Rapport d’étudiants ingénieurs

Deuxième année

Filière 5 : Réseaux et sécurité informatique

Logiciel de gestion de patrimoine immobilier

*Présenté par* : **Sylvain BESSONNEAU et Clément DIF**

Tutrice ISIMA : Myoung-Ah KANG **03 février 2020**

Campus des Cézeaux. 1 rue de la Chébarde. TSA 60125. 63178 Aubière CEDEX

# Table des matières

[Table des matières 2](#_Toc33562132)

[Table des illustrations 3](#_Toc33562133)

[Remerciements 4](#_Toc33562134)

[Introduction 5](#_Toc33562135)

[Etude générale du projet 5](#_Toc33562136)

[I. Introduction de l’étude 5](#_Toc33562137)

[1. Contexte et activité du client 5](#_Toc33562138)

[2. Analyse du besoin 5](#_Toc33562139)

[3. Objectifs du travail et attentes du client 6](#_Toc33562140)

[II. Matériel et méthodes 6](#_Toc33562141)

[1. Etude du problème et questionnements préliminaires 6](#_Toc33562142)

[2. Conception approfondie de la solution 7](#_Toc33562143)

[3. Mise en œuvre de la solution 9](#_Toc33562144)

[a) Ressources et organisation du travail 9](#_Toc33562145)

[b) Architecture logicielle de la solution 12](#_Toc33562146)

[c) Le *backend* ou l’aspect fonctionnel 13](#_Toc33562147)

[d) Le *frontend* ou la relation directe avec l’utilisateur 18](#_Toc33562148)

[III. Résultats et perspectives 22](#_Toc33562149)

[1. Utilisation de la solution 22](#_Toc33562150)

[a) Gestion des locataires 22](#_Toc33562151)

[b) Gestion des locaux 23](#_Toc33562152)

[c) Gestion des contrats 23](#_Toc33562153)

[d) Gestion des documents 24](#_Toc33562154)

[2. Perspectives d’amélioration 24](#_Toc33562155)

[Conclusion 25](#_Toc33562156)

[Résumé 25](#_Toc33562157)

[Abstract 26](#_Toc33562158)

[Glossaire 27](#_Toc33562159)

[Références bibliographiques 27](#_Toc33562160)

[Annexes 27](#_Toc33562161)

# Table des illustrations

[Figure 1: Diagramme de Gantt prévisionnel 7](file:///C:\MAMP\htdocs\REAM\Documents\french%20report.docx#_Toc33562162)

[Figure 2 : Représentation schématique de la base de données utilisée au sein du projet 8](file:///C:\MAMP\htdocs\REAM\Documents\french%20report.docx#_Toc33562163)

[Figure 3 : Logo officiel du logiciel libre Apache HTTP Server créé et maintenu au sein de la fondation Apache. 10](file:///C:\MAMP\htdocs\REAM\Documents\french%20report.docx#_Toc33562164)

[Figure 4 : MySQL, système de gestion de bases de données relationnelles. 10](file:///C:\MAMP\htdocs\REAM\Documents\french%20report.docx#_Toc33562165)

[Figure 5 : Représentation schématique de la séparation frontend et backend dans un projet de développement Web. 11](file:///C:\MAMP\htdocs\REAM\Documents\french%20report.docx#_Toc33562166)

[Figure 6 : Logo officiel de la bibliothèque JavaScript jQuery, libre et multiplateforme. 12](file:///C:\MAMP\htdocs\REAM\Documents\french%20report.docx#_Toc33562167)

[Figure 7 : Représentation hiérarchique de l'architecture logicielle du projet. 13](file:///C:\MAMP\htdocs\REAM\Documents\french%20report.docx#_Toc33562168)

[Figure 8 : Exemple d'entité 14](file:///C:\MAMP\htdocs\REAM\Documents\french%20report.docx#_Toc33562169)

[Figure 9 : Exemple de contrôleur 15](file:///C:\MAMP\htdocs\REAM\Documents\french%20report.docx#_Toc33562170)

[Figure 10 : Exemple d'utilisation du kit Material Design for Bootstrap 18](file:///C:\MAMP\htdocs\REAM\Documents\french%20report.docx#_Toc33562171)

[Figure 11 : Implémentation d'un formulaire côté frontend. 20](file:///C:\MAMP\htdocs\REAM\Documents\french%20report.docx#_Toc33562172)

[Figure 12 : Exemple de code JavaScript d’une requête AJAX. 21](file:///C:\MAMP\htdocs\REAM\Documents\french%20report.docx#_Toc33562173)

# Remerciements

Nous tenons à remercier notre tutrice de projet Mme Myoung-Ah KANG pour l’aide qu’elle a su nous fournir lors de nos différentes rencontres. Elle aura pu notamment nous guider et nous orienter lors de la création de bases de données et de l’agencement des fichiers de l’utilisateur au sein de notre projet.

Par ailleurs nous portons une attention particulière à M. Vincent MAZENOD et M. Rémi DUBOURGNOUX avec qui nous avons pu échanger au sujet de multiples questionnements que nous avons pu avoir au cours de l’élaboration de notre projet.

Enfin, adressons nos remerciements à Mme Murielle MOUZAT pour l’encadrement de la partie expression et communication et pour nous avoir donné les directions pour la rédaction de notre rapport.

# Introduction

Notre projet est un logiciel de gestion de gestion de patrimoine immobilier. Il comprend ainsi la gestion de contrats, de logements, de locataires ainsi que des documents qui en découlent. Celui-ci a été développé dans le but de répondre au besoin d’un particulier qui gère un parc d’appartements.

Dans une première partie, nous présenterons le projet en détail en analysant le besoin du client ainsi que les objectifs. Puis, nous parlerons des problèmes auxquels nous avons dû faire face, aux solutions trouvées ainsi que du fonctionnement complet du logiciel. Enfin, nous évoquerons les résultats obtenus avec notre logiciel et également les éventuelles perspectives que nous pouvons effectuer pour l’améliorer.

# Etude générale du projet

## Introduction de l’étude

### Contexte et activité du client

Le client est un particulier gérant la location de plusieurs appartements et hangars au sein d’un parc immobilier. Pour mener son activité, il administre les contrats de bail et les formalités administratives manuellement. Cependant, cela lui nécessite actuellement un temps précieux car il lui est obligatoire de remplir ces documents de manière manuscrite à la venue d’un potentiel locataire.

La création de contrats étant également cause de nombreux échanges, le nombre élevé de courriels envoyés pour le partage d’informations entre les deux parties et la gestion qu’il incombe forme, dans une autre mesure, contrainte au propriétaire.

### Analyse du besoin

Lorsque l’on examine le contexte dans sa globalité, l’on peut en effet noter qu’il pourrait être particulièrement judicieux d’offrir l’automatisation du remplissage des documents au propriétaire. Cette complétion informatisée serait alors mise en place grâce à une saisie de toutes les informations nécessaires par l’utilisateur du logiciel.

L’idée de notre projet et la solution qu’il présente est donc effectivement de proposer un tel service à partir de l’entrée d’informations sur le locataire mais également sur le bien destiné à être loué et sur le type de contrat liant le propriétaire et son client.

Evidemment, des solutions similaires existent déjà sur le marché mais celles-ci sont malheureusement trop coûteuses ou non adaptées à une gestion de patrimoine à échelle non professionnelle.

### Objectifs du travail et attentes du client

Dans un souci de gain de temps, le client a émis le souhait d’avoir un logiciel capable de gérer des locataires, des locaux et des contrats de location de manière détaillée et d’informatiser l’ensemble des données pour une dématérialisation complète.

Le logiciel que nous devons mettre en place doit en conséquence être capable de créer des locataires à partir d’un formulaire, de les modifier, de les rechercher, de les afficher et de leur assigner un contrat tout cela au travers d’une interface graphique. Par ailleurs, la solution doit également permettre une gestion similaire de locaux en autorisant leur ajout et leur suppression au besoin.

Également, notre programme se doit d’apporter un service équivalent concernant les baux de locations. Cela se traduit d’une part par leur création en assignant un local et un locataire et d’autre part, par la génération automatique de documents à partir de patrons fournis par l’utilisateur du logiciel, propriétaire des biens immobiliers.

Enfin, une des principales attentes de l’utilisateur, bien que non directement liée aux fonctionnalités propres du logiciel, était la présence d’une interface propre et surtout rapide d’utilisation. Le client souhaite effectivement se dispenser de tout type de manuel et pouvoir accéder et prendre en main le logiciel de manière tout à fait intuitive et apprécier par ailleurs son esthétisme.

## Matériel et méthodes

### Etude du problème et questionnements préliminaires

Les objectifs étant fixés et déterminés, nous avons dû par la suite rechercher l’ensemble des technologies possibles afin de sélectionner celles permettant de réaliser une solution viable et durable.

Premièrement, le réflexe que nous avions eu a été d’opter pour une solution « application de bureau » ou « desktop application » en anglais c’est-à-dire un logiciel applicatif affichant son interface graphique dans un environnement de bureau. Une telle solution aurait ainsi pu être implémentée à l’aide de langages fournissant cette possibilité comme le C++ ou le Java.

Néanmoins, bien que satisfaisante de prime abord, nous nous sommes rapidement rendus compte qu’une telle idée n’était pas à conserver et qu’il était d’ailleurs plus judicieux d’aliéner notre intérêt pour ces dernières. En effet, l’une des contraintes les plus importantes était, comme nous l’avons mentionné en introduction d’étude, la présence d’une interface esthétique et intuitive. En dépit du fait que le C++ et le Java offrent des cadres d'applications et bibliothèques pour un affichage graphique, les solutions apportées n’égalaient pas les exigences de départ. Par ailleurs, la qualité de réflexion nécessaire à la mise en place d’une application à l’aide du langage C++ formait un inconvénient considérable car, bien que très complet, sa complexité intrinsèque aurait risqué de ralentir considérablement le développement de la solution sans pour autant combler une nécessité de performance.

C’est donc à juste titre que nous avons préféré éviter les bibliothèques relevant de ces deux langages comme GTKmm ou Qt pour le premier et JavaFX pour le second, pour nous concentrer sur des technologies plus adaptées à notre problème.

Nous nous sommes alors rapprochés de techniques employées au sein du World Wide Web[[1]](#endnote-1) car elles disposent d’une flexibilité notable et procurent une réelle segmentation entre l’architecture logicielle interne et la création de l’interface. Cette occasion de fractionner le développement a d’ailleurs formé un atout notable qui nous a largement permis de mettre en place une organisation solide et robuste du travail d’équipe comme nous le verrons dans la suite de notre étude. En outre, du fait de leur présence récurrente dans l’univers numérique, la possibilité de créer des interfaces « familières » a été un avantage non négligeable pour la suite du projet et a donc, par la même occasion, confirmé notre sélection.

Ensuite, dès lors que nous avions déterminé une piste de développement et de création, il nous a été obligatoire, comme nous allons l’expliquer en sous-partie suivante, de conceptualiser la solution et de déterminer un plan d’implémentation afin de cadrer notre travail.

### Conception approfondie de la solution

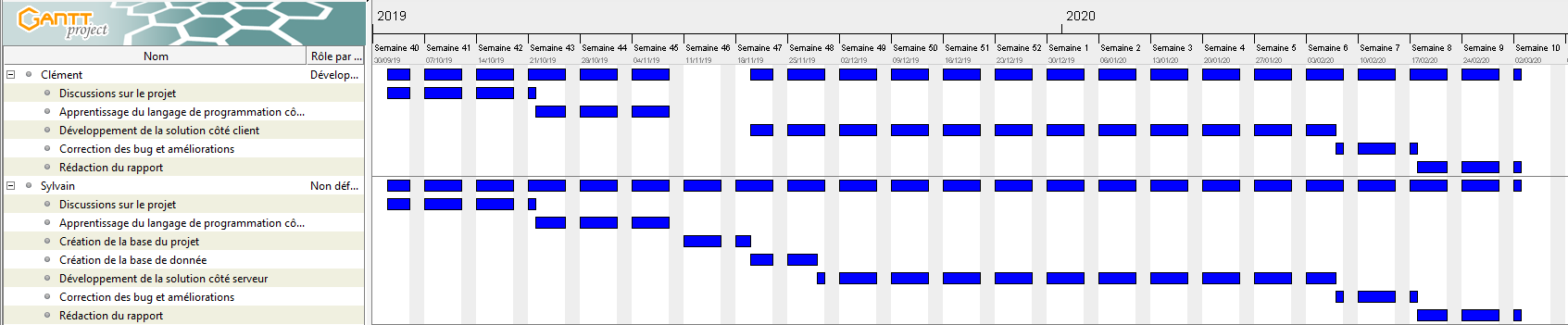
Au tout début de la phase de conception, avant même de commencer le projet, nous avons élaborer un diagramme de Gantt prévisionnel afin de pouvoir savoir à tout instant ce que nous avons accompli et ce qu’il nous reste à accomplir. Voici ci-après le diagramme de Gantt prévisionnel :

Figure 1: Diagramme de Gantt prévisionnel

La phase de conception, essentielle au bon déroulement d’un projet a eu lieu en lien direct avec le client final. Effectivement, il était important, dès les prémisses de la création de la solution, d’être en accord avec l’utilisateur afin de proposer un résultat adéquat à ses besoins que nous avons développés précédemment.

La première étape a été de comprendre quelles fonctionnalités souhaitait le client et de quelle manière pouvaient-elles être affichées à sa convenance. La maquette ainsi réalisée de l’interface dans son ensemble donnait une idée de la représentation globale du logiciel et de son interface générale. C’est alors ici que le client s’est montré relativement souple dans la création de chaque cas d’utilisation et nous a offert la liberté de la création des interfaces dans ces cas spécifiques. Evidemment la contrainte d’une utilisation abordable et autodidacte du logiciel restait maîtresse de nos ambitions.

**[INSERER UNE IMAGE DE LA MAQUETTE + L’EXPLIQUER]**

Néanmoins, afin de pouvoir générer des documents facilement et rapidement, il était évident de devoir coucher sur le papier l’ensemble des données devant être renseignées au sein du logiciel. Il a donc été nécessaire de recenser une liste d’informations pour chaque entité prenant part à la création d’un contrat comme, à titre d’exemple, le locataire ou le local.

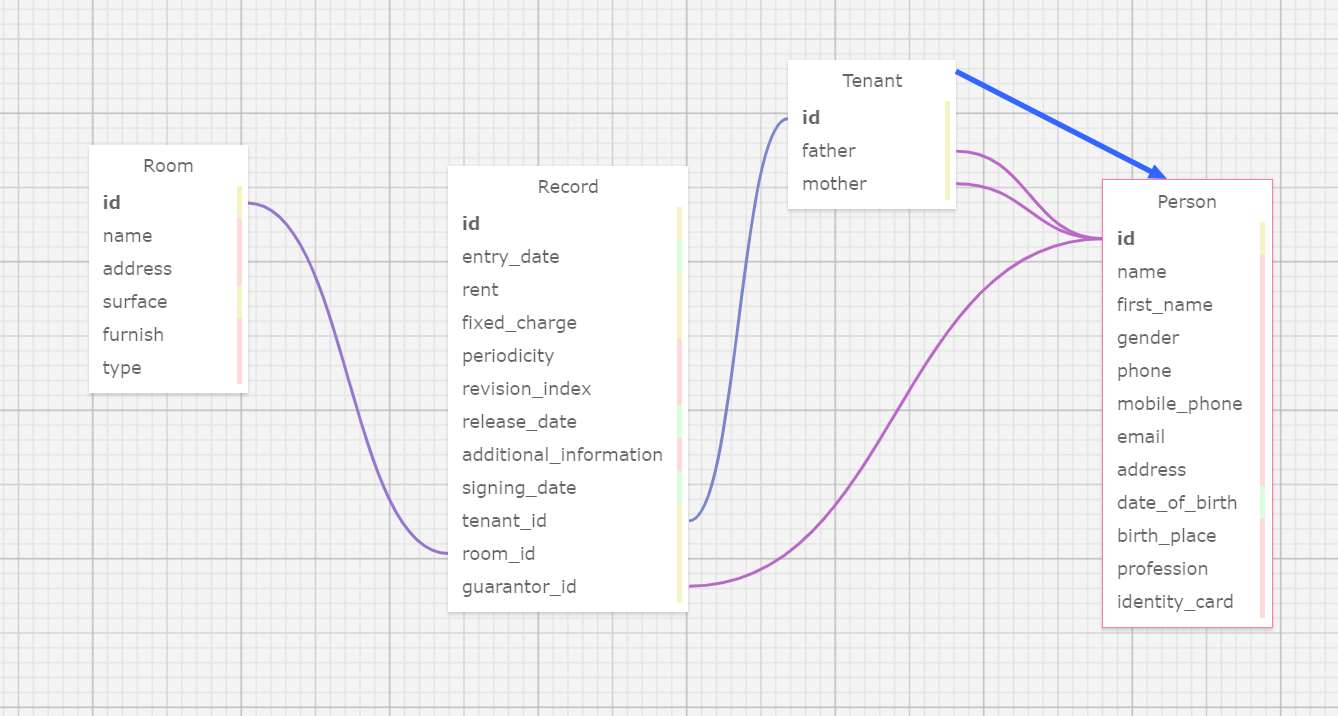
Subséquemment, nous avons dû nous pencher sur la manière qui pourrait nous permettre de stocker et garder ces données de façon structurée. Immanquablement, la création d’une base de données prévue à cet effet à de suite été étudiée. Nous avons alors pris soin d’échanger au cours d’une réunion avec notre tutrice afin de pouvoir agencer, avec la disposition la plus logique et rigoureuse possible, l’ensemble des données mentionnées. De cette rencontre productive a émergé le schéma suivant qui a de suite été adopté et qui d’ailleurs nous aura suivi jusqu’à la fin du projet.

Figure 2 : Représentation schématique de la base de données utilisée au sein du projet

Comme cela peut être constaté aisément, l’ensemble de la base de données et d’ailleurs, par extension, l’ensemble du code de notre application est en anglais. Ce choix est justifié de par le fait que même si l’interface est en français, l’anglais est la langue utilisée en informatique dans le milieu professionnel. Nous avons en conséquence souhaité reprendre cette pratique usuelle et courante et l’appliquer à notre projet.

Dans le prolongement de la conception de notre solution, s’est alors posé le problème de comment récupérer les patrons de documents, comment les interpréter et surtout comment les classer. En premier lieu, suivant les conseils et explications du client, il était particulièrement judicieux et commode d’élaborer une stratégie pour catégoriser les documents. Nous avons alors décidé de considérer ces modèles textuels sous forme de trois catégories que l’on retrouve classées ci-dessous du cas le plus spécifique cas le plus général :

* Les patrons uniques ou concernant un local particulier

Ils regroupent les modèles liés à un seul local présent en base de données. L’on retrouve par exemple les documents relatifs aux risques sismiques et technologiques d’un appartement ou hangar.

* Les patrons typés ou relatif à un type de local

Cette catégorie rassemble les documents relatifs aux appartements seulement ou aux hangars uniquement. L’attestation de consentement d’HADOPI est par exemple un modèle correspondant aux appartements et n’est jamais présente lors de la location des hangars car ces derniers ne prévoient pas un accès internet.

* Les patrons globaux

Les modèles figurant dans cette catégorie sont les plus généraux et ne sont en conséquence pas liés à un type de contrat en particulier, ils sont toujours présents et doivent en conséquence être toujours générés dans n’importe quel type de location.

Connaissant alors, de par notre étude conceptuelle, parfaitement les objectifs du projet et la manière par laquelle celui ici devait être façonné, nous devions alors amorcer le développement de ce dernier. Nous avons alors entrepris de mettre en place l’architecture définie à l’aide de technologies Web comme expliqué antérieurement. Nous verrons, en sous-partie suivante, dans quelle mesure nous avons mis en œuvre la solution, avec quelles ressources et dans quelle perspective d’organisation.

### Mise en œuvre de la solution

#### Ressources et organisation du travail

Avant de débuter l’écriture du code à proprement parlé, nous avons décidé de nous organiser grâce à *GitHub*[[2]](#endnote-2), service web d'hébergement et de gestion de développement de logiciels, utilisant le logiciel de gestion de versions *Git*[[3]](#endnote-3). Celui-ci permet en effet de planifier les étapes du projet grâce à un *« project board »* et de garder à l’esprit les bugs et les divers disfonctionnements à corriger au travers des *« issues »*. L’outil *Git* nous a également permis de faciliter le déroulement du projet par la mise en commun de nos travaux et la gestion des conflits inhérente à la fusion des codes sources.

Notre dépôt est accessible à l’URL suivante : <https://github.com/Clem9963/REAM> et recense l’historique de notre projet que l’on peut donc qualifier *« d’open source »*.

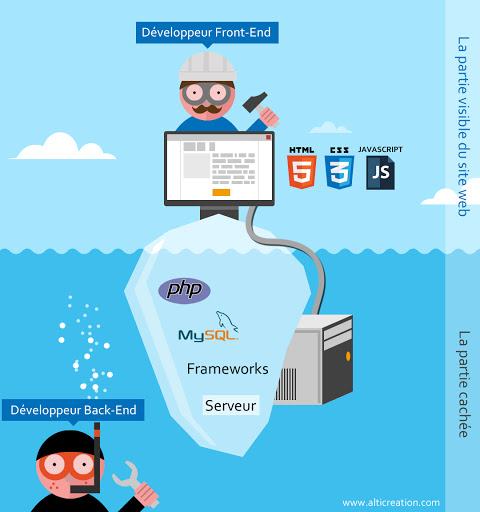
Successivement à la mise en place de notre plan d’organisation, nous avons pris le temps de choisir les outils que nous allions utiliser pour créer la solution. Comme nous l’avons précisé précédemment, nous avions décidé de nous concentrer sur des technologies Web. C’est donc alors que nous avons souhaité mettre en place un cadre de travail avec un serveur HTTP[[4]](#endnote-4) et une base de données en local c’est-à-dire sans hébergement sur internet.

La préparation d’un environnement n’a pas été tâche ingrate car de nombreux outils libres de droits existent. Nous nous sommes alors décidés d’opter pour les services d’un serveur Apache (confère figure 2 ci-contre) et d’un système de gestion de base de données MySQL (confère figure 3 ci-dessous). Malgré leur ampleur, leurs fonctionnalités nous ont été très utiles dès les premières ébauches de notre projet. En effet, leur configuration rapide nous a permis de nous concentrer rapidement sur le code source et d’obtenir ainsi aussitôt quelque chose de fonctionnel à discuter avec le client.

Figure 3 : Logo officiel du logiciel libre Apache HTTP Server créé et maintenu au sein de la fondation Apache.

L’environnement étant installé et parfaitement fonctionnel, nous avons alors décidé de nous séparer la charge de travail pour éviter, au cours du développement, de produire des codes divergents qui auraient nécessité une mise en commun plus que chronophage.

Figure 4 : MySQL, système de gestion de bases de données relationnelles.

Cela a amplement été facilité par la qualité intrinsèque du développement Web qu’est la séparation de la programmation côté serveur de celle côté client. Il a donc été tout à fait judicieux d’attribuer l’une et l’autre à chacun des membres du binôme. Sylvain BESSONNEAU s’est alors tenu responsable d’étudier et de mettre en place l’architecture *« backend[[5]](#endnote-5) »* tandis que Clément DIF s’est concentré sur la partie frontale du projet plus communément dénommée *« frontend [[6]](#endnote-6)»*.

Concernant les langages, nous avons d’une part décidé d’utiliser le PHP pour le côté *« backend »* et le JavaScript, CSS et HTML d’autre part pour son alter ego frontal.

Figure 5 : Représentation schématique de la séparation frontend et backend dans un projet de développement Web.

*Crédits :* [*www.alticreation.com*](http://www.alticreation.com)

Du fait de sa flexibilité et de sa compatibilité avec la plupart des systèmes de gestion de base de données, le PHP était en effet une technologie pertinente compte tenu des objectifs du projet. Par ailleurs, il dispose d’un vaste panel de *« frameworks »* aidant à la création de sites Web et très largement utilisés dans le milieu professionnel. Nous avons alors considéré qu’utiliser la version 4 du cadriciel Symfony pouvait être très adapté à la situation car il s’agit d’un cadre d’applications fournissant de nombreux outils. L’on retrouve par exemple la génération et le remplissage de documents, la gestion des données en base et également le puissant moteur de templates Twig.

Ce moteur établit un lien direct avec le *« frontend »* car il permet de générer des pages dynamiques selon une structure définie. Le développeur se chargeant de l’interface a donc la responsabilité de créer cette structure tandis que son homologue s’occupant de la partie métier met en place un ensemble d’informations qui seront transférées à l’interface pour la complétion automatique de la page et donc à posteriori l’affichage du rendu visuel.

Cela nous conduit ainsi à citer les outils employés par le développement frontal. Comme nous avons parlé précédemment, nous dû user des langages HTML, CSS et JavaScript de par leur support par le navigateur auquel nous avons ajouté, de manière très similaire au *« backend »* des frameworks pour faciliter, accélérer et stabiliser le développement. Nous pouvons notamment relever le kit de création d’interface Material Design for Bootstrap qui nous aura été d’une aide précieuse au cours de la création de notre projet. Vient également s’ajouter la bibliothèque JavaScript jQuery (confère figure 5 ci-contre) offrant des services de manipulation du DOM [[7]](#endnote-7)(Document Object Model) et permettant ainsi, comme nous le verrons dans la suite de l’étude, la création aisée de scripts et une amélioration notable de leur maintenabilité.

Figure 6 : Logo officiel de la bibliothèque JavaScript jQuery, libre et multiplateforme.

Nous venons de mettre en exergue les ressources que nous avons utilisées. Néanmoins, notre étude ne s’est pas encore intéressée à la façon technique dont nous avons construit notre solution. Nous allons donc désormais nous pencher sur l’architecture de l’application c’est-à-dire la manière avec laquelle les fichiers du projet, les fichiers utilisateurs et tous les *« assets [[8]](#endnote-8)»* gravitant autour de la solution sont organisés.

#### Architecture logicielle de la solution

Comme facilement constatable sur notre dépôt *GitHub*, la solution que nous avons apportée se décompose en plusieurs dossiers et fichiers. Nous listons dans cette succincte sous-partie quels sont les objectifs remplis par chaque segment de notre travail mais nous ne nous attardons pas ni sur les détails du code source ni sur l’implémentation pure. En effet, nous réservons cette analyse pour la suite de l’étude étant donné qu’il est d’abord nécessaire d’expliquer le contexte de la solution. En page suivante figure l’énumération des dossiers constituants le logiciel accompagnés d’une description rappelant leur objectif.

#### C:\Users\Clément\AppData\Local\Microsoft\Windows\INetCache\Content.Word\Screenshot from 2020-02-23 20-13-13.pngLe *backend* ou l’aspect fonctionnel

Figure 7 : Représentation hiérarchique de l'architecture logicielle du projet.

Nous allons maintenant voir le côté backend de la solution. C’est-à-dire le côté fonctionnel, celui que l’utilisateur ne voit pas, et n’a d’ailleurs pas à s’en soucier. C’est au niveau du backend que se gère tous les éléments de notre logiciel : les locataires, les locaux, les contrats et les documents.

Le langage de programmation utilisé est le PHP. Nous utilisons le Framework Symfony afin de nous faire gagner un temp précieux lors du développement. En effet, ce Framework permet une gestion simplifiée des formulaires ainsi que la possibilité d’ajouter ou de modifier facilement des éléments en base de données.

Au niveau des données, nous les stockons dans une base de données, afin que celles-ci ne se perdent pas. Au début nous avons utilisé le serveur de base de données MySQL. Puis, nous voulions que notre projet se suffise à lui-même, c’est-à-dire que lorsque nous téléchargeons celui-ci, il n’y a pas besoin d’utiliser d’autres outils ou librairies externes au projet. Ainsi, nous sommes passé sur une base de données SQLite, dont le fichier se trouve directement dans notre projet. Cela permet un meilleur déploiement de notre solution. De plus, ce type de base de données est directement gérée au niveau de Symfony. Nous n’avons pas besoin de nous en occuper. Il y a juste une ligne à changer dans les fichiers de configuration.

Au niveau du serveur, nous avions besoin d’un serveur web afin d’exécuter le code PHP. Pour se faire, nous avions commencé par utiliser le serveur Apache. C’est un serveur très connu et qui fonctionne très bien. Cependant, toujours dans un souci de déploiement, nous préférions un serveur qui soit intégré à notre dossier de projet. Comme le logiciel est utilisé en local, c’est-à-dire qu’il n’a pas besoin de connexion à Internet pour fonctionner, nous avons pu nous servir du serveur web intégré à Symfony. Ce qui nous a permis de nous passer d’Apache et ainsi supprimer une dépendance de taille au projet. Nous lançons le serveur avec une commande très simple : *php bin/console server:start localhost:8000*

*php signifie que l’on exécute du PHP*

*bin/console c’est l’endroit où se trouve le gestionnaire de Symfony*

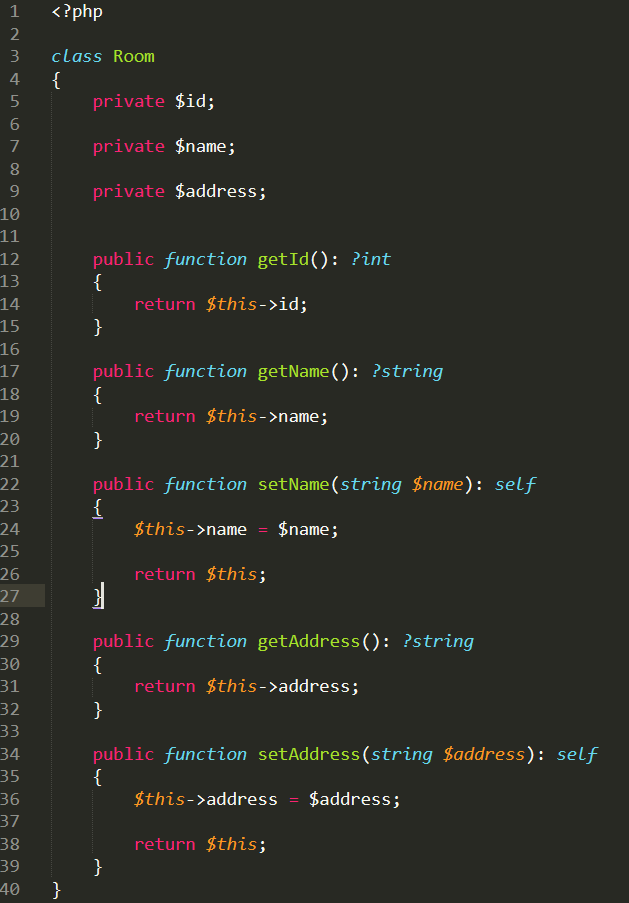
*server:start pour lancer le serveur*

*localhost:8000 c’est pour dire que l’on lance le server en local (localhost) et 8000 c’est*

*le port d’écoute du serveur web*

Une fois le serveur web lancé et la base de données configurée, nous pouvons commencer à faire le code métier permettant de gérer les différents éléments du projet.

Le Framework Symfony nous permet de nous abstenir de faire des requêtes SQL à la base de données. En effet, avec ce Framework, tout est objet, même les éléments en base de données. D’ailleurs ces éléments sont appelés des entités.

Voici un exemple d’entité utilisée pour les appartements :

On peut ainsi manipuler des objets Room qui ont un Id, un nom et une adresse. On peut les modifier et les récupérer.

Une fois que cet objet est rempli comme il faut, on peut le sauvegarder en base de données en une ligne de code. C’est le Framework Symfony qui se charge du reste.

Figure 8 : Exemple d'entité

Tous ces objets et toute la logique métier en général est gérée au niveau de ce que l’on appelle un contrôleur. Un contrôleur est appelé quand on va entrer une URL du projet dans le navigateur web, par exemple : *localhost:8000/index.php/room*. Cette url permet d’afficher tous les locaux qui sont sauvegardés dans le logiciel.

Cette URL va appeler le contrôleur RoomController.php. En voici un exemple :



Figure 9 : Exemple de contrôleur

Sur cette image, on peut remarquer des commentaires au-dessus de chaque fonction. Cela s’appelle des annotations. Elles permettent de paramétrer l’utilisation des fonctions en renseignant ce que l’on appelle des routes. Par exemple, dans mon exemple d’URL précédent (*localhost:8000/index.php/room*), la route est */room*.

Du coup, la fonction *index* qui permet de lister les locaux, est appelée si on renseigne la route */room* avec la méthode GET.

Plus en détail, dans la fonction *index*, on appelle *Doctrine* c’est le gestionnaire de base de données de Symfony et on lui demande de récupérer la table des locaux. (Ligne 13)

On récupère ensuite tous les enregistrements des locaux à l’aide de la méthode findAll (Ligne 14)

La méthode GET permet de récupérer une page, alors que la méthode POST permet d’envoyer des données au serveur, par exemple des données renseignées dans un formulaire d’ajout d’un local.

La fonction *add* permet d’afficher le formulaire d’ajout d’un local si la méthode utilisée est GET. On peut remarquer que Symfony gère tout seul les champs du formulaire à envoyer au frontend grâce à l’entité *Room*. (Ligne 27)

Au contraire, si la méthode est POST, alors cela veut dire qu’on soumet le formulaire d’ajout d’un local. On récupère donc les données et on va sauvegarder ces données en base en faisant ce que l’on appelle de la persistance. On va persister les données en base. (Lignes 35-36).

Nous avons décrit deux fonctions du contrôleur des locaux. La syntaxe est la même pour chaque fonction. Seul le code métier à l’intérieur change. A savoir que c’est un peu prêt la même chose pour la gestion des locataires et des contrats.

Enfin, les lignes 38 à 42 permettent la gestion des documents. Dans notre exemple, il permet de créer un dossier relatif au local que l’on vient de créer.

En effet, notre logiciel permet de gérer de manière précise et structurée les documents relatifs aux locataires, locaux et contrats.

Nous avons aussi besoin de remplir des documents génériques ou templates automatiquement. Pour ce faire, nous avions d’abord pensé à utiliser des fichier PDF remplissable. Cependant, cette solution nous contraignait à devoir acheter une licence Adobe afin de pouvoir remplir les fichiers PDF. Nous avons donc cherché d’autres solutions, et nous avons décidé d’utiliser plutôt des fichiers Word (DOCX). Ceux-ci sont en effet remplissable directement par le code PHP et les modules Symfony.

Voici en détails les différents types de documents utilisés :

* Les documents relatifs à un local : ce sont des documents qui diffèrent d’un local à un autre. Ces documents sont ainsi gérés directement dans la partie du local en question

On peut en ajouter, en supprimer et les télécharger. Le format accepté est le format Word (docx). C’est obligatoirement ce format car il faut pouvoir le lire et le remplir automatiquement. C’est un format qui nous permet de remplir facilement le document à partir du code PHP.

* Les documents générés automatiquement : Ce sont les documents qui sont générés lors de la création d’un contrat. Ceux-ci sont accessible directement à partir du contrat auxquels ils sont rattachés. Pour ce faire, nous utilisons une librairie PHP qui prend en entrée un fichier Word (docx) et qui va le remplir grâce à des balises présentes dans le fichier. En sortie, nous obtenons un fichier PDF qui a donc été généré automatiquement.
* Les documents relatifs à un locataire : ce sont tous les documents obligatoires ou non relatifs à un locataire en particulier. Ils sont ainsi accessibles à partir de la page du locataire en question. Le logiciel accepte des fichiers PDF, JPEG et PNG. Il est possible d’en ajouter, d’en supprimer ou d’en télécharger un. De plus, un bouton permet de télécharger tous les fichiers d’un locataire sous forme d’archive.

De plus, les documents ainsi que les templates sont stockés selon une arborescence de dossiers bien précise :

* users
* default
* generated Dossier des templates générés automatiquement
* record1 Document templates pour le contrat numéro 1
* record2 Document templates pour le contrat numéro 2
* ……
* templates Dossier des contenant les templates vides
* general Contient les templates généraux
* typed Contient les templates typés
* apartments Contient les templates pour les appartements
* hangars Contient les templates pour les hangars
* unique Contient documents pour les locaux
* room1 Contient les documents relatifs au local numéro 1
* room2 Contient les documents relatifs au local numéro 2
* …..
* tenants Dossier des contenant les documents des locataires
* tenant1 Contient les documents relatifs au locataire numéro 1
* tenant2 Contient les documents relatifs au locataire numéro 2
* …

Une fois le traitement attendu effectué au niveau du contrôleur, il faut bien envoyer quelque chose à l’utilisateur pour qu’il sache où en est le logiciel. Pour ce faire, le contrôleur va envoyer des objets PHP directement au frontend afin que celui-ci puisse afficher les données correctement à l’utilisateur (Ligne 44).

#### Le *frontend* ou la relation directe avec l’utilisateur

A présent, nous allons nous pencher sur la réalisation détaillée de l’interface utilisateur. Afin de plonger dans les détails du code source, il est judicieux de rappeler les outils utilisés et d’expliquer leur contexte d’utilisation et leur fonctionnement dans une approche orientée développement.

Concernant le cadriciel MDB (Material Design for Bootstrap), celui-ci se définit comme un kit permettant de créer aisément des interfaces esthétiques et s’utilise directement dans le code HTML. Ce dernier étant un langage déclaratif utilisant le principe des balises pour créer la structure et l’architecture de la page, il suffit alors de placer un composant MDB dans la page pour permettre son affichage. Cela est rendu possible grâce aux attributs *« class »* des nœuds HTML. Ci-dessous figure d’ailleurs un exemple d’utilisation d’un composant MDB au sein de la page d’accueil de notre projet.

Figure 10 : Exemple d'utilisation du kit Material Design for Bootstrap

Comme nous pouvons aisément le remarquer, il s’agit d’un composant *« Card »* affichant un contenu constitué d’un titre, d’une image et d’un corps. L’aspect Material Design offre des ombres, des profondeurs et des effets de survol caractéristiques. Ce composant permet en outre un affichage rapide et instinctif des informations à présenter à l’utilisateur.

Même si l’on peut comprendre sans peine la sémantique du code source, il peut être intéressant d’appréhender le fonctionnement de l’affichage du composant par le navigateur. Cela est en fait permis grâce à l’inclusion en début de fichier HTML des fichiers de style CSS du kit MDB. Lorsque le navigateur rencontre les attributs *« class »* au cours de l’analyse syntaxique ou *parsing* de la page, il sait alors quels styles appliquer en fonction de la valeur de cet attribut et procède au rendu de la page de manière adéquate.

Ce procédé est par ailleurs extrêmement puissant car il est possible de redéfinir, dans un autre fichier CSS, les styles appliqués à un composant ou même à un autre nœud de la page. Cela renforce a fortiori la flexibilité et les possibilités de création par le développeur *frontend*. En outre, le langage HTML permet l’attribution de plusieurs classes à un nœud et permet en conséquence l’ajout supplémentaire de règles de style aux composants MDB.

Dans l’exemple en figure 6, nous avons également pu remarquer la présence d’accolades lors de la définition du corps du composant. Ceci est une conséquence directe de l’utilisation du moteur de *templates* Twig et représente la mise en place d’une page dynamique. En effet, lorsque la page est rendue par le navigateur, celle-ci doit d’abord être générée par le *backend*.

Cependant, se pose la question évidente du lien entre le code métier et la création de l’interface. C’est alors que les objets PHP prennent part à la situation. En effet, ceux-ci sont transférés au moteur et peuvent donc être utilisés directement dans les pages à rendre. Il est alors tout à fait convenable d’accéder aux attributs de ces objets dans le code HTML afin d’afficher les informations utiles à l’utilisateur. Dans l’exemple présenté, les directives utilisées permettent de créer des liens vers d’autres pages selon le nom du contrôleur PHP ou d’accéder à des ressources images. En ce qui concerne les liens hypertexte, le chemin absolu n’est donc pas à renseigner ce qui provoque un gain considérable en termes de flexibilité.

Dans une autre mesure, le moteur de *templates* Twig est très utile car il facilite la création de formulaires du côté *frontend*. En effet, le développeur frontal peut ornementer le design de la page en reprenant les champs fournis par le *backend*. La liaison avec ce dernier est alors largement facilitée.

Ci-contre apparaît d’ailleurs un exemple de formulaire pour l’ajout d’un nouveau local (confère figure 8). L’on peut ici remarquer que le responsable de l’interface a ici placé librement les divers champs incluant leur *label* et leur *widget* en leur associant par la même occasion des classes qui sont utilisées dans des fichiers de style séparés.

Figure 11 : Implémentation d'un formulaire côté frontend.

Toujours dans la continuité de l’étude du moteur de *templates*, au cœur de notre projet, il est judicieux de mentionner le procédé d’héritage qu’il permet entre pages. Cela provoque assurément un gain plus qu’appréciable en maintenabilité car les interfaces ressemblantes peuvent spécialiser un même modèle père. La majorité des pages, suivant un *design* similaire afin de ne pas perdre l’utilisateur, héritent toutes d’une page de base qui définit la structure globale. Chaque code HTML fils vient alors spécifier les différentes sections du code HTML père en fonction de leur objectif.

Afin d’éviter de s’éloigner de la description de la relation entre l’interface et le code métier, il est indispensable d’également expliciter la manière par laquelle l’utilisateur peut fournir des documents à générer ou des pièces justificatives de contrats. Cela implique donc d’étudier la transmission, de manière non textuelle, d’informations depuis l’utilisateur vers le cœur de l’application c’est-à-dire sans user de formulaires.

Cela est mis en place au sein de notre projet au moyen de l’utilisation des requêtes asynchrones AJAX[[9]](#endnote-9). Celles-ci permettent en effet, l’envoi de commandes http de manière non synchrone c’est-à-dire sans nécessaire réactualisation de la page. Le serveur ou application PHP est alors conçu pour proposer des URL consommables en méthodes *POST*, *GET* et *DELETE* pour respectivement modifier, récupérer et supprimer ces fichiers binaires.

Si l’on prend l’exemple de l’ajout d’un patron de document de contrat par l’utilisateur du logiciel. Ce dernier se doit d’accéder à la page prévue à cet effet et de fournir la pièce à envoyer par l’intermédiaire d’une fenêtre contextuelle. Alors, l’application cliente (ici le *frontend*) prépare une requête http en direction du serveur (le *backend* Symfony) en méthode POST en ayant défini les headers et le corps de manière adéquate. Le serveur récupère alors cette requête, l’analyse puis stocke le document dans les fichiers de l’application.

Bien que le protocole http soit un des protocoles les plus utilisés sur internet, il n’est pas incohérent de rappeler que les requêtes effectuées et récupérées par notre application circulent sur la machine locale et ne sont donc à aucun moment transmises sur le réseau mondial. Par ailleurs, comme énoncé précédemment, les documents ne sont pas suivis sur le dépôt *GitHub*. Les problèmes de sécurité et de confidentialité en sont ainsi largement réduits.

De manière plus détaillée, ci-contre se présente en figure 8 le code source d’une requête AJAX pour l’ajout d’un document :

Tout d’abord, nous créons un objet *FormData*, ensemble de clef/valeurs qui sera écrit dans le corps de la requête. Ce dernier nous permet en effet d’ajouter le fichier *file* à envoyer en lui associant la clef *document* comprise par le *backend*.

Dans la suite de la fonction, nous définissons les paramètres de la requête : l’url, passée en argument de la fonction ; la méthode http, ici POST car l’on vient ajouter une ressource ; les données de la requête, ici le *formData* et enfin la désactivation du traitement automatique des données d’une part, car nous envoyons un fichier binaire, et la désactivation de l’en-tête *contentType* d’autre part car le fichier envoyé n’est pas textuel*.*

Figure 12 : Exemple de code JavaScript d’une requête AJAX.

Il est par ailleurs intéressant de noter qu’un *header*[[10]](#endnote-10)additionnel peut également être envoyé. Celui-ci permet de spécifier le type de document et est utilisé pour les pièces justificatives obligatoires d’un contrat. Par exemple, lorsque l’utilisateur renseigne la carte d’identité d’un locataire, cet en-tête sera défini à *« identityCard »*. Enfin, la requête est envoyée grâce au principe des promesses JavaScript et la réponse est inscrite dans la console du navigateur pour faciliter le débogage.

## Résultats et perspectives

### Utilisation de la solution

#### Gestion des locataires

Grace à notre interface « user friendly » il est très facile de gérer les locataires. On peut en ajouter avec le formulaire d’ajout d’un locataire. On peut par ailleurs renseigner son père, sa mère ou les deux. Ceux-ci sont ainsi sauvegardés en base de données lors de la soumission du formulaire.

Lors de l’ajout d’un locataire, plusieurs champs sont à renseignés :

Au niveau du locataire :

Nom,

Prénom,

Sexe,

Téléphone fixe

Téléphone mobile,

Email,

Adresse postale,

Date de naissance,

Lieu de naissance

Au niveau du/des parents :

Nom,

Prénom,

Sexe,

Téléphone fixe

Téléphone mobile,

Email,

Adresse postale,

Date de naissance,

Lieu de naissance,

Profession

Tous ces champs sont obligatoires et nécessaires à la création d’un contrat par la suite.

Une page permet aussi de visionner tous les locataires enregistrés : on peut ainsi les modifier, les supprimer ou les consulter de manière fluide pour l’utilisateur.

Sur la page de consultation d’un locataire, il est possible de rajouter des documents en lien avec le locataire. Par exemple, il est possible d’ajouter la carte d’identité de celui-ci. Celle-ci fait d’ailleurs parti des documents obligatoires que doit posséder l’utilisateur du logiciel. Mais ces documents peuvent être rajoutés plus tard, par exemple, si au moment de la création du contrat, le locataire n’est pas en possession de sa carte d’identité. L’utilisateur est notifié, sur la page principale de l’application, des locataires dont la carte d’identité n’a pas encore été renseignée. Cela lui permet ainsi de toujours savoir quels locataires ne sont pas tout à fait en règle.

#### Gestion des locaux

Notre logiciel permet aussi la gestion des locaux. Ceux-ci comprennent les appartements et les hangars. En effet, il est possible de louer soit un appartement, soit un hangar, soit les deux.

Pour ce faire, tout une partie du logiciel est consacrée aux locaux.

On peut ajouter de nouveaux locaux grâce au formulaire d’ajout approprié.

Plusieurs champs obligatoires sont à renseignés :

Le nom du local, son adresse, sa surface, son type (appartement / hangar) et s’il est meublé ou non.

Après avoir créé un local, l’utilisateur peut associer des documents patrons au local. Comme ceci, chaque local possède ses propres documents patrons qui seront remplis automatiquement lors de la création d’un contrat.

Comme pour les locataires, il est possible de lister les locaux, de les modifier ou de les supprimer.

#### Gestion des contrats

Une fois que le local et le locataire ont été renseigné, il est alors possible de créer un contrat.

Pour ce faire, nous avons élaborer un formulaire de création de contrat, dans lequel le logiciel va demander de remplir certains champs :

Date de début du contrat,

Montant du loyer,

Montant des charges fixes,

Périodicité du loyer,

Indice de révision

Date de fin de contrat,

Date de signature du contrat,

Informations additionnelles (champ facultatif),

Le locataire relatif au contrat,

Le local relatif au contrat

Le champ permettant de choisir le locataire est en fait une liste de sélection qui contient tous les locataires renseignés. Il suffit ainsi de sélectionner celui-ci à qui l’utilisateur veut attribuer le contrat. C’est la même chose pour la sélection du local.

Lors de la création du contrat, il faut aussi renseigner un garant. Celui-ci peut être le père, la mère ou une autre personne.

Si c’est une autre personne, il faut remplir les champs relatifs au garant : ce sont les mêmes que pour ceux pour l’ajout du père ou de la mère lors de la création d’un locataire.

Si c’est le père ou la mère qui est renseignée, une vérification est effectuée lors de la soumission du formulaire. En effet, par exemple, on ne peut pas choisir un garant qui est la mère si, lors de la création du locataire, on a renseigné le père. Du coup, ce qu’il faudrait faire, c’est soit changer le garant, soit modifier le locataire afin de rajouter un parent.

Comme pour les locataires et les locaux, il est tout à fait possible de visualiser la liste des contrats générés et de les supprimer. Il n’est par contre pas possible de les modifier car sinon ce n’est plus le même contrat.

De plus, lors de la création d’un contrat, des documents sont générés automatiquement grâce aux informations fournies par le formulaire. En effet, l’un des buts principaux de ce logiciel est de pouvoir remplir automatiquement des documents lors de la création d’un contrat, afin de faire gagner du temps à l’utilisateur. Il faut juste que l’utilisateur fournisse les documents templates aux logiciels et c’est le logiciel qui s’occupe de les remplir. C’est le cas notamment pour le contrat de bail, pour l’attestation ADOPI ou encore pour l’inventaire.

#### Gestion des documents

Bonne idée de parler des documents, de leur classement et de leur génération mais il serait peut-être plus approprié d’en parler dans la partie sur le backend…

Ici, il faudrait plutôt placer une capture d’écran et expliquer l’utilisation du logiciel en lien avec les objectifs attendus.

### Perspectives d’amélioration

Notre logiciel est actuellement fonctionnel. Il correspond aux attentes du client. Cependant, des éléments peuvent être améliorés afin de le rendre encore plus performant.

Tout d’abord, il serait utile de rajouter un module interactif permettant de gérer les inventaires : l’utilisateur n’aurait qu’à rajouter des champs et des valeurs pour chaque champ et notre logiciel s’occuperait de remplir automatiquement un document faisant l’inventaire des éléments se trouvant dans l’appartement en question.

Ensuite, un élément central que nous n’avons pas eu le temps de mettre en œuvre est le fait de pouvoir envoyer par mail les documents générés automatiquement. Cela permet de faire gagner un temps précieux à l’utilisateur. Cette fonctionnalité se fait de manière quasiment automatique avec le Framework Symfony, c’est pourquoi cette perspective d’amélioration sera effective d’ici peu.

De plus, l’utilisation du Framework Symfony nous a permis de gagner un temps de développement précieux car les fonctionnalités de base comme la gestion de la base de données est déjà implémentée. Cependant, un lien persiste entre le backend et le frontend. En effet, le backend envoi des objets PHP au frontend, ce qui signifie que le frontend doit toujours savoir ce que le backend envoi afin de recevoir et traiter les données correctement. Nous pourrions limiter cela en utilisant, par exemple, le générateur d’application Jhipster. C’est un outil développé par Microsoft et cela permet de générer automatiquement un backend et un frontend fonctionnel qui utilise le fonctionnement des API REST. Cela permet ainsi de bien séparer le backend du frontend. De plus, Jhipster est bien adapté pour ce genre de projet qui utilise des formulaires et des entités. Il gère aussi l’intégration et le déploiement continu et il est très facile de déployer le site sur le cloud.

Enfin, notre logiciel s’utilise actuellement avec un navigateur web. Cela aurait pu être une bonne idée d’encapsuler notre logiciel dans un Framework permettant de générer une application desktop. Le Framework Electron convient parfaitement à ce besoin. Il est facile d’utilisation et s’adapte parfaitement aux projets utilisant Symfony. C’est pourquoi nous pensons utiliser Electron dans les mois prochains afin d’améliorer notre logiciel.

# Conclusion

Ainsi, après avoir beaucoup analysé le besoin du client et appréhendé son domaine d’activité, nous avons été en mesure de concevoir une solution adaptée et répondant à ses attentes. En pesant le pour et le contre de plusieurs outils et méthodes, nous avons finalement décidé de nous diriger vers une solution web, plus facile à mettre en œuvre que les solution desktop avec une interface utilisateur beaucoup plus agréable et soignée. Nous avons ainsi pu déployer le Backend et le Frontend de manière séparée. Notre logiciel peut ainsi gérer des locataires, des locaux et des contrats. Les documents relatifs à chaque élément sont stockés de manière structurée. Les templates se remplissent automatiquement à a création d’un contrat. L’utilisateur peut ainsi utiliser le logiciel en l’état. Cependant, des perspectives d’amélioration peuvent être envisagée, notamment l’ajout d’envoi de mail automatique lors de la création d’un contrat ou encore la gestion d’un inventaire propre à chaque local.

# Résumé

Le but de ce projet est de réaliser un logiciel de gestion de locataires, de locaux et de contrats de location. Dans ce logiciel, l’utilisateur peut renseigner de nouveaux locataires, ajouter des locaux, associer des locataires à un local en créant de nouveaux contrats. Le logiciel permet aussi la gestion de tous les documents relatifs aux différents contrats, locataires et locaux. Cette application est une application web, utilisable avec un navigateur web. Elle a été développée avec le langage PHP et le Framework Symfony côté backend et les langages CSS et JAVASCRIPT avec le Framework Material Design For Bootstrap coté frontend. Le logiciel fonctionne en mode déconnecté car le serveur web et la base de données se trouvent en local.

Le développement a été réalisé avec les logiciels de programmation Visual Studio Code et Sublime Text. Pour le serveur web, nous utilisons le serveur intégré dans Symfony et une base de données SQLite.

Actuellement, le logiciel fonctionne parfaitement et peut d’ores et déjà être utilisé par le client.

Mots-clés : Web, PHP, Symfony, HTML, CSS, JAVASCRIPT, MDB, SQLite

# Abstract

The goal of the project is to realize a software for the management of tenants, rooms and records. In this software, the user can add new tenants, add rooms and associate tenants to a room by creating new records. The software also allows the management of all documents relative to records, tenants and rooms. This application is a web application, usable with a web browser. It is developed with the PHP language and the Symfony Framework for the backend and with CSS and JAVASCRIPT languages and the Material Design for Bootstrap Framework for the frontend. The software works int offline mode because the web server and the database are located locally.

The development was realized with the programming software Visual Studio Code and Sublime Text. For the web server, we use the Symfony integrated web server and the SQLite database.

Currently, the software works perfectly and can already be used by the user.

Keywords: Web, PHP, Symfony, HTML, CSS, JAVASCRIPT, MDB, SQLite

# Glossaire

# Références bibliographiques

# Annexes

1. World Wide Web [↑](#endnote-ref-1)
2. GitHub [↑](#endnote-ref-2)
3. Git [↑](#endnote-ref-3)
4. Serveur HTTP [↑](#endnote-ref-4)
5. Backend [↑](#endnote-ref-5)
6. Frontend [↑](#endnote-ref-6)
7. Document Object Model [↑](#endnote-ref-7)
8. Asset [↑](#endnote-ref-8)
9. AJAX [↑](#endnote-ref-9)
10. Header HTTP [↑](#endnote-ref-10)