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 **02 mars 2020**

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

# Table des matières

[Table des matières 2](#_Toc33727916)

[Table des illustrations 3](#_Toc33727917)

[Remerciements 4](#_Toc33727918)

[Introduction 5](#_Toc33727919)

[Etude générale du projet 5](#_Toc33727920)

[I. Introduction de l’étude 5](#_Toc33727921)

[1. Contexte et activité du client 5](#_Toc33727922)

[2. Analyse du besoin 5](#_Toc33727923)

[3. Objectifs du travail et attentes du client 6](#_Toc33727924)

[II. Matériel et méthodes 6](#_Toc33727925)

[1. Etude du problème et questionnements préliminaires 6](#_Toc33727926)

[2. Conception approfondie de la solution 7](#_Toc33727927)

[3. Mise en œuvre de la solution 11](#_Toc33727928)

[a) Ressources et organisation du travail 11](#_Toc33727929)

[b) Architecture logicielle de la solution 13](#_Toc33727930)

[c) Le *backend* ou l’aspect fonctionnel 14](#_Toc33727931)

[d) Le *frontend* ou la relation directe avec l’utilisateur 18](#_Toc33727932)

[III. Résultats et perspectives 22](#_Toc33727933)

[1. Diagramme de Gantt réalisé 22](#_Toc33727934)

[2. Utilisation de la solution 24](#_Toc33727935)

[a) Commentaires préalables 24](#_Toc33727936)

[b) Page d’accueil et prise en main 24](#_Toc33727937)

[c) Premiers pas et ajout d’un local 25](#_Toc33727938)

[d) Création des patrons de documents d’un local 26](#_Toc33727939)

[e) Renseignement d’un locataire 27](#_Toc33727940)

[f) Elaboration d’un contrat 29](#_Toc33727941)

[g) Gestion des documents typés et généraux 31](#_Toc33727942)

[3. Perspectives d’amélioration 31](#_Toc33727943)

[Conclusion 32](#_Toc33727944)

[Résumé 32](#_Toc33727945)

[Abstract 33](#_Toc33727946)

[Glossaire 34](#_Toc33727947)

[Références 35](#_Toc33727948)

[1. Bibliographie 35](#_Toc33727949)

[2. Webographie 35](#_Toc33727950)

# Table des illustrations

[Figure 1: Diagramme de Gantt prévisionnel 8](file:///C:\Users\Clément\Desktop\french%20report.docx#_Toc33727815)

[Figure 2 : Maquette générale de l'application 9](file:///C:\Users\Clément\Desktop\french%20report.docx#_Toc33727816)

[Figure 3 : Représentation schématique de la base de données utilisée au sein du projet 10](file:///C:\Users\Clément\Desktop\french%20report.docx#_Toc33727817)

[Figure 4 : Logo officiel du logiciel libre Apache HTTP Server créé et maintenu au sein de la fondation Apache. 11](file:///C:\Users\Clément\Desktop\french%20report.docx#_Toc33727818)

[Figure 5 : MySQL, système de gestion de bases de données relationnelles. 12](file:///C:\Users\Clément\Desktop\french%20report.docx#_Toc33727819)

[Figure 6 : Représentation schématique de la séparation frontend et backend dans un projet Web. 12](file:///C:\Users\Clément\Desktop\french%20report.docx#_Toc33727820)

[Figure 7 : Logo officiel de la bibliothèque JavaScript jQuery, libre et multiplateforme. 13](file:///C:\Users\Clément\Desktop\french%20report.docx#_Toc33727821)

[Figure 8 : Représentation hiérarchique de l'architecture logicielle du projet. 14](file:///C:\Users\Clément\Desktop\french%20report.docx#_Toc33727822)

[Figure 9 : Exemple d'entité 15](file:///C:\Users\Clément\Desktop\french%20report.docx#_Toc33727823)

[Figure 10 : Exemple de contrôleur 16](file:///C:\Users\Clément\Desktop\french%20report.docx#_Toc33727824)

[Figure 11 : Organisation hiérarchique des documents de l’utilisateur. 18](file:///C:\Users\Clément\Desktop\french%20report.docx#_Toc33727825)

[Figure 12 : Extrait d'utilisation du kit Material Design for Bootstrap 19](file:///C:\Users\Clément\Desktop\french%20report.docx#_Toc33727826)

[Figure 13 : Implémentation d'un formulaire côté frontend. 20](file:///C:\Users\Clément\Desktop\french%20report.docx#_Toc33727827)

[Figure 14 : Exemple de code JavaScript d’une requête AJAX. 21](file:///C:\Users\Clément\Desktop\french%20report.docx#_Toc33727828)

[Figure 15 : Diagramme de Gantt réalisé 23](file:///C:\Users\Clément\Desktop\french%20report.docx#_Toc33727829)

[Figure 16 : Aperçu de la page d'accueil du logiciel 24](file:///C:\Users\Clément\Desktop\french%20report.docx#_Toc33727830)

[Figure 17 : Aperçu du formulaire d'ajout d'un local 25](file:///C:\Users\Clément\Desktop\french%20report.docx#_Toc33727831)

[Figure 18 : Extrait de modèle de document. 26](file:///C:\Users\Clément\Desktop\french%20report.docx#_Toc33727832)

[Figure 19 : Interface de gestion d'un local. 26](file:///C:\Users\Clément\Desktop\french%20report.docx#_Toc33727833)

[Figure 20 : Formulaire d'ajout d'un nouveau locataire. 28](file:///C:\Users\Clément\Desktop\french%20report.docx#_Toc33727834)

[Figure 21 : Exemple de consultation d'un locataire. 28](file:///C:\Users\Clément\Desktop\french%20report.docx#_Toc33727835)

[Figure 22 : Formulaire d'ajout d'un contrat 30](file:///C:\Users\Clément\Desktop\french%20report.docx#_Toc33727836)

[Figure 23 : Extrait de document généré. 30](file:///C:\Users\Clément\Desktop\french%20report.docx#_Toc33727837)

# 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 la base 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, M. Rémi DUBOURGNOUX et M. Julien USSON avec qui nous avons pu échanger au sujet de multiples questionnements que nous avons pu avoir au cours de l’élaboration de notre projet. Ils nous auront notamment apporté une aide non négligeable concernant le choix des *frameworks* ou encore des outils de création d’interfaces.

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 patrimoine immobilier. Il comprend ainsi l’administration 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 et également les éventuelles perspectives que nous effectuerons dans un futur proche 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, destinés en priorité à des étudiants, 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 des 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 leur 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, l’une des principales attentes de l’utilisateur, bien que non directement liée aux fonctionnalités propres du logiciel, est la présence d’une interface épurée, esthétique 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.

## 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 sur le long terme.

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*.

Nonobstant, 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 cette dernière. 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 » à l’aide de ces technologies 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 élaboré un diagramme de Gantt prévisionnel afin de pouvoir savoir à tout instant ce que nous avions accompli et ce qu’il nous restait à réaliser. Voici ci-après le diagramme de Gantt prévisionnel :

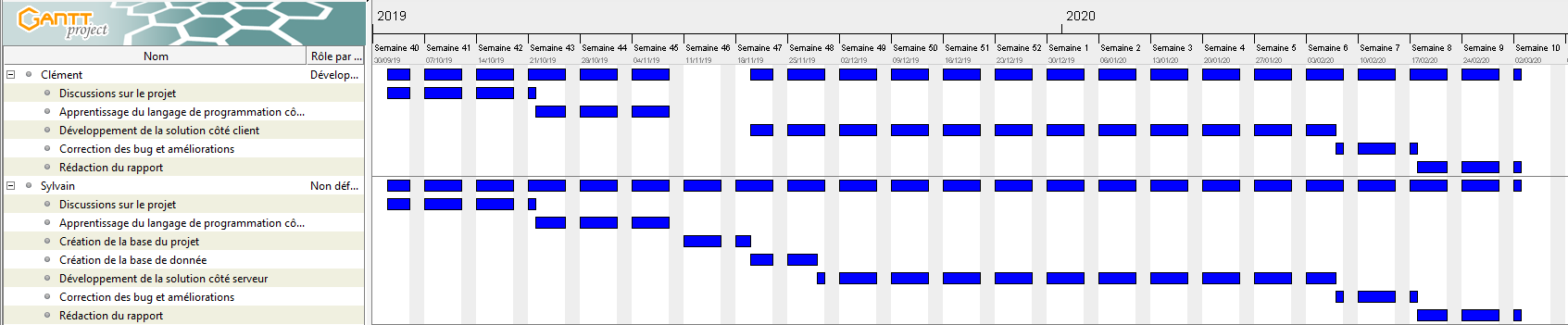


Figure 1: Diagramme de Gantt prévisionnel

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 en adéquation avec ses besoins, que nous avons développés précédemment.

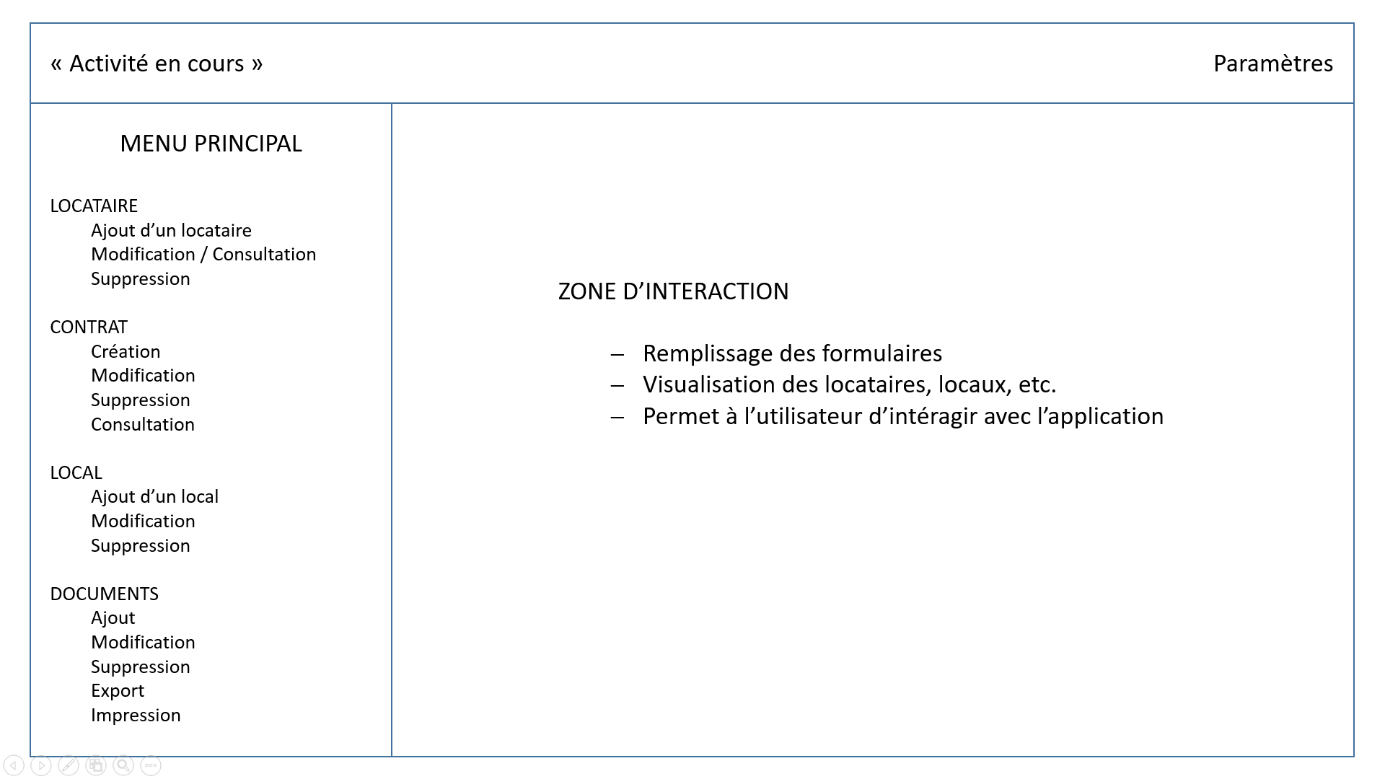
La première étape a été de comprendre quelles fonctionnalités souhaitait le client et de quelle manière pouvaient-elles être affichées à sa convenance. La maquette ainsi réalisée de l’interface dans son ensemble (confère figure 2) donnait une idée de la représentation globale du logiciel et de son interface 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 situations spécifiques. Evidemment la contrainte d’une utilisation abordable et autodidacte du logiciel restait maîtresse de nos ambitions.

Figure 2 : Maquette générale de l'application

De surcroît, 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.

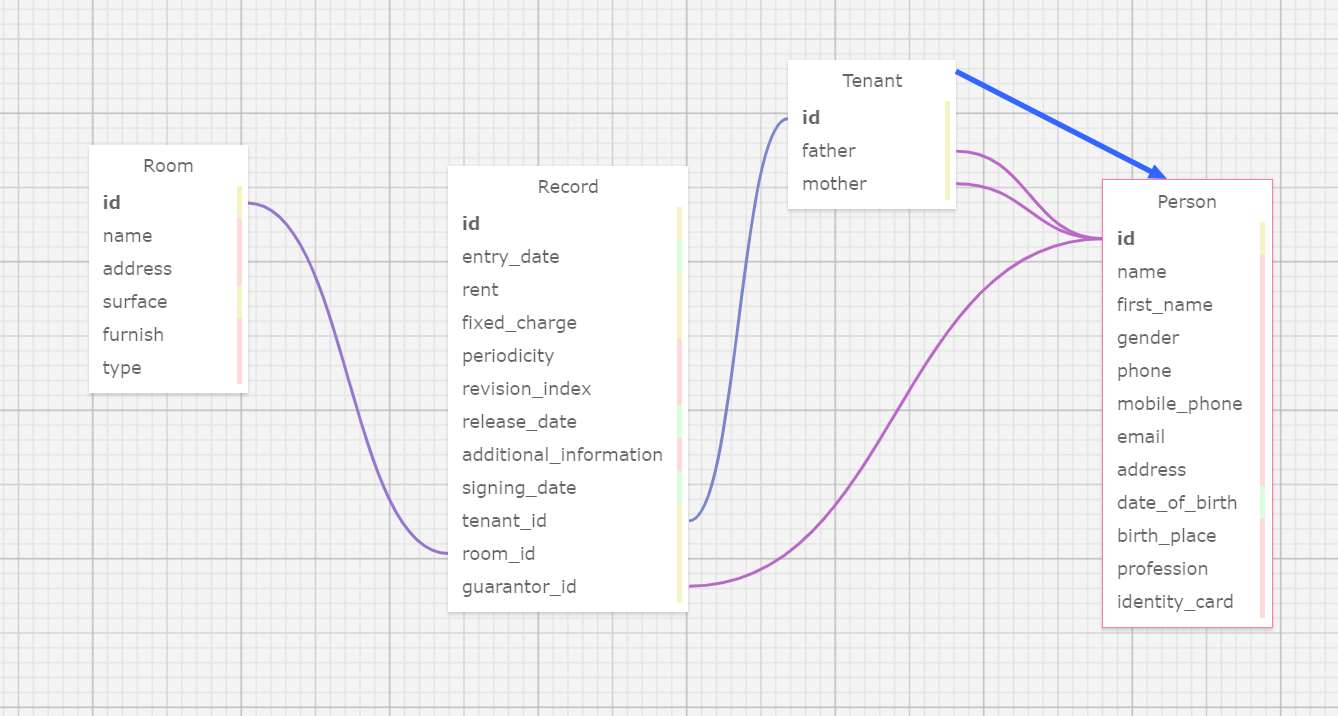
Subséquemment, nous avons dû nous pencher sur la manière qui pourrait nous permettre de stocker et garder ces données de façon structurée. Immanquablement, la création d’une base de données prévue à cet effet a de suite été étudiée. Nous avons alors pris soin d’échanger au cours d’une réunion avec notre 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, malgré quelques modifications légères, jusqu’à la fin du projet.

Figure 3 : 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 afin de les catégoriser. 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 au 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 cet ensemble sont les plus généraux et ne sont en conséquence pas liés à un type de contrat en particulier, ils sont toujours présents et doivent donc 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 du logiciel à proprement parlé, nous avons décidé de nous organiser grâce à *GitHub*[[2]](#endnote-2), service web d'hébergement et de conduite de développement d’applications, utilisant l’outil 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 offert la possibilité de faciliter le déroulement du projet par la mise en commun de nos travaux et par 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 [[4]](#endnote-4)»*.

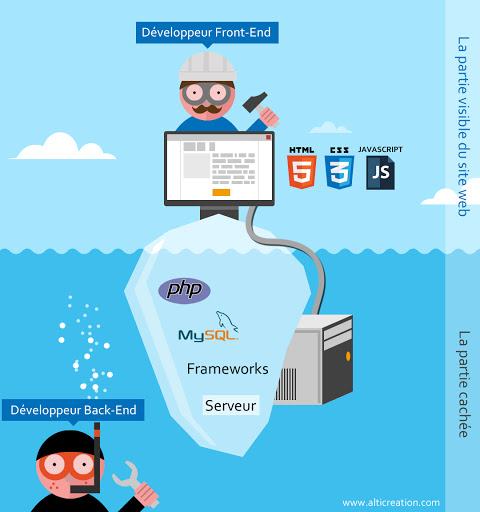
Successivement à la mise en place de notre plan d’organisation, nous avons pris le temps de choisir les outils que nous allions utiliser pour créer la solution. Comme nous l’avons précisé précédemment, nous avions décidé de nous concentrer sur des technologies Web. C’est donc alors que nous avons souhaité mettre en place un cadre de travail avec un serveur HTTP[[5]](#endnote-5) et une base de données en local c’est-à-dire sans hébergement sur internet.

Figure 4 : Logo officiel du logiciel libre Apache HTTP Server créé et maintenu au sein de la fondation Apache.

La préparation d’un environnement n’a pas été tâche ingrate car de nombreux outils libres de droits existent. Nous avons alors émis le souhait d’opter pour les services d’un serveur *Apache* (confère figure 4 ci-dessus) et d’un système de gestion de base de données *MySQL* (confère figure 5 ci-contre). 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 aussitôt sur le code source et d’obtenir ainsi promptement quelque chose de fonctionnel à discuter avec le client.

Figure 5 : MySQL, système de gestion de bases de données relationnelles.

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.

Cela a amplement été facilité par la qualité intrinsèque du développement Web qu’est la séparation nette 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[[6]](#endnote-6) »* tandis que Clément DIF s’est concentré sur la partie frontale du projet plus communément dénommée *« frontend [[7]](#endnote-7)»*.

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 6 : Représentation schématique de la séparation frontend et backend dans un projet 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 un choix de technologie pertinent compte tenu des objectifs du projet. Par ailleurs, il dispose d’un vaste panel de *« frameworks[[8]](#endnote-8) »* aidant à la création de sites Web et très largement utilisés dans le milieu professionnel. Nous avons ainsi 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 ladite 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 Web et donc à posteriori l’affichage du rendu visuel.

Cela nous conduit ainsi à citer les outils employés par le développement frontal. Comme nous l’avons évoqué précédemment, nous souhaitions user des langages *HTML*, *CSS* et *JavaScript* de par leur support par le navigateur. Evidemment, 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 7 ci-contre) offrant des services de manipulation du *DOM* [[9]](#endnote-9)(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 en matière de maintenabilité.

Figure 7 : 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 [[10]](#endnote-10)»* 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 listerons dans cette succincte sous-partie quels sont les objectifs remplis par chaque segment de notre travail mais nous ne nous attarderons pas ni sur les détails du code source ni sur l’implémentation pure. En effet, nous réserverons 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 composant 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 8 : 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 dont il n’a d’ailleurs pas à s’en soucier. C’est au niveau du *backend* que se gère toutes les entités de notre logiciel : les locataires, les locaux, les contrats et les documents.

Pour mémoire, le langage de programmation utilisé est le *PHP,* accompagné du *framework Symfony,* afin de nous faire gagner un temps précieux lors du développement. En effet, ce *framework* permet une gestion simplifiée des formulaires ainsi que la possibilité d’ajouter ou de modifier aisément des éléments en base de données.

Au niveau des données, nous avions premièrement utilisé le système de gestion de base de données *MySQL* comme annoncé précédemment. Puis, nous voulions que notre projet se suffise à lui-même, c’est-à-dire que lorsque nous téléchargions celui-ci, il n’y ait pas besoin d’utiliser d’autres outils ou bibliothèques[[11]](#endnote-11) externes au projet. Ainsi, nous sommes passés 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 géré au niveau de *Symfony* et nous n’en avons donc pas besoin de nous en occuper. Il y a juste une ligne à changer dans les fichiers de configuration.

Concernant l’aspect fonctionnel, nous avions besoin d’un serveur web afin d’exécuter le code *PHP*. Pour ce faire, nous avions commencé par utiliser l’outil *Apache* dans l’optique d’être capables d’offrir rapidement une première ébauche de solution. 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. Suivant les conseils de M. MAZENOD et 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*. Cette manipulation nous a permis de nous passer d’*Apache* et ainsi de supprimer une dépendance de taille au projet. Nous lançons le serveur allégé avec une commande très simple : *php bin/console server:start localhost:8000*

*php fait appel à l’interpréteur PHP*

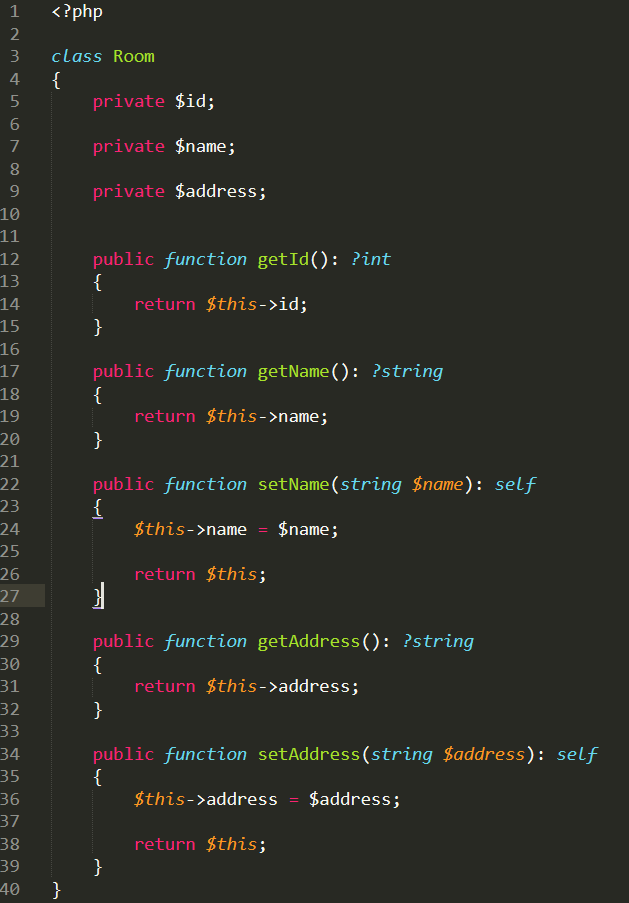
*bin/console est l’endroit où se trouve le gestionnaire de Symfony*

*server:start lance le serveur*

*localhost:8000 informe que l’on lance le server en local sur le port d’écoute 8000*

Une fois le serveur web lancé et la base de données configurée, nous pouvions 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 cadriciel, tout est objet, même les éléments en base de données dénommés entités.

Voici, en figure 9, un exemple d’entité utilisée pour les appartements :

On peut ainsi manipuler des objets « Room » qui ont un identifiant, 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.

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, comme : *localhost:8000/index.php/room*. Cette url permet au demeurant d’afficher tous les locaux qui sont sauvegardés dans le logiciel et va appeler le contrôleur *RoomController.php* dont voici un exemple en page suivante :

Figure 9 : Exemple d'entité

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

Figure 10 : Exemple de contrôleur

Du coup, la fonction *index* qui permet de lister les locaux, est appelée si on renseigne la route */room* avec la méthode HTTP[[12]](#endnote-12) *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 au client c’est-à-dire à l’interface de récupérer une page, alors que la méthode *POST* permet d’envoyer des données au serveur, par exemple des informations renseignées dans un formulaire d’ajout d’un local.

La fonction *add* offre la possibilité 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*, cela veut dire que l’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 (Lignes 35 et 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. Egalement, la gestion des locataires et des contrats demeure très semblable.

Ensuite, les lignes 38 à 42 permettent la gestion des documents. Dans notre exemple, elles offrent la possibilité de créer un dossier relatif au local que l’on vient de renseigner.

Finalement, une fois le traitement attendu effectué au niveau du contrôleur, il faut 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).

Pour rappel, notre logiciel permet de gérer de manière précise et structurée les documents relatifs aux locataires, locaux et contrats. C’est notamment ce que nous avons expliqué il y a quelques instants. Nous avons aussi besoin de remplir des documents génériques ou *templates* automatiquement. Pour ce faire, nous avions d’abord pensé utiliser des fichiers *PDF* remplissables. Cependant, cette solution nous contraignait à devoir acheter une licence *Adobe* afin de pouvoir les remplir. Nous avons donc cherché d’autres solutions, et nous avons décidé d’utiliser plutôt des fichiers *Word (.docx)*. Ceux-ci sont en effet remplissables directement par le code *PHP* et les modules *Symfony*.

Voici, en lieu et place de rappel, les différents types de documents et modèles utilisés et la manière avec laquelle ils sont traités au sein du *backend* :

* Les documents modèles : ce sont des documents qui diffèrent d’un local à un autre, qui sont spécifiques à un type particulier de location ou qui sont relatifs à toute sorte de contrats. 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 patron 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 directement accessibles à partir du contrat auquel ils sont rattachés. Pour ce faire, nous utilisons une bibliothèque *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 pièces relatives à un locataire : ce sont tous les documents obligatoires ou facultatifs concernant un locataire en particulier comme, à titre d’exemple, sa carte d’identité. 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. De plus, une fonctionnalité offre l’éventualité de récupérer tous les fichiers d’un locataire sous forme d’archive.

Les documents complétés ainsi que les *templates* sont stockés selon une arborescence de dossiers bien précise comme définie ci-dessous :

* users Fichiers utilisateur (*templates*, documents générés, etc.)
* default Utilisateur par défaut
* generated Dossier des documents générés automatiquement
  + - record1 Documents pour le contrat numéro 1
    - record2 Documents pour le contrat numéro 2
    - …
* templates Dossier contenant les *templates* à générer
  + - general Contient les *templates* généraux
    - typed Contient les *templates* typés
    - apartments Contient les *templates* limités aux appartements
    - hangars Contient les *templates* limités aux hangars
    - unique Contient les *templates* uniques à un local
    - room1 Contient les patrons de documents relatifs au local numéro 1
    - room2 Contient les patrons de documents relatifs au local numéro 2
    - …
    - tenants Dossier contenant les pièces justificatives des locataires
    - tenant1 Contient les pièces relatives au locataire numéro 1
    - tenant2 Contient les pièces relatives au locataire numéro 2
    - …

Figure 11 : Organisation hiérarchique des documents de l’utilisateur.

#### 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 de balisage pour créer la structure et l’architecture de la vue utilisateur, 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 12 : Extrait 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 pour 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 12, 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 13). L’on peut ici remarquer que le responsable de l’interface a placé librement les divers champs incluant leur *label* et leur *widget* en leur associant par la même occasion des classes qui sont définies dans des fichiers de style séparés.

Figure 13 : 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 suscite 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*[[13]](#endnote-13). 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 14 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* interprétée 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 14 : Exemple de code JavaScript d’une requête AJAX.

Il est par ailleurs intéressant de noter qu’un *header*[[14]](#endnote-14)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.

Nous venons d’exposer sans réserve l’ensemble des qualités techniques de notre projet et les détails d’implémentation du logiciel. Nous pouvons donc désormais aborder le manuel d’utilisation et comprendre comment et dans quelle mesure le logiciel peut être employé.

## Résultats et perspectives

### Diagramme de Gantt réalisé

En page suivante, apparaît le diagramme de Gantt représentant l’ensemble des tâches réellement effectuées. Il diffère de manière remarquable avec le diagramme prévisionnel car certains objectifs ont été revus au cours de la réalisation de la solution. En effet, lors des différentes entrevues avec le client, nous avons régulièrement eu l’occasion de dresser la liste des modifications à apporter ou des fonctionnalités qui ne convenaient pas tout à fait à cet utilisateur final.

Nous avons choisi de réaliser ce diagramme autrement que par l’intermédiaire du logiciel *Gantt Project* car celui-ci ne permettait pas de représenter le déroulement chronologique comme nous l’entendions.

Le graphique se décompose en plusieurs catégories de tâches et l’avancement de ces dernières est représenté par une couleur donnée en fonction du développeur les ayant réalisées.

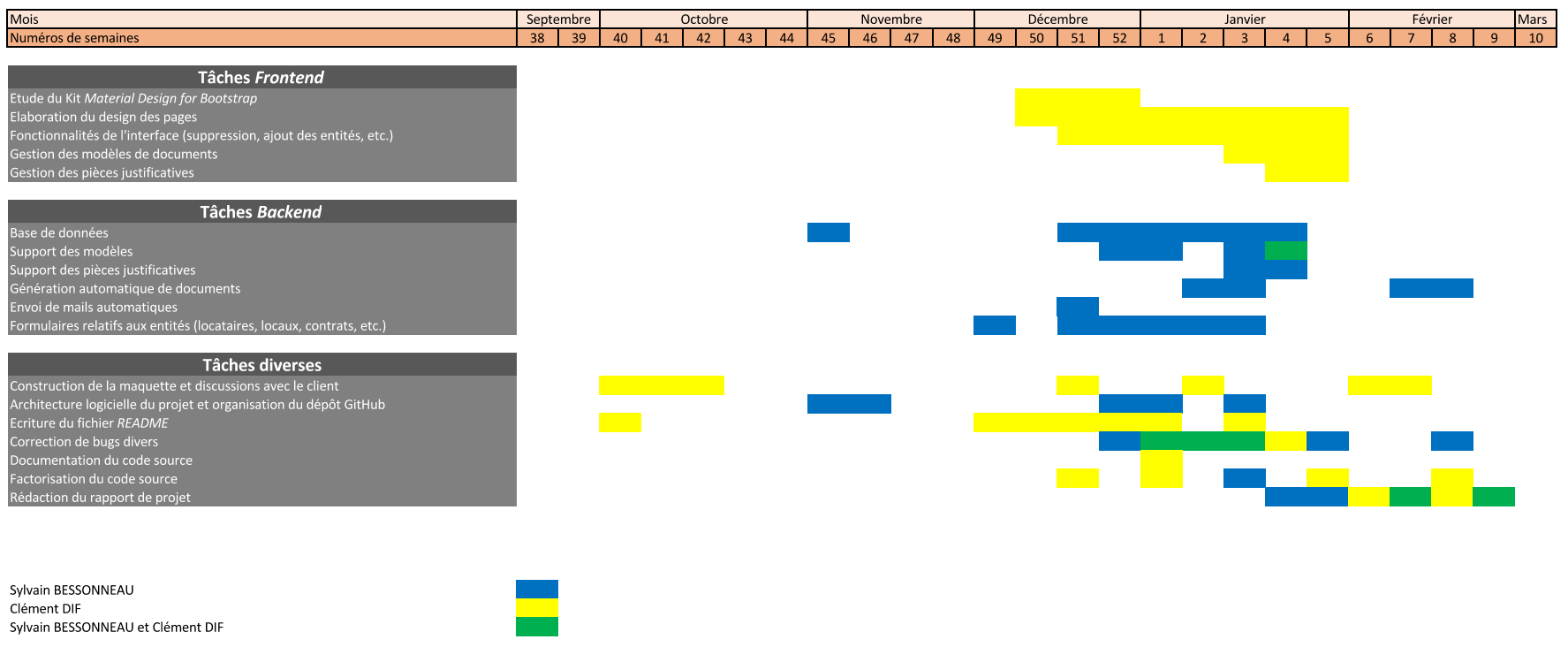


Figure 15 : Diagramme de Gantt réalisé

### Utilisation de la solution

#### Commentaires préalables

Comme expliqué dans la partie relative au *backend*, nous avons opté pour une solution *« standalone »* se suffisant à elle-même. Néanmoins, il est toujours possible d’utiliser un serveur *Apache* et une base de données *MySQL*, la compatibilité avec nos outils de départ est donc conservée. Le fichier *« readme »* du dépôt *GitHub* indique d’ailleurs comment installer un environnement de développement mais cette méthode reste désuète et nous conseillons plutôt de se référer aux indications d’exécution données dans rapport au sein de la partie relative au *backend*. Le fichier *« readme »* sera bientôt adapté pour proposer des directives d’installation en mode *« standalone »*.

Tout au long de cette partie, nous étudierons un cas pratique en nous identifiant à un nouvel utilisateur. Nous commencerons ainsi par découvrir l’interface et les possibilités du logiciel dans sa globalité puis nous insérerons un local et un document associé à générer. Ensuite, nous procéderons à l’ajout d’un locataire. Puis, dernièrement, nous créerons un contrat relatif aux entités que nous venons à l’instant de mentionner.

#### Page d’accueil et prise en main

Pour accéder au logiciel, il suffit de démarrer le serveur et d’ouvrir un navigateur puis d’entrer l’URL suivante : *http://localhost:8000/index.php*

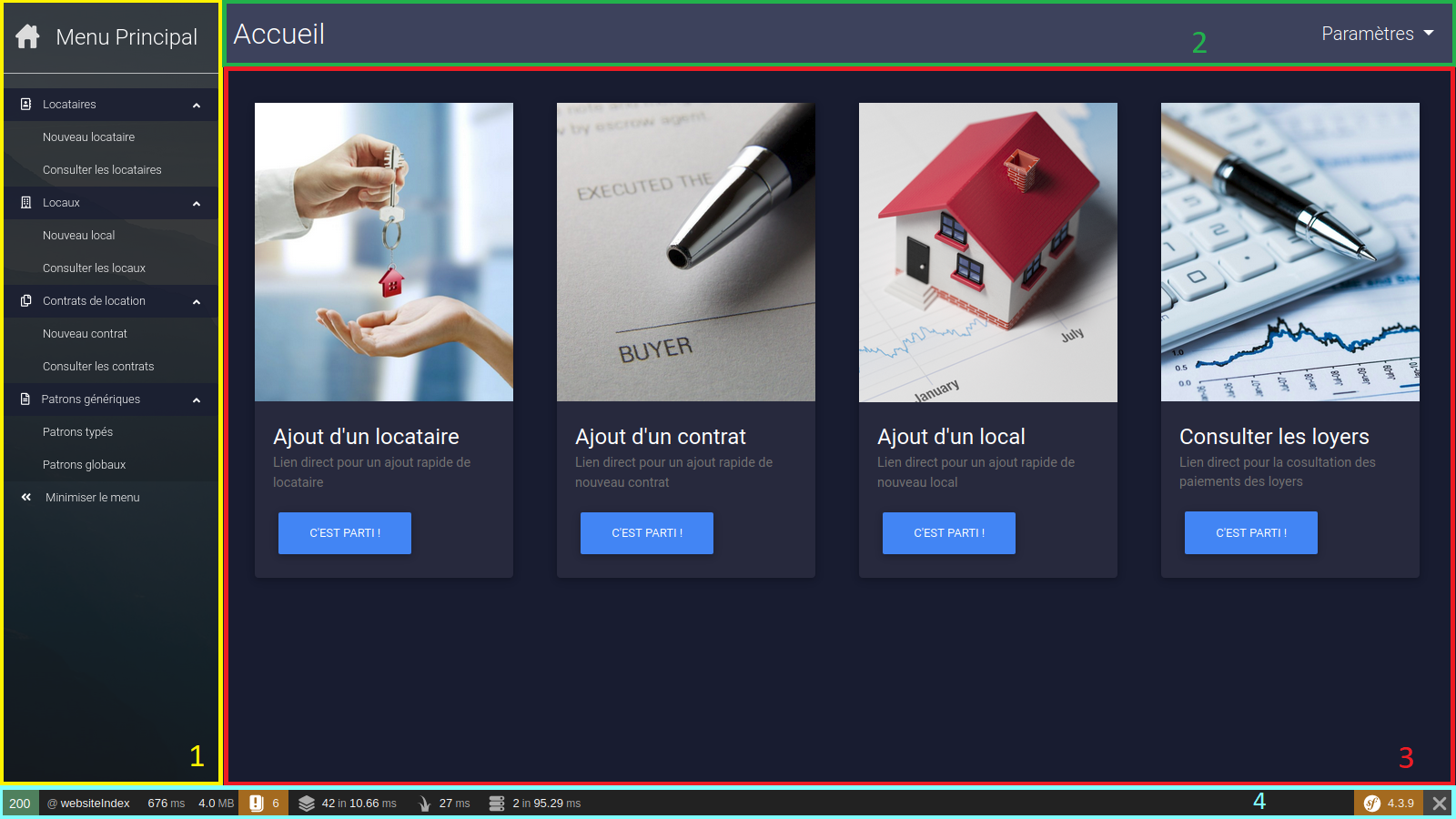
L’on arrive alors sur la page d’accueil de notre projet, baptisé REAM *(Real Estate Asset Manager),* que l’on retrouve d’ailleurs ci-dessous en figure 16.

Figure 16 : Aperçu de la page d'accueil du logiciel

Nous pouvons remarquer la ressemblance avec la maquette initiale et la proposition d’un accès rapide aux différentes fonctionnalités. Sont mises ici en surbrillance quatre zones au sein de l’interface ; chacune accomplissant un objectif précis :

* La zone numéro 1 en jaune est le menu principal de l’application. Il permet de naviguer entre les différentes pages du logiciel et est constitué de plusieurs sous-menus offrant la gestion des différentes entités comme les locataires, les locaux ou encore les contrats.
* La section numéro 2, encadrée en vert, représente l’activité en cours et indique à l’utilisateur où il se situe. Il contient également un lien vers les paramètres, non encore fonctionnel au moment de l’écriture du rapport.
* La partie numéro 3 de couleur rouge, nommée « zone d’interactions », comprend ici des raccourcis vers les principales fonctionnalités du logiciel. Dans d’autres pages, elle sera amenée à contenir les différents champs des formulaires ou les divers boutons d’interactions avec les documents utilisateur.
* Le pied de page en bleu, portant le numéro 4, rassemble des outils fournis par le *framework Symfony* et offre des opportunités de débogage. Evidemment, ce service n’est présent qu’en environnement de développement et ne sera pas déployé en production.

#### Premiers pas et ajout d’un local

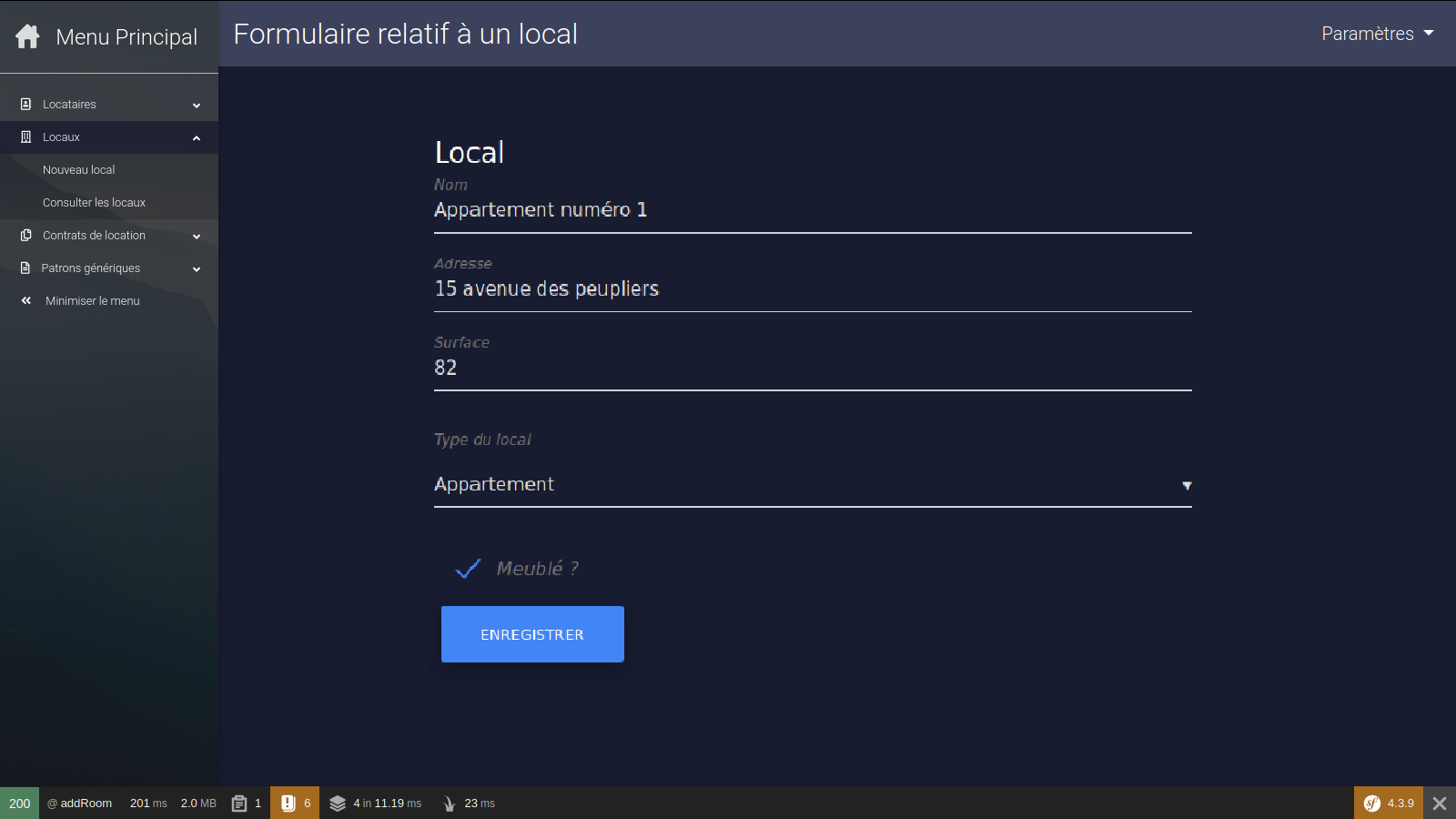
L’utilisation de notre solution commence en premier lieu par l’ajout d’un local c’est-à-dire un appartement ou un hangar. Ci-dessous est présenté en figure 17 un exemple de création de local via l’interface proposée par le logiciel.

Figure 17 : Aperçu du formulaire d'ajout d'un local

Les différents champs peuvent être complétés par l’utilisateur et le local est créé dès lors que l’on presse le bouton « enregistrer ».

#### Création des patrons de documents d’un local

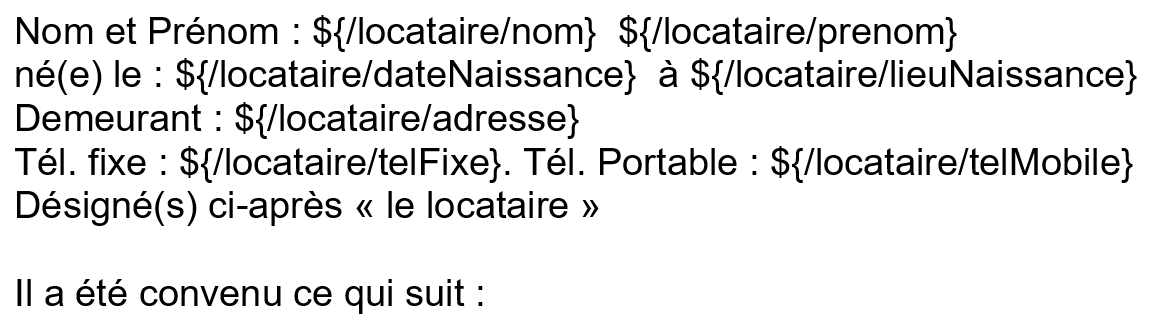
Nous avons choisi pour cet exemple d’associer au local que nous venons de créer un document spécifique qui sera généré a posteriori par l’application. Ce document doit être préparé par l’utilisateur à l’aide de balises prédéfinies. Ci-dessous est d’ailleurs affiché un extrait d’exemple de modèle.

Figure 18 : Extrait de modèle de document.

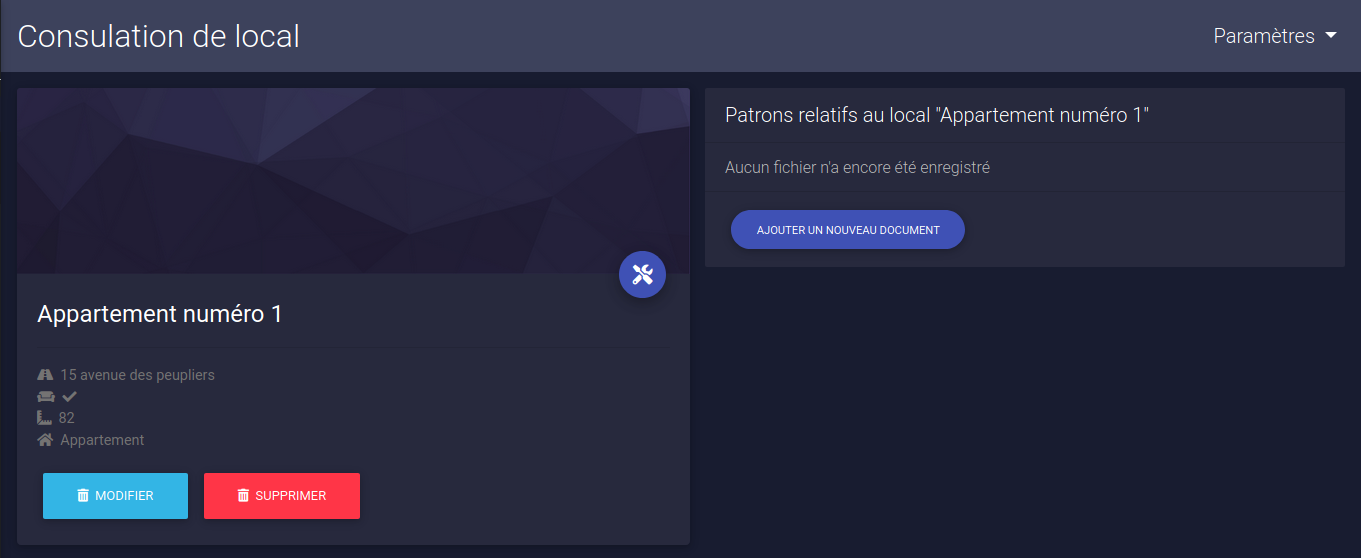
 Sur la page correspondant au local, présente en figure 19, il est possible d’ouvrir une fenêtre contextuelle pour l’importation d’un patron. Dans ce but, il suffit d’appuyer sur le bouton « ajouter un nouveau document » sur la droite de l’écran.

Figure 19 : Interface de gestion d'un local.

#### Renseignement d’un locataire

Suivant notre cas pratique, il est maintenant obligatoire d’indiquer quel client s’apprête à louer le bien. L’on peut directement effectuer cette opération depuis la page prévue à cet effet. Le formulaire de création est plus long que celui relatif à l’ajout d’un appartement et est exposé en page suivante, en figure 20.

Grace à notre interface, il est très facile de créer un locataire. L’on peut par ailleurs renseigner son père, sa mère ou les deux comme exigé par le client final du logiciel. Ceux-ci sont ainsi sauvegardés en base de données lors de la soumission du formulaire.

Au cours de l’ajout d’un locataire, plusieurs champs sont à renseigner comme notamment le nom et le prénom ou encore l’adresse parentale puisque les locataires sont des étudiants et donc souvent mineurs. Tous ces champs sont obligatoires et nécessaires à la création d’un contrat par la suite.

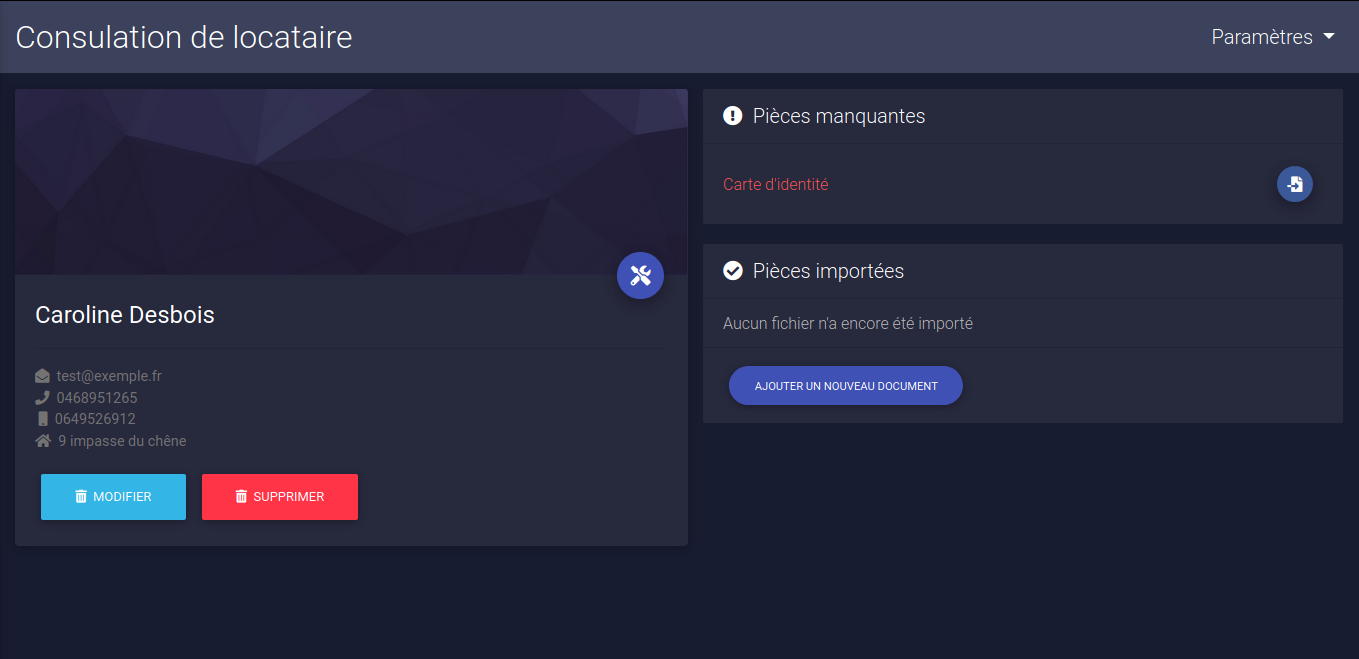


Figure 20 : Formulaire d'ajout d'un nouveau locataire.

Figure 21 : Exemple de consultation d'un locataire.

Sur la page de consultation d’un locataire, en figure 21, il est possible de rajouter des pièces justificatives en lien avec ce dernier. Par exemple, l’on peut spécifier sa carte d’identité. Celle-ci fait d’ailleurs parti des documents obligatoires que doit posséder l’utilisateur du logiciel mais ces documents peuvent être rajoutés plus tard. A titre d’exemple, si au moment de la création d’un locataire, ce dernier n’est pas en possession de sa carte d’identité, l’utilisateur peut tout de même continuer jusqu’à la création d’un contrat. L’interface d’ajout d’une pièce justificative étant très similaire à celle responsable de l’insertion d’un patron de document, l’apport de la pièce peut être effectué ultérieurement. D’autre part, en conséquence de cette similitude, nous avons choisi de ne pas représenter cette interface au sein de notre mémoire pour éviter toute redondance.

#### Elaboration d’un contrat

Postérieurement au renseignement du local et du locataire, il est possible de créer un contrat. Pour ce faire, nous proposons un formulaire dans lequel le logiciel va demander de remplir certains champs en suivant le même principe que pour les deux autres entités.

La section offrant la capacité de choisir le locataire est en fait une liste de sélection parmi ceux renseignés en base de données locale. Il suffit alors de choisir celui à qui l’utilisateur veut attribuer le contrat. L’on retrouve le même principe pour la sélection du local.

Sur demande du client de la solution, un contrat doit obligatoirement mentionner une personne se portant caution. Celle-ci peut être le père ou la mère du locataire ou encore une tierce personne. Si l’on est dans le dernier cas, il faut remplir les champs relatifs à la caution identiques à ceux pour l’ajout du père ou de la mère lors de la création d’un locataire.

Si l’on s’affranchit de la création d’une caution et que l’on sélectionne l’un des deux parents, une vérification est effectuée lors de la soumission du formulaire. Cela permet de pallier le cas où, lors de la création du locataire, l’on aurait renseigné le père et que l’on souhaiterait intégrer la mère comme responsable de la caution. Dans une telle situation, l’on pourrait soit changer la personne se portant garante, soit modifier le locataire afin de renseigner le deuxième parent.

Comme pour les locataires et les locaux, il est tout à fait possible de visualiser la liste des contrats créés et de les supprimer. Il n’est par contre pas possible de les modifier.

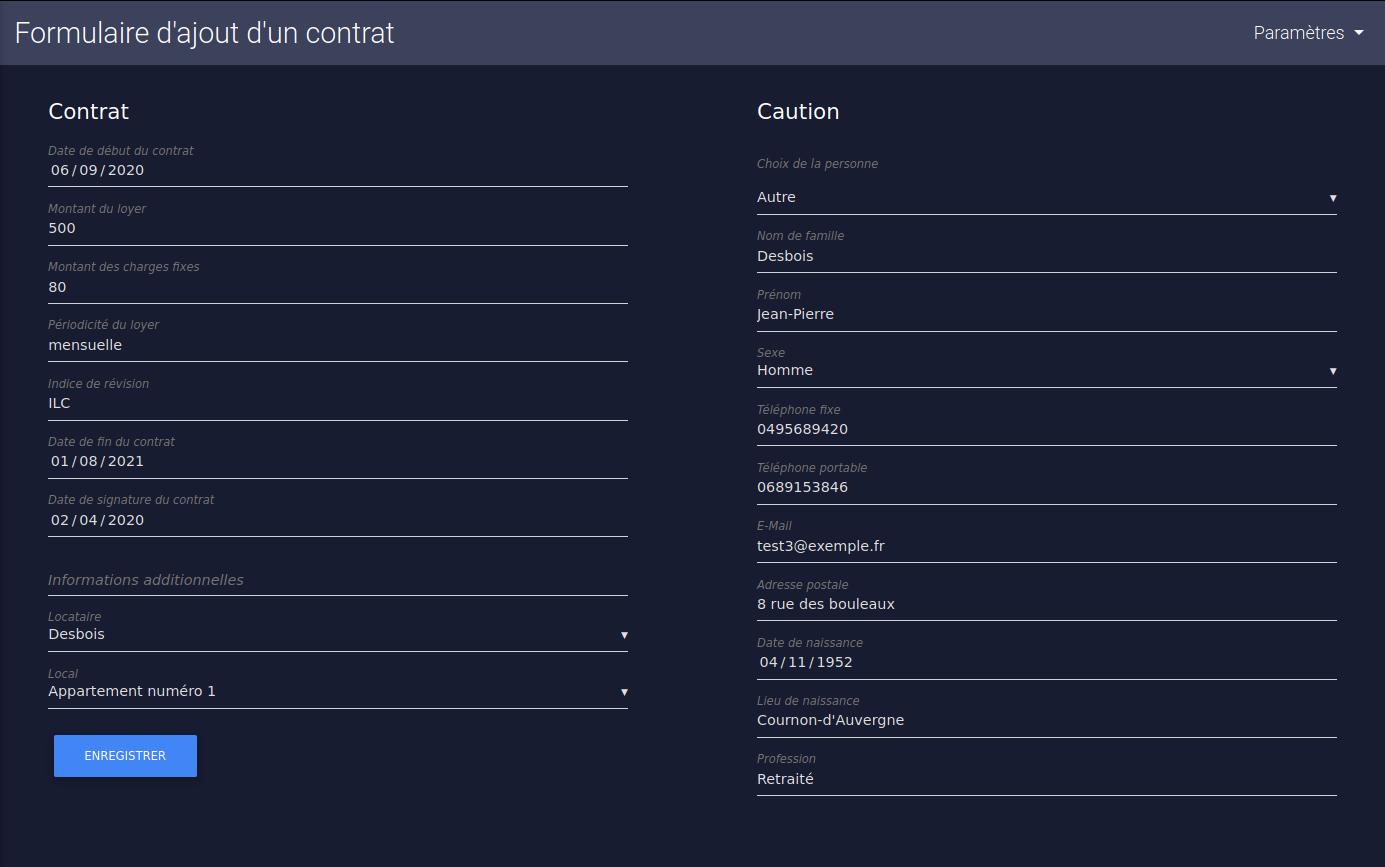
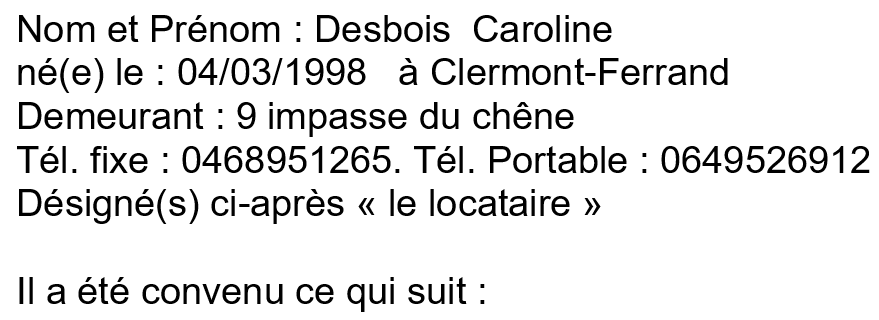
Est représenté en figure 22 ci-dessus, le formulaire de création d’un nouveau contrat. Suite à l’envoi de ce dernier, les patrons de documents sont complétés automatiquement grâce aux informations fournies par l’ensemble des formulaires et présentes en base de données. L’utilisateur peut alors les récupérer et les placer sur sa machine. Les documents ainsi créés sont actuellement au format *PDF* pour faciliter leur impression. Ci-dessous est représenté le même extrait de document qu’en figure 18 mais après génération.

Figure 22 : Formulaire d'ajout d'un contrat

Figure 23 : Extrait de document généré.

#### Gestion des documents typés et généraux

Nous avons examiné précédemment comment intégrer à l’application un modèle de document relatif à un local. Cependant, comme nous l’avons précisé dans la conception approfondie de la solution, les patrons sont classés en trois catégories. La fonctionnalité offrant la possibilité de saisir les modèles relatifs à un type de local ou ceux concernant l’ensemble des contrats est encore en cours de développement, c’est pourquoi aucune illustration n’est possible pour cette sous-partie.

### Perspectives d’amélioration

Notre logiciel est actuellement fonctionnel et correspond à la grande majorité des attentes du client. Cependant, des éléments seront encore ajoutés ou améliorés dans un futur proche 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 insérer des champs et des valeurs et notre logiciel s’occuperait de remplir automatiquement un document listant les biens se trouvant dans l’appartement en question comme les meubles, la vaisselle, etc.

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

Par ailleurs, un lien persiste entre le *backend* et le *frontend*. En effet, le *backend* envoie des objets *PHP* au *frontend*, ce qui signifie que ce dernier doit toujours tenir à jour une liste de ce que son alter ego lui transmet afin de recevoir et traiter les données correctement. Nous pourrions limiter cela en utilisant, par exemple, le générateur d’applications *Jhipster*. Il s’agit d’un outil développé par *Microsoft* qui permet de générer automatiquement un *backend* et un *frontend* fonctionnels en utilisant le fonctionnement des API REST[[15]](#endnote-15). Cela offre ainsi l’atout de pouvoir séparer distinctement le *backend* du *frontend*. De plus, Jhipster est très adapté pour ce genre de projet qui utilise des formulaires et des entités et gère en outre l’intégration et le déploiement continu.

Enfin, notre logiciel s’utilise actuellement avec un navigateur web. Cela pourrait donc être une bonne éventualité de perfectionnement que d’encapsuler notre logiciel dans un *framework* permettant de générer une application *desktop*. Le cadre de travail *Electron*, conseillé par M. USSON, l’un de nos enseignants, convient parfaitement à ce besoin. Il est facile d’utilisation et s’adapte parfaitement aux projets utilisant *Symfony*. C’est pourquoi nous pensons utiliser cet outil dans les mois prochains afin de compléter 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 solutions *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 offre la gestion des locataires, des locaux et des contrats. Les documents relatifs à chaque élément sont stockés de manière structurée. Les *templates* se remplissent automatiquement à la création d’un contrat. L’utilisateur peut donc utiliser le logiciel en l’état. Cependant, des perspectives d’amélioration sont envisagées, notamment l’ajout d’envois de mails automatiques lors de la création d’un contrat ou encore la gestion d’un inventaire propre à chaque local.

# Résumé

Le but de ce projet était de réaliser un logiciel de gestion de locataires, de locaux et de contrats de locations. Dans notre application, l’utilisateur peut renseigner de nouveaux locataires, ajouter des locaux et 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 *HTML*, *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 environnements de développement intégrés *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, AJAX, jQuery, MDB, SQLite.

# Abstract

This project aimed to develop a software for the management of tenants, rooms and rental contracts. In our application, the user can inform new tenants, add rooms and associate tenants to a room by creating new contracts. The software also allows the management of all the documents related to the different contracts, tenants and premises. This application is a web application, usable within a web browser. It was developed with *PHP* and the *Symfony* *framework* on the *backend* side and *HTML*, *CSS* and JavaScript with the *Material Design for Bootstrap framework* on the *frontend* side. The software works in offline mode since the web server and the database are located locally.

The development was done with the integrated development environments *Visual Studio Code* and *Sublime Text*. For the webserver, we use the server integrated into *Symfony* and an *SQLite* database.

Currently, the software is running perfectly and can already be used by the client.

Keywords: Web, PHP, Symfony, HTML, CSS, JavaScript, AJAX, jQuery, MDB, SQLite.

# Glossaire

**1 World Wide Web :** Système hypertexte public fonctionnant sur Internet. Le Web permet de consulter, avec un navigateur, des pages accessibles sur des sites

**2 GitHub :** Service web d'hébergement et de gestion de développement de logiciels, utilisant le logiciel de gestion de versions Git.

**3 Git :** Logiciel de gestion de versions décentralisé. C'est un logiciel libre créé par Linus Torvalds, auteur du noyau Linux, et distribué selon les termes de la licence publique générale GNU version 2.

**4 Open Source :** La désignation Open Source, s'applique aux logiciels dont les possibilités de libre redistribution, d'accès au code source et de création de travaux dérivés sont autorisées.

**5 Serveur HTTP :** Serveur informatique répondant à des requêtes du World Wide Web, en utilisant principalement le protocole de communication client-serveur HTTP.

**6 Backend :** Représente la partie interne d’un projet web et comprend souvent la base de données et le code métier s’y appliquant.

**7 Frontend :** Inclut l’ensemble des éléments et composants affichés sur une interface avec lesquels l’utilisateur peut interagir.

**8 Framework :** En programmation informatique, un *framework* (appelé aussi cadre de travail, cadre d’applications ou encore cadriciel) désigne un ensemble cohérent de composants logiciels structurels, qui sert à créer les fondations ainsi que les grandes lignes de tout ou d’une partie d'une application informatique.

**9 Document Object Model :** Interface de programmation normalisée par le W3C, qui permet à des scripts d'examiner et de modifier le contenu du navigateur web. Par le DOM, la composition d'un document HTML est représentée sous forme d'un jeu d'objets reliés selon une structure en arbre.

**10 Asset :** Un asset est défini comme une ressource basique telle une image ou un son et pouvant être affichée dans un navigateur web.

**11 Bibliothèque :** En informatique, une bibliothèque logicielle est une collection de routines, qui peuvent être déjà compilées et prêtes à être utilisées par des programmes.

**12 Méthode HTTP** **:** Dans le protocole HTTP, une méthode est une commande spécifiant un type de requête, c'est-à-dire qu'elle demande au serveur d'effectuer une action. En général l'action concerne une ressource identifiée par l'URL qui suit le nom de la méthode.

**13 AJAX :** Architecture informatique qui permet de construire des applications Web et des sites web dynamiques interactifs sur le poste client en offrant des possibilités de dialogue asynchrone avec le serveur.

**14 Header HTTP :** Permettent au client et au serveur de transmettre des informations supplémentaires avec la requête ou la réponse du protocole HTTP.

**15 API REST :** Interface de programmation normalisée entre deux entités. Les API REST sont souvent utilisées dans des modèles fournisseur-consommateur.

Remarque : certaines définitions sont inspirées de l’encyclopédie libre Wikipédia.

# Références

### Bibliographie

PARKER, Daniel. *JavaScript with Promises: Managing Asynchronous Code*. " O'Reilly Media, Inc.", 2015 : 77-78

POREBSKI, Bartosz, PRZYSTALSKI, Karol, et NOWAK, Leszek. *Building PHP Applications with Symfony, CakePHP, and Zend Framework*. John Wiley and Sons, 2011.

VAN LANCKER, Luc. *JQuery: le framework JavaScript du Web 2.0*. Editions ENI, 2011 : 437-480

WOYCHOWSKY, Edmond et WOYCHOWSKY, Edmond. *AJAX: Creating web pages with asynchronous JavaScript and XML*. Prentice Hall, 2006.

### Webographie

SZYMANSKI, Michal. *Bootstrap 4 tutorial – best & free guide of responsive web design*, date de consultation : 2 janvier 2020.

Disponible à l’adresse : <https://mdbootstrap.com/education/bootstrap>

Auteur(s) non mentionné(s). Symfony Documentation, date de consultation : 24 novembre 2020.

Disponible à l’adresse : <https://symfony.com/doc/>

Auteur(s) non mentionné(s). *Wikipédia, l'encyclopédie libre*, date de consultation : 15 février 2020.

Disponible à l’adresse : <https://fr.wikipedia.org>

Auteur(s) non mentionné(s). *Ajax | jQuery API Documentation*, date de consultation : 17 janvier 2020.

Disponible à l’adresse : <https://api.jquery.com/jquery.ajax/>

Auteur(s) non mentionné(s). *CSS reference - CSS: Cascading Style Sheets*, date de mise à jour : 26 octobre 2019.

Disponible à l’adresse : <https://developer.mozilla.org/en-US/docs/Web/CSS/Reference>

1. World Wide Web [↑](#endnote-ref-1)
2. GitHub [↑](#endnote-ref-2)
3. Git [↑](#endnote-ref-3)
4. Open source [↑](#endnote-ref-4)
5. Serveur HTTP [↑](#endnote-ref-5)
6. Backend [↑](#endnote-ref-6)
7. Frontend [↑](#endnote-ref-7)
8. Framework [↑](#endnote-ref-8)
9. Document Object Model [↑](#endnote-ref-9)
10. Asset [↑](#endnote-ref-10)
11. Bibliothèque [↑](#endnote-ref-11)
12. Méthode HTTP [↑](#endnote-ref-12)
13. AJAX [↑](#endnote-ref-13)
14. Header HTTP [↑](#endnote-ref-14)
15. API REST [↑](#endnote-ref-15)