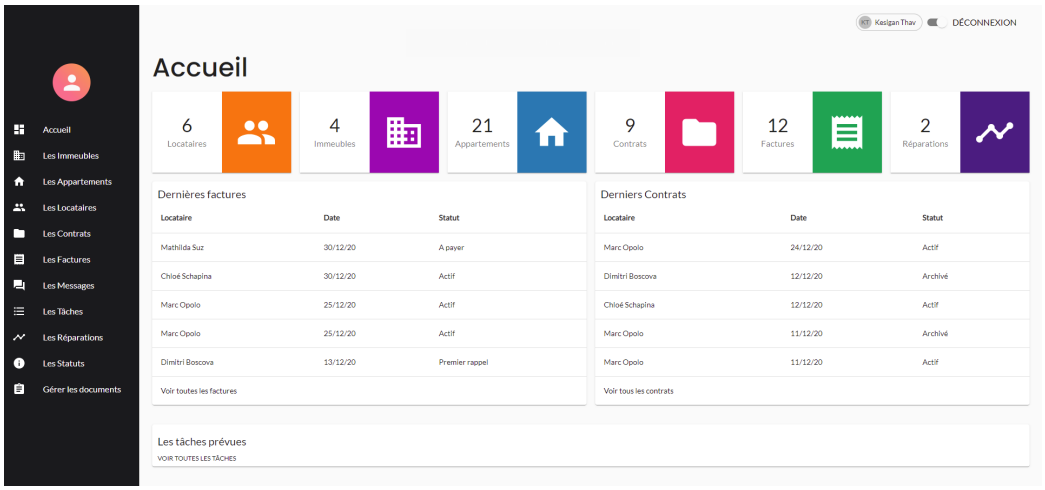


Figure 4.6. – Page d'accueil du propriétaire

Une fois le propriétaire authentifié, il accédera à la page présentée par la figure 4.6. Cette page d'accueil contient un récapitulatif de ses données : le nombre de locataires, d'immeubles, d'appartements, de contrats, de factures et de réparations. Il est possible d'accéder aux pages correspondantes en cliquant sur les vignettes ou en utilisant le tiroir présent sur la gauche. Cette page possède également la liste des dernières factures et contrats ainsi que leurs statuts. Le propriétaire y trouve aussi la liste des tâches prévues.

Page des appartements

La figure 4.7 représente la page des appartements. Elle contient un tableau avec les données concernant l'appartement ainsi que leurs statuts.

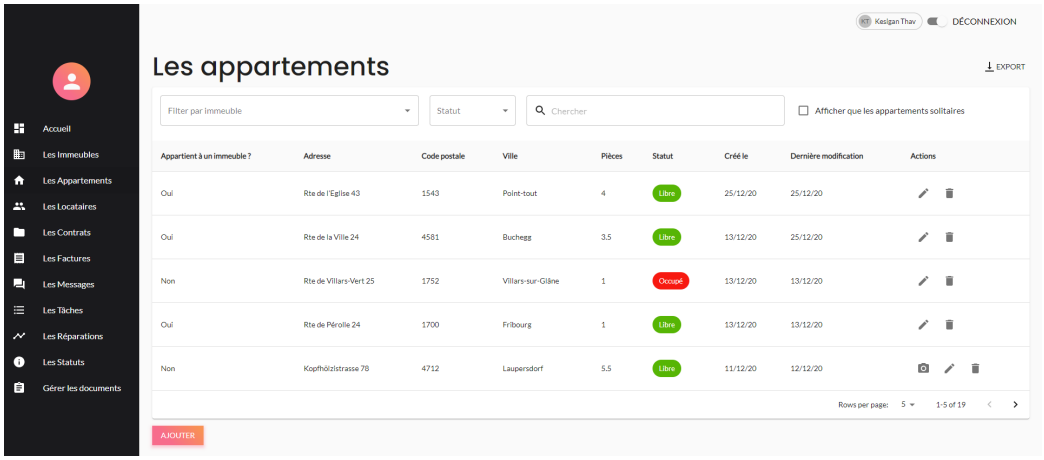


Figure 4.7. – Page des appartements

Au-dessus du tableau, le propriétaire a accès à plusieurs options de filtrage. Les appartements peuvent être filtrés en utilisant :

- la barre de recherche
- l'appartenance ou non appartenance à un immeuble
- le statut

Les options de filtrage varient en fonction des cas d'utilisation. Les tableaux peuvent être exportés en fichier Excel. L'exportation est sensible au filtrage, c'est-à-dire que les données exportées correspondent à celles filtrées.

Si l'appartement possède des images, l'icône "appareil photo" apparaît dans la liste des actions possibles. En cliquant dessus, les images sont visualisables. (Figure 4.8)

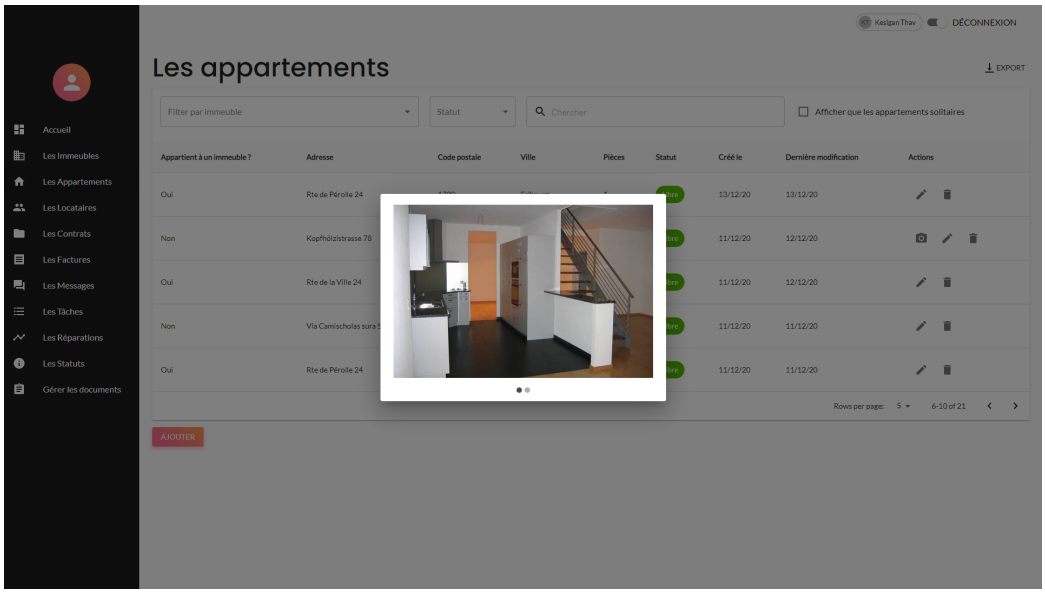


Figure 4.8. – Visualiser les images

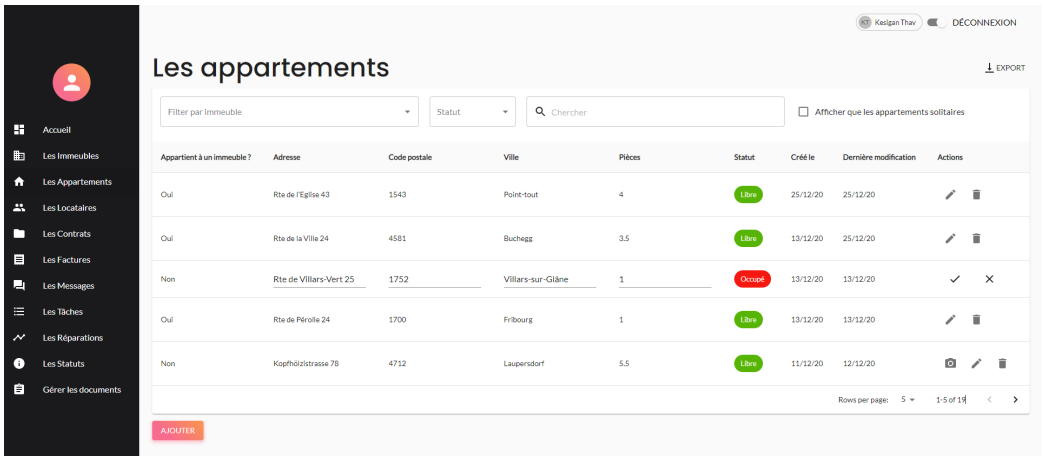


Figure 4.9. – Modification des données

Le tableau est éditable en cliquant sur l'action "éditer", ce qui aura pour conséquence de transformer les cellules de la ligne que l'on souhaite éditer en champ de texte (Figure 4.9). Il suffit alors de faire les modifications nécessaires et de cliquer sur le bouton "confirmer" afin de les appliquer. Si la modification a pu être effectuée, un message de confirmation apparaît. Sinon ce sera un message d'erreur.

Dans le cas où le propriétaire tente de quitter une page en cours d'édition, une alerte s'affichera et le propriétaire devra confirmer son action. (figure 4.10)

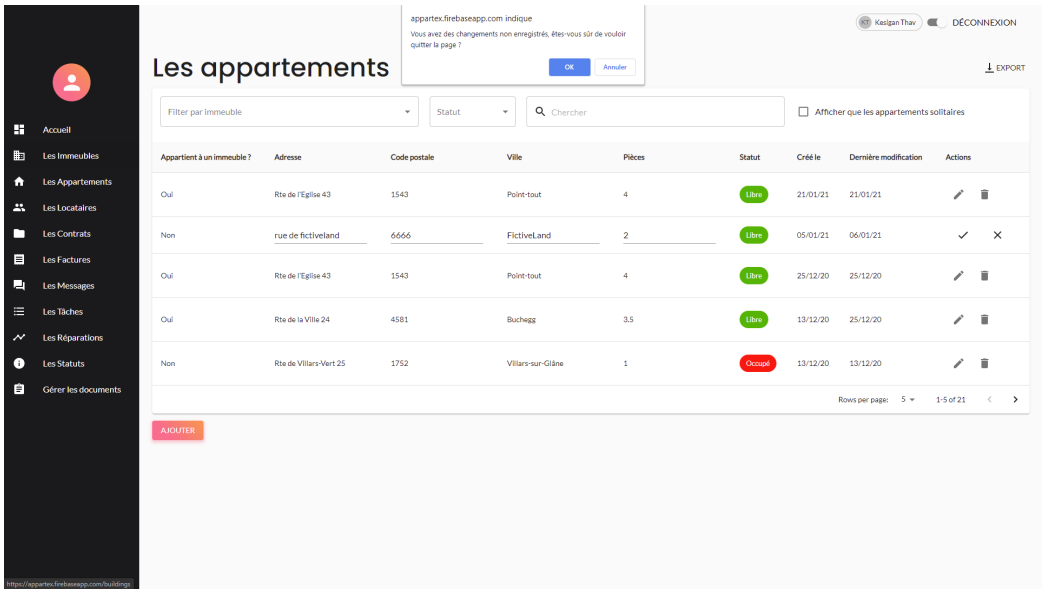


Figure 4.10. – Message d'alerte si la page change en cours de modification

La création des appartements se fait en cliquant sur le bouton "Ajouter". Une boîte de dialogue s'ouvre et les champs obligatoires marqués d'un astérisque doivent être complétés. La vérification des champs est faite grâce aux options HTML. Une fois les champs remplis, le système vérifie que les données sont bien conformes à celles demandées lorsque le bouton "valider" est cliqué. Si ce n'est pas le cas, une alerte apparaît (figure 4.11).

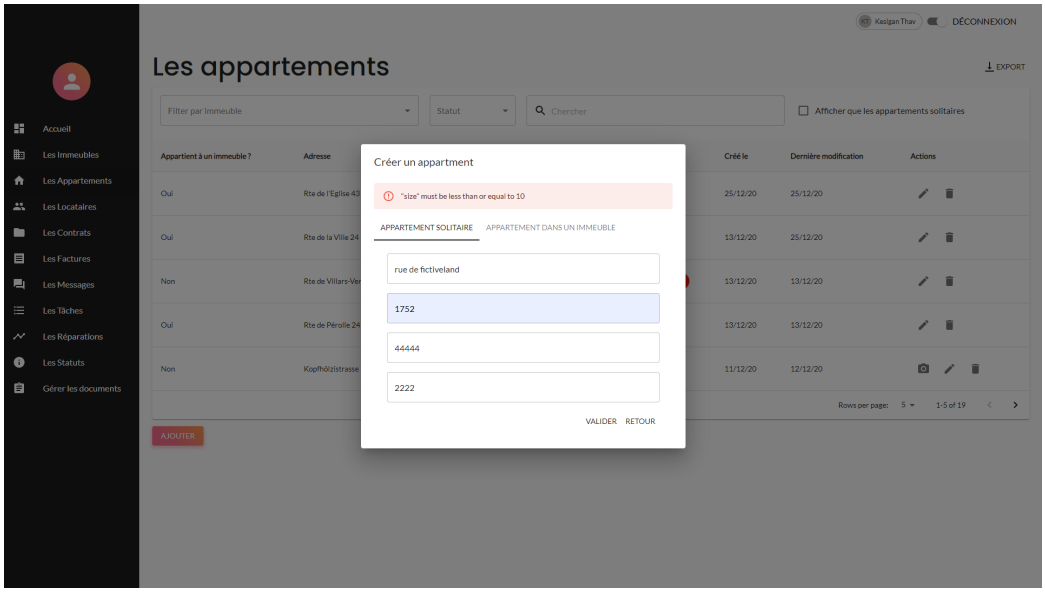


Figure 4.11. – Détection d'erreur

Le propriétaire devra alors corriger les champs déclenchant l'erreur et valider à nouveau. Si tous les champs sont correctes, le système crée la donnée dans sa base de données. (figure 4.12)

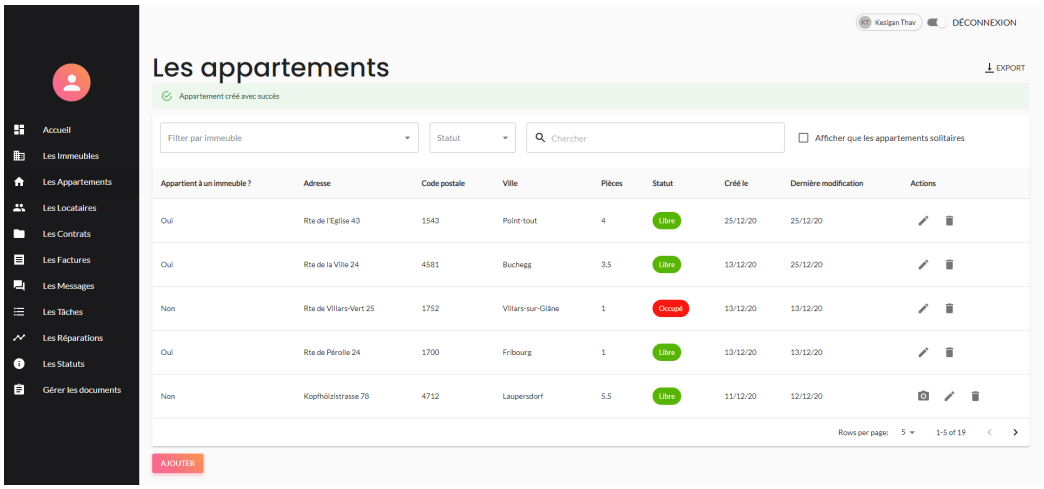


Figure 4.12. – Confirmation de la création

La suppression d'un document se fait un cliquant sur la corbeille se trouvant dans la colonne "Actions". Une boîte de dialogue avertit le propriétaire des conséquences qui en découlent. L'action est irréversible.

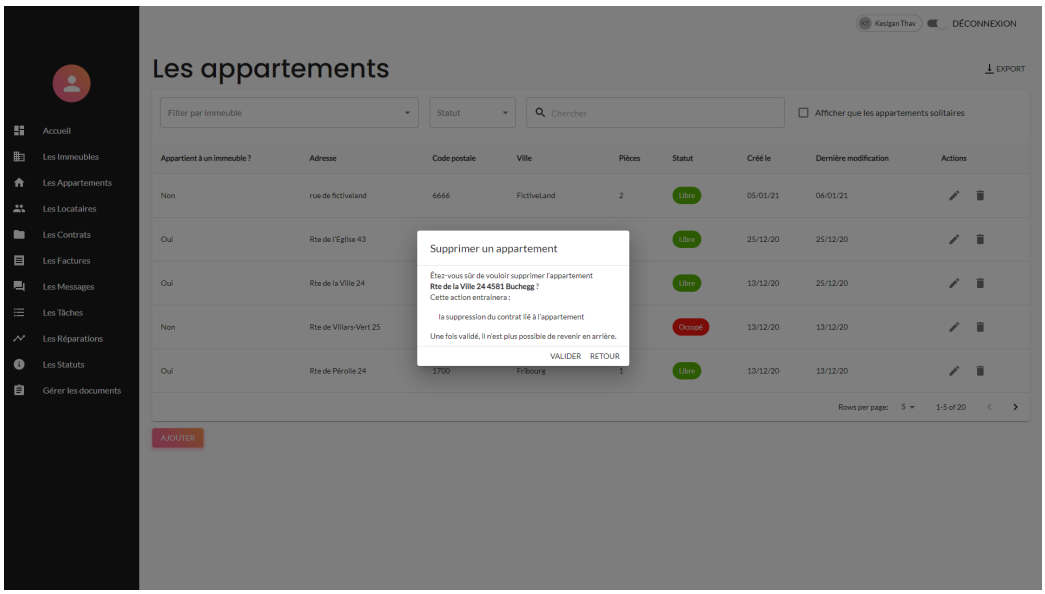


Figure 4.13. – Suppression d'un appartement

Page des messages

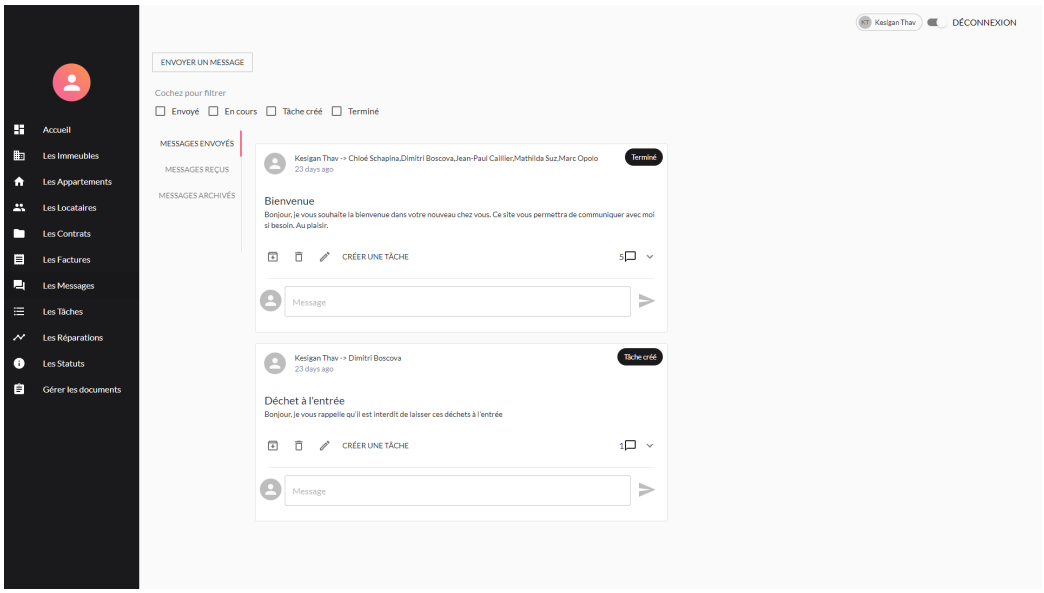


Figure 4.14. – Page des messages

La page des messages permet aux propriétaires et aux locataires de communiquer entre eux. En cliquant sur le bouton "Envoyer un message", une boîte de dialogue s'ouvre et permet de sélectionner le ou les locataires destinataires, d'ajouter un titre et qu'un contenu. Les champs obligatoires sont marqués d'un astérisque et un message d'erreur s'affiche lorsque des données sont manquantes ou erronées.

Pour chaque message, le propriétaire a la possibilité de l'archiver, de le supprimer, d'éditer le statut ou de créer une tâche. Il possède également un compteur de commentaires. En cliquant sur la flèche à droite du message, le propriétaire peut étendre les commentaires.

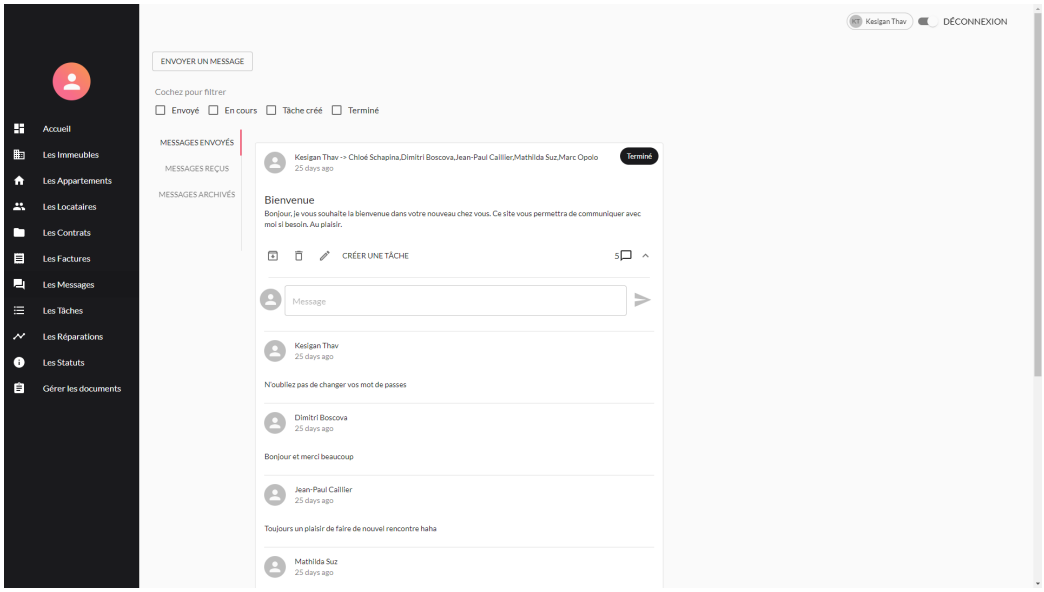


Figure 4.15. – Les commentaires étendus

Pour ajouter un commentaire, il suffit de compléter le champ de texte. Pour l'envoyer, il suffit de cliquer sur la flèche à droite ou d'appuyer sur la touche "Entrée".

Si un message est archivé, il ne peut pas être commenté, ni édité, ni utilisé pour créer une tâche. Le propriétaire devra le désarchiver afin de disposer de ces fonctions.

Page de la gestion des documents

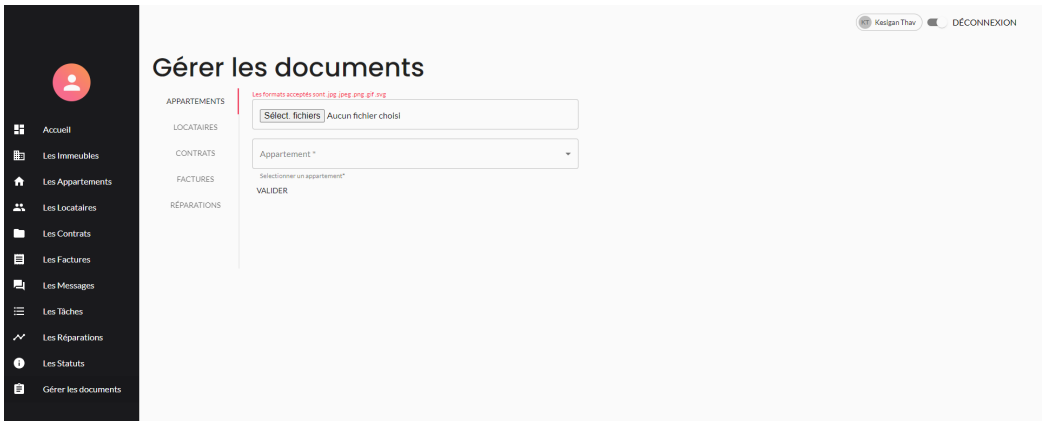


Figure 4.16. – Page pour la gestion des documents

La gestion des documents se fait via la page "Gérer les documents" . Le propriétaire peut ajouter des images à la section "APPARTEMENTS" et des documents dans les sections "LOCATAIRES", "CONTRATS", "FACTURES" et "REPARATIONS". La sélection du document se fait grâce à un menu déroulant. Si l'élément sélectionné possède des documents, une liste de ceux-ci s'affiche. Ils peuvent être téléchargés en cliquant dessus.

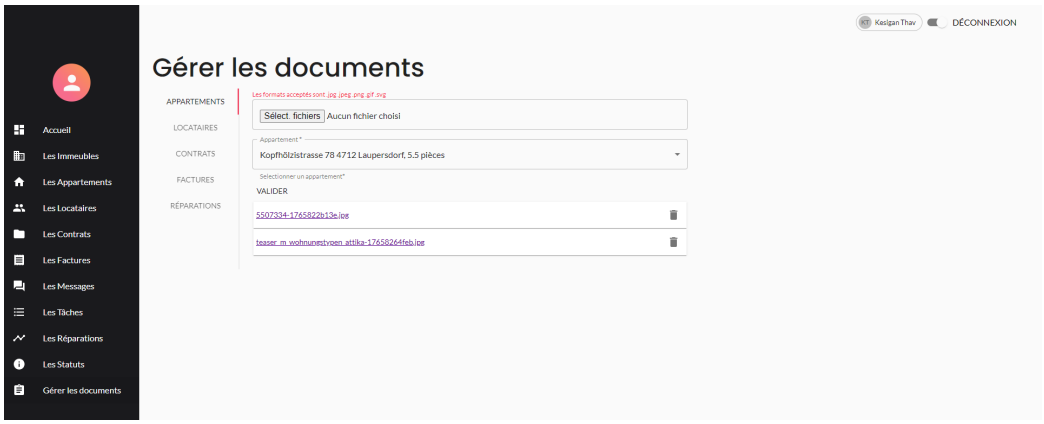


Figure 4.17. – Affichage des documents

La suppression des documents se fait de la même manière que pour les appartements, en cliquant sur la corbeille et en validant la boîte de dialogue.

Page des tâches

Comme vu dans le cas d'utilisation des tâches, une tâche peut être créée au bon vouloir du propriétaire ou suite à une demande d'un locataire. La tâche peut être créée en cliquant sur le bouton "Créer une tâche". Celui-ci est disponible sur les messages ou depuis la page des tâches. Si la tâche est liée à un message, le propriétaire a la possibilité de visualiser le message ainsi que les commentaires de la discussion directement depuis la page des tâches (figure 4.18).

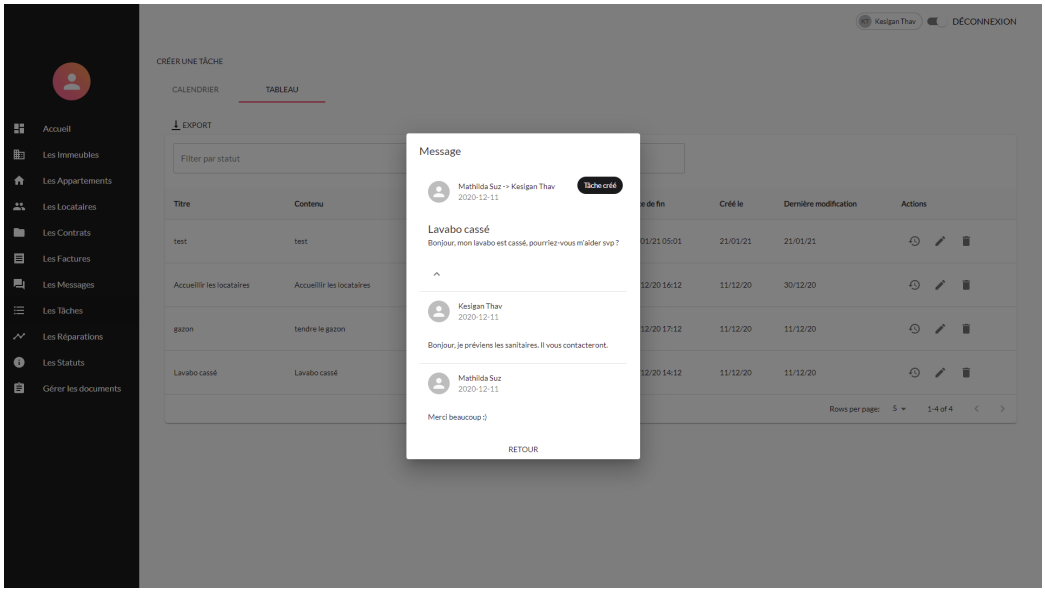


Figure 4.18. – Afficher un message lié à une tâche

Afficher l'historique

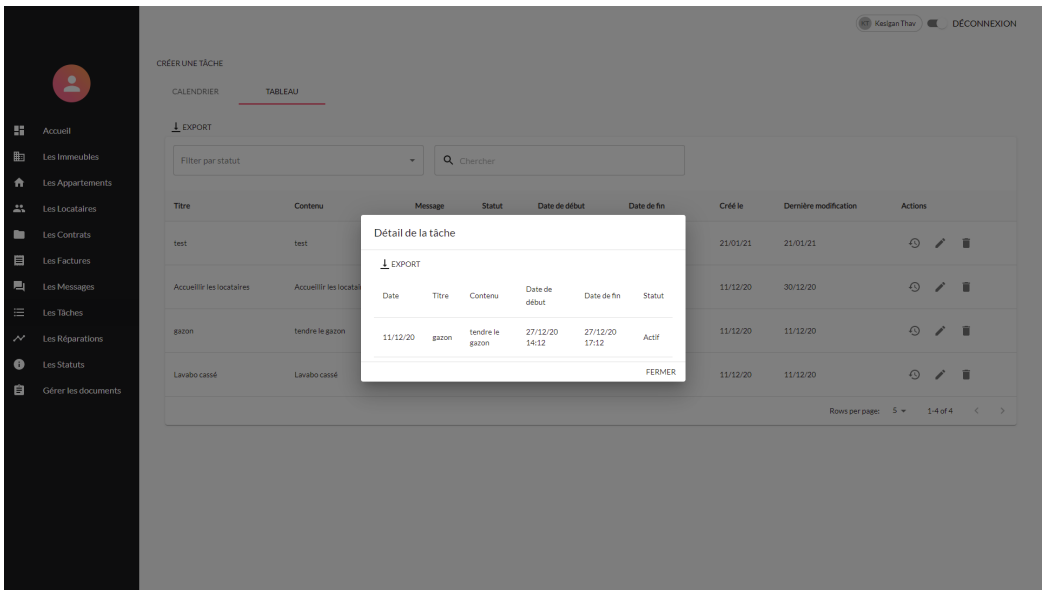


Figure 4.19. – Affichage de l'historique

Comme expliqué dans les cas d'utilisation, il est possible de voir l'historique des factures, des tâches et des réparations. L'historique s'affiche lorsque nous cliquons sur le bouton "détails" se trouvant dans la colonne "Actions".

4.3.2. Scénario locataire

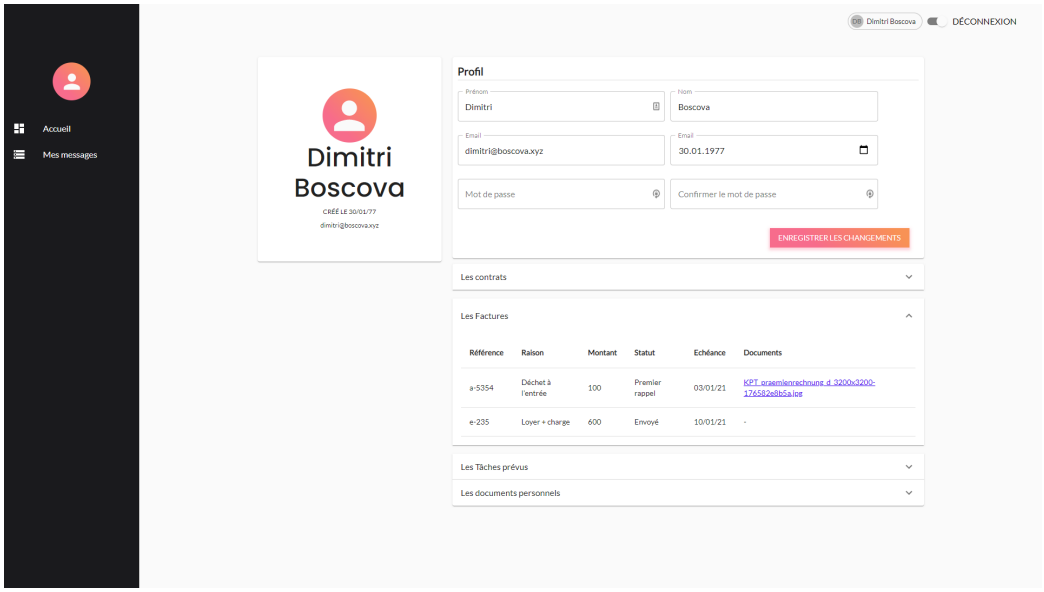


Figure 4.20. – Page d'accueil du locataire

Le scénario du locataire se différencie du scénario du propriétaire. En effet, les actions du locataire sont plus limitées.

Lorsqu'un locataire s'authentifie, il est redirigé vers la page présentée par la figure 4.20. Cette page contient ses informations personnels, la liste de ses factures, de ses contrats, de ses documents personnels, ainsi que les tâches prévus pour lui. Le locataire peut visualiser ses éléments en cliquant sur la flèche à droite de chaque section. Si des documents sont présents, ils peuvent être téléchargés en cliquant dessus.

Le locataire peut mettre à jour ses informations en modifiant les champs du "Profil" et en cliquant sur "Enregistrer les changements".

Le propriétaire a également accès à cette page.

La page des messages du locataire est exactement la même que celui du propriétaire. Le locataire pourra seulement lire et commenter les messages.

4.4. Une application responsive

L'application web est dite "responsive", c'est-à-dire qu'elle s'adapte à la taille de l'écran et au support utilisé. À l'air des smartphones où une grande majorité des pages sont consultées depuis un appareil mobile, il est indispensable que l'application puisse convenir à des personnes qui ne souhaitent pas l'utiliser depuis un ordinateur. Les différentes données sont représentées par des tableaux qui ne sont pas pensés pour un affichage mobile. Il a donc fallu réfléchir à l'organisation et

à la structure des tableaux. Appartex a été rendu responsive grâce à l'utilisation des formules CSS. Elles ont permis de changer radicalement la structure des tableaux et des pages en tenant compte des divers formats d'écran. La figure 4.21 illustre l'application vue depuis un smartphone.

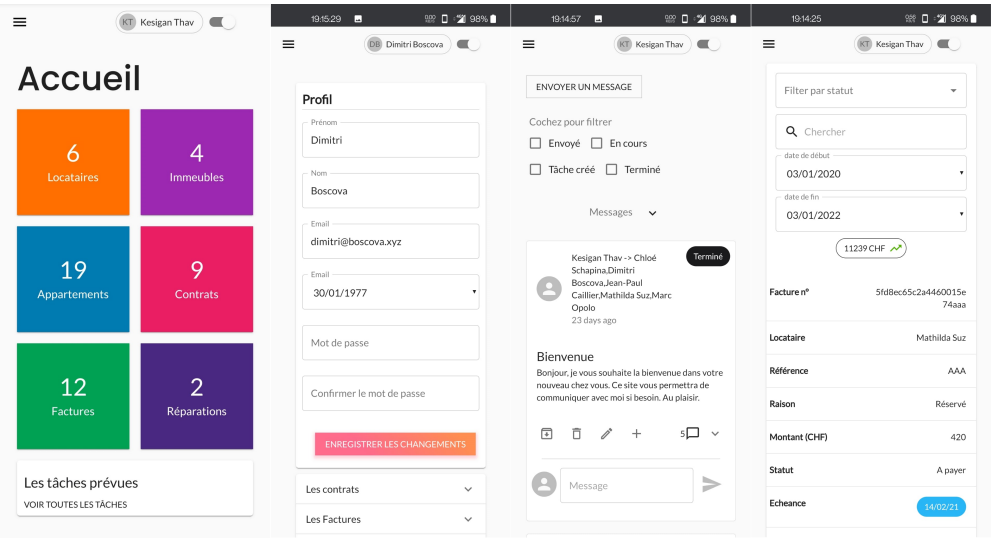


Figure 4.21. – Vue mobile

4.5. Améliorations possibles

L'application web Appartex fournit un panel de fonctions afin que le propriétaire puisse gérer ses appartements et ses locataires. La forme actuelle serait suffisante pour la plupart des propriétaires qui souhaitent simplement maintenir une gestion de leurs appartements ainsi qu'une cetaine autonomie.

Après l'avoir fait testé à plusieurs personnes, quelques améliorations possibles ont été évoquées :

- L'application permet un suivi des états des factures mais ne possède pas la fonction de paiement. L'implémentation d'une fonction de paiement en ligne serait très utile pour le locataire. De plus, un changement automatique des statuts des factures permettraient aux propriétaires d'économiser du temps. Ces deux changements augmenteraient fortement l'intérêt des propriétaires pour cette application web.
- Les documents sont actuellement stockés dans la base de données sous la forme de fichiers binaires. Ces fichiers sont récupérés par le frontend et convertis en document lorsqu'ils sont appelés. L'avantage de cette méthode est qu'il est possible de déployer l'application en ligne gratuitement sans avoir à payer une base de données. L'inconvénient principal et non des moindres est le fait que les fichiers binaires sont lourds. Les requêtes mettent plus de temps à être effectuées dû à l'imposante quantités de données à traiter. La conversion en fichier binaire est pratique si le propriétaire possède deux ou trois documents. Le stockage des documents sous forme binaire ne permet pas la migration vers une autre base de données.
- Certaines fonctions, comme la réparition des factures entre locataires, pourraient être bénéfiques. En effet, les factures partagées tels que les frais de chauffage seraient répartis en fonction de la surface des appartements. Les calculs se feraient automatiquement et les factures seraient créées et distribuées entre les locataires. Cette fonction pourrait être encore plus poussée. Prenons l'exemple de cinq appartements liés à un immeuble. Seuls deux appartements sont occupés. La fonction distribuerait le montant selon la réparatition des deux locataires. Le reste serait payé par le propriétaire.
- Aucun système de sécurité, à part le jeton d'authentification, n'a été implémenté. Étant un travail de bachelor, il faudrait plus de temps pour mettre en place un système de sécurité fonctionnel. Excepté le mot de passe qui est stocké sous forme de hash, le reste des données sont stockées en brut dans la base de données. Il est dès lors très facile pour une personne mal intentionnée d'effectuer une attaque et de voler les données. Par exemple, une attaque de type man-in-the-middle. Ce genre d'attaque n'est pas à la portée de tout le monde mais il faut quand même précisé que nous n'en sommes pas à l'abri.

5 Conclusion

Ce travail de bachelor a contribué à la mise en pratique des différents aspects théoriques étudiés en cours.

Nous avons pu définir les besoins fonctionnels et non-fonctionnels du propriétaire et des locataires. Nous les avons représentés et analysés grâce à des cas d'utilisation. Puis, nous avons utilisé ces besoins pour construire la base de données et le serveur grâce à un diagramme entité-relation. Finalement, l'interface client a été développée et l'application Appartex a pris forme.

La théorie sur la programmation web a permis de consolider les notions apprises et de créer Appartex en partant de zéro. L'approche par les principes REST a fixé une base solide à ce projet. En effet, le couplage faible client-serveur a été un avantage considérable lors du développement de cette application web.

Nous avons conclu que l'application pouvait être améliorée sur certains points. En effet, à l'état actuel, il s'agit d'une première phase vers l'avancement d'une alternative solide à une régie. Cependant, Appartex peut tout de même être utilisé par des petits propriétaires indépendants qui souhaitent gérer leurs appartements et leurs locataires eux-mêmes. Ainsi cela leur éviterait les coûts liés à une régie et leur permettrait de retrouver une certaine indépendance.

Lors de ce projet, certaines difficultés ont été rencontrées. La principale difficulté était l'apprentissage des divers langages de programmation. Les concepts ne sont pas toujours évidents à comprendre. Heureusement, nous vivons aujourd'hui dans un monde où la technologie est omniprésente, ce qui facilite le partage de connaissances. L'apprentissage s'est fait en trois phases : la première a été l'étude des concepts, la deuxième a été la mise en pratique en suivant des tutoriels et finalement la dernière en se lançant dans des projets personnels. Certains cours en ligne et StackOverflow sont des outils très puissants si nous savons nous en servir. Ceux-ci m'ont beaucoup aidé dans la mise en forme de l'application.

En conclusion, ce travail m'a permis de me fixer des objectifs à atteindre et à les surmonter. La prochaine étape serait de développer l'application en prenant en compte les améliorations citées précédemment tels que l'implémentation d'un système de sécurité ainsi que d'un paiement en ligne. Ce projet m'a confirmé la complexité et la rigueur nécessaire lors de la création d'une application. Dans le monde du travail, il n'est pas possible de recommencer à chaque nouveau changement. Les coûts sont très élevés. Une base solide est donc nécessaire. Ce travail m'a appris l'importance d'une bonne structuration avant le développement d'une application.

A
Annexe

A.1. Cas d'utilisation

A.1.1. Système

Titre	S'inscrire	
Description	Un nouveau propriétaire arrive sur le système et veux s'inscrire.	
Acteurs	Nouveau propriétaire non inscrit dans le système	
Pré-conditions	Le propriétaire n'a pas de compte dans le système.	
Post-conditions	En cas de succès	Le propriétaire pourra s'authentifier.
	En cas d'échec	Le propriétaire ne pourra pas s'authentifier.
Scénario principal	1. Un nouveau propriétaire se trouve devant la page d'inscription. 2. Le nouveau propriétaire remplit le formulaire d'inscription. 3. Le système vérifie que toutes les informations obligatoires sont remplies. 4. Le système inscrit le propriétaire dans sa base de données.	
Déroulement alternatif	Des données sont manquantes	1. Le système avertit le propriétaire. 2. Le propriétaire doit corriger les erreurs. 3. Le cas continue à l'étape 3 du scénario principal.
	Le serveur ne répond pas.	1. Le système avertit le propriétaire. 2. Fin du cas d'utilisation.
Autres informations	Les données sont vérifiées par le client et par le serveur.	

Table A.1. – Cas d'utilisation "S'inscrire"

Titre	S'authentifier	
Description	Un utilisateur veut s'authentifier.	
Acteurs	Utilisateur non authentifié	
Pré-conditions	Pour le propriétaire : Il n'est pas authentifié et possède un compte. Pour le Locataire : Le propriétaire lui a créé un compte.	
Post-conditions	En cas de succès	L'utilisateur est authentifié
	En cas d'échec	L'utilisateur n'est pas authentifié.
Scénario principal	1. L'utilisateur se trouve devant la page de connexion. 2. L'utilisateur remplit les informations nécessaires. 3. Le système vérifie que toutes les informations sont remplies et correctes. 4. Le système authentifie l'utilisateur. 5. L'utilisateur est authentifié.	
Déroulement alternatif	Des données sont manquantes ou erronées.	1. Le système avertit l'utilisateur. 2. L'utilisateur doit corriger les erreurs. 3. Le cas continue à l'étape 3 du scénario principal.
	Le serveur ne répond pas.	1. Le système avertit l'utilisateur. 2. Fin du cas d'utilisation.
Autres informations	Les données sont vérifiées par le client et par le serveur.	

Table A.2. – Cas d'utilisation "S'authentifier"

A.1.2. Gestion des immeubles

Titre	Créer un immeuble	
Description	Le propriétaire crée un immeuble.	
Acteurs	Propriétaire authentifié	
Pré-conditions	Le Propriétaire est authentifié sur le système.	
Post-conditions	En cas de succès	L'immeuble est créé.
	En cas d'échec	L'immeuble n'est pas créé.
Scénario principal	1. Le propriétaire se trouve sur la page des immeubles et clique sur "Ajouter". 2. Le propriétaire remplit les champs obligatoires. 3. Le système vérifie que les champs obligatoires soient remplis. 4. L'immeuble est créé.	
Déroulement alternatif	Des données sont manquantes ou erronées.	1. Le système avertit le propriétaire. 2. Le propriétaire doit corriger les erreurs. 3. Le cas continue à l'étape 3 du scénario principal.
	Le propriétaire quitte la page sans terminer le processus.	1. Le système avertit le propriétaire. 2. Le propriétaire valide. Fin du cas d'utilisation. 3. Sinon le cas continue à l'étape 3 du scénario principal.
	Le serveur ne répond pas.	1. Le système avertit le propriétaire. 2. Fin du cas d'utilisation.
Autres informations	Les données sont vérifiées par le client et par le serveur.	

Table A.3. – Cas d'utilisation "Créer un immeuble"

Titre	Modifier un immeuble	
Description	Le propriétaire modifie un immeuble.	
Acteurs	Propriétaire authentifié	
Pré-conditions	Le Propriétaire est authentifié sur le système. L'immeuble existe dans le système.	
Post-conditions	En cas de succès	L'immeuble est modifié.
	En cas d'échec	L'immeuble n'est pas modifié.
Scénario principal	1. Le propriétaire modifie les champs souhaités. 2. Le système vérifie que les champs obligatoires soient remplis. 3. Le système demande si le propriétaire souhaite enregistrer les changements. 4. L'immeuble est modifié.	
Déroulement alternatif	Des données sont manquantes ou erronées	1. Le système avertit le propriétaire. 2. Le propriétaire doit corriger les erreurs. 3. Le cas continue à l'étape 2 du scénario principal.
	Le propriétaire quitte la page sans terminer le processus.	1. Le système avertit le propriétaire. 2. Le propriétaire valide. Fin du cas d'utilisation. 3. Sinon le cas continue à l'étape 2 du scénario principal.
	Le serveur ne répond pas.	1. Le système avertit le propriétaire. 2. Fin du cas d'utilisation.
Autres informations	Les données sont vérifiées par le client et par le serveur.	

Table A.4. – Cas d'utilisation "Modifier un immeuble"

Titre	Supprimer un immeuble	
Description	Le propriétaire supprime un immeuble.	
Acteurs	Propriétaire authentifié	
Pré-conditions	Le Propriétaire est authentifié sur le système. L'immeuble existe dans le système.	
Post-conditions	En cas de succès	L'immeuble est supprimé.
	En cas d'échec	L'immeuble n'est pas supprimé.
Scénario principal	1. Le propriétaire clique sur supprimer l'immeuble. 2. Le système avertit le propriétaire et lui demande une confirmation. 3. Le propriétaire confirme. 4. Le système supprime l'immeuble de sa base de données. 5. L'immeuble est supprimé. 6. Les appartements ayant la clé étrangère de l'immeuble sont supprimés. 7. Les contrats liés aux appartements supprimés sont supprimés.	
Déroulement alternatif	Le serveur ne répond pas.	1. Le système avertit le propriétaire. 2. Fin du cas d'utilisation.
Autres informations	-	

Table A.5. – Cas d'utilisation "Supprimer un immeuble"

Titre	Voir la liste des locataires d'un immeuble.	
Description	Le propriétaire veut voir la liste des locataires d'un immeuble.	
Acteurs	Propriétaire authentifié	
Pré-conditions	Le propriétaire est authentifié sur le système.	
Post-conditions	En cas de succès	Les locataires sont affichés.
	En cas d'échec	Les locataires ne sont pas affichés.
Scénario principal	1. Le propriétaire se trouve sur la pages des immeubles. 2. Le propriétaire clique sur détails. 3. Le système affiche la liste des locataires.	
Déroulement alternatif	Le serveur ne répond pas.	1. Le système avertit le propriétaire. 2. Fin du cas d'utilisation.
Autres informations	-	

Table A.6. – Cas d'utilisation "Voir la liste des locataires d'un immeuble"

Titre	Voir le nombre d'appartements que l'immeuble possède	
Description	Le propriétaire veut voir le nombre d'appartements que l'immeuble possède.	
Acteurs	Propriétaire authentifié	
Pré-conditions	Le propriétaire est authentifié sur le système.	
Post-conditions	En cas de succès	Le nombre d'appartements que l'immeuble possède est affiché.
	En cas d'échec	Le nombre d'appartements que l'immeuble possède n'est pas affiché.
Scénario principal	1. Le propriétaire se trouve sur la pages des immeubles. 2. Le système affiche le nombre d'appartements possédés par chaque immeuble.	
Déroulement alternatif	Le serveur ne répond pas.	1. Le système avertit le propriétaire. 2. Fin du cas d'utilisation.
Autres informations	-	

Table A.7. – Cas d'utilisation "Voir le nombre d'appartements qu'un immeuble possède"

Titre	Voir le nombre d'appartements occupés	
Description	Le propriétaire veut voir le nombre d'appartements occupés.	
Acteurs	Propriétaire authentifié	
Pré-conditions	Le propriétaire est authentifié sur le système.	
Post-conditions	En cas de succès	Le nombre d'appartements occupés est affiché.
	En cas d'échec	Le nombre d'appartements occupés n'est pas affiché.
Scénario principal	1. Le propriétaire se trouve sur la pages des immeubles. 2. Le système affiche, pour chaque immeuble, le nombre d'appartements occupés.	
Déroulement alternatif	Le serveur ne répond pas.	1. Le système avertit le propriétaire. 2. Fin du cas d'utilisation.
Autres informations	-	

Table A.8. – Cas d'utilisation "Voir le nombre d'appartements occupés"

A.1.3. Gestion des appartements

Titre	Créer un appartement	
Description	Le propriétaire crée un appartement.	
Acteurs	Propriétaire authentifié	
Pré-conditions	Le Propriétaire est authentifié sur le système.	
Post-conditions	En cas de succès	L'appartement est créé.
	En cas d'échec	L'appartement n'est pas créé.
Scénario principal	1. Le propriétaire se trouve sur la page des appartements et clique sur "Ajouter". 2. Le propriétaire choisit s'il veut créer un appartement solitaire ou un appartement lié à un immeuble. 3. Le propriétaire remplit les champs obligatoires. 4. Le système vérifie que les champs obligatoires soient remplis. 5. L'appartement est créé. 6. si l'appartement appartient à un immeuble, le compteur d'appartement de l'immeuble augmente de 1.	
Déroulement alternatif	Des données sont manquantes ou erronées.	1. Le système avertit le propriétaire. 2. Le propriétaire doit corriger les erreurs. 3. le cas continue à l'étape 4 du scénario principal.
	Le propriétaire quitte la page sans terminer le processus.	1. Le système avertit le propriétaire. 2. Le propriétaire valide. Fin du cas d'utilisation. 3. Le cas continue à l'étape 4 du scénario principal.
	Le serveur ne répond pas	1. Le système avertit le propriétaire. 2. Fin du cas d'utilisation.
Autres informations	Les données sont vérifiées par le client et par le serveur.	

Table A.9. – Cas d'utilisation "Créer un appartement"

Titre	Modifier un appartement	
Description	Le propriétaire modifie un appartement.	
Acteurs	Propriétaire authentifié	
Pré-conditions	Le Propriétaire est authentifié sur le système L'appartement existe dans le système.	
Post-conditions	En cas de succès	L'appartement est modifié.
	En cas d'échec	L'appartement n'est pas modifié.
Scénario principal	1. Le propriétaire modifie les champs souhaités. 2. Le système vérifie que les champs obligatoires soient remplis. 3. Le système demande si le propriétaire souhaite enregistrer les changements. 4. L'appartement est modifié.	
Déroulement alternatif	Des données sont manquantes ou erronées.	1. Le système avertit le propriétaire. 2. Le propriétaire doit corriger les erreurs. 3. Le cas continue à l'étape 2 du scénario principal.
	Le propriétaire quitte la page sans terminer le processus.	1. Le système avertit le propriétaire. 2. Le propriétaire valide. Fin du cas d'utilisation. 3. Sinon le cas continue à l'étape 2 du scénario principal.
	Le serveur ne répond pas	1. Le système avertit le propriétaire. 2. Fin du cas d'utilisation.
Autres informations	Les données sont vérifiées par le client et par le serveur.	

Table A.10. – Cas d'utilisation "Modifier un appartement"

Titre	Supprimer un appartement	
Description	Le propriétaire supprime un appartement.	
Acteurs	Propriétaire authentifié	
Pré-conditions	Le Propriétaire est authentifié sur le système. L'appartement existe dans le système.	
Post-conditions	En cas de succès	L'appartement est supprimé.
	En cas d'échec	L'appartement n'est pas supprimé.
Scénario principal	1. Le propriétaire clique sur supprimer l'appartement. 2. Le système avertit le propriétaire et lui demande une confirmation. 3. Le propriétaire confirme. 4. Le système supprime l'appartement de sa base de données. 5. le contrat lié à l'appartement supprimé est supprimé. 6. si l'appartement appartient à un immeuble, le compteur d'appartement de l'immeuble diminue de 1.	
Déroulement alternatif	Le serveur ne répond pas.	1. Le système avertit le propriétaire. 2. Fin du cas d'utilisation.
Autres informations	-	

Table A.11. – Cas d'utilisation "Supprimer un appartement"

Titre	Voir les images d'un appartement.	
Description	Le propriétaire veut voir les images d'un appartement.	
Acteurs	Propriétaire authentifié	
Pré-conditions	Le Propriétaire est authentifié L'appartement existe dans le système.	
Post-conditions	En cas de succès	Les images de l'appartement sont affichées.
	En cas d'échec	Les images de l'appartement ne sont pas affichées.
Scénario principal	1. Le propriétaire se trouve sur la page des appartements. 2. Le propriétaire clique sur détails. 3. Le système affiche les images de l'appartement.	
Déroulement alternatif	Le serveur ne répond pas.	1. Le système avertit le propriétaire. 2. Fin du cas d'utilisation.
Autres informations	-	

Table A.12. – Cas d'utilisation "Voir les images d'un appartement"

A.1.4. Gestion des locataires

Titre	Créer un locataire	
Description	Le propriétaire crée un locataire.	
Acteurs	Propriétaire authentifié	
Pré-conditions	Le Propriétaire est authentifié sur le système.	
Post-conditions	En cas de succès	Le locataire est créé.
	En cas d'échec	Le locataire n'est pas créé.
Scénario principal	1. Le propriétaire se trouve sur la page locataire et clique sur "Ajouter".. 2. Le propriétaire remplit les champs obligatoires. 3. Le système vérifie que les champs obligatoires soient remplis. 4. Le locataire est créé.	
Déroulement alternatif	Des données sont manquantes ou erronées.	1. Le système avertit le propriétaire. 2. Le propriétaire doit corriger les erreurs. 3. Le cas continue à l'étape 3 du scénario principal.
	Le propriétaire quitte la page sans terminer le processus.	1. Le système avertit le propriétaire. 2. Le propriétaire valide. Fin du cas d'utilisation. 3. Sinon le cas continue à l'étape 3 du scénario principal.
	Le serveur ne répond pas.	1. Le système avertit le propriétaire. 2. Fin du cas d'utilisation.
Autres informations	Les données sont vérifiées par le client et par le serveur.	

Table A.13. – Cas d'utilisation "Créer un locataire"

Titre	Modifier un locataire	
Description	Le propriétaire modifie un locataire.	
Acteurs	Propriétaire authentifié	
Pré-conditions	Le Propriétaire est authentifié sur le système Le locataire existe dans le système.	
Post-conditions	En cas de succès	Le locataire est modifié.
	En cas d'échec	Le locataire n'est pas modifié.
Scénario principal	1. Le propriétaire modifie les champs souhaités. 2. Le système vérifie que les champs obligatoires soient remplis. 3. Le système demande si le propriétaire souhaite enregistrer les changements. 4. Le propriétaire valide. 5. Le locataire est modifié.	
Déroulement alternatif	Des données sont manquantes ou erronées.	1. Le système avertit le propriétaire. 2. Le propriétaire doit corriger les erreurs. 3. Le cas continue à l'étape 2 du scénario principal.
	Le propriétaire quitte la page sans terminer le processus.	1. Le système avertit le propriétaire. 2. Le propriétaire valide. Fin du cas d'utilisation. 3. Sinon le cas continue à l'étape 2 du scénario principal.
	Le serveur ne répond pas.	1. Le système avertit le propriétaire. 2. Fin du cas d'utilisation.
Autres informations	Les données sont vérifiées par le client et par le serveur.	

Table A.14. – Cas d'utilisation "Modifier un locataire"

Titre	Supprimer un locataire	
Description	Le propriétaire supprime un locataire.	
Acteurs	Propriétaire authentifié	
Pré-conditions	Le Propriétaire est authentifié sur le système. Le locataire existe dans le système.	
Post-conditions	En cas de succès	Le locataire est supprimé.
	En cas d'échec	Le locataire n'est pas supprimé.
Scénario principal	1. Le propriétaire clique sur supprimer le locataire. 2. Le système avertit le propriétaire et lui demande une confirmation. 3. Le propriétaire confirme. 4. Le système supprime le locataire de sa base de données. 5. les contrats liés au locataire sont supprimés.	
Déroulement alternatif	Le serveur ne répond pas.	1. Le système avertit le propriétaire. 2. Fin du cas d'utilisation.
Autres informations	-	

Table A.15. – Cas d'utilisation "Supprimer un locataire"

Titre	Désactiver un locataire	
Description	Le propriétaire désactive un locataire.	
Acteurs	Propriétaire authentifié	
Pré-conditions	Le Propriétaire est authentifié sur le système. Le locataire existe dans le système.	
Post-conditions	En cas de succès	Le locataire est désactivé.
	En cas d'échec	Le locataire n'est pas désactivé.
Scénario principal	1. Le propriétaire modifie le statut du locataire. 2. Le système avertit le propriétaire et lui demande une confirmation. 3. Le propriétaire confirme. 4. Le système désactive le locataire de sa base de données. 5. Le locataire ne pourra plus être utilisé.	
Déroulement alternatif	Le serveur ne répond pas	1. Le système avertit le propriétaire. 2. Fin du cas d'utilisation.
Autres informations	-	

Table A.16. – Cas d'utilisation "Désactiver un locataire"

Titre	Voir les données personnelles d'un locataire.	
Description	Le propriétaire veut voir les données personnelles d'un locataire.	
Acteurs	Propriétaire authentifié	
Pré-conditions	Le propriétaire est authentifié sur le système. Le locataire existe dans le système.	
Post-conditions	En cas de succès	Les données personnelles du locataire sont affichées.
	En cas d'échec	Les données personnelles du locataire ne sont pas affichées.
Scénario principal	1. Le propriétaire se trouve sur la page locataire. 2. Le propriétaire clique sur détails. 3. Le système affiche la page du locataire avec ses factures, ses contrats, ses tâches prévues et ses documents personnels.	
Déroulement alternatif	Le serveur ne répond pas	1. Le système avertit le propriétaire. 2. Fin du cas d'utilisation.
Autres informations	-	

Table A.17. – Cas d'utilisation "Voir les données personnelles d'un locataire"

A.1.5. Gestion des contrats

Titre	Créer un contrat	
Description	Le propriétaire crée un contrat.	
Acteurs	Propriétaire authentifié	
Pré-conditions	Le Propriétaire est authentifié dans le système. Un ou des locataires existent dans le système. Un ou des appartements existent dans le système.	
Post-conditions	En cas de succès	Le contrat est créé.
	En cas d'échec	Le contrat n'est pas créé.
Scénario principal	1. Le propriétaire se trouve sur la page des contrats et clique sur "Ajouter". 2. Le propriétaire remplit les informations nécessaires et confirme. 3. Le système vérifie si les données obligatoires sont remplies. 4. Le système crée le contrat. 5. Le statut de l'appartement choisi change en "occupé". 6. Si un appartement lié à un immeuble est choisi, le compteur d'appartements occupés de l'immeuble augmente de 1.	
Déroulement alternatif	Des données sont manquantes ou erronées.	1. Le système avertit le propriétaire. 2. Le propriétaire doit corriger les erreurs. 3. Le cas continue à l'étape 3 du scénario principal.
	Le propriétaire quitte la page sans terminer le processus.	1. Le système avertit le propriétaire. 2. Le propriétaire valide. Fin du cas d'utilisation. 3. Sinon le cas continue à l'étape 3 du scénario principal.
	Le serveur ne répond pas.	1. Le système avertit le propriétaire. 2. Fin du cas d'utilisation.
Autres informations	Les données sont vérifiées par le client et par le serveur.	

Table A.18. – Cas d'utilisation "Créer un contrat"

Titre	Modifier un contrat	
Description	Le propriétaire modifie un contrat.	
Acteurs	Propriétaire authentifié	
Pré-conditions	Le Propriétaire est authentifié sur le système. Le contrat existe dans le système.	
Post-conditions	En cas de succès	Le contrat est modifié.
	En cas d'échec	Le contrat n'est pas modifié.
Scénario principal	1. Le propriétaire modifie les champs souhaités. 2. Le système vérifie que les champs obligatoires soient remplis. 3. Le système demande si le propriétaire souhaite enregistrer les changements. 4. Le contrat est modifié.	
Déroulement alternatif	Des données sont manquantes ou erronées	1. Le système avertit le propriétaire. 2. Le propriétaire doit corriger les erreurs. 3. Le cas continue à l'étape 2 du scénario principal.
	Le propriétaire quitte la page sans terminer le processus.	1. Le système avertit le propriétaire. 2. Le propriétaire valide. Fin du cas d'utilisation. 3. Sinon le cas continue à l'étape 2 du scénario principal.
	Le serveur ne répond pas.	1. Le système avertit le propriétaire. 2. Fin du cas d'utilisation.
Autres informations	Les données sont vérifiées par le client et par le serveur.	

Table A.19. – Cas d'utilisation "Modifier un contrat"

Titre	Supprimer un contrat	
Description	Le propriétaire supprime un contrat.	
Acteurs	Propriétaire authentifié	
Pré-conditions	Le Propriétaire est authentifié sur le système. Le contrat existe dans le système.	
Post-conditions	En cas de succès	Le contrat est supprimé.
	En cas d'échec	Le contrat n'est pas supprimé.
Scénario principal	1. Le propriétaire clique sur supprimer le contrat. 2. Le système avertit le propriétaire et lui demande une confirmation. 3. Le propriétaire confirme. 4. Le système supprime le contrat de sa base de données. 5. Si un appartement solitaire est choisi, son statut change en Libre dans le système. 6. Si un appartement lié à un immeuble est choisi, son statut change en Libre dans le système et le compteur d'appartement occupé de l'immeuble diminue de 1.	
Déroulement alternatif	Le serveur ne répond pas	1. Le système avertit le propriétaire. 2. Fin du cas d'utilisation.
Autres informations	-	

Table A.20. – Cas d'utilisation "Supprimer un contrat"

Titre	Archiver un contrat	
Description	Le propriétaire archive un contrat.	
Acteurs	Propriétaire authentifié	
Pré-conditions	Le Propriétaire est authentifié sur le système. Le contrat existe dans le système.	
Post-conditions	En cas de succès	Le contrat est archivé.
	En cas d'échec	Le contrat n'est pas archivé.
Scénario principal	1. Le propriétaire modifie le statut du contrat. 2. Le système avertit le propriétaire et lui demande une confirmation. 3. Le propriétaire confirme. 4. Le système modifie le statut du contrat dans sa base de données.	
Déroulement alternatif	Le serveur ne répond pas.	1. Le système avertit le propriétaire. 2. Fin du cas d'utilisation.
Autres informations	-	

Table A.21. – Cas d'utilisation "Archiver un contrat"

Titre	Voir les documents liés à un contrat	
Description	Le propriétaire veut voir les documents liés à un contrat.	
Acteurs	Propriétaire authentifié	
Pré-conditions	Le propriétaire est authentifié sur le système. Le contrat existe dans le système.	
Post-conditions	En cas de succès	Les documents liés au contrat sont affichés.
	En cas d'échec	Les documents liés au contrat ne sont pas affichés.
Scénario principal	1. Le propriétaire se trouve sur la page contrats. 2. Le propriétaire clique sur détails. 3. Les documents liés au propriétaire sont affichés.	
Déroulement alternatif	Le serveur ne répond pas.	1. Le système avertit le propriétaire. 2. Fin du cas d'utilisation.
Autres informations	-	

Table A.22. – Cas d'utilisation "Voir les documents liés à un contrat"

A.1.6. Gestion des factures

Titre	Créer une facture	
Description	Le propriétaire crée une facture	
Acteurs	Propriétaire authentifié	
Pré-conditions	Le Propriétaire est authentifié sur le système. Le locataire existe dans le système. Divers statuts existent dans le système.	
Post-conditions	En cas de succès	La facture est créée.
	En cas d'échec	la facture n'est pas créée.
Scénario principal	1. Le propriétaire se trouve sur la page des factures et clique sur "Ajouter". 2. Le propriétaire choisit un locataire dans la liste déroulante. 3. Le propriétaire remplit les autres informations. 4. Le propriétaire confirme. 5. Le système vérifie si les champs obligatoires sont remplis. 6. Le système crée la facture.	
Déroulement alternatif	Des données sont manquantes ou erronées	1. Le système avertit le propriétaire. 2. Le propriétaire doit corriger les erreurs. 3. Le cas continue à l'étape 5 du scénario principal.
	Le propriétaire quitte la page sans terminer le processus.	1. Le système avertit le propriétaire. 2. Le propriétaire valide. Fin du cas d'utilisation. 3. Sinon le cas continue à l'étape 5 du scénario principal.
	Le serveur ne répond pas.	1. Le système avertit le propriétaire. 2. Fin du cas d'utilisation.
Autres informations	Les données sont vérifiées par le client et par le serveur.	

Table A.23. – Cas d'utilisation "Créer une facture"

Titre	Modifier une facture	
Description	Le propriétaire modifie une facture.	
Acteurs	Propriétaire authentifié	
Pré-conditions	Le Propriétaire est authentifié sur le système. La facture existe dans le système.	
Post-conditions	En cas de succès	La facture est modifiée.
	En cas d'échec	La facture n'est pas modifiée.
Scénario principal	1. Le propriétaire modifie les champs souhaités. 2. Le système vérifie que les champs obligatoires soient remplis. 3. Le système demande si le propriétaire souhaite enregistrer les changements. 4. La facture est modifiée.	
Déroulement alternatif	Des données sont manquantes ou erronées.	1. Le système avertit le propriétaire. 2. Le propriétaire doit corriger les erreurs. 3. Le cas continue à l'étape 2 du scénario principal.
	Le propriétaire quitte la page sans terminer le processus.	1. Le système avertit le propriétaire. 2. Le propriétaire valide. Fin du cas d'utilisation. 3. Sinon le cas continue à l'étape 2 du scénario principal.
	Le serveur ne répond pas.	1. Le système avertit le propriétaire. 2. Fin du cas d'utilisation.
Autres informations	Les données sont vérifiées par le client et par le serveur.	

Table A.24. – Cas d'utilisation "Modifier une facture"

Titre	Supprimer une facture	
Description	Le propriétaire supprime une facture.	
Acteurs	Propriétaire authentifié	
Pré-conditions	Le Propriétaire est authentifié sur le système. La facture existe dans le système.	
Post-conditions	En cas de succès	La facture est supprimée.
	En cas d'échec	La facture n'est pas supprimée.
Scénario principal	1. Le propriétaire clique sur supprimer la facture. 2. Le système avertit le propriétaire et lui demande une confirmation. 3. Le propriétaire confirme. 4. Le système supprime la facture de sa base de données.	
Déroulement alternatif	Le serveur ne répond pas.	1. Le système avertit le propriétaire. 2. Fin du cas d'utilisation.
Autres informations	-	

Table A.25. – Cas d'utilisation "Supprimer une facture"

Titre	Voir l'historique d'une facture	
Description	Le propriétaire veut voir l'historique d'une facture.	
Acteurs	Propriétaire authentifié	
Pré-conditions	Le Propriétaire est authentifié sur le système. La facture existe dans le système.	
Post-conditions	En cas de succès	L'historique de la facture est affiché.
	En cas d'échec	L'historique de la facture n'est pas affiché.
Scénario principal	1. Le propriétaire se trouve sur la page des factures. 2. Le propriétaire clique sur détails. 3. Le système affiche l'historique de la facture.	
Déroulement alternatif	Le serveur ne répond pas.	1. Le système avertit le propriétaire. 2. Fin du cas d'utilisation.
Autres informations	-	

Table A.26. – Cas d'utilisation "Voir l'historique d'une facture"

Titre	Voir les documents liés à une facture	
Description	Le propriétaire veut voir les documents liés à une facture.	
Acteurs	Propriétaire authentifié	
Pré-conditions	Le Propriétaire est authentifié sur le système. La facture existe dans le système.	
Post-conditions	En cas de succès	Les documents de la facture sont affichés.
	En cas d'échec	Les documents de la facture ne sont pas affichés.
Scénario principal	1. Le propriétaire se trouve sur la page des factures. 2. Le propriétaire clique sur détails. 3. Le système affiche les documents de la facture.	
Déroulement alternatif	Le serveur ne répond pas.	1. Le système avertit le propriétaire. 2. Fin du cas d'utilisation.
Autres informations	-	

Table A.27. – Cas d'utilisation "Voir les documents liés à une facture"

A.1.7. Gestion des messages

Titre	Envoyer un message	
Description	L'utilisateur envoie un message.	
Acteurs	Utilisateur authentifié	
Pré-conditions	L'utilisateur est authentifié sur le système.	
Post-conditions	En cas de succès	Le message est envoyé.
	En cas d'échec	Le message n'est pas envoyé.
Scénario principal	1. L'utilisateur se trouve sur la pages des messages et clique sur envoyer un message. 2. L'utilisateur remplit les champs obligatoires. 3. L'utilisateur confirme. 4. Le système vérifie si les champs obligatoires sont remplis. 5. Le système envoie le message. 6. L'utilisateur reçoit une confirmation.	
Déroulement alternatif	Des données sont manquantes ou erronées.	1. Le système avertit l'utilisateur. 2. L'utilisateur doit corriger les erreurs. 3. Le cas continue à l'étape 4 du scénario principal.
	L'utilisateur quitte la page sans terminer le processus.	1. Le système avertit l'utilisateur. 2. L'utilisateur valide. Fin du cas d'utilisation. 3. Sinon le cas continue à l'étape 4 du scénario principal.
	Le serveur ne répond pas.	1. Le système avertit le l'utilisateur. 2. Fin du cas d'utilisation.
Autres informations	Les données sont vérifiées par le client et par le serveur.	

Table A.28. – Cas d'utilisation "Envoyer un message"

Titre	Editer le statut d'un message	
Description	Le propriétaire veut éditer le statut d'un message.	
Acteurs	Propriétaire authentifié	
Pré-conditions	Le propriétaire est authentifié sur le système.	
Post-conditions	En cas de succès	Le statut du message est modifié.
	En cas d'échec	Le statut du message n'est pas modifié.
Scénario principal	1. Le propriétaire se trouve sur la page des messages. 2. Le propriétaire clique sur éditer le statut. 3. Le système affiche la liste des statuts disponibles. 4. Le propriétaire choisit le statut voulu et valide. 5. Le système modifie le statut.	
Déroulement alternatif	Le serveur ne répond pas.	1. Le système avertit le propriétaire. 2. Fin du cas d'utilisation.
Autres informations	-	

Table A.29. – Cas d'utilisation "Editer le statut d'un message"

Titre	Supprimer un message	
Description	Le propriétaire supprime un message.	
Acteurs	Propriétaire authentifié	
Pré-conditions	Le propriétaire est authentifié sur le système. Le message existe dans le système.	
Post-conditions	En cas de succès	Le message est supprimé.
	En cas d'échec	Le message n'est pas supprimé.
Scénario principal	1. Le propriétaire clique sur supprimer le message. 2. Le système avertit le propriétaire et lui demande une confirmation. 3. Le propriétaire confirme. 4. Le système supprime le message de sa base de données.	
Déroulement alternatif	Le serveur ne répond pas.	1. Le système avertit le propriétaire. 2. Fin du cas d'utilisation.
Autres informations	-	

Table A.30. – Cas d'utilisation "Supprimer un message"

Titre	Archiver une message	
Description	Le propriétaire archive un message.	
Acteurs	Propriétaire authentifié	
Pré-conditions	Le Propriétaire est authentifié sur le système. Le message existe dans le système.	
Post-conditions	En cas de succès	Le message est archivé.
	En cas d'échec	Le message n'est pas archivé.
Scénario principal	1. Le propriétaire clique sur archiver un message. 2. Le système archive le message.	
Déroulement alternatif	Le serveur ne répond pas.	1. Le système avertit le propriétaire. 2. Fin du cas d'utilisation.
Autres informations	-	

Table A.31. – Cas d'utilisation "Archiver un message"

Titre	Désarchiver une message	
Description	Le propriétaire désarchive un message.	
Acteurs	Propriétaire authentifié	
Pré-conditions	Le Propriétaire est authentifié sur le système. Le message existe dans le système et est archivé.	
Post-conditions	En cas de succès	Le message est désarchivé.
	En cas d'échec	Le message n'est pas désarchivé.
Scénario principal	1. Le propriétaire clique sur désarchiver un message. 2. Le système désarchive le message.	
Déroulement alternatif	Le serveur ne répond pas.	1. Le système avertit le propriétaire 2. Fin du cas d'utilisation
Autres informations	-	

Table A.32. – Cas d'utilisation "Désarchiver un message"

Titre	Ajouter un commentaire	
Description	L'utilisateur commente le message.	
Acteurs	Utilisateur authentifié	
Pré-conditions	L'utilisateur est authentifié sur le système. Le message existe dans le système.	
Post-conditions	En cas de succès	Le message est commenté.
	En cas d'échec	Le message n'est pas commenté.
Scénario principal	1. L'utilisateur écrit son commentaire et valide. 2. Le système vérifie si le champ de texte est rempli. 3. Le système ajoute le commentaire.	
Déroulement alternatif	Le serveur ne répond pas.	1. Le système avertit l'utilisateur. 2. Fin du cas d'utilisation.
Autres informations	-	

Table A.33. – Cas d'utilisation "Ajouter un commentaire"

A.1.8. Gestion des tâches

Titre	Créer une tâche	
Description	Le propriétaire crée une tâche.	
Acteurs	Propriétaire authentifié	
Pré-conditions	Le Propriétaire est authentifié sur le système. Le propriétaire a reçu un ou plusieurs messages nécessitant la création d'une tâche ou le propriétaire souhaite effectuer une tâche.	
Post-conditions	En cas de succès	La tâche est créée
	En cas d'échec	La tâche n'est pas créée.
Scénario principal	1. Le propriétaire clique sur créer une tâche. 2. Le propriétaire remplit les champs obligatoires. 3. Le propriétaire confirme. 4. Le système vérifie si les données obligatoires sont remplies. 5. Le système crée la tâche dans sa base de données.	
Déroulement alternatif	Des données sont manquantes ou erronées.	1. Le système avertit le propriétaire. 2. Le propriétaire doit corriger les erreurs. 3. Le cas continue à l'étape 4 du scénario principal.
	Le propriétaire quitte la page sans terminer le processus.	1. Le système avertit le propriétaire. 2. Le propriétaire valide. Fin du cas d'utilisation. 3. Sinon le cas continue à l'étape 4 du scénario principal.
	Le serveur ne répond pas.	1. Le système avertit le propriétaire. 2. Fin du cas d'utilisation.
Autres informations	Les données sont vérifiées par le client et par le serveur.	

Table A.34. – Cas d'utilisation "Créer une tâche"

Titre	Modifier une tâche	
Description	Le propriétaire modifie une tâche.	
Acteurs	Propriétaire authentifié	
Pré-conditions	Le Propriétaire est authentifié sur le système. La tâche existe dans le système.	
Post-conditions	En cas de succès	La tâche est modifiée.
	En cas d'échec	La tâche n'est pas modifiée.
Scénario principal	1. Le propriétaire modifie les champs souhaités. 2. Le système vérifie que les champs obligatoires soient remplis. 3. Le système demande si le propriétaire souhaite enregistrer les changements. 4. La tâche est modifiée.	
Déroulement alternatif	Des données sont manquantes ou erronées.	1. Le système avertit le propriétaire. 2. Le propriétaire doit corriger les erreurs. 3. Le cas continue à l'étape 2 du scénario principal.
	Le propriétaire quitte la page sans terminer le processus.	1. Le système avertit le propriétaire. 2. Le propriétaire valide. Fin du cas d'utilisation. 3. Sinon le cas continue à l'étape 2 du scénario principal.
	Le serveur ne répond pas.	1. Le système avertit le propriétaire. 2. Fin du cas d'utilisation.
Autres informations	Les données sont vérifiées par le client et par le serveur.	

Table A.35. – Cas d'utilisation "Modifier une tâche"

Titre	Supprimer une tâche	
Description	Le propriétaire supprime une tâche.	
Acteurs	Propriétaire authentifié	
Pré-conditions	Le Propriétaire est authentifié sur le système. La tâche existe dans le système.	
Post-conditions	En cas de succès	La tâche est supprimée.
	En cas d'échec	La tâche n'est pas supprimée.
Scénario principal	1. Le propriétaire clique sur supprimer la tâche. 2. Le système avertit le propriétaire et lui demande une confirmation. 3. Le propriétaire confirme. 4. Le système supprime la tâche de sa base de données.	
Déroulement alternatif	Le serveur ne répond pas.	1. Le système avertit le propriétaire. 2. Fin du cas d'utilisation.
Autres informations	-	

Table A.36. – Cas d'utilisation "Supprimer une tâche"

Titre	Voir l'historique d'une tâche	
Description	Le propriétaire veut voir l'historique d'une tâche.	
Acteurs	Propriétaire authentifié	
Pré-conditions	Le Propriétaire est authentifié sur le système. La tâche existe dans le système.	
Post-conditions	En cas de succès	L'historique de la tâche est affiché.
	En cas d'échec	L'historique de la tâche n'est pas affiché.
Scénario principal	1. Le propriétaire se trouve sur la page des tâches. 2. Le propriétaire clique sur détails. 3. Le système affiche l'historique de la tâche.	
Déroulement alternatif	Le serveur ne répond pas.	1. Le système avertit le propriétaire. 2. Fin du cas d'utilisation.
Autres informations	-	

Table A.37. – Cas d'utilisation "Voir l'historique d'une tâche"

A.1.9. Gestion des réparations

Titre	Créer une réparation	
Description	Le propriétaire crée une réparation	
Acteurs	Propriétaire authentifié	
Pré-conditions	Le Propriétaire est authentifié sur le système. Une tâche nécessitant l'ouverture d'une réparation existe. Divers statuts existent dans le système	
Post-conditions	En cas de succès	La réparation est créée.
	En cas d'échec	la réparation n'est pas créée.
Scénario principal	1. Le propriétaire se trouve sur la page des réparations. 2. Le propriétaire clique sur "enregister une facture de réparation". 3. Le propriétaire remplit les informations et confirme. 4. Le système vérifie si les données obligatoires sont remplies. 5. Le système crée la réparation.	
Déroulement alternatif	Des données sont manquantes ou erronées.	1. Le système avertit le propriétaire. 2. Le propriétaire doit corriger les erreurs. 3. Le cas continue à l'étape 4 du scénario principal.
	Le propriétaire quitte la page sans terminer le processus.	1. Le système avertit le propriétaire. 2. Le propriétaire valide. Fin du cas d'utilisation . 3. Sinon le cas continue à l'étape 4 du scénario principal.
	Le serveur ne répond pas	1. Le système avertit le propriétaire. 2. Fin du cas d'utilisation.
Autres informations	Les données sont vérifiées par le client et par le serveur.	

Table A.38. – Cas d'utilisation "Créer une réparation"

Titre	Modifier une réparation	
Description	Le propriétaire modifie une réparation.	
Acteurs	Propriétaire authentifié	
Pré-conditions	Le Propriétaire est authentifié sur le système. La réparation existe dans le système.	
Post-conditions	En cas de succès	La réparation est modifiée.
	En cas d'échec	La réparation n'est pas modifiée.
Scénario principal	1. Le propriétaire modifie les champs souhaités. 2. Le système vérifie que les champs obligatoires soient remplis. 3. Le système demande si le propriétaire souhaite enregistrer les changements. 4. La réparation est modifiée.	
Déroulement alternatif	Des données sont manquantes ou erronées.	1. Le système avertit le propriétaire. 2. Le propriétaire doit corriger les erreurs. 3. Le cas continue à l'étape 2 du scénario principal.
	Le propriétaire quitte la page sans terminer le processus.	1. Le système avertit le propriétaire. 2. Le propriétaire valide. Fin du cas d'utilisation. 3. Sinon le cas continue à l'étape 2 du scénario principal.
	Le serveur ne répond pas.	1. Le système avertit le propriétaire. 2. Fin du cas d'utilisation.
Autres informations	Les données sont vérifiées par le client et par le serveur.	

Table A.39. – Cas d'utilisation "Modifier une réparation"

Titre	Supprimer une réparation	
Description	Le propriétaire supprime une réparation.	
Acteurs	Propriétaire authentifié	
Pré-conditions	Le Propriétaire est authentifié sur le système. La réparation existe dans le système.	
Post-conditions	En cas de succès	La réparation est supprimée.
	En cas d'échec	La réparation n'est pas supprimée.
Scénario principal	1. Le propriétaire clique sur supprimer la réparation. 2. Le système avertit le propriétaire et lui demande une confirmation. 3. Le propriétaire confirme. 4. Le système supprime la réparation de sa base de données.	
Déroulement alternatif	Le serveur ne répond pas.	1. Le système avertit le propriétaire. 2. Fin du cas d'utilisation.
Autres informations	-	

Table A.40. – Cas d'utilisation "Supprimer une réparation"

Titre	Voir l'historique d'une réparation	
Description	Le propriétaire veut voir l'historique d'une réparation.	
Acteurs	Propriétaire authentifié	
Pré-conditions	Le Propriétaire est authentifié sur le système. La réparation existe dans le système.	
Post-conditions	En cas de succès	L'historique de la réparation est affiché.
	En cas d'échec	L'historique de la réparation n'est pas affiché.
Scénario principal	1. Le propriétaire se trouve sur la page des réparations. 2. Le propriétaire clique sur détails. 3. Le système affiche l'historique de la réparation.	
Déroulement alternatif	Le serveur ne répond pas.	1. Le système avertit le propriétaire. 2. Fin du cas d'utilisation.
Autres informations	-	

Table A.41. – Cas d'utilisation "Voir l'historique d'une réparation"

Titre	Voir les documents liés à une réparation.	
Description	Le propriétaire veut voir les documents liés à une réparation.	
Acteurs	Propriétaire authentifié	
Pré-conditions	Le Propriétaire est authentifié sur le système. La réparation existe dans le système.	
Post-conditions	En cas de succès	Les documents de la réparation sont affichés.
	En cas d'échec	Les documents de la réparation ne sont pas affichés.
Scénario principal	1. Le propriétaire se trouve sur la page des réparations. 2. Le propriétaire clique sur détails. 3. Le système affiche les documents de la réparation.	
Déroulement alternatif	Le serveur ne répond pas.	1. Le système avertit le propriétaire. 2. Fin du cas d'utilisation.
Autres informations	-	

Table A.42. – Cas d'utilisation "Voir les documents liés à une réparation"

A.1.10. Gestion des statuts

Titre	Créer un statut	
Description	Le propriétaire crée un statut.	
Acteurs	Propriétaire authentifié	
Pré-conditions	Le Propriétaire est authentifié sur le système.	
Post-conditions	En cas de succès	Le statut est créé
	En cas d'échec	Le statut n'est pas créé.
Scénario principal	1. Le propriétaire clique sur créer un statut. 2. Le propriétaire remplit les champs obligatoires. 3. Le système vérifie que les champs obligatoires soient remplis. 4. Le statut est créé.	
Déroulement alternatif	Des données sont manquantes ou erronées	1. Le système avertit le propriétaire. 2. Le propriétaire doit corriger les erreurs. 3. Le cas continue à l'étape 3 du scénario principal.
	Le propriétaire quitte la page sans terminer le processus.	1. Le système avertit le propriétaire. 2. Le propriétaire valide. Fin du cas d'utilisation. 3. Sinon le cas continue à l'étape 3 du scénario principal.
	Le serveur ne répond pas.	1. Le système avertit le propriétaire. 2. Fin du cas d'utilisation.
Autres informations	Les données sont vérifiées par le client et par le serveur.	

Table A.43. – Cas d'utilisation "Créer un statut"

Titre	Modifier un statut	
Description	Le propriétaire modifie un statut.	
Acteurs	Propriétaire authentifié	
Pré-conditions	Le Propriétaire est authentifié sur le système. Le statut existe dans le système.	
Post-conditions	En cas de succès	Le statut est modifié.
	En cas d'échec	Le statut n'est pas modifié.
Scénario principal	1. Le propriétaire modifie les champs souhaités. 2. Le système vérifie que les champs obligatoires soient remplis. 3. Le système demande si le propriétaire souhaite enregistrer les changements. 4. Le statut est modifié.	
Déroulement alternatif	Des données sont manquantes ou erronées.	1. Le système avertit le propriétaire. 2. Le propriétaire doit corriger les erreurs. 3. Le cas continue à l'étape 2 du scénario principal.
	Le propriétaire quitte la page sans terminer le processus.	1. Le système avertit le propriétaire. 2. Le propriétaire valide. Fin du cas d'utilisation. 3. Sinon le cas continue à l'étape 2 du scénario principal.
	Le serveur ne répond pas.	1. Le système avertit le propriétaire. 2. Fin du cas d'utilisation.
Autres informations	Les données sont vérifiées par le client et par le serveur.	

Table A.44. – Cas d'utilisation "Modifier un statut"

Titre	Supprimer un statut	
Description	Le propriétaire supprime un statut.	
Acteurs	Propriétaire authentifié	
Pré-conditions	Le Propriétaire est authentifié sur le système. Le statut existe dans le système. Le statut n'est pas utilisé dans une facture, une réparation ou une tâche.	
Post-conditions	En cas de succès	Le statut est supprimé
	En cas d'échec	Le statut n'est pas supprimé.
Scénario principal	1. Le propriétaire clique sur supprimer le statut. 2. Le système avertit le propriétaire et lui demande une confirmation. 3. Le propriétaire confirme. 4. Le système supprime le statut de sa base de données.	
Déroulement alternatif	Le statut est lié à une facture, une tâche, ou une réparation.	1. Le système avertit le propriétaire. 2. Le propriétaire doit supprimer la facture, la tâche ou la réparation ou modifier leur statut. 3. Sinon le cas continue à l'étape 2 du scénario principal.
	Le serveur ne répond pas.	1. Le système avertit le propriétaire. 2. Fin du cas d'utilisation.
Autres informations	-	

Table A.45. – Cas d'utilisation "Supprimer un statut"

A.1.11. Cas d'utilisation communs

Titre	Exporter les données en fichier Excel.	
Description	Le propriétaire veut exporter les données en fichier Excel.	
Acteurs	Propriétaire authentifié	
Pré-conditions	Le Propriétaire est authentifié dans le système.	
Post-conditions	En cas de succès	Les données sont exportées en fichier Excel.
	En cas d'échec	Les données ne sont pas exportées en fichier Excel.
Scénario principal	1. Le propriétaire se trouve sur une page qui possède l'option "EXPORT". 2. Le Propriétaire clique sur "EXPORT". 3. Les données sont exportées en fichier Excel.	
Déroulement alternatif	Le serveur ne répond pas.	1. Le système avertit le propriétaire. 2. Fin du cas d'utilisation.
Autres informations	-	

Table A.46. – Cas d'utilisation "Exporter les données en fichier Excel"

Titre	Ajouter un document	
Description	Le propriétaire veut ajouter un document à un appartement, un locataire, un contrat, une facture ou une réparation	
Acteurs	Propriétaire authentifié	
Pré-conditions	Le Propriétaire est authentifié sur le système. l'appartment/le locataire/le contrat/la facture/la réparation existe.	
Post-conditions	En cas de succès	Le document est ajouté.
	En cas d'échec	Le document n'est pas ajouté.
Scénario principal	1. Le propriétaire se trouve sur la page gérer les documents. 2. Le propriétaire choisit dans quelle section il veut ajouter le document. 3. Le système vérifie que les champs obligatoires soient remplis. 4. Le document est ajouté.	
Déroulement alternatif	Des données sont manquantes ou erronées.	1. Le système avertit le propriétaire. 2. Le propriétaire doit corriger les erreurs. 3. Le cas continue à l'étape 3 du scénario principal.
	Le propriétaire quitte la page sans terminer le processus.	1. Le système avertit le propriétaire. 2. Le propriétaire valide. Fin du cas d'utilisation. 3. Sinon le cas continue à l'étape 3 du scénario principal.
	Le serveur ne répond pas.	1. Le système avertit le propriétaire. 2. Fin du cas d'utilisation.
Autres informations	Les données sont vérifiées par le client et par le serveur.	

Table A.47. – Cas d'utilisation "Ajouter un document"

Titre	Supprimer un document	
Description	Le propriétaire supprime une document.	
Acteurs	Propriétaire authentifié	
Pré-conditions	Le Propriétaire est authentifié sur le système. Le document existe dans le système.	
Post-conditions	En cas de succès	Le document est supprimé.
	En cas d'échec	Le document n'est pas supprimé.
Scénario principal	1. Le propriétaire choisit le document et clique sur supprimer. 2. Le système avertit le propriétaire et lui demande une confirmation. 3. Le propriétaire confirme. 4. Le système supprime le document de sa base de données.	
Déroulement alternatif	Le serveur ne répond pas.	1. Le système avertit le propriétaire. 2. Fin du cas d'utilisation.
Autres informations	-	

Table A.48. – Cas d'utilisation "Supprimer un document"

A.1.12. Cas d'utilisation spécifiques aux locataires

Titre	Voir les factures personnelles	
Description	Le locataire veut voir ses factures.	
Acteurs	Locataire authentifié	
Pré-conditions	Le Locataire est authentifié sur le système.	
Post-conditions	En cas de succès	Les factures sont affichées.
	En cas d'échec	les factures ne sont pas affichées.
Scénario principal	1. Le locataire clique pour voir les factures personnelles. 2. Le système affiche les factures du locataire.	
Déroulement alternatif	Le serveur ne répond pas.	1. Le système avertit le locataire. 2. Fin du cas d'utilisation.
Autres informations	-	

Table A.49. – Cas d'utilisation "Voir les factures personnelles"

Titre	Voir les contrats personnels	
Description	Le locataire veut voir ses contrats.	
Acteurs	Locataire authentifié	
Pré-conditions	Le locataire est authentifié sur le système.	
Post-conditions	En cas de succès	Les contrats sont affichés.
	En cas d'échec	Les contrats ne sont pas affichés.
Scénario principal	1. Le locataire clique pour voir les contrats personnels. 2. Le système affiche les contrats du locataire.	
Déroulement alternatif	Le serveur ne répond pas.	1. Le système avertit le locataire. 2. Fin du cas d'utilisation.
Autres informations	-	

Table A.50. – Cas d'utilisation "Voir les contrats personnels"

Titre	Voir les tâches prévues	
Description	Le locataire veut voir les tâches le concernant.	
Acteurs	Locataire authentifié	
Pré-conditions	Le locataire est authentifié sur le système.	
Post-conditions	En cas de succès	Les tâches sont affichées.
	En cas d'échec	Les tâches ne sont pas affichées.
Scénario principal	1. Le locataire clique pour voir les tâches personnels. 2. Le système affiche les tâches du locataire.	
Déroulement alternatif	Le serveur ne répond pas.	1. Le système avertit le locataire. 2. Fin du cas d'utilisation.
Autres informations	-	

Table A.51. – Cas d'utilisation "Voir les tâches prévues"

Titre	Voir les documents personnels	
Description	Le locataire veut voir ses documents personnels.	
Acteurs	Locataire authentifié	
Pré-conditions	Le locataire est authentifié sur le système.	
Post-conditions	En cas de succès	Les documents sont affichés.
	En cas d'échec	Les documents ne sont pas affichés.
Scénario principal	1. Le locataire clique pour voir les documents personnels. 2. Le système affiche les documents personnels du locataire.	
Déroulement alternatif	Le serveur ne répond pas.	1. Le système avertit le locataire. 2. Fin du cas d'utilisation.
Autres informations	-	

Table A.52. – Cas d'utilisation "Voir les documents personnels"

Titre	Modifier les données personnelles	
Description	Le locataire modifie ses données personnelles.	
Acteurs	Locataire authentifié	
Pré-conditions	Le locataire est authentifié sur le système.	
Post-conditions	En cas de succès	Les données personnelles sont modifiées.
	En cas d'échec	Les données personnelles ne sont pas modifiées.
Scénario principal	1. Le locataire va dans ses données personnelles et clique sur modifier. 2. Le locataire modifie ses données. 3. Le système avertit le locataire et lui demande une confirmation. 4. Le locataire confirme. 5. Le système vérifie que les champs obligatoires sont remplis. 6. Le système met à jour sa base de données.	
Déroulement alternatif	Des données sont manquantes ou erronées.	1. Le système avertit le locataire. 2. Le locataire doit corriger les erreurs. 3. lLe cas continue à l'étape 2 du scénario principal.
	Le locataire quitte la page sans terminer le processus.	1. Le système avertit le locataire. 2. Le locataire valide. Fin du cas d'utilisation. 3. Sinon le cas continue à l'étape 2 du scénario principal.
	Le serveur ne répond pas.	1. Le système avertit le locataire. 2. Fin du cas d'utilisation.
Autres informations	Les données sont vérifiées par le client et par le serveur.	

Table A.53. – Cas d'utilisation "Modifier les données personnelles"

A.2. Endpoints

A.2.1. Authentification

POST /auth/register Register as owner	
Description	Inscription du propriétaire
Accès	Propriétaire non authentifié
Corps de la requête	Les données de base du propriétaire
Paramètres	-
Réponse	Propriétaire créé

POST /auth/login Login as owner	
Description	Authentification du propriétaire
Accès	Propriétaire non authentifié
Corps de la requête	Email et mot de passe du propriétaire
Paramètres	-
Réponse	Token d'authentification

POST /auth/tenant/login Login as tenant	
Description	Authentification du locataire
Accès	Locataire non authentifié
Corps de la requête	Email et mot de passe du locataire
Paramètres	-
Réponse	Token d'authentification

A.2.2. Locataire

GET /tenants Return all tenants	
Description	Retourne la liste des locataires
Accès	Propriétaire authentifié
Corps de la requête	-
Paramètres	-
Réponse	Liste des locataires

GET /tenants/{tenant_id} Return one tenant	
Description	Retourne les données d'un locataire
Accès	Propriétaire authentifié
Corps de la requête	-
Paramètres	L'identifiant du locataire
Réponse	Les données du locataire demandé

POST /tenants/add Create a tenant	
Description	Ajouter un locataire
Accès	Propriétaire authentifié
Corps de la requête	Les données du locataire
Paramètres	-
Réponse	Locataire créé

PUT /tenants/update/{tenant_id} Update a tenant

Description	Modifier un locataire
Accès	Propriétaire authentifié
Corps de la requête	Les nouvelles données du locataire
Paramètres	L'identifiant du locataire
Réponse	Locataire modifié

DELETE /tenants/delete/{tenant_id} Delete a tenant

Description	Supprimer un locataire
Accès	Propriétaire authentifié
Corps de la requête	-
Paramètres	L'identifiant du locataire
Réponse	Locataire supprimé

GET /tenants/contracts/{tenant_id} Return tenant contracts

Description	Retourne les contrats d'un locataire
Accès	Propriétaire authentifié & le locataire dont l'identifiant est passé en paramètre
Corps de la requête	-
Paramètres	L'identifiant du locataire
Réponse	Liste des contrats du locataire

GET /tenants/bills/{tenant_id} Return tenant bills

Description	Retourne les factures d'un locataire
Accès	Propriétaire authentifié & le locataire dont l'identifiant est passé en paramètre
Corps de la requête	-
Paramètres	L'identifiant du locataire
Réponse	Liste des factures du locataire

GET /tenants/tasks/{tenant_id} Return tenant tasks

Description	Retourne les tâches d'un locataire
Accès	Propriétaire authentifié & le locataire dont l'identifiant est passé en paramètre
Corps de la requête	-
Paramètres	L'identifiant du locataire
Réponse	Liste des tâches du locataire

PUT /tenants/upload/{tenant_id} Add file to tenant

Description	Ajouter des documents à un locataire
Accès	Propriétaire authentifié
Corps de la requête	Documents à ajouter
Paramètres	L'identifiant du locataire
Réponse	Document(s) ajouté(s) au locataire

PUT /appartments/upload/{appart_id} Add image to appartment	
Description	Supprimer un document à un locataire
Accès	Propriétaire authentifié
Corps de la requête	L'identifiant de l'image à supprimer
Paramètres	L'identifiant du locataire
Réponse	Document supprimé du locataire

A.2.3. Immeuble

GET /buildings Return all buildings	
Description	Retourne la liste des immeubles
Accès	Propriétaire authentifié
Corps de la requête	-
Paramètres	-
Réponse	Liste des immeubles

GET /buildings/{building_id} Return one building	
Description	Retourne les données d'un immeuble
Accès	Propriétaire authentifié
Corps de la requête	-
Paramètres	L'identifiant de l'immeuble
Réponse	Les données de l'immeuble demandé

GET `/buildings/tenants/{building_id}` Return tenants from one building

Description	Retourne les locataire d'un immeuble
Accès	Propriétaire authentifié
Corps de la requête	-
Paramètres	L'identifiant de l'immeuble
Réponse	Les locataires d'un immeuble

POST `/buildings/add` Create a building

Description	Ajouter un immeuble
Accès	Propriétaire authentifié
Corps de la requête	Les données de l'immeuble
Paramètres	-
Réponse	Immeuble créé

PUT `/appartments/update/{appart_id}` Update an appartment

Description	Mettre à jour un immeuble
Accès	Propriétaire authentifié
Corps de la requête	Les nouvelles données de l'immeuble
Paramètres	L'identifiant de l'immeuble
Réponse	Immeuble modifié

DELETE `/appartments/delete/{appart_id}` Delete an appartment

Description	Supprimer un immeuble
Accès	Propriétaire authentifié
Corps de la requête	-
Paramètres	L'identifiant de l'immeuble
Réponse	Immeuble supprimé

A.2.4. Appartement

GET `/appartments` Return all appartments

Description	Retourne la liste des appartements
Accès	Propriétaire authentifié
Corps de la requête	-
Paramètres	-
Réponse	Liste des appartements

GET `/appartments/{appart_id}` Return one appartment

Description	Retourne les données d'un appartement
Accès	Propriétaire authentifié
Corps de la requête	-
Paramètres	L'identifiant de l'appartement
Réponse	Les données de l'appartement demandé

POST `/appartments/add` Create an appartment

Description	Ajouter un appartement
Accès	Propriétaire authentifié
Corps de la requête	Les données de l'appartement
Paramètres	-
Réponse	Appartement créé

PUT /appartments/update/{appart_id} Update an appartment	
Description	Mettre à jour un appartement
Accès	Propriétaire authentifié
Corps de la requête	Les nouvelles données de l'appartement
Paramètres	L'identifiant de l'appartement
Réponse	Appartement modifié

DELETE /appartments/delete/{appart_id} Delete an appartment	
Description	Supprimer un appartement
Accès	Propriétaire authentifié
Corps de la requête	-
Paramètres	L'identifiant de l'appartement
Réponse	Appartement supprimé

PUT /appartments/upload/{appart_id} Add image to appartment	
Description	Ajouter des documents à un appartement
Accès	Propriétaire authentifié
Corps de la requête	Image(s) à ajouter
Paramètres	L'identifiant de l'appartement
Réponse	Image(s) ajoutée(s) à l'appartement

PUT /appartments/delete/file/{appart_id} delete image from appartment	
Description	Supprimer un document à un appartement
Accès	Propriétaire authentifié
Corps de la requête	L'identifiant de l'image à supprimer
Paramètres	L'identifiant de l'appartement
Réponse	Image supprimée de l'appartement

A.2.5. Contrat

GET /contracts Return all contracts	
Description	Retourne la liste des contrats
Accès	Propriétaire authentifié
Corps de la requête	-
Paramètres	-
Réponse	Liste des contrats

GET /contracts/{contract_id} Return one contract	
Description	Retourne les données d'un contrat
Accès	Propriétaire authentifié
Corps de la requête	-
Paramètres	L'identifiant du contrat
Réponse	Les données du contrat demandé

POST /contracts/add Create a contract	
Description	Ajouter un contrat
Accès	Propriétaire authentifié
Corps de la requête	Les données du contrat
Paramètres	-
Réponse	Contrat créé

PUT `/contracts/update/{contract_id}` Update a contract

Description	Mettre à jour un contrat
Accès	Propriétaire authentifié
Corps de la requête	Les nouvelles données du contrat
Paramètres	L'identifiant du contrat
Réponse	Contrat modifié

PUT `/contracts/archive/{contract_id}` Archive a contract

Description	Archiver un contrat
Accès	Propriétaire authentifié
Corps de la requête	-
Paramètres	L'identifiant du contrat
Réponse	Contrat archivé

DELETE `/contracts/delete/{contract_id}` Delete a contract

Description	Supprimer un contrat
Accès	Propriétaire authentifié
Corps de la requête	-
Paramètres	L'identifiant du contrat
Réponse	Contrat supprimé

PUT `/contracts/upload/{contract_id}` Add file to contract

Description	Ajouter des documents à un contrat
Accès	Propriétaire authentifié
Corps de la requête	Documents(s) à ajouter
Paramètres	L'identifiant du contrat
Réponse	Document(s) ajouté(s) au contrat

PUT /contracts/delete/file/{contract_id} delete file from contract

Description	Supprimer un document à un contrat
Accès	Propriétaire authentifié
Corps de la requête	L'identifiant du document à supprimer
Paramètres	L'identifiant du contrat
Réponse	Document supprimé du contrat

A.2.6. Facture

GET /bills Return all bills

Description	Retourne la liste des factures
Accès	Propriétaire authentifié
Corps de la requête	-
Paramètres	-
Réponse	Liste des factures

GET /bills/{bill_id} Return one bill

Description	Retourne les données d'une facture
Accès	Propriétaire authentifié
Corps de la requête	-
Paramètres	L'identifiant de la facture
Réponse	Les données de la facture demandée

POST /bills/add Create a bill

Description	Ajouter un facture
Accès	Propriétaire authentifié
Corps de la requête	Les données de la facture
Paramètres	-
Réponse	facture créée

PUT /bills/update/{bill_id} Update a status

Description	Mettre à jour une facture
Accès	Propriétaire authentifié
Corps de la requête	Les nouvelles données de la facture
Paramètres	L'identifiant de la facture
Réponse	facture modifiée

DELETE /bills/delete/{bill_id} Delete a bill

Description	Supprimer une facture
Accès	Propriétaire authentifié
Corps de la requête	-
Paramètres	L'identifiant de la facture
Réponse	facture supprimée

PUT /bills/upload/{bill_id} Add file to bill

Description	Ajouter des documents à une facture
Accès	Propriétaire authentifié
Corps de la requête	Documents(s) à ajouter
Paramètres	L'identifiant de la facture
Réponse	Document(s) ajouté(s) de la facture

PUT /bills/delete/file/{bill_id} delete file from bill	
Description	Supprimer un document à une facture
Accès	Propriétaire authentifié
Corps de la requête	L'identifiant du document à supprimer
Paramètres	L'identifiant de la facture
Réponse	Document supprimé de la facture

A.2.7. Statut

GET /status Return all status	
Description	Retourne la liste des statuts
Accès	Propriétaire authentifié
Corps de la requête	-
Paramètres	-
Réponse	Liste des statuts

GET /status/{status_id} Return one status	
Description	Retourne un statut
Accès	Propriétaire authentifié
Corps de la requête	-
Paramètres	L'identifiant du statut
Réponse	Le nom du statut demandé

POST /status/add Create a status

Description	Ajouter un statut
Accès	Propriétaire authentifié
Corps de la requête	Le nom du statut
Paramètres	-
Réponse	Statut créé

PUT /status/update/{status_id} Update a status

Description	Mettre à jour un statut
Accès	Propriétaire authentifié
Corps de la requête	Le nouveau nom du statut
Paramètres	L'identifiant du statut
Réponse	Statut modifié

DELETE /status/delete/{status_id} Delete a status

Description	Supprimer un statut
Accès	Propriétaire authentifié
Corps de la requête	-
Paramètres	L'identifiant du statut
Réponse	Statut supprimé

A.2.8. Historique

GET /history/bills Return all bills histories	
Description	Retourne tous les historiques de toutes les factures
Accès	Propriétaire authentifié
Corps de la requête	-
Paramètres	-
Réponse	Liste des historiques de toutes les factures

GET /history/bills/{bill_id} Return all histories of one bill	
Description	Retourne l'historique d'une facture
Accès	Propriétaire authentifié
Corps de la requête	-
Paramètres	L'identifiant de la facture
Réponse	L'historique de la facture demandée

GET /history/tasks Return all tasks histories	
Description	Retourne tous les historiques de toutes les tâches
Accès	Propriétaire authentifié
Corps de la requête	-
Paramètres	-
Réponse	Liste des historiques de toutes les tâches

GET `/history/bills/{bill_id}` Return all histories of one bill

Description	Retourne l'historique d'une tâche
Accès	Propriétaire authentifié
Corps de la requête	-
Paramètres	L'identifiant de la tâche
Réponse	L'historique de la tâche demandée

GET `/history/repairs` Return all repairs histories

Description	Retourne tous les historiques de toutes les réparations
Accès	Propriétaire authentifié
Corps de la requête	-
Paramètres	-
Réponse	Liste des historiques de toutes les réparations

GET `/history/tasks/{task_id}` Return all histories of one task

Description	Retourne l'historique d'une tâche
Accès	Propriétaire authentifié
Corps de la requête	-
Paramètres	L'identifiant de la tâche
Réponse	L'historique de la tâche demandée

A.2.9. Message

GET /messages/sended Return all sended messages	
Description	Retourne la liste des messages envoyés par l'utilisateur
Accès	Utilisateur authentifié
Corps de la requête	-
Paramètres	-
Réponse	Liste des messages envoyés

GET /messages/received Return all received messages	
Description	Retourne la liste des messages reçus par l'utilisateur
Accès	Utilisateur authentifié
Corps de la requête	-
Paramètres	-
Réponse	Liste des messages reçus

GET /messages/archived Return all archive messages	
Description	Retourne la liste des messages archivés par le propriétaire
Accès	Propriétaire authentifié
Corps de la requête	-
Paramètres	-
Réponse	Liste des messages archivés

POST `/messages/add` Create a message

Description	Envoyer un message
Accès	Utilisateur authentifié
Corps de la requête	Le titre et le contenu du message
Paramètres	-
Réponse	Message envoyé

POST `/messages/comment/add/{message_id}` Add a comment

Description	Ajouter un commentaire à un message
Accès	Utilisateur authentifié
Corps de la requête	Le contenu du commentaire
Paramètres	L'identifiant du message
Réponse	Commentaires ajoutés

PUT `/messages/archive/{message_id}` Archive a message

Description	Archiver un message
Accès	Propriétaire authentifié
Corps de la requête	-
Paramètres	L'identifiant du message
Réponse	Message archivé

PUT `/messages/unarchive/{message_id}` Unarchive a message

Description	Désarchiver un message
Accès	Propriétaire authentifié
Corps de la requête	-
Paramètres	L'identifiant du message
Réponse	Message désarchivé

PUT /messages/update/status/{message_id} update message status	
Description	Mettre à jour le statut d'un message
Accès	Propriétaire authentifié
Corps de la requête	le nouveau statut du message
Paramètres	L'identifiant du message
Réponse	Message modifié

DELETE /messages/delete/{message_id} Delete a message	
Description	Supprimer un message
Accès	Propriétaire authentifié
Corps de la requête	-
Paramètres	L'identifiant du message
Réponse	Message supprimé

A.2.10. Tâche

GET /tasks Return all tasks	
Description	Retourne la liste des tâches
Accès	Propriétaire authentifié
Corps de la requête	-
Paramètres	-
Réponse	Liste des tâches

POST /tasks/add Create a task

Description	Ajouter une tâche
Accès	Propriétaire authentifié
Corps de la requête	Le nom de la tâche
Paramètres	-
Réponse	Tâche créé

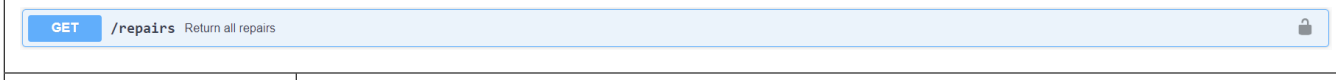
PUT /tasks/update/{task_id} Update a task

Description	Mettre à jour une tâche
Accès	Propriétaire authentifié
Corps de la requête	Les nouvelles données de la tâche
Paramètres	L'identifiant de la tâche
Réponse	Tâche modifiée

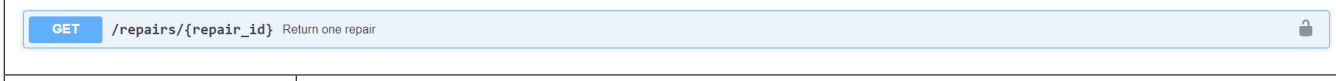
DELETE /tasks/delete/{task_id} Delete a task

Description	Supprimer une tâche
Accès	Propriétaire authentifié
Corps de la requête	-
Paramètres	L'identifiant de la tâche
Réponse	Tâche supprimée

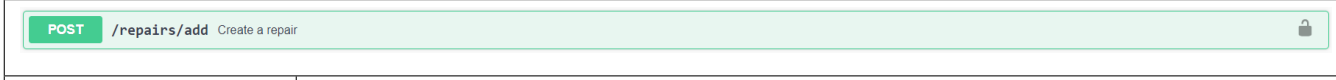
A.2.11. Réparation



Description	Retourne la liste des réparations
Accès	Propriétaire authentifié
Corps de la requête	-
Paramètres	-
Réponse	Liste des réparations



Description	Retourne les données d'une réparation
Accès	Propriétaire authentifié
Corps de la requête	-
Paramètres	L'identifiant de la réparation
Réponse	Les données de la réparation demandée



Description	Ajouter un réparation
Accès	Propriétaire authentifié
Corps de la requête	Les données de la réparation
Paramètres	-
Réponse	réparation créée

PUT /repairs/update/{repair_id} Update a repair

Description	Mettre à jour une réparation
Accès	Propriétaire authentifié
Corps de la requête	Les nouvelles données de la réparation
Paramètres	L'identifiant de la réparation
Réponse	réparation modifiée

DELETE /repairs/delete/{repair_id} Delete a repair

Description	Supprimer une réparation
Accès	Propriétaire authentifié
Corps de la requête	-
Paramètres	L'identifiant de la réparation
Réponse	réparation supprimée

PUT /repairs/upload/{repair_id} Add file to repair

Description	Ajouter des documents à une réparation
Accès	Propriétaire authentifié
Corps de la requête	Documents(s) à ajouter
Paramètres	L'identifiant de la réparation
Réponse	Document(s) ajouté(s) de la réparation

PUT /repairs/delete/file/{repair_id} delete file from repair

Description	Supprimer un document à une réparation
Accès	Propriétaire authentifié
Corps de la requête	L'identifiant du document à supprimer
Paramètres	L'identifiant de la réparation
Réponse	Document supprimé de la réparation

A.3. Liste des modèles

```
const Mongoose = require("mongoose");

const ownerSchema = new Mongoose.Schema(
  {
    name: {
      type: String,
    },
    lastname: {
      type: String,
    },
    email: {
      type: String,
      required: true,
    },
    password: {
      type: String,
      required: true,
    },
    role: {
      type: String,
      default: "Admin",
    },
  },
  { timestamps: true }
);

module.exports = Mongoose.model("Owner", ownerSchema);
```

Listing A.1 – Owner Schema

```
const Mongoose = require("mongoose");

const tenantSchema = new Mongoose.Schema(
  {
    name: {
      type: String,
    },
    lastname: {
      type: String,
    },
    email: {
      type: String,
      required: true,
    },
    password: {
      type: String,
      required: true,
    },
    role: {
      type: String,
      default: "User",
    },
    status: {
      type: String,
      default: "Actif",
    },
    dateofbirth: {
      type: Date,
    },
    createdBy: {
      type: Mongoose.Schema.Types.ObjectId,
      required: true,
      ref: "Owner",
    },
    file: [{ type: Mongoose.Schema.Types.ObjectId, ref: "File" }],
  },
  { timestamps: true }
);

module.exports = Mongoose.model("Tenant", tenantSchema);
```

Listing A.2 – tenant Schema

```
const Mongoose = require("mongoose");

const buildingSchema = new Mongoose.Schema(
  {
    counter: {
      type: Number,
      default: 0,
    },
    numberofAppart: {
      type: Number,
      default: 0,
    },
    adress: {
      type: String,
      required: true,
    },
    postalcode: {
      type: Number,
      required: true,
    },
    city: {
      type: String,
      required: true,
    },
    createdBy: {
      type: Mongoose.Schema.Types.ObjectId,
      required: true,
      ref: "Owner",
    },
  },
  { timestamps: true }
);

module.exports = Mongoose.model("Building", buildingSchema);
```

Listing A.3 – Building Schema

```
const Mongoose = require("mongoose");

const appartSchema = new Mongoose.Schema(
  {
    size: {
      type: Number,
      required: true,
    },
    adress: {
      type: String,
      default: "",
    },
    postalcode: {
      type: Number,
      default: -1,
    },
    city: {
      type: String,
      default: "",
    },
    building: {
      type: Mongoose.Schema.Types.ObjectId,
      ref: "Building",
      default: null,
    },
    picture: [
      {
        type: Mongoose.Schema.Types.ObjectId,
        ref: "File",
      },
    ],
    status: {
      type: String,
      default: "Libre",
    },
    createdBy: {
      type: Mongoose.Schema.Types.ObjectId,
      required: true,
      ref: "Owner",
    },
  },
  { timestamps: true }
);

module.exports = Mongoose.model("Appart", appartSchema);
```

Listing A.4 – Appartment Schema

```
const Mongoose = require("mongoose");

const billSchema = new Mongoose.Schema(
  {
    tenant: {
      type: Mongoose.Schema.Types.ObjectId,
      required: true,
      ref: "Tenant",
    },
    endDate: {
      type: Date,
      required: true,
    },
    reference: {
      type: String,
    },
    amount: {
      type: Number,
      required: true,
    },
    file: [{ type: Mongoose.Schema.Types.ObjectId, ref: "File" }],
    reason: {
      type: String,
      required: true,
    },
    status: {
      type: Mongoose.Schema.Types.ObjectId,
      required: true,
      ref: "Status",
    },
    createdBy: {
      type: Mongoose.Schema.Types.ObjectId,
      required: true,
      ref: "Owner",
    },
  },
  { timestamps: true }
);

module.exports = Mongoose.model("Bill", billSchema);
```

Listing A.5 – Bill Schema

```
const { date } = require("joi");
const Mongoose = require("mongoose");

const billstatusSchema = new Mongoose.Schema(
  {
    billid: {
      type: Mongoose.Schema.Types.ObjectId,
      required: true,
      ref: "Bill",
    },
    endDate: {
      type: Date,
      required: true,
    },
    status: {
      type: Mongoose.Schema.Types.ObjectId,
      required: true,
      ref: "Status",
    },
    createdBy: {
      type: Mongoose.Schema.Types.ObjectId,
      required: true,
      ref: "Owner",
    },
  },
  { timestamps: true }
);

module.exports = Mongoose.model("Billstatus", billstatusSchema);
```

Listing A.6 – Bill status Schema

```
const Mongoose = require("mongoose");

const repairSchema = new Mongoose.Schema(
  {
    amount: {
      type: Number,
      required: true,
    },
    status: {
      type: Mongoose.Schema.Types.ObjectId,
      required: true,
      ref: "Status",
    },
    reason: {
      type: String,
      required: true,
    },
    taskid: {
      type: Mongoose.Schema.Types.ObjectId,
      required: true,
      ref: "Task",
    },
    file: [{ type: Mongoose.Schema.Types.ObjectId, ref: "File" }],
    createdBy: {
      type: Mongoose.Schema.Types.ObjectId,
      required: true,
      ref: "Owner",
    },
  },
  { timestamps: true }
);

module.exports = Mongoose.model("Repair", repairSchema);
```

Listing A.7 – Repair Schema

```
const Mongoose = require("mongoose");

const repairstatusSchema = new Mongoose.Schema(
  {
    repairid: {
      type: Mongoose.Schema.Types.ObjectId,
      required: true,
      ref: "Repair",
    },
    status: {
      type: Mongoose.Schema.Types.ObjectId,
      required: true,
      ref: "Status",
    },
    createdBy: {
      type: Mongoose.Schema.Types.ObjectId,
      required: true,
      ref: "Owner",
    },
  },
  { timestamps: true }
);

module.exports = Mongoose.model("Repairstatus", repairstatusSchema);
```

Listing A.8 – Repair status Schema

```
const Mongoose = require("mongoose");

const messageSchema = new Mongoose.Schema(
  {
    sendedTo: [
      {
        type: Mongoose.Schema.Types.ObjectId,
        required: true,
        refPath: "sendedToType",
      },
    ],
    sendedToType: {
      type: String,
      required: true,
      enum: ["Owner", "Tenant"],
    },
    content: {
      type: String,
      required: true,
    },
    status: {
      type: String,
      default: "envoy[U+FFFD]",
    },
    title: {
      type: String,
      required: true,
    },
    createdBy: {
      type: Mongoose.Schema.Types.ObjectId,
      required: true,
      refPath: "createdByType",
    },
    createdByType: {
      type: String,
      required: true,
      enum: ["Owner", "Tenant"],
    },
    comments: [{ type: Mongoose.Schema.ObjectId, ref: "Comment" }],
  },
  { timestamps: true }
);
module.exports = Mongoose.model("Message", messageSchema);
```

Listing A.9 – Message Schema

```
const Mongoose = require("mongoose");

const commentSchema = new Mongoose.Schema(
  {
    content: {
      type: String,
      required: true,
    },
    createdBy: {
      type: Mongoose.Schema.Types.ObjectId,
      required: true,
      refPath: "createdByType",
    },
    createdByType: {
      type: String,
      required: true,
      enum: ["Owner", "Tenant"],
    },
  },
  { timestamps: true }
);
module.exports = Mongoose.model("Comment", commentSchema);
```

Listing A.10 – Comment Schema

```
const Mongoose = require("mongoose");

const contractSchema = new Mongoose.Schema(
  {
    charge: {
      type: Number,
      required: true,
    },
    rent: {
      type: Number,
      required: true,
    },
    tenant: {
      type: Mongoose.Schema.Types.ObjectId,
      required: true,
      ref: "Tenant",
    },
    appartmentid: {
      type: Mongoose.Schema.Types.ObjectId,
      ref: "Appart",
      required: true,
    },
    file: [{ type: Mongoose.Schema.Types.ObjectId, ref: "File" }],
    other: {
      type: String,
      default: null,
    },
    status: {
      type: String,
      default: "Actif",
    },
    createdBy: {
      type: Mongoose.Schema.Types.ObjectId,
      required: true,
      ref: "Owner",
    },
  },
  { timestamps: true }
);

module.exports = Mongoose.model("Contract", contractSchema);
```

Listing A.11 – Contract Schema

```
const Mongoose = require("mongoose");

const taskSchema = new Mongoose.Schema(
  {
    content: {
      type: String,
      required: true,
    },
    title: {
      type: String,
      required: true,
    },
    startDate: {
      type: Date,
      required: true,
    },
    endDate: {
      type: Date,
      required: true,
    },
    status: {
      type: Mongoose.Schema.Types.ObjectId,
      required: true,
      ref: "Status",
    },
    messageid: {
      type: Mongoose.Schema.Types.ObjectId,
      ref: "Message",
    },
    createdBy: {
      type: Mongoose.Schema.Types.ObjectId,
      required: true,
      ref: "Owner",
    },
  },
  { timestamps: true }
);

module.exports = Mongoose.model("Task", taskSchema);
```

Listing A.12 – Task Schema

```
const Mongoose = require("mongoose");

const taskstatusSchema = new Mongoose.Schema(
  {
    title: {
      type: String,
      required: true,
    },
    content: {
      type: String,
      required: true,
    },
    startDate: {
      type: Date,
      required: true,
    },
    endDate: {
      type: Date,
      required: true,
    },
    taskid: {
      type: Mongoose.Schema.Types.ObjectId,
      required: true,
      ref: "Task",
    },
    status: {
      type: Mongoose.Schema.Types.ObjectId,
      required: true,
      ref: "Status",
    },
    createdBy: {
      type: Mongoose.Schema.Types.ObjectId,
      required: true,
      ref: "Owner",
    },
  },
  { timestamps: true }
);

module.exports = Mongoose.model("Taskstatus", taskstatusSchema);
```

Listing A.13 – Task status Schema

```
const Mongoose = require("mongoose");

const statusSchema = new Mongoose.Schema(
  {
    name: {
      type: String,
      required: true,
    },
    createdBy: {
      type: Mongoose.Schema.Types.ObjectId,
      required: true,
      ref: "Owner",
    },
  },
  { timestamps: true }
);

module.exports = Mongoose.model("Status", statusSchema);
```

Listing A.14 – Status Schema

```
const Mongoose = require("mongoose");

const fileSchema = new Mongoose.Schema(
  {
    data: {
      type: Buffer,
    },
    contentType: {
      type: String,
    },
    name: {
      type: String,
    },
  },
  { timestamps: true }
);

module.exports = Mongoose.model("File", fileSchema);
```

Listing A.15 – Status Schema

Sites Web

[1] *API Documentation & Design Tools for Teams | Swagger*. URL : https://swagger.io/ (visité le 26/11/2020).

[2] AUTH0.COM. *JWT.IO - JSON Web Tokens Introduction*. en. URL : http://jwt.io/introduction (visité le 31/01/2021).

[3] *HTTP Methods for RESTful Services*. URL : https://www.restapitutorial.com/lessons/httpmethods.html (visité le 16/01/2021).

[4] *HTTP Status Codes - REST API Tutorial*. en-US. Mai 2018. URL : https://restfulapi.net/http-status-codes/ (visité le 16/01/2021).

[5] *Identifiez les avantages d'une API REST*. fr. URL : https://openclassrooms.com/fr/courses/6573181-adoptez-les-api-rest-pour-vos-projets-web/6817216-identifiez-les-avantages-d-une-api-rest (visité le 17/01/2021).

[6] *Koa - next generation web framework for node.js*. URL : https://koajs.com/ (visité le 19/11/2020).

[7] *Koa - Upload files using koa-multer (upload restrictions, error handling)*. URL : https://programmer.group/koa-upload-files-using-koa-multer-upload-restrictions-error-handling.html (visité le 19/11/2020).

[8] *Managed MongoDB Hosting | Database-as-a-Service*. en-us. URL : https://www.mongodb.com/cloud/atlas (visité le 06/01/2021).

[9] *Material-UI : A popular React UI framework*. en. URL : https://material-ui.com/ (visité le 19/11/2020).

[10] NODE.JS. *Node.js*. en. URL : https://nodejs.org/en/ (visité le 23/01/2021).

[11] *node.js - How can I split my koa routes into separate files ?* URL : https://stackoverflow.com/questions/30285683/how-can-i-split-my-koa-routes-into-separate-files (visité le 19/11/2020).

[12] *React – Une bibliothèque JavaScript pour créer des interfaces utilisateurs*. fr. URL : https://fr.reactjs.org/ (visité le 19/11/2020).

[13] Colin REILLY. *Send, Store and Show Images With React, Express and MongoDB*. en. Août 2018. URL : https://medium.com/@colinrlly/send-store-and-show-images-with-react-express-and-mongodb-592bc38a9ed (visité le 12/12/2020).

[14] The WebShala. *Responsive table design using only html css*. 2019. URL : https://www.youtube.com/watch?v=ZtopjfXhUZI (visité le 20/12/2020).

[15] *What is Use Case Diagram ?* URL : https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-use-case-diagram/ (visité le 25/01/2021).

UNIVERSITÉ DE FRIBOURG
UNIVERSITÄT FREIBURG

Faculté des sciences économiques et sociales
Wirtschafts- und sozialwissenschaftliche Fakultät
Boulevard de Pérolles 90
CH-1700 Fribourg

D E C L A R A T I O N

Par ma signature, j'atteste avoir rédigé personnellement ce travail écrit et n'avoir utilisé que les sources et moyens autorisés, et mentionné comme telles les citations et paraphrases.

J'ai pris connaissance de la décision du Conseil de Faculté du 09.11.2004 l'autorisant à me retirer le titre conféré sur la base du présent travail dans le cas où ma déclaration ne correspondrait pas à la vérité.

De plus, je déclare que ce travail ou des parties qui le composent, n'ont encore jamais été soumis sous cette forme comme épreuve à valider, conformément à la décision du Conseil de Faculté du 18.11.2013.

……, le ..... 20.....

.....
(signature)