



Application mobile

Et son écosystème

Julien Bertacco - 2018



Introduction

Dans le cadre de ma formation Concepteur Développeur Informatique, j'ai effectué mes deux années d'alternance au sein de l'entreprise SDGI Bretagne.

Présentation

J'ai eu l'occasion de travailler sur plusieurs projets avec différentes tailles d'équipe :

- Projet Module d'export tiers, développé en JAVA. Travail en autonomie mais en liaison avec la personne chargée du développement. Le module consiste à envoyer sur une API les données liés à des lots vacants pour les publier sur des sites tel que leboncoin, seloger etc.
- Projet « AltImmo », application sous VUEjs et Laravel. Travail en équipe avec la personne chargée du projet. Il s'agit ici d'un back office qui permet la gestion complète du SEO pour un site de location saisonnière.
- Réalisation de divers sites internet en lien avec l'immobilier. Travail en liaison direct avec le client et parfois avec des graphistes.
- Rédaction de diverses documentations en lien avec les projets précédent.
- Divers utilitaires pour Gestion Locative. Travail en autonomie, en liaison avec la personne chargée du développement. Un exemple d'utilitaire : L'ajout de champs dans la fiche de description d'un lot pour effectuer l'export tiers.

Ce rapport a pour but de présenter le projet « Etat des lieux », les différentes étapes de sa conception jusqu'à la mise en production. L'application évolue de jour en jour, travaillant actuellement dessus au moment où je rédige mon rapport.

Table des matières

Introduction.....	2
Remerciements	6
Abstract	6
Liste des compétences du référentiel couvertes par le projet	7
Présentation de l'entreprise.....	8
L'entreprise	8
Activité.....	8
Organigramme de l'entreprise :	9
Cahier des charges.....	10
Contexte du projet	10
Analyse de l'existant.....	10
Problématique	11
Objectifs du projet.....	12
Etude de faisabilité	13
Vue d'ensemble du projet.....	14
Livrables.....	15
Loi A.I.U.R – Décret n°2016-382 du 30 mars 2016	15
Gestion du projet.....	17
L'équipe dédiée au projet	17
Méthodologie	17
Estimation de la charge	17
Planning.....	18
Eviter l'effet tunnel.....	19
Présentation régulière.....	19
Spécifications fonctionnelles.....	20
Diagramme de cas d'utilisation	20
Diagramme du cas d'utilisation « Synchronisation des données ».....	22
Maquettage de l'application	23
Liste des états des lieux.....	23
Récapitulatif avant le débuter d'un état des lieux	24
Passage d'un état des lieux – Liste des pièces	25
Passage d'un état des lieux – Élément d'une pièce	26
Gestion des modèles	27
Paramètres de l'application	27

Page d'assistance.....	28
Spécifications techniques.....	29
Technologies utilisées sur le projet.....	29
JavaScript.....	29
TypeScript.....	29
Ionic.....	30
Laravel.....	31
INSOMNIA.....	32
IndexedDB.....	33
Java.....	33
H2.....	34
Réalisation.....	35
Préambule.....	35
Module d'export des données.....	35
Les requêtes.....	36
Les modèles d'objet « Adaptateur ».....	36
L'IHM.....	37
API.....	38
Préambule.....	38
Conception de la base de données.....	38
Mise en place de la base de données.....	38
Fichier de migration pour la table lot :.....	39
Fournir des données au déploiement de l'API sur un serveur.....	40
Routes.....	40
Les classes d'objets métier.....	41
Application cross platform Etat des lieux.....	43
Design pattern MVVM.....	43
Base de données.....	45
Couche d'accès aux données.....	45
Optimisation et recherche de performance.....	47
Les classes d'objets métier « Model ».....	50
Les classes « ViewModel ».....	51
Les classes d'accès aux données.....	51
Les vues.....	52
Génération d'un PDF en JavaScript.....	54

Création des jeux de test.....	55
Les Tests	56
Tests unitaires	56
Tests fonctionnels	56
Fiche de tests.....	57
Maintenabilité	58
Déploiement.....	59
Affichage des messages utilisateurs.....	59
Gestions des logs.....	59
Documentation.....	61
Documentation technique	61
Conclusion	61
Annexes	62
1. Etude préalable	62
2. Documentation technique	62
3. Maquettage de l'application	62
4. Méthodes de l'utilitaire pour le module Export Tier.....	62
5. MCD	62
6. Exemple d'état des lieux au format PDF	62

Remerciements

Je tiens tout particulièrement à remercier M. Dominique Barbier pour son accueil chaleureux et pour m'avoir accepté en tant qu'alternant.

Je remercie également M. Renaud Dugarin mon tuteur pour le soutien technique et les conseils tout au long des projets. M. Guillaume Rues pour les conseils et l'aide technique pour le développement sur le logiciel Gestion Locative. Ainsi que M. Simon Jouault, collègue d'alternance.

D'une façon plus générale, je remercie l'équipe de SDGIB, pour l'intérêt qu'ils m'ont porté tout au long de mon alternance, ainsi que pour leur aide et précision.

Je tiens aussi à remercier l'ensemble des formateurs et personnels de l'administration de l'ENI Ecole Informatique.

Abstract

Facing the current state of the real estate market, the SDGIB company entrusted me with the task of designing and implementing a mobile application. The goal of this application is to enhance the company's flagship software: « Gestion Locative ». This software provides the complete management of a real estate agency and the accounting related to this activity. The mobile application must cover the inventory of fixtures management linked to the tenants of an agency. To do that, the application must be able to collect and process data from the « Gestion Locative » software. This application is named: « Etat des lieux ». To achieve this, it will be necessary to integrate a data export module and to design and build a server-side application that can collect and render data. The application will have to be able to facilitate the making of an inventory of fixtures. While being in compliance with the Alur law of the decree n°2016-382 of March 30th, 2016.

Liste des compétences du référentiel couvertes par le projet

Maquetter une application	✓
Développer une interface utilisateur	✓
Développer des composants d'accès aux données	✓
Développer des pages web en lien avec une base de données	✓
Concevoir une base de données	✓
Mettre en place une base de données	✓
Développer des composants dans le langage d'une base de données	✓
Utiliser l'anglais dans son activité professionnelle en informatique	✓
Concevoir une application	✓
Collaborer à la gestion d'un projet informatique	✓
Développer des composants métier	✓
Construire une application organisée en couches	✓
Développer une application de mobilité numérique	✓
Préparer et exécuter les plans de tests d'une application	✓
Préparer et exécuter le déploiement d'une application	✓

Présentation de l'entreprise

L'entreprise

L'entreprise SDGI Bretagne est une société éditrice de logiciels composée de 7 personnes.

Basée à Saint-Brice-en-Coglès, en Ille et Vilaine, cette entreprise familiale possédant un fort savoir-faire dans le domaine de l'immobilier, est née de la cession d'une ancienne entité nommée SDGI au groupe SEPTEO au cours de l'année 2012. Une entité fut alors créée dans le but de gérer les régions Normandie et Bretagne.

L'entreprise SDGI Bretagne commercialisait alors les logiciels de SDGI. Par la suite, il fut décidé de reconstruire une gamme de logiciels propres à SDGI Bretagne.

L'entreprise commercialise aujourd'hui un logiciel de Gestion Locative, un logiciel de location saisonnière, des services Web comme la gestion d'annonces immobilières intégrées sur différentes plateformes connues (Le bon Coin, seLoger.com), ainsi que divers services d'Extranet à destination des clients de nos clients et l'application « Etat des lieux » (projet présenté).

A ces différents services et produits s'ajoutent aussi la création de sites WEB, ainsi que la location de serveurs pour héberger les logiciels vendus et assurer ainsi un accès permanent à ceux-ci.

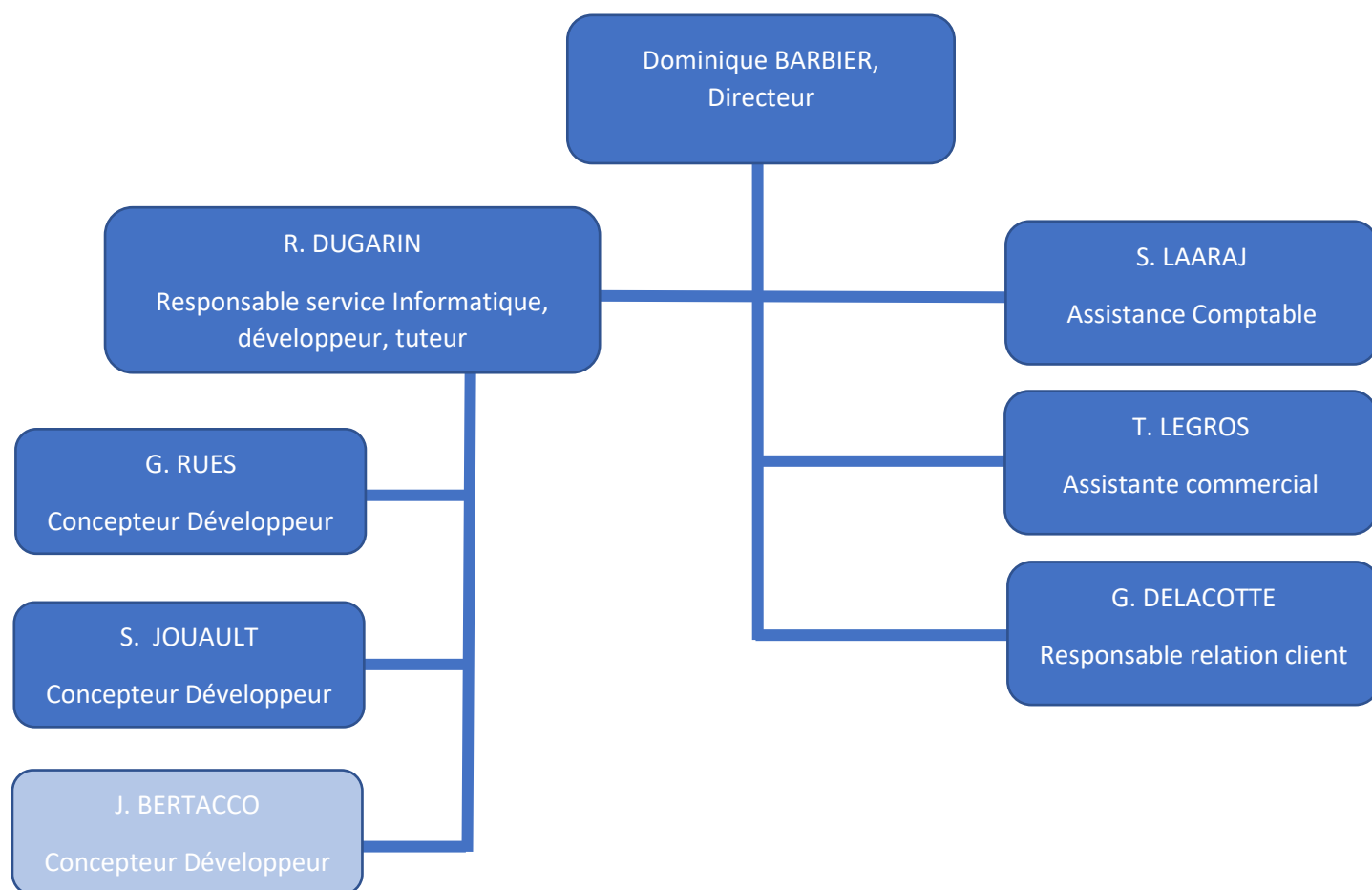
SDGI Bretagne assure aussi la formation et l'assistance sur les différents logiciels de sa gamme, tant sur la partie comptabilité que fonctionnelle. A ce titre les différents logiciels font l'objet de mises à jour régulières.

Activité

Les principales prestations de SDGIB en tant qu'entreprise de services sont :

- Le développement et la vente des outils de gestion.
- La formation à ces outils.
- Un support technique et fonctionnel.

Organigramme de l'entreprise :



Cahier des charges

Contexte du projet

L'entreprise SDGIB dispose d'un socle de logiciels développés dans les années 2000. Avec l'avancement des technologies et de leurs modes d'utilisations, il est devenu important de proposer des logiciels en adéquation avec les plateformes dont dispose les utilisateurs et également faciliter le support et l'assistance.

L'application Etat des lieux développé et proposé par l'entreprise depuis les années 2000 n'est disponible que sur des appareils sous système d'exploitation Windows. De plus, cette application ne communique qu'avec l'ancien logiciel de Gestion Locative.

Gestion Locative est un logiciel développé sous JAVA. Ce dernier propose la gestion complète d'une agence dans le domaine locatif. Mais également la comptabilité liée à cette activité.

Une nouvelle version du logiciel Gestion Locative ayant vue le jour depuis maintenant 4 ans, le développement d'une nouvelle version de l'état des lieux s'inscrit dans une suite logique.

Analyse de l'existant

L'ancienne version de l'état des lieux proposait aux clients de synchroniser les données du logiciel de gestion locative dans le but d'éviter la saisie fastidieuse de toutes les informations concernant le lot, les propriétaires et les locataires. L'application permettait également de récupérer les états des lieux effectués, dans le logiciel de gestion locative, afin d'avoir un suivi intégral d'un lot en gestion.

L'application était perçue comme un outil indispensable à partir du moment où les données pouvaient être récupérées des deux côtés.

Concernant l'application en elle-même, il s'agissait donc de faire des états des lieux de sortie ou d'entrée, de pouvoir créer ou éditer des modèles de lot et de générer des PDF contenant toutes les informations.

Problématique

L'ancienne application n'étant disponible que sur Windows, il est indispensable de développer une version pour Android et IOS.

Il est également essentiel de pouvoir récupérer les données de l'ancien logiciel d'état des lieux qui sont stockées sous Access 97.

L'application devant pouvoir gérer les données hors ligne et être synchronisée avec la nouvelle application de gestion locative, il est essentiel de mettre en place une API faisant le lien entre ces deux applications.

L'application doit être disponible sous Android et IOS et permettre aux clients de synchroniser les données une fois la connexion internet activée sur l'appareil. Le traitement et la conservation des données hors ligne et donc importante. Il faut également pouvoir proposer la signature électronique qui n'est pas simplement une zone dessinable sur l'écran, mais bien un système de fichier certifié par un organisme avec dépôt dans un coffre-fort numérique sur des serveurs sécurisé.

L'application doit également proposer la création et la modification de modèle de lot, composé de pièces elles même composées d'éléments divers tel que mur, sol, porte etc. D'intégrer la reconnaissance vocale pour les champs d'observation, ainsi que la prise de photographies.

Lors du passage d'un état des lieux, l'utilisateur doit pouvoir modifier le modèle uniquement pour cet état des lieux. En ajoutant des pièces et/ou des éléments.

Le modèle d'origine doit être conservé sans les modifications, toutefois, les modifications apportées, doivent être retrouvées lors d'un autre état des lieux sur le lot.

Objectifs du projet

L'objectif est de créer une solution permettant aux clients de disposer d'une application mobile sur tablette pouvant se synchroniser avec le logiciel Gestion locative ou traiter toutes les données en mode hors ligne sans synchronisation. Elle doit garantir une ergonomie satisfaisante pour l'utilisateur, une rapidité d'exécution et d'une manière générale, la qualité, la fiabilité et la sécurité des données.

Ce qui implique :

- De fournir une interface de saisie ergonomique capable de récupérer et envoyer des données à Gestion locative (application client lourd) via une API en fonction des entrées et sortie de locataires.
- De conserver les données liées aux états des lieux sur l'API.
- D'être capable de restituer les données des états des lieux effectué sous forme d'un PDF et les données lié à la modification d'un modèle sous forme de JSON dans l'API.
- De mettre en place une API ayant l'intégralité des clients d'SDGIB en base et pouvant recueillir l'ensemble des données lié à un lot, ainsi que de communiquer avec Gestion locative et l'application mobile EDL.
- D'ajouter un module à Gestion locative permettant d'envoyer à l'API les données des lots ayant des locataires entrants ou sortant et de récupérer les données liées à un état des lieux effectué.

Etude de faisabilité

Scénario 1 : Utilisation d'une solution externe déjà existante (non retenu)

Il existe des solutions gratuites ou payantes sur le marché permettant de réaliser des états des lieux disponibles sur les diverses plateformes (Android/iOS). L'idée était de tester les applications afin de trouver une solution fiable pour nos clients.

Avantages :

- Temps de développement.
- Support / Maintenance utilisateur.

Inconvénients :

- Aucune possibilité de synchroniser les données avec notre suite de logiciel.
- Aucune prise en charge des données existantes.
- Evolution dépendante de l'éditeur.

Scénario 2 : Développement en interne de la solution (retenu)

Développement d'une application mobile sous le framework Ionic, mise en place d'une API sous Laravel et développement d'un module pour Gestion location sous JAVA.

Avantages :

- Ergonomie de l'interface (HTML, CSS)
- Large possibilité d'évolution
- Maîtrise des données du client et récupération via l'API
- Synchronisation des données avec Gestion Locative
- Transfert des données en SSL
- Maîtrise complète de l'application

Inconvénients :

- Nécessite une montée en compétence dans plusieurs langages/Frameworks
- Temps de développement

J'ai rédigé un document « étude préalable » afin de présenter le projet à l'ensemble des personnes concernées.

Voir annexe 1

Vue d'ensemble du projet

La solution développée est une application mobile de création d'état des lieux pouvant être synchronisée avec le logiciel Gestion Locative, déployable sur Android ou IOS. L'application peut stocker des données métier ou les récupérer via l'API REST mis en place en amont.

L'application doit pouvoir gérer des « modèles » de lot, constitué de pièces, elles-mêmes constituées d'éléments.

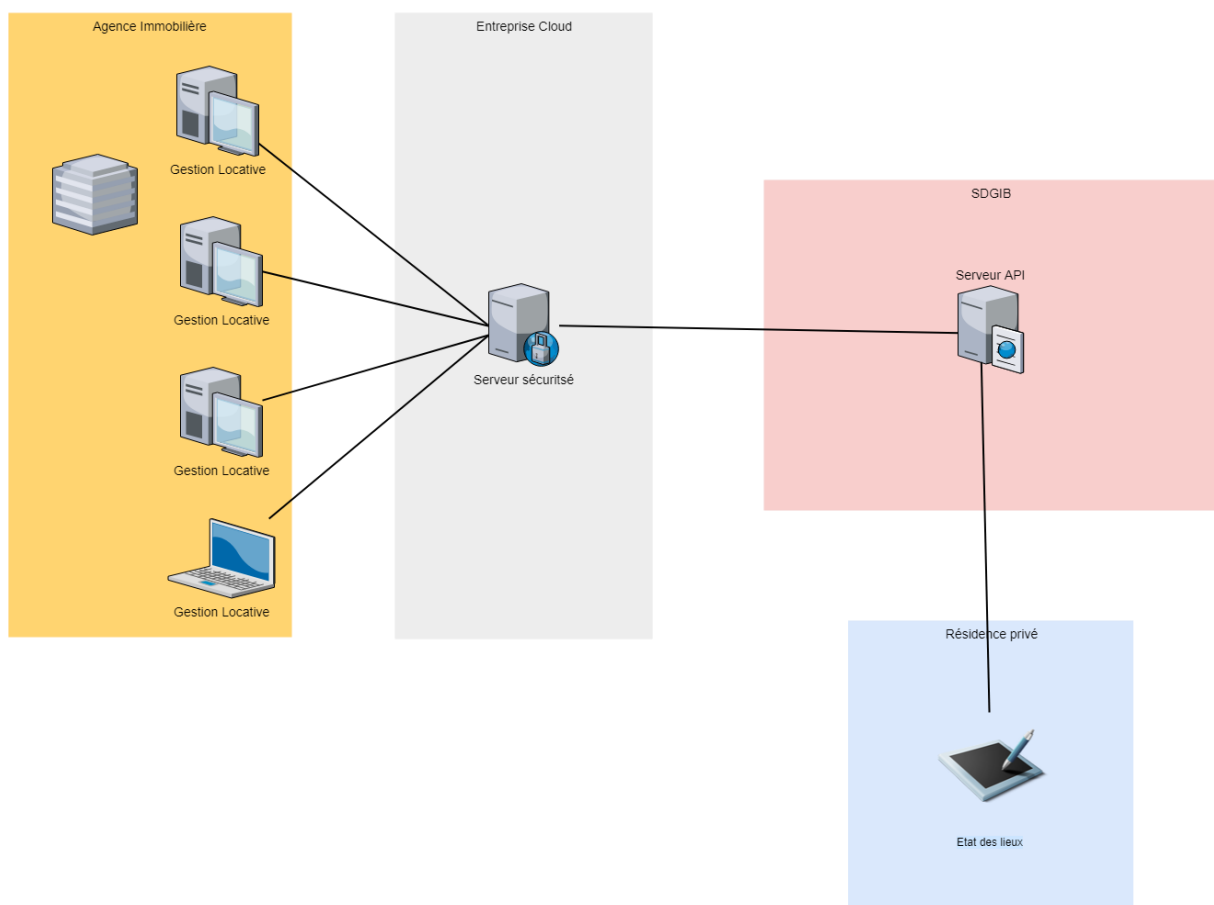
Les modèles doivent disposer obligatoirement de certains éléments (compteurs, clés, équipement énergétique).

L'utilisateur final doit pouvoir modifier un modèle pendant le passage d'un état des lieux en ajoutant des pièces et des éléments à ces pièces.

La gestion des modèles doit pouvoir être effectuée hors état des lieux. Il faut également pouvoir dupliquer un modèle.

La source de données de l'API est le logiciel Gestion Locative, développé depuis plusieurs années au sein de l'entreprise et produit phare. Il permet de gérer les lots de propriétaires, ainsi que les locataires, baux et tout ce qui gravite autour de la gestion locative ainsi que la gestion complète de la partie comptable pour une agence.

Schéma du projet



Livrables

Voici une liste exhaustive des livrables attendus par SDGIB :

- L'application (.apk, .ipa)
- Les sources de l'application mobile
- Les sources de l'API
- Une documentation technique de l'application mobile
- Une documentation technique de l'API
- Un mode opératoire destiné aux utilisateurs de l'application mobile
- Un mode opératoire de déploiement des nouvelles versions et de suivis des logs
- La mise en place de l'API sur un serveur
- Une version de Gestion Locative contenant le module pour l'export des données vers l'API ainsi qu'une documentation technique

Loi A.l.u.r – Décret n°2016-382 du 30 mars 2016

La loi Alur « Accès au Logement et Urbanisme Rénové » met en place certaines spécifications dans le cadre de la création d'un état des lieux :

L'état des lieux prévu à l'[article 3-2 de la loi du 6 juillet 1989 susvisée](#) doit porter sur l'ensemble des locaux et équipements d'usage privatif mentionnés au contrat de bail et dont le locataire a la jouissance exclusive.

L'état des lieux décrit le logement et constate son état de conservation. Il comporte au moins les informations suivantes :

1° A l'entrée et à la sortie du logement :

a) Le type d'état des lieux : d'entrée ou de sortie ;

b) Sa date d'établissement ;

c) La localisation du logement ;

d) Le nom ou la dénomination des parties et le domicile ou le siège social du bailleur ;

e) Le cas échéant, le nom ou la dénomination et le domicile ou le siège social des personnes mandatées pour réaliser l'état des lieux ;

f) Le cas échéant, les relevés des compteurs individuels de consommation d'eau ou d'énergie ;

g) Le détail et la destination des clés ou de tout autre moyen d'accès aux locaux à usage privatif ou commun ;

h) Pour chaque pièce et partie du logement, la description précise de l'état des revêtements des sols, murs et plafonds, des équipements et des éléments du logement. Il peut être complété d'observations ou de réserves et illustré d'images ;

i) La signature des parties ou des personnes mandatées pour réaliser l'état des lieux ;

2° A la sortie du logement :

a) L'adresse du nouveau domicile ou du lieu d'hébergement du locataire ;

b) La date de réalisation de l'état des lieux d'entrée ;

c) Eventuellement, les évolutions de l'état de chaque pièce et partie du logement constatées depuis l'établissement de l'état des lieux d'entrée.

À cet effet, l'application devra pouvoir fournir à l'agent la possibilité de gérer les éléments d'une pièce (murs, plafond, sol, équipement etc.), d'en saisir l'état, ainsi qu'une description.

Le PDF généré devra fournir les informations sur le bailleur, le propriétaire et les locataires, ainsi que le type d'état des lieux, la ou les dates. Le PDF devra également afficher d'une part l'état des lieux d'entrée et l'état de sortie.

Voir annexe 6

Gestion du projet

L'équipe dédiée au projet

Dans une entreprise de seulement quatre employés en développement, chaque projet est géré par une seule personne. Tout au long de la durée de vie du projet nous nous concertons pour permettre d'avoir les conseils avisés de collègues plus expérimentés. J'ai donc été chargé de la conception de ce projet en étant appuyé par mes collègues.

Méthodologie

En collaboration avec mon tuteur et la personne responsable du logiciel Gestion locative, nous avons commencé une première phase ayant pour but de formaliser le besoin.

Dans un premier temps, nous nous sommes réunis pour formaliser le besoin, puis, dans un second temps, j'ai présenté les différents scénarios étudiés. Pour finir, j'ai rédigé un cahier des charges, basé sur le scénario retenu et présenté lors d'une troisième réunion pour en valider le contenu (besoin, spécifications fonctionnelles, maquettes, ...). S'en est suivi la phase de conception et développement, le recettage, des présentations régulières aux clients et une démonstration à l'équipe.

Estimation de la charge

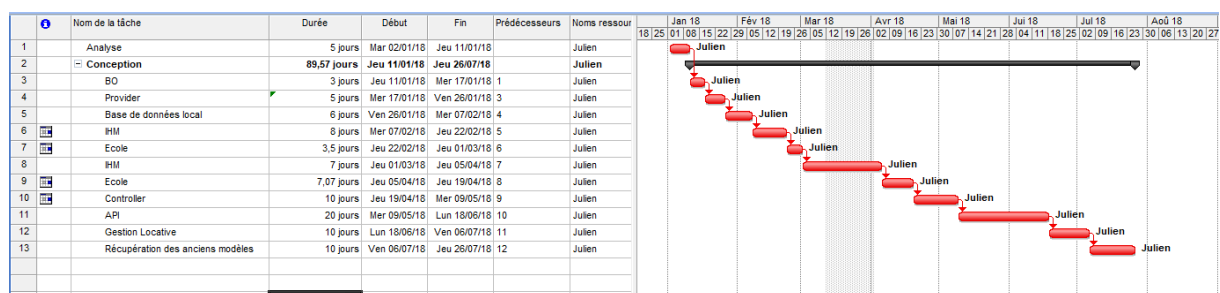
Etant le seul en charge du projet et n'en étant pas à ma première application sous Ionic j'ai pu estimer mon temps de travail sur les divers composants. Tout en prenant en compte la montée en compétence sur les langages/framework que je ne maîtrisais pas. Pour combler mon manque d'expérience j'ai mis en place un fichier sous Excel permettant de comparer le temps estimé au temps passé.

Dans le fichier Excel, je liste toutes les tâches liées au projet. Pour chaque tâche, je renseigne un temps de développement. Ensuite, lors du développement, je renseigne le temps que j'ai pris pour réaliser cette tâche.

Tâche	Temps estimé (H)	Temps réel (H)	Etat	Commentaire
Etat des lieux				
Mise en place du projet	2	1	DONE	
Création de la BO	2	3	DONE	
Ajout des providers	2	4	DONE	
IHM sans fonctionnalités	5	4	DONE	
				Ajout de la réorganisation des pièces et des éléments
Gestion des modèles de lot	20	27	DONE	Ajout de la duplication d'un modèle, d'une pièce et d'un élément
Ajout d'un état des lieux d'exemple	1	1	DONE	
Ajout d'un modèle d'exemple	1	1	DONE	
Affichage de la liste des états des lieux	2	2	DONE	
Affichage des détails d'un état des lieux	3	4	DONE	
				Mise en place d'un style plus adapté avec une liste d'éléments à gauche et les champs de l'élément sélectionné à droite
Liste des pièces lors du passage d'un EDL	30	42	DONE	
Ajouter une pièce lors du passage d'un EDL	2	3	DONE	
Ajouter un élément lors du passage d'un EDL	2	1	DONE	
				Affichage d'une demande de confirmation avant de supprimer une photo, appuyer sur la photo l'affiche en plein écran, appuyer long affiche la demande de confirmation avant suppression.
Gestion des photos dans un élément	6	10	DONE	

Planning

Voici un aperçu du diagramme de Gantt du projet.



Le planning n'a pu être respecté étant donné que j'ai dû travailler sur d'autres projets entre temps. Cependant celui-ci était estimatif et a pesé dans la balance des pour et des contre lors de l'étude de faisabilité.

Les tâches sont assignées à une seule ressource étant seul sur le projet. On peut constater que les jours de formation sont inclus.

Eviter l'effet tunnel

L'effet tunnel, c'est la première cause d'échec des projets. Dans un projet, il y a toujours deux acteurs principaux : le client, également appelé maître d'ouvrage, qui a exprimé un besoin et attend une solution répondant à ce besoin, et le maître d'œuvre qui est en charge de la conception et de la réalisation de la solution.

Un projet dure en général de quelques mois à plusieurs années. L'effet tunnel c'est lorsque pendant toute la durée du projet, le maître d'ouvrage et le maître d'œuvre ne se parlent pas.

Conséquence de cet état de fait, le résultat est le plus souvent extrêmement décevant, quand le projet n'est pas complètement raté.

Présentation régulière

Afin d'éviter l'effet tunnel, j'ai régulièrement effectué des présentations à des clients. Par exemple avec une agence immobilière, basée à Toulouse. Cela me permet de recueillir des avis différents sur l'utilisation d'un même logiciel.

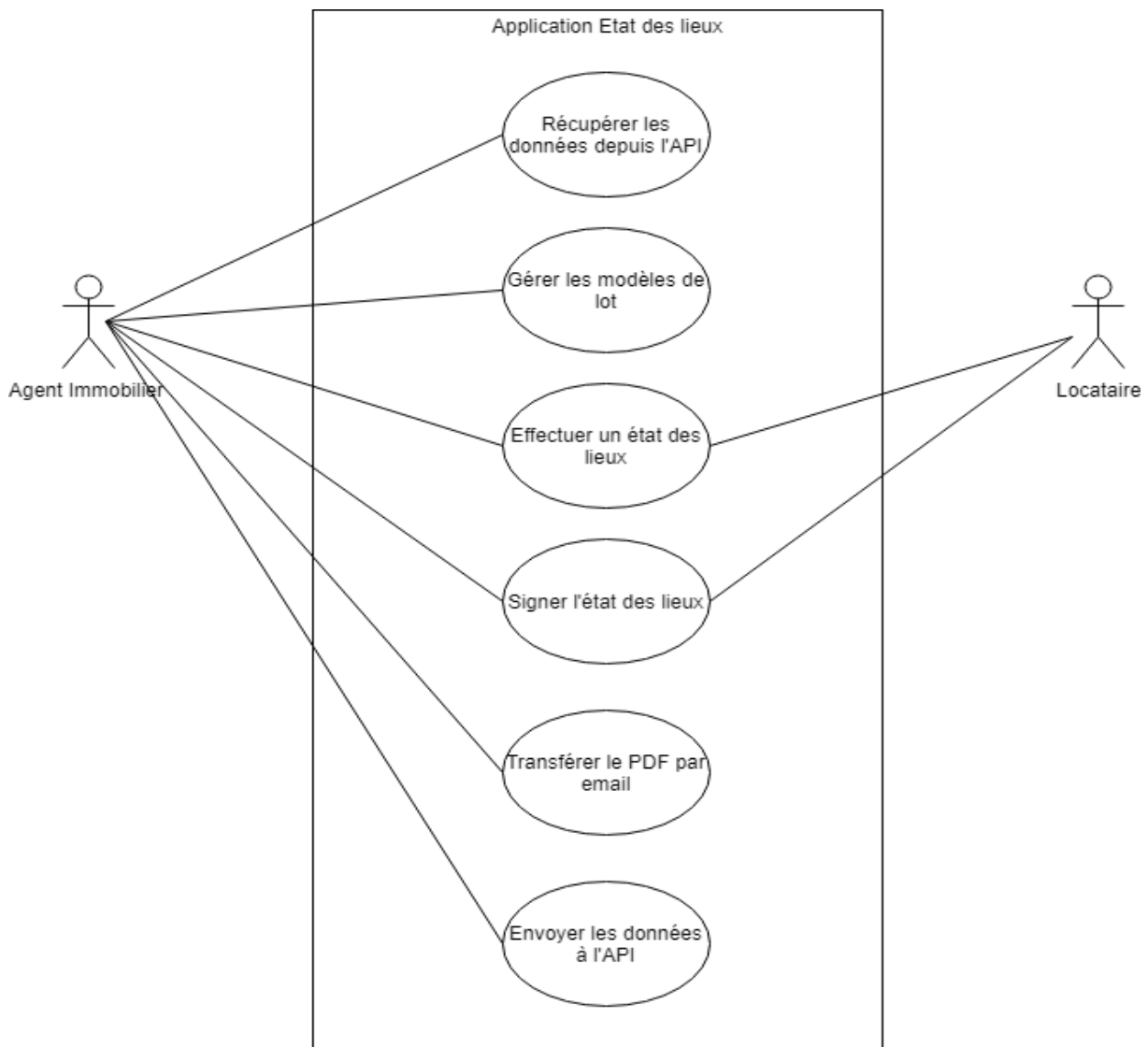
Dans le cas de l'agence, le client souhaite ne plus imprimer de document et être en règle avec la loi Alur décret du 30 mars 2016. Deux points qui sont déjà prévus dans l'application avec la signature électronique et la gestion des modèles afin de retrouver les données d'un état des lieux précédemment effectué.

Spécifications fonctionnelles

Diagramme de cas d'utilisation

Dans le schéma ci-dessous, on observe le fonctionnement de l'application réduit à son minimum.

L'utilisateur n'a pas besoin de s'authentifier pour accéder à l'API. L'authentification est configurée par nos soins. Si les champs de connexion à l'API ne sont pas renseignés, l'application est automatiquement passée en mode hors ligne et n'essaiera pas de se connecter. Dans ce cas précis, l'utilisateur disposera de plus d'options. Tel que l'ajout de lot, locataire, propriétaire et la possibilité d'ajouter des états des lieux. Ces options ne sont pas disponibles en mode synchronisation du fait qu'elles demandent beaucoup de saisies, ce que la majorité de nos clients ne souhaitent pas. De plus, les données saisies sur l'application (hors passage d'un état des lieux) ne doivent pas remonter à l'API. Dans un tel contexte, si les champs de connexion sont renseignés, les options sont désactivées.



Les deux acteurs d'un état des lieux sont l'agent immobilier et le(s) locataire(s) :

L'agent immobilier doit pouvoir :

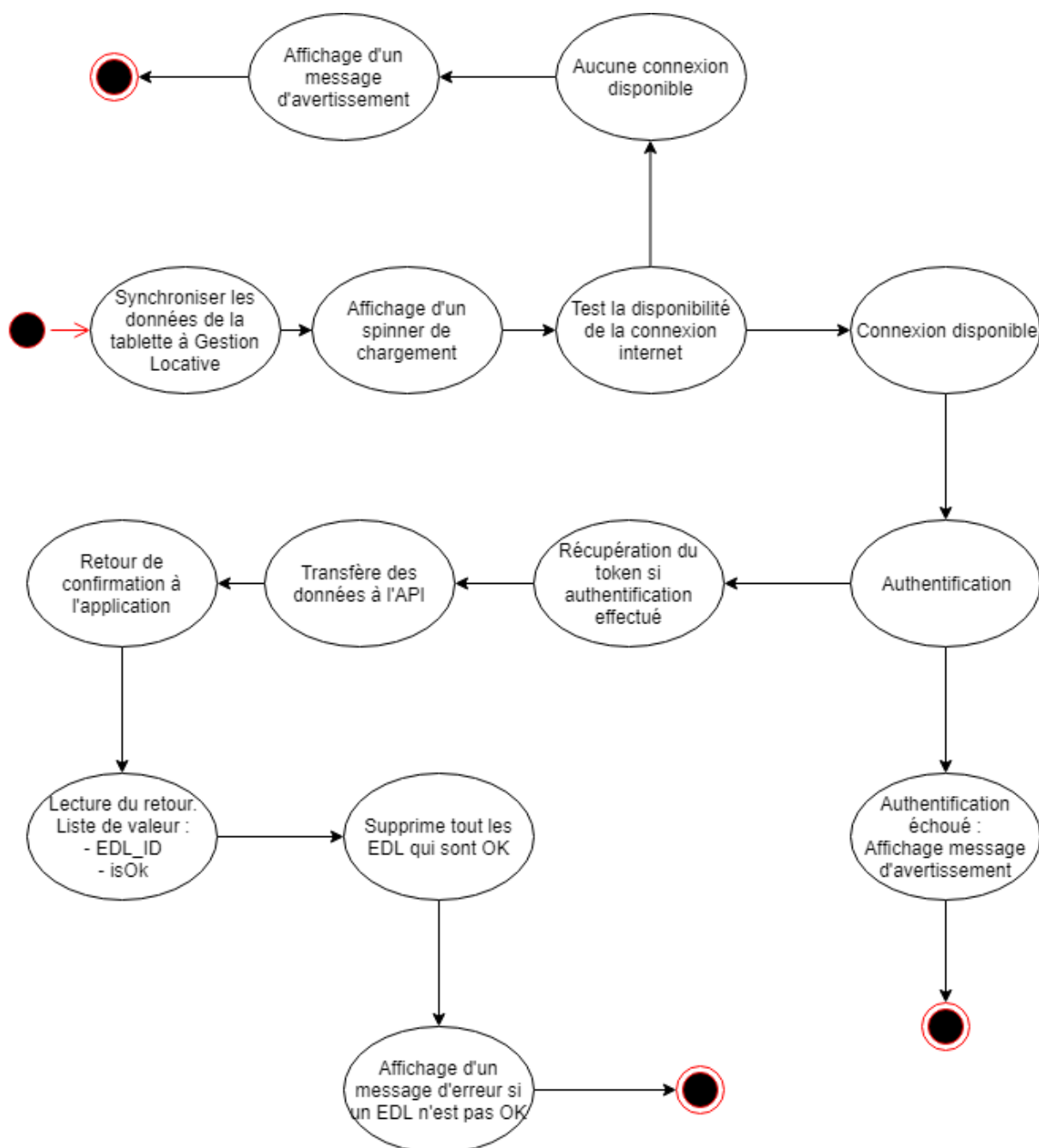
- Récupérer les données envoyées depuis le logiciel Gestion Locative.
- Gérer les modèles de lot.
- Effectuer un état des lieux.
- Signer l'état des lieux.
- Transférer le document généré aux locataires.
- Transférer l'état des lieux au logiciel Gestion Locative.

Le locataire doit pouvoir signer l'état des lieux.

Evidemment le locataire interagit avec l'agent immobilier lors du passage de l'état des lieux.

Diagramme du cas d'utilisation « Synchronisation des données »

Certains de nos clients ne souhaitent pas avoir de tablette connectée en permanence à « des ondes ». Choix que je respecte et pour cela, afin de synchroniser les données avec l'API, sachant qu'il faut tout de même connecter la tablette à internet, il leur suffit de cliquer sur le bouton « Synchroniser les données » une fois qu'ils sont connectés.



Si la connexion internet est disponible, l'authentification s'effectuera. Un « Token » d'une durée de validité de 12h sera retourné et enregistré en local. Le « Token » servira pour toutes les requêtes envoyées à l'API.

Une fois cette étape terminée, l'application va récupérer les états des lieux effectués et les transférer à l'API. De son côté, l'API va lire la liste des états des lieux envoyés, va ensuite tenter de les récupérer depuis la base de données et mettre à jour tous les attributs puis les enregistrer. Si l'API n'a pas pu récupérer un état des lieux depuis la base de données via l'ID envoyé par l'état des lieux, l'opération sera immédiatement arrêtée. L'ID ainsi que la valeur false sera ajouté à un tableau. Si l'opération a réussi dans son ensemble l'ID et la valeur true seront ajoutés au tableau.

Le tableau contenant la liste des ID des états des lieux envoyé ainsi qu'une valeur associée true ou false à chacun sera retournée à l'application « Etat des lieux ».

De son côté, l'application « Etat des lieux » va lire le tableau récupéré. Puis va supprimer dans la liste des états des lieux effectué tout ceux ayant la valeur true. Ceux ayant la valeur false ne seront pas supprimés et une entrée dans les logs sera ajoutée avec comme niveau ERROR, l'application affichera alors un message d'erreur à l'utilisateur, lui indiquant de nous contacter.

Maquettage de l'application

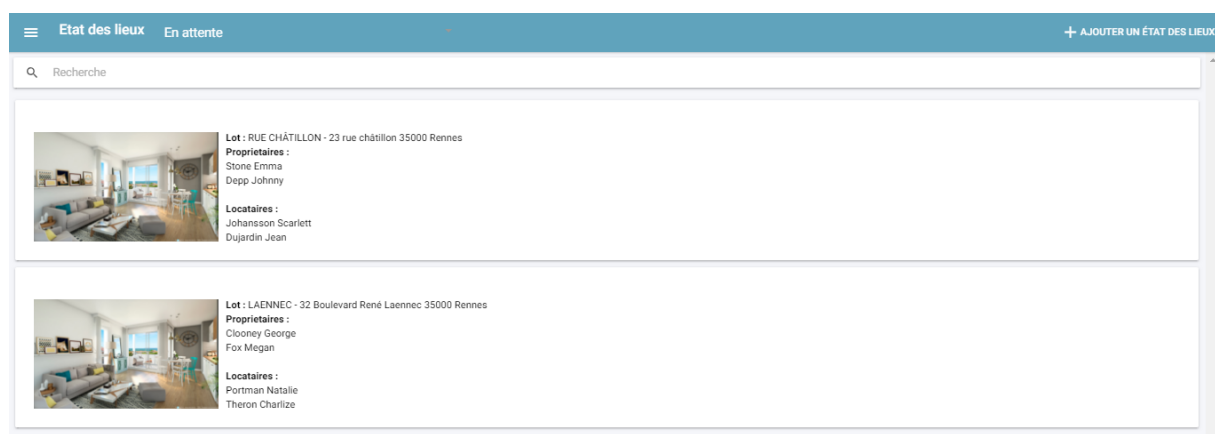
Le maquettage d'une application est une phase importante pour présenter les diverses fonctionnalités.

Il est généralement préférable de présenter une maquette dépouillée afin que le client ne se concentre pas sur l'aspect visuel de l'application, mais bien sur les fonctionnalités.

Les écrans suivants sont constitués de la version actuelle de l'application. La maquette réalisée avec Balsamiq Mockup est en annexe.

Voir annexe 3.

Liste des états des lieux



La liste d'états des lieux permet entre autres de filtrer ceux qui sont en attente ou terminés, d'ajouter un nouvel état des lieux et d'effectuer une recherche qui se fera sur les noms des propriétaires et locataires ainsi que sur l'identifiant du lot et son adresse.


En appuyant sur l'un des états des lieux présent, l'utilisateur est dirigé vers la page de récapitulation.

Les états des lieux sont divisés en deux tables, ceux qui sont en attentes et ceux terminés. Afin de garantir une rapidité d'exécution lors de l'affichage.

Récapitulatif avant le début d'un état des lieux

La page récapitulative permet d'afficher les informations concernant le lot, les propriétaires et locataires et propose de sélectionner un modèle de lot.

← Etat des lieux



Identifiant : Résidence Rue Cesson - Type d'état des lieux : Entrée

Informations sur le lot

Adresse : 43 Rue de cesson Rennes 35000

Nombre de pièce : 4

Surface (m²) : 62

Type de chauffage : Electrique individuel

Description :

Informations Propriétaires

Nom : Clooney

Prenom : George

Adresse : 26 Boulevard René Laennec Rennes 35000

Informations Locataires

Nom : Dujardin

Prenom : Jean

Adresse : 43 rue de cesson Rennes 35000

Modèle :

Résidence les prés verts

DÉBUT L'ÉTAT DES LIEUX


Dans le cas où un état des lieux en cours a dû être stoppé, pour quelque raison que ce soit, l'intégralité des données sont enregistrées dans le modèle qui est lui-même enregistré dans l'état des lieux.


Si l'utilisateur final souhaite changer de modèle, une fenêtre de confirmation s'affiche à l'écran lui indiquant que toutes les données seront alors supprimées.

Passage d'un état des lieux – Liste des pièces

La page principale du passage d'un état des lieux affiche uniquement les informations sur le lot ainsi que la liste des pièces présentes dans le modèle.

L'utilisateur a la possibilité d'ajouter une pièce ou d'accéder aux éléments présents dans une pièce déjà créée.

 **Etat des lieux**



Identifiant : Résidence Rue Cesson - **Type d'état des lieux :** Entrée

Informations sur le lot

Adresse : 43 Rue de cesson Rennes 35000

Nombre de pièce : 4

Type de chauffage : Electrique individuel

Description :

+AJOUTER UNE PIÈCE

Cuisine
Eléments effectué : 1/3

Compteurs
Eléments effectué : 0/3

Équipements
Eléments effectué : 0/4

Clés
Eléments effectué : 0/ 7

VALIDER

La progression de l'état des lieux est affichée dans la carte représentant une pièce.

L'utilisateur peut à tout instant terminer le passage en validant. Cela va générer et afficher un PDF qu'il sera possible d'envoyer par email ou d'enregistrer.

Passage d'un état des lieux – Élément d'une pièce

Les informations importantes sont présentes, à savoir :

- Reconnaissance vocale
- Prise de photo
- Liste des photos prises
- Champ d'observation
- Choix de l'état de l'élément
- Pouvoir ajouter une pièce / un élément
- Poursuivre à l'élément suivant ou retourner à l'élément précédent

← Cuisine

+AJOUTER UN ÉLÉMENT

Mur ✓

Portes

Fenêtres

Observation :

Observation

Etat

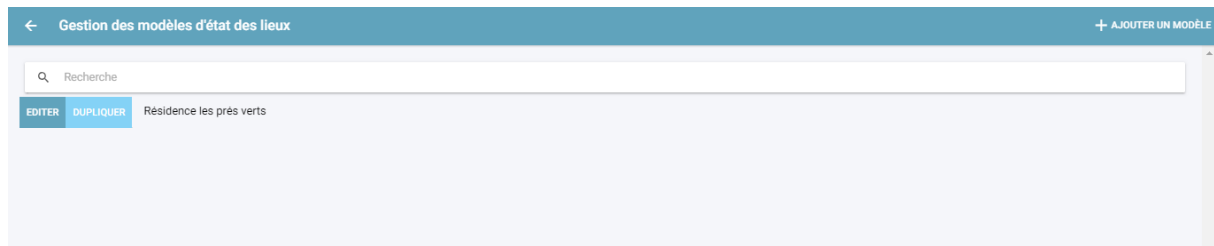
Très bon état

Réparation à la charge : Charge du locataire

VALIDER

Chaque fois que l'utilisateur valide un élément, il passe automatiquement au suivant. Une fois tous les éléments validés, l'utilisateur est redirigé vers la liste des pièces. Les données sont enregistrées au fur et à mesure de la progression.

Gestion des modèles



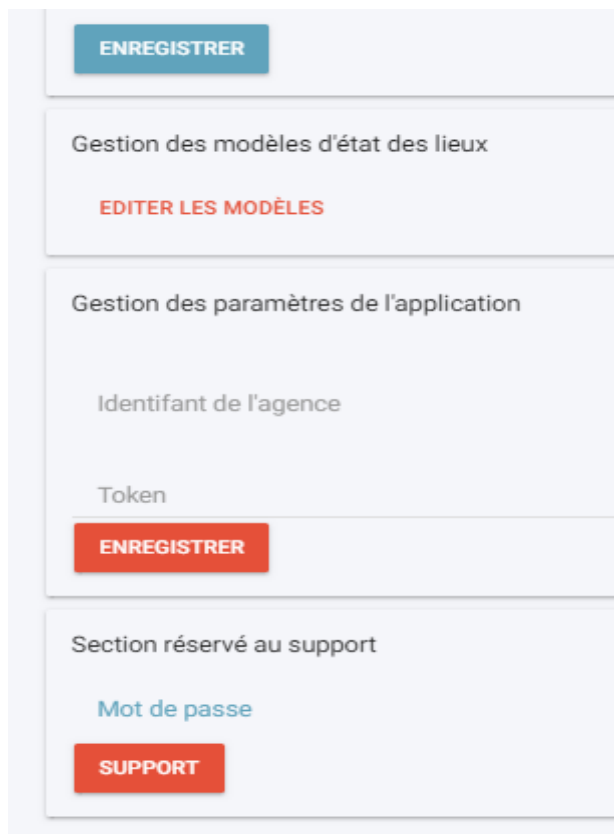
La gestion des modèles permet d'ajouter, d'éditer, de supprimer ou de dupliquer un modèle.

Dans le fonctionnement, on retrouve un modèle qui contient une liste de pièce et pour chaque pièce une liste d'élément.

Il est possible de dupliquer un modèle, une pièce contenue dans un modèle ou l'élément d'une pièce.

Paramètres de l'application

Les paramètres de l'application comprennent, la gestion des informations relative à une agence (raison sociale, coordonnées, etc...), un accès à la gestion des modèles, la configuration des informations d'authentification à l'API, ainsi qu'un accès à la page d'assistance.



La page d'assistance permet d'effectuer des opérations sur la base de données à partir de la tablette sans passer par un branchement sur un ordinateur. Car il est possible de faire une maintenance sur une application cross plateforme utilisant des webView via les outils de développeur du navigateur Google Chrome.

← Support

Executer du code JavaScript

Accéder à la base de données local :

```
this.storage.get('NOM_DE_LA_TABLE').then((VARIABLE_TEMP)=>{ console.log(VARIABLE_TEMP)});
```

Enregistrer dans la base de données :

```
this.storage.set('NOM_DE_LA_TABLE', VARIABLE);
```

Console :

```
this.storage.get('edls').then((edlsTemp)=>{
  for(let i=0; i< edlsTemp.length; i++){
    if(i == 0) {
      edlsTemp[i].id = 1;
    } else {
      edlsTemp[i].id = 2;
    }
  }
  this.storage.remove('edls').then(()=>{
    this.storage.set('edls', edlsTemp).then((newEdls)=>{
      console.log(newEdls);
    });
  });
});
```

["id":1,"model":null,"ordre":0,"valide":false,"count":0,"pdf":"","lot":"","id":1,"photo":"assets/imos/Jadeo-appartement.jpg","nbPiece":4,"proprietaires":

EXECUTER EFFACER

Gestion des logs :

AFFICHER LES LOGS DANS LA CONSOLE NETTOYER LES LOGS

Dans l'exemple ci-dessus, lors de l'ajout des états des lieux, j'ai oublié d'assigner un ID automatiquement. Une fois mon code corrigé, plutôt que de recréer un jeu d'essai, j'ai simplement modifié l'ID de mes deux états des lieux.

La « Console » est une simple balise HTML textarea dans laquelle tout code entré est évalué puis exécuté. Elle est loin d'être aussi ergonomique et puissante que les consoles disponibles sur les navigateurs, mais permet tout de même d'afficher le résultat.

Il est possible d'afficher et de nettoyer les logs de l'application dans la console afin de les analyser.

La principale fonction de la console est d'évaluer le code entré dans le champ, en cas d'erreur dans la saisie d'un code, une fenêtre d'alerte s'affiche.

Si le résultat peut être évalué correctement, il peut être affiché via un console.log.

J'ai override console.log afin que la sortie soit le champ textarea. En ce qui concerne l'affichage des objets et des tableaux, ils sont passés à la fonction JSON.stringify pour les rendre lisibles. Je pense améliorer l'ergonomie et les fonctionnalités de la console dans une version future.

Afin de faciliter d'avantage le support, j'envisage de mettre en place une façon plus visuelle de gérer la base de données. Sur un modèle tel que phpMyAdmin avec des fonctionnalités plus basiques.

Spécifications techniques

Technologies utilisées sur le projet

JavaScript

C'est un langage orienté objet à prototype, c'est-à-dire que les bases du langage et ses principales interfaces sont fournies par des objets qui ne sont des instances de classes, mais qui sont chacun équipés de constructeurs permettant de créer leurs propriétés.

JavaScript est un langage de programmation de scripts principalement employé dans les pages web interactives mais aussi pour les serveurs avec l'utilisation (par exemple) de Node.js.

TypeScript

TypeScript est un sur-ensemble de JavaScript. Il permet d'améliorer et de sécuriser la production du code JavaScript. Pour ce faire il est transpilé en JavaScript pouvant ainsi être interprété par n'importe quel navigateur web. Un des points intéressant de TypeScript est la ressemblance avec un langage client lourd. À mes débuts je n'avais que très peu de connaissance en JavaScript, et grâce à TypeScript j'ai pu facilement monter en compétence.

On doit certainement la ressemblance avec un langage client lourd du fait que le Co créateur de TypeScript soit l'inventeur du C#.

Aujourd'hui, j'ai une bien meilleure maîtrise de JavaScript, me permettant à l'occasion de mettre à disposition de la communauté des exemples de code pour des fonctionnalités.

Exemples : Calculer le nombre de jours dans un mois sur MDN – JavaScript DATE

Dans cet exemple, une fonction qui permet de calculer le nombre de jours dans un mois donné. Cette fonction utilise le comportement de Date.

Ionic

D'après la vision d'origine des développeurs du Framework, Ionic devait être un cadre d'interface utilisateur qui pourrait fonctionner avec n'importe quelle technologie choisie par un développeur web.

Cependant, dès le début ils ont opté pour un fonctionnement avec Angular. À l'heure actuelle, Ionic est passé en version 4 et est dorénavant indépendant de la technologie choisie.

Concrètement cela est une grande avancée non seulement pour eux en terme de diffusion via les différents Frameworks JavaScript populaires, mais aussi pour moi développeur sous Angular, car il m'est possible de mettre à jour la version de ce dernier indépendamment de Ionic et ainsi de profiter des avantages.

Pour recentrer sur le principe de Ionic, il s'agit d'un Framework visuel « basé » sur Angular permettant de réaliser des applications mobiles en utilisant Cordova.

Pour cela il apporte un grand nombre de composants visuels axés sur l'expérience utilisateur sur mobile. Une CLI (Command-Line Interface) permettant de transpiller le projet et de générer une application mobile grâce à Cordova.

Ainsi que l'utilisation de fonctionnalités native d'un appareil ou de développer ses propres fonctionnalités (en Java pour Android et Objective-C pour IOS) toujours grâce à Cordova.

Pour schématiser le concept :

- Développement d'un site web via Angular.
- Apport d'élément graphique via Ionic et facilitation de l'utilisation des fonctionnalités native.
- Encapsulation du site dans une WebView et accès aux fonctionnalités native via Apache Cordova.

Il est possible de créer des plugins pour cordova, en JAVA pour Android ou en Objective-C pour IOS.

Lors de la réalisation de l'application « Etat des lieux » je n'ai pas eu le besoin de créer un plugin.

Angular

Framework JavaScript permettant de réaliser des sites « Single Page Application ».

L'avantage principal d'un tel framework par rapport au développement de site classique réside dans la délocalisation d'une partie du traitement des données, qui auparavant s'effectuait coté serveur chez l'utilisateur.

Permettant ainsi de libérer des ressources sur le dit serveur et ainsi de pouvoir recevoir plus d'utilisateurs, mais également de rendre le site plus réactif car nécessitant moins d'appels au serveur, le traitement s'effectuant directement chez l'utilisateur. Une autre caractéristique des « Single Page Application » est la possibilité de continuer à exécuter en local l'application.

Le Framework, baptisé à la base AngularJs a été abandonné par ses concepteurs, qui l'ont fait basculer dans le domaine open source. Il a depuis été repris en grande partie par Google.

Depuis la version 2.0 (considérablement différente). Le Framework est rebaptisé Angular n*.

*Numéro de version.

L'une des différences majeures ayant entraînée le changement de nom est l'utilisation de TypeScript en tant que langage conseillé.

Laravel

Laravel est un framework PHP permettant de réaliser des sites internet et/ou des API.

Il embarque un ORM très bien doté, nommé Eloquent, permettant de gérer facilement les relations entre les données et la base de données.

L'utilisation de Laravel est facilitée grâce à la CLI de php artisan. Elle permet entre autres de générer rapidement des templates pour diverses couches. Par exemple la génération des migrations, des seeders, des modèles, de démarrer un serveur local de test, d'effectuer les tests unitaires, etc.

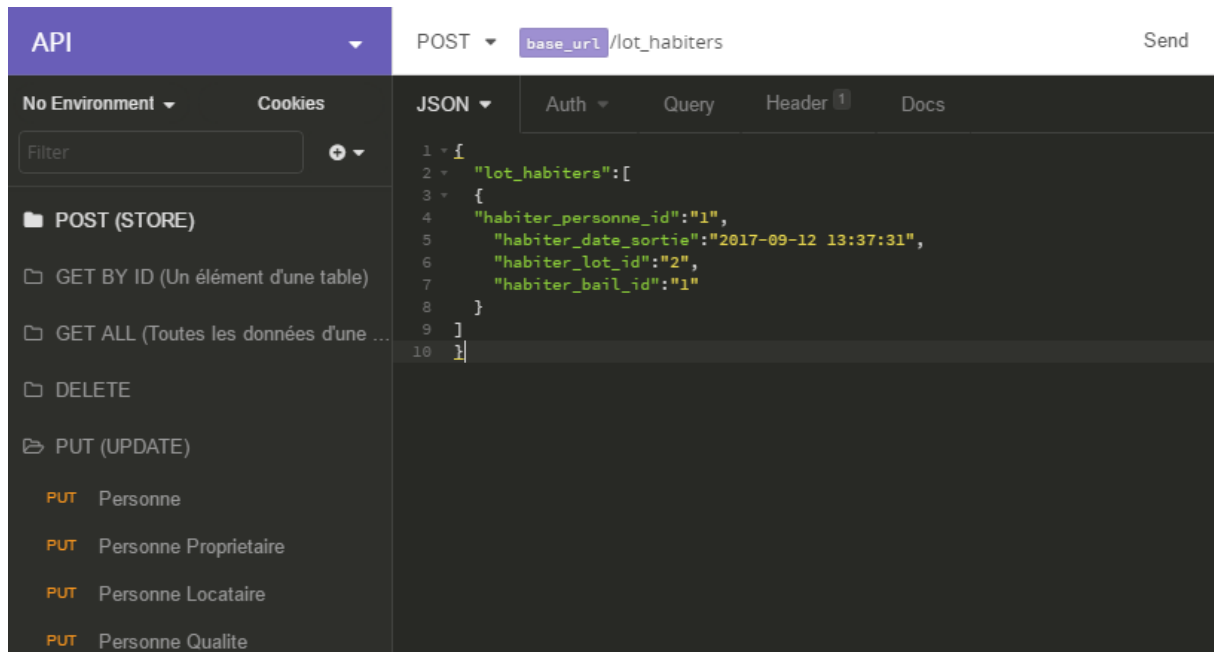
Laravel fournit des fonctionnalités en termes de routage de requête, de mapping objet-relationnel, d'authentification, de vue (avec Blade) et de migration de données.

INSOMNIA

INSOMNIA est un outil de test d'API REST.

Il permet :

- De créer des requêtes http : Spécifier l'URL, le payload, header et d'envoyer.
- D'afficher les détails de la réponse : Statut code, body, headers, cookies.
- D'organiser : Un espace de travail exportable composé de dossiers, composé de requêtes.



INSOMNIA peut s'avérer plus pratique que les tests de Laravel. Et il permet une bonne visualisation des données.

IndexedDB

« IndexedDB est un système de gestion de base de données transactionnel, similaire à un SGBD relationnel basé sur SQL. Cependant contrairement aux SGBD relationnels, qui utilisent des tables avec des colonnes fixes, IndexedDB est une base de données orientée objet pour JavaScript. IndexedDB permet de stocker et de récupérer des objets qui sont indexés avec une clef ; tout objet supporté par l'algorithme de clonage structuré peut être stocké. »

Source : MDN

Petit retour sur l'algorithme de clonage structuré. Il fait partie des nouvelles spécifications du HTML5 pour la sérialisation d'objets JavaScript complexes. Il est plus puissant que JSON du fait qu'il supporte la sérialisation d'objets contenant des graphes cycliques (des objets peuvent faire référence à des objets faisant référence à d'autres objets dans le même graphe cyclique).

De plus dans certains cas l'algorithme de clonage structuré peut être plus efficace que JSON.

En somme, il est possible de sauvegarder des tableaux d'objets contenant des objets qui contiennent eux même des tableaux d'objets et ainsi de suite, le tout référencé par une clé.

Java

Le projet Gestion Locative est développé en JAVA 8. Ce langage comprend une bibliothèque graphique nommée JAVA FX. Celle-ci ressemble au HTML et part donc d'un langage à balise nommé FXML. Ensuite un Controller est codé en JAVA, et le code s'exécute de manière traditionnelle en JAVA.

L'intérêt de cette technologie réside en son aspect moderne pour l'utilisateur, couplé aux implémentations. Il est en effet possible d'utiliser du CSS, pour paramétrer sa charte graphique. Le FXML étant extrêmement simple d'utilisation, celui-ci s'est présenté comme une évidence face à des technologies vieillissantes comme le SWING.

H2

La base de données du logiciel Gestion Locative est gérée avec H2. Initialement le système de gestion de base de données était SQLite. Cependant, la complexité de la gestion d'un accès en réseau local a conduit les développeurs de Gestion Locative à utiliser H2.

Ce système de gestion de base de données permet en effet un accès aisé en réseau local via un serveur embarqué, mais également un accès aisé depuis plusieurs sessions.

La dernière contrainte concernant la base de données était la nécessité d'un fichier de base de données facilement transportable.

Voici une liste (non exhaustive) des technologies utilisées pour le projet :

- Langage de l'application mobile : TypeScript
- Langage de l'API : Php 7
- Langage du module de Gestion Locative : JEE (JDK 1.8)
- JavaFX
- HTML
- CSS/SCSS
- Windows Server
- Base de données : PostgreSQL, IndexedDB, H2
- Outil de test d'API : INSOMNIA
- Environnement de développement : PHPStorm, Visual Studio Code, IntelliJ
- Outil de maquettage : Mokup
- Outil de diagramme : draw.io
- Framework de l'API : Laravel 5
- Framework de l'application mobile : Ionic 4 (basé sur Angular 4).
- MCD : Mysql Workbench
- Versionning : Gogs, GitKraken, SVN
- Suivi du projet : Trello

Réalisation

Préambule

L'application mobile cross platform « Etat des lieux » étant dépendante de plusieurs projets, je tenais à les présenter.

Module d'export des données

Afin de pouvoir fournir les données à diverses applications, il fallait mettre en place un module d'export des données sur le logiciel Gestion Locative.

Gestion Locative est un logiciel client lourd sous Java développé depuis plusieurs années en interne.

L'architecture est basée sur le design pattern MVC. Mais certains Design Pattern sont également implémentés :

- Façade : ReseauController conditionne l'accès à tout le module réseau qui contient plusieurs classes (Emission.java, Reception.java, TestUnique.java etc.). L'ensemble de ces fonctionnements étant très complexe et critique, l'ensemble des fonctionnalités est accessible via ReseauController.java
- Factory : pour l'ensemble des DAO. Cependant, l'implémentation du pattern n'est pas parfaite, puisque à l'utilisation des classes DAOFactory, DAOComptaFactory et MainDAOFactory, ce ne sont pas des objets de l'interface qui sont retournés mais bien des instances des classes implémentant l'interface DAI. Ce choix délibéré vient du fait que l'utilisation d'instances de l'interface limite l'utilisation de l'instance aux méthodes disponibles dans l'interface. Or, la plupart des classes DAO contiennent des méthodes spécifiques aux objets qu'elles manipulent. Ceci se rapproche alors du pattern Bridge.
- Prototype : les classes permettant la gestion des redditions notamment, implémentent une méthode clone(). Le but est de recréer un objet identique pour modifier la copie et éviter ainsi les conflits de référence inhérents au JAVA.

La phase de conception du module d'export c'est effectué en collaboration avec Mr. Guillaume Rues, qui s'occupe à plein temps du développement du logiciel.

On retrouve donc dans le module d'export, des modèles d'objet, une DAO pour requêter la base de données et des vues afin de proposer à l'utilisateur le choix des données à exporter.

Le module doit pouvoir envoyer les données, mais également récupérer les PDF au format Base64 puis les écrire sur le poste client dans un dossier qui correspond au lot.

Les requêtes

Afin de ne pas alourdir d'avantage la base de données de Gestion Locative, je n'ai pas ajouté de table. Les requêtes me permettant de récupérer les données dont l'application « Etat des lieux » a besoin lors des différentes étapes.

Une première requête va rechercher les baux dont la date expire le mois suivant.

Vient ensuite une requête pour vérifier si des locataires sont fraîchement ajoutés à un lot.

Puis via la DAOFactory, je récupère toutes les données liées à un lot, locataires, propriétaires etc.

N'ayant pas encore commencé le développement du module, je mets en annexe une méthode d'un utilitaire pour le module Export Tier. Cette méthode est composée de nombreuses requêtes et création de tables.

Voir annexe 4.

Les modèles d'objet « Adaptateur »

Les modèles d'objet de Gestion Locative contenant trop d'informations par rapport aux besoins de l'application « Etat des lieux », j'ai décidé de faire des classes « Adaptateur » afin d'alléger la structure des objets que le module allait envoyer sur l'API et de les adapter aux besoins futurs de l'extranet.

La classe ExtranetEtatDesLieuxAdapter.java est composée de plusieurs méthodes, qui correspondent aux divers objets à transférer. Par exemple, pour un propriétaire :

```
public JSONObject exportProprietaire(Proprietaire proprietaire) {
    JSONObject jsonProprietaire = new JSONObject();

    jsonProprietaire.addProperty("proprietaire_code", proprietaire.getCode());
    jsonProprietaire.addProperty("personne_nom", proprietaire.getNom());
    jsonProprietaire.addProperty("personne_prenom", proprietaire.getPrenom());
    jsonProprietaire.addProperty("personne_profession", proprietaire.getProfession());

    jsonProprietaire.addProperty("personne_date_naissance", proprietaire.getDateNaissance().toString());

    .....

    return jsonProprietaire;
}
```

Cette méthode me permet de récupérer uniquement les informations dont l'API a besoin, mais également de retourner un objet au format JSON.

L'IHM

Ayant déjà mis en place une interface pour le module d'export tiers, la personne en charge du développement de Gestion Locative ma incité à la réutiliser afin de ne pas alourdir d'avantage la structure du logiciel.

En fonction d'où provient l'utilisateur, il lui sera affiché le module export tiers ou le module export état des lieux.

L'IHM est composé d'une liste de lot dont les locataires sont soit en fin de bail soit récemment rattaché à un lot.

L'utilisateur peut alors sélectionner les lots et via un bouton situé en bas de page, transférer les données sur l'API.

Le travail n'ayant pas commencé, voici une capture de l'IHM du module Export Tier :

ACCUEIL

EXPORT DES ANNONCES

RECHERCHE SIMPLE

Mots clés...

ATTENTION si les champs seLoger obligatoires(*) ne sont pas remplis, le lot ne sera pas exporté.

RESULTAT : 317 lot(s) trouvé(s)

	CODE	TYPE	ADRESSE	VILLE	REMARQUE	DISPONIBILITE
<input checked="" type="checkbox"/>	524001	F2		LE PERREUX SUR MARNE		24/05/2015
<input checked="" type="checkbox"/>	866412	Maison		CHAMPIGNY SUR MARNE		02/01/2016
<input checked="" type="checkbox"/>	301505	parking		PARIS		01/01/2016
<input checked="" type="checkbox"/>	301506	parking		PARIS		01/05/2015
<input checked="" type="checkbox"/>	301507	parking		PARIS		01/03/2016
<input checked="" type="checkbox"/>	301509	parking		PARIS		18/03/2015
<input checked="" type="checkbox"/>	850101	F4		NOISY LE GRAND		18/11/2014
<input checked="" type="checkbox"/>	934002	Garage		PONTAULT COMBAULT		02/08/2016
<input checked="" type="checkbox"/>	867801	Studio		VILLENEUVE SAINT GEORGES		04/12/2015
<input checked="" type="checkbox"/>	867802	F/2 duplex		Villeneuve Saint Georges		06/06/2017
<input checked="" type="checkbox"/>	867803	F2		VILLENEUVE SAINT GEORGES		20/02/2016
<input checked="" type="checkbox"/>	821001	Studio		LE PLESSIS TREVISE		01/09/2014
<input checked="" type="checkbox"/>	868706	parking		VILLIERS SUR MARNE		26/09/2016
<input checked="" type="checkbox"/>	866903	F2		PARIS		04/11/2016
<input checked="" type="checkbox"/>	868402	Mobil home		GRISY SUISNES		01/06/2014
<input checked="" type="checkbox"/>	433001	F2		VILLIERS SUR MARNE		28/10/2015
<input checked="" type="checkbox"/>	472001	F2		OZOIR LA FERRIERE		01/11/2016
<input checked="" type="checkbox"/>	510101	F2		PARIS		27/07/2014
<input checked="" type="checkbox"/>	510202	parking		SUCY EN BRIE		09/01/2017
<input checked="" type="checkbox"/>	868201	Maison		BONDY		19/02/2015
<input checked="" type="checkbox"/>	697001	F2	VILLEMOMBLE		01/02/2017	
<input checked="" type="checkbox"/>	719001	F4	NOISY LE GRAND		10/07/2016	
<input checked="" type="checkbox"/>	753001	F4	LE PLESSIS TREVISE		01/12/2016	

EXPORTER

API

Préambule

L'API pourrait couvrir une grande partie du rapport mais ne suffirait pas à compléter l'ensemble du référentiel à elle seule.

Un point hors application mais devant passer dans le giron du développement : L'API doit aussi pouvoir gérer un Extranet pour les locataires / propriétaires de nos clients sous Gestion Locative. Elle doit également pouvoir stocker, traiter et retourner les données liées à un lot aux différents annonceurs (LeBonCoin, Seloger, etc).

Cela comporte beaucoup de dépendances : qualité, adresse, assurance, type de paiement etc. Au total plus d'une quarantaine de tables à mettre en place.

Conception de la base de données

Lors de la conception de la base de données je me suis mis en relation avec la personne qui conçoit et développe le logiciel Gestion Locative, car la majeure partie des données traitées proviennent de ce logiciel. Le plus important était de pouvoir retrouver les données essentielles à la réalisation d'un état des lieux et de pouvoir stocker/récupérer les données provenant de ce dernier.

Il ne s'agissait pas simplement de prendre chaque table de G.L et de les mettre dans l'API. Toutes les données n'ont pas lieux d'être.

Afin de bien visualiser l'enchaînement des requêtes et d'être sûr de ne rien oublier, j'ai fait un MCD sur Mysql Workbench bien que la base de données soit sous PostgreSQL.

Voir annexe 5

Mise en place de la base de données

Avec Php Artisan il est possible de générer des modèles avec des migrations :

```
php artisan make:model test2 --migration
```

Le fichier de migration permet de mettre en place les tables dans la base de données, mais aussi les colonnes et foreign key.

Fichier de migration pour la table lot :

Le fichier de migration permet de mettre en place une table en spécifiant le nom de la table, ainsi que le type et le nom des colonnes qui la compose.

La fonction up() ajoute la table à la base.

La fonction down() la supprime.

```
/**
 * Run the migrations.
 *
 * @return void
 */
public function up()
{
    Schema::create('personne_locataire', function (Blueprint $table) {
        $table->increments('locataire_id');
        $table->string('locataire_code');
        $table->integer('locataire_jour_prelevement');
        $table->integer('locataire_personne_id');
        $table->integer('locataire_adresse_quittance_id');
        $table->string('locataire_nom_quittance');
        $table->string('locataire_prenom_quittance');
        $table->integer('locataire_qualite_quittance_id');
        $table->integer('locataire_organisme_id');
        $table->timestamps();
    });
}

/**
 * Reverse the migrations.
 *
 * @return void
 */
public function down()
{
    Schema::dropIfExists('personne_locataire');
}
```

Fournir des données au déploiement de l'API sur un serveur

Afin de faciliter le déploiement de l'API, j'ai mis en place des « seeder » pour fournir les données « statique » à la base de données. Par exemple, les civilités (M/Mme) :

```
public function run()
{
    DB::table('personne_qualite')->delete();

    DB::table('personne_qualite')->insert([
        'qualite_libelle' => 'M.',
        'qualite_libelle_long' => 'Monsieur'
    ]);
    DB::table('personne_qualite')->insert([
        'qualite_libelle' => 'Mme',
        'qualite_libelle_long' => 'Madame'
    ]);
    DB::table('personne_qualite')->insert([
        'qualite_libelle' => 'M. et Mme.',
        'qualite_libelle_long' => 'Madame, Monsieur'
    ]);
}
```

Un seeder permet également d'ajouter les différents types de lot (F1, F2, T4 etc.), les catégories de lot (résidence, appartement etc), ainsi que l'ensemble des adresses et des informations de nos clients, et bien d'autres.

Routes

Voici un exemple des routes utilisées par l'API :

```
Route::get('/v1/edls', 'EtatDesLieuxController@index');
Route::get('/v1/edls/{id}', 'EtatDesLieuxController@show');
Route::post('/v1/edls', 'EtatDesLieuxController@store');
Route::put('/v1/edls/{id}/update', 'EtatDesLieuxController@update');
Route::delete('/v1/edls/{id}/delete', 'EtatDesLieuxController@delete');
```

Je me suis inspiré de ce que j'ai utilisé comme API afin de composer les routes.

Les routes commencent toutes par « v1 » afin de garantir la maintenabilité. Chaque route pointe vers un Controller et une méthode.

Les classes d'objets métier

La classe métier est composée du nom de la table (\$table), de la clé primaire (\$primaryKey), d'une liste de champs (\$fillable) assignable en masse, optionnellement des champs qui ne doivent pas être retourné (\$hidden).

```
/**
 * Nom de la table
 */
protected $table = 'personne';

/**
 * Clef primaire de la table
 */
protected $primaryKey = 'personne_id';

/**
 * Champs de la table qui sont modifiable
 */
protected $fillable = [
    'personne_nom',
    'personne_prenom',
    'personne_profession',
    ...
];
protected $hidden =
['personne_adresse_id', 'personne_qualite_id', 'personne_agence_id', ...];
```

Viennent ensuite les relations :

« HasOne » relation One to One

```
public function personne_proprietaire() {
    return $this->hasOne('App\Models\PersonneProprietaire',
        'proprietaire_personne_id');
}
```

Une relation individuelle, par exemple ici, une personne peut être associé à un propriétaire. Pour définir cette relation, une méthode `personne_proprietaire()` est défini dans le modèle `personne`. La méthode `personne_proprietaire` fait appel à la méthode `hasOne` et retourne son résultat.

Le premier argument transmet à la méthode `hasOne` est le nom du modèle associé. Eloquent détermine la clé étrangère de la relation en fonction du nom du modèle, Dans ce cas, le modèle `personne_proprietaire` est automatiquement supposé avoir une clé étrangère `personne_proprietaire_id`, par précaution la clé étrangère est passée en second argument.

« hasMany » relation One to Many

```
public function personne()
{
    return $this->hasMany('App\Models\Personne', 'personne_agence_id',
    'agence_id');
}
```

Une relation « un à plusieurs », est utilisée pour définir des relations dans lesquelles un seul modèle possède une quantité quelconque d'autres modèles. Par exemple ici, dans le modèle Agence, celle-ci peut contenir un nombre infini de personne. Comme toutes les autres relations avec Eloquent, mes relations « one to many » sont définies en plaçant une fonction dans le modèle. Comme pour le « one to one » il suffit de préciser le nom du modèle associé puis la clé étrangère.

« belongsToMany » relation Many to Many

```
public function casquette(){
    return $this->belongsToMany('App\Models\Casquette',
    'Personne_Casquette', 'personne_id', 'casquette_id');
}
```

Les relations « plusieurs à plusieurs » sont légèrement plus compliquées que « un à un » et « un à plusieurs ». Par exemple, ici, une personne peut avoir plusieurs casquettes. Les casquettes sont comme des rôles. Une personne peut être à la fois locataire et propriétaire.

Pour définir une telle relation, il faut trois tables de base de données : Personnes, Casquettes et Personne_Casquette.

Les relations « plusieurs à plusieurs » sont définies en écrivant une méthode qui renvoie le résultat de la méthode belongsToMany.

Application cross platform Etat des lieux

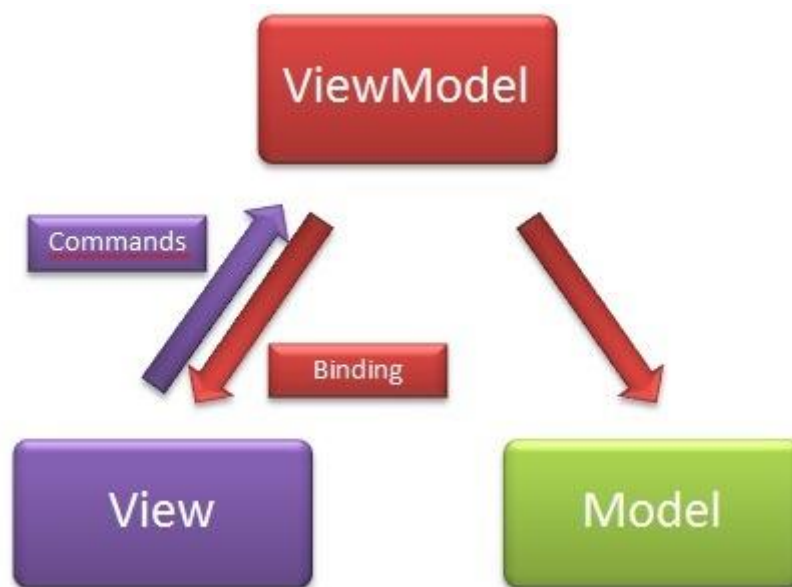


Design pattern MVVM

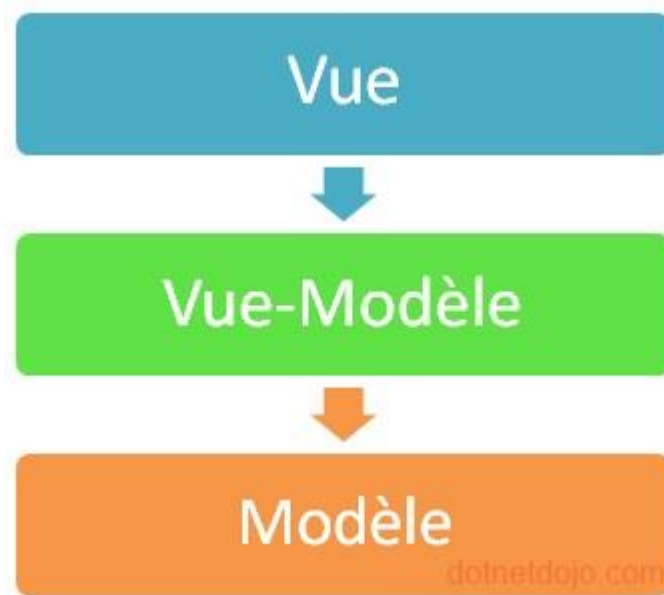
Ce pattern a spécialement été conçu pour améliorer la séparation entre les données et la vue qui les affiche. Le lien entre la vue et le modèle de données est fait par des mécanismes de binding.

Le binding est un mécanisme qui permet de faire des liaisons entre les données de manière dynamique. Ce qui veut dire que si A et B sont liés, le fait de modifier A va être répercuté sur B et inversement.

Le design pattern « Model View View-Model » permet de faire communiquer le modèle avec la vue-modèle et ce dernier avec la vue :



Les composantes de MVVM :



Model (Modèle) : le modèle contient les données. Généralement, ces données proviennent d'une base de données ou d'un service externe comme une API dans le cas de l'application « Etat des lieux ».

View (Vue) : la vue correspond à ce qui est affiché (la page web dans notre cas). La vue contient les différents composants graphiques (boutons, liens, listes) ainsi que le texte.

View-Model (Vue-Modèle) : ce composant fait le lien entre le modèle et la vue. Il s'occupe de gérer les liaisons de données et les éventuelles conversions. C'est ici qu'intervient le binding.

L'idée avec MVVM est simple : la vue ne doit jamais traiter des données. Elle s'occupe uniquement de les afficher. Le View-Model aura en charge les conversions et les accès au modèle de données.

Base de données

L'application dispose d'une base de données sous indexedDB.

IndexedDB est une API de stockage côté client qui permet de gérer des quantités importantes de données structurées, incluant des fichiers/blobs. Cette API utilise des index afin de permettre des recherches performantes sur ces données.

IndexedDB est un système de gestion de base de données transactionnel, similaire à un SGBD relationnel basé sur SQL. Cependant contrairement aux SGBD relationnels, qui utilisent des tables avec des colonnes fixes, IndexedDB est une base de données orienté objet pour JavaScript.

IndexedDB permet de stocker et de récupérer des objets qui sont indexés avec une clé.

Dans un but de faciliter la maintenance mais également l'évolution de l'application, les index sont définis dans un fichier de configuration global que j'ai mis en place afin de centraliser les informations importantes.

La base de données est donc composée de plusieurs clés :

- EDLS_WAITING
- EDLS_DONE
- LOTS
- LOCATAIRES
- PROPRIETAIRES
- MODELES
- AGENCE
- PDFS
- SETTINGS

Couche d'accès aux données

La couche d'accès aux données est mise en place dans des providers.

Ces derniers sont passés en « Singleton » à l'application.

Dans l'application il y a deux types d'accès aux données, le premier que nous allons voir, permet l'accès aux données de la base locale, pour un traitement entièrement hors ligne. Le seconde d'accéder aux données de l'API, que nous verrons après.

Par exemple dans le provider qui fournit à l'application les données liées aux états des lieux :

- Une variable déclarée en global pour stocker les états des lieux.
- Le constructeur effectue un appelle à la base de données pour assigner les états des lieux à la variable.
- Des méthodes afin de retourner ou sauvegarder les données.

La méthode qui suit permet de charger paresseusement les états des lieux en attente :

```

public getEdlsWaitLazy(length: number): Edl[] {
  let lazyEdlsWait = [];
  lazyEdlsWait = this.EdlsWait.slice();
  if (lazyEdlsWait.length > 20) {
    if (length === 0) {
      lazyEdlsWait = lazyEdlsWait.splice(0, 20);
    } else {
      if (lazyEdlsWait.length > length) {
        if ((lazyEdlsWait.length - length) > 20) {
          lazyEdlsWait = lazyEdlsWait.splice(0, (length + 20));
        }
      }
    }
  }
  return lazyEdlsWait
}

```

Le getter qui est utilisé :

```

get EdlsWait(): Edl[] {
  return this.edlsWait;
}

```

La requête à la base de données, qui est effectué dans le constructeur du provider :

```

this.storage.get(this.globals.DB_EDLS_WAIT).then((edlsWaitTemp) => {
  if (edlsWaitTemp === null || edlsWaitTemp === undefined) {
    this.EdlsWait = [];
  } else {
    this.EdlsWait = edlsWaitTemp;
  }
});

```

Et le setter :

```

set EdlsWait(edlsWaitTemp: Edl[]) {
  this.edlsWait = edlsWaitTemp;
}

```

Optimisation et recherche de performance

Effectivement, une application native sera bien plus performante qu'une application basée sur les WebView dans de nombreux domaines. Toutefois, pour une application de gestion de données, il n'y a pas de grande différence avec du natif.

Peu importe que l'application soit développée sous WebView ou natif, il est important de prendre en compte le fait qu'il s'agisse d'une application pour appareil mobile.

Dans un tel contexte il faut bien faire attention à l'utilisation de la batterie, du processeur et de la RAM.

L'utilisation de la batterie va grandement dépendre des autres points. Un processeur très demandé est énergivore. Tout fois, le fait de simplement modifier la couleur des pages, passant d'un blanc à un gris clair peut avoir un impact sur une journée entière d'utilisation.

Quelles sont les limites d'une application mobile de gestion ?

L'application doit pouvoir être utilisée hors connexion à l'API par un client ne souhaitant pas faire le lien entre l'application « Etat des lieux » et le logiciel « Gestion Locative », il est essentiel de connaître le nombre d'état des lieux que l'application va pouvoir gérer et d'adapter son architecture pour en accroître le nombre. J'ai rarement trouvé d'information à ce sujet et ce n'est pas abordé lors des cours. Pourtant c'est un point essentiel pour une application devant être mise en production.

Le PDF d'état des lieux d'entrée doit pouvoir être modifié pour y ajouter les informations de l'état des lieux de sortie. Il faut donc sauvegarder les données liées à l'état des lieux d'entrée. Le plus lourd dans ces données sont les photographies.

Doit-on les conserver sur l'appareil avec un espace de stockage limité ? Ou doit-on les enregistrer au format base64 et risquer des pertes de performance voir des « out of memory » ?

Le flag -prod

Il s'agit de l'une des modifications les plus simples mais ayant un gros impact. Lors du déploiement d'une version n'étant pas destinée au débogage, il est recommandé d'utiliser le flag -prod. Celui-ci n'est pas actif par défaut.

Sa fonction principale est de minifier le code et de compiler en avance ou cela est pertinent. Malheureusement, la documentation sur le flag est très rare.

Angular enableProdMode()

Ici, il s'agit surtout de faire attention à la console JavaScript du navigateur pendant le développement de l'application. Tant que « `enableProdMode()` » n'est pas indiqué dans le fichier principal de l'application (`main.ts`) un message d'avertissement s'affiche dans le console du navigateur.

La documentation d'Angular indique que cela désactive les assertions et autres vérifications dans la structure. En approfondissant les recherches à son sujet, il s'avère que cela désactive une deuxième exécution de détection des modifications. Cette deuxième exécution est présente en mode développement pour s'assurer que les modèles et méthodes retournent bien la même valeur que lors de la première exécution.

Mark-and-sweep

Mark-and-sweep « Marquer et balayer » est un algorithme qui s'apparente à un « garbage collector ». Il est implémenté dans l'ensemble des navigateurs web depuis 2012.

Le concept principal de cet algorithme par rapport aux « garbage collector » traditionnels est qu'il ne compte pas sur les références d'un objet pour déterminer s'il est toujours utile de le conserver dans la mémoire. Car un objet JavaScript fait toujours référence à un prototype.

En effet, il va plutôt chercher à savoir si cet objet peut être atteint. Si ce n'est pas le cas, la mémoire sera libérée.

Dans un tel contexte, il suffit de travailler sur des variables locales dans des fonctions retournant un résultat. Évitant ainsi la multiplication des variables déclarées globalement à une page ou un composant.

Lazy Loading

Chaque page de l'application est configurée en chargement paresseux afin d'améliorer la fluidité lors du démarrage de l'application.

Le point le plus important à charger paresseusement est la liste des états des lieux effectués.

Si le client n'a pas souhaité la version de l'application synchronisée avec Gestion Locative, la liste des états des lieux effectués peut à moyen long terme devenir volumineuse.

Afin d'éviter à l'application de devoir charger et générer graphiquement une liste de plusieurs centaines d'éléments, j'ai mis en place un chargement fragmenté de la liste. Les 20 premiers états des lieux sont chargés et un bouton situé en bas de page, à la fin de la liste permet de charger les 20 suivants. Cette opération est répétée jusqu'à atteindre l'intégralité de la liste. Normalement le client ne devrait pas avoir à charger toute la liste, car il a la possibilité d'effectuer une recherche dans celle-ci.

En ce qui concerne la liste des états des lieux à effectuer, on estime que celle-ci ne devrait pas dépasser les 20 entrées par semaine. Une charge acceptable pour un appareil mobile.

La librairie JavaScript utilisée pour la génération du PDF (PDFMake) prend uniquement des images au format Base 64 ce qui est un problème pour les performances de l'application. Si toutes les images sont converties directement lors du passage de l'état des lieux et stockées dans un objet image, lui-même stocké dans une liste d'images rattachées à un objet Element. L'application finirait par un joli « out of memory ». De plus en termes de stockage, une image au format base64 est 1.37 fois plus volumineuse que l'originale.

Le choix a été de ne conserver que les chemins d'accès aux images. Celle-ci sont converties à la volée lors de la génération du PDF.

Afin de ne pas encombrer l'espace de stockage de l'appareil, la prise de photographie est configurée pour s'adapter directement au PDF. À savoir que le PDF fait une largeur d'environ 596 pixels en 72 DPI. La photographie est donc directement convertie à 250 pixels afin de conserver une marge tout autour et de pouvoir aligner une deuxième photographie à côté d'elle. Cela permet de réduire leur poids. Ainsi le poids d'une photographie ne dépasse pas les 700 kilooctets.

Toutefois, les photographies prises lors d'un état des lieux d'entrée sont supprimées lors de la génération de l'état des lieux de sortie du même lot. De même pour les photographiques d'un état des lieux d'entrée, les photographies de l'état des lieux de sortie sont supprimées. Cela permet de diviser au minimum par deux l'espace de stockage nécessaire pour les photos.

De plus le client a la possibilité de changer de carte SD ou de transférer les photographies sur un stockage externe et de les remettre quand il souhaite effectuer un état des lieux de sortie.

Conclusion

Actuellement, l'application peut gérer plusieurs dizaines de milliers d'états des lieux effectués. Plus exactement 1 000 000 d'états des lieux, avec les images au format base64, directement enregistrées dans l'objet état des lieux. L'application ne s'arrête pas à la suite d'une erreur d'insuffisance mémoire et fonctionne parfaitement.

Sachant que le lazy loading des états des lieux est mis en place et que les photographies sont stockées sur le périphérique et non pas en base64, le chargement est instantané.

Si nous tablons sur un maximum de 20 états des lieux effectués par mois, avec une durée de vie de l'application de 20 ans, cela revient à 4800 états des lieux à gérer.

Quant à l'espace de stockage, il n'est plus un problème, sachant que Apple commercialise des tablettes ayant 512GO de stockage. Avec une moyenne de 40 photographies par état des lieux, sur un total de 4800 états des lieux, suivant la méthode de stockage des photographies précédemment détaillé, l'application ne devrait pas prendre plus de 70GO d'espace en 20 ans.

Avec 1 million d'états des lieux, il est impératif que la recherche s'effectue de façon asynchrone. De plus seuls les 20 premiers résultats sont retournés. Avec cela il est possible d'affirmer que l'architecture de l'application supporte la montée en charge.

Les classes d'objets métier « Model »

Un objet métier est un conteneur de données d'application, tel qu'un client ou une facture. Les données sont échangées entre les composants par l'intermédiaire d'objets métier.

L'application comporte quelques classes d'objets métier :

- Agence
- Edl
- Element
- Image
- Personne
- Lot
- Model
- Piece
- ...

Certaines classes implémentent d'autres classes :

- Locataire implémente Personne
- Propriétaire implémente Personne
- Element-energetique implémente Element
- Piece-compteur implémente Piece
- ...

Les classes disposent d'attributs visibles, il n'y a qu'un seul constructeur, vide, car il n'est pas possible de faire du polymorphisme dans une même classe, il n'y a donc pas de deuxième constructeur avec surcharge.

Pas de getter/setter. Arrivant sur JavaScript avec un bagage en client lourd, les getters et setters me semblaient logique. De plus TypeScript implémente get et set. Cependant à force de pratiquer, j'ai trouvé cela plus contraignant, générant plus d'erreurs, mais également alourdissant le code et ainsi impactant de manière moindre les performances.

Certaines classes d'objets, disposent de getter pour retourner des données, construites à partir de plusieurs attributs, comme par exemple retourner l'adresse complète, composer du numéro de la voie, de la voie, du code postal et de la ville.

Plus particulièrement, à la suite d'un problème d'instance, quelques classes ont un getter qui retourne un « token » ou plus exactement un nombre aléatoire compris entre 1 et 100 milliards. Le « token » est assigné directement à un attribut de la classe, afin de le générer pour chaque nouvelle instance.

Le problème est survenu lors de la récupération des données depuis la base de données local. Certains objets ayant des attributs identiques, par exemple deux objets de type « Element », avec un libellé « murs » et un id « 1 » étant assigné l'un à une pièce « Salon », l'autre à « Cuisine » étaient considérés comme une seule instance du même objet lors du mappage automatique des données.

C'est donc afin de palier à ce problème que la génération d'un « token » aléatoire à fait son entrée.

Grâce à lui les deux objets ont un attribut qui les distingue.

```
export class Element {

  id: number = 0;
  libelle: string = '';
  token: number = this.generateToken();
  .....
  constructor() {
  }

  generateToken(): number {
    return Math.floor(Math.random() * 100000000000) + 1);
  }
  .....
}
```

Les classes « ViewModel »

Une classe ViewModel est chargée de préparer et de gérer les données d'une View. Il gère également la communication de la vue avec le reste de l'application. Un ViewModel est toujours créé en association avec un scope et sera conservé tant que la portée est active. Par exemple, si c'est une page, jusqu'à ce qu'elle soit quittée.

En d'autres termes, cela signifie qu'un ViewModel ne sera pas détruit si son propriétaire est détruit pour un changement de configuration (par exemple, la rotation de l'écran). La nouvelle instance du propriétaire sera simplement reconnectée au ViewModel.

Par ailleurs, le ViewModel relate la logique métier de l'interface graphique.

Les classes d'accès aux données

Dans l'application, il y a plusieurs classes d'accès aux données, qui fonctionnent en Singleton étant déclarées dans le fichier app.module.ts.

Dans l'exemple ci-dessous, cette classe s'occupe d'effectuer les requêtes à l'API via diverses méthodes. Chaque méthode retourne une promesse « Promise », celle de résoudre la requête ou d'échouer. Permettant ainsi des requêtes asynchrones :

```
getEdl() {
  const httpOptions = {
    headers: new HttpHeaders({
      'Content-Type': 'application/json',
    })
  }
```

```

};
return new Promise((resolve) => {
  this.http.get(this.globals.API_URL, httpOptions).subscribe(res => {
    resolve(res);
  });
}).then(res => this.mapEdl(res)).catch((err)=>{
  let params = [];
  params.push(this.log.paramConstructor('Erreur',err.toString()));
  this.log.write(this.iLog.ERRORS, 'API SERVICE','getEdl', `Erreur lors de
l'envoi des données`, params);
  this.log.presentToast('Erreur lors de la synchronisation');
});
}

```

La méthode `getEdl()` retourne un jeu de données une fois la résolution effectuée. Les données sont d'abord envoyées à la méthode `mapEdl()` qui s'occupe de mapper les données reçues avec des modèles d'objets.

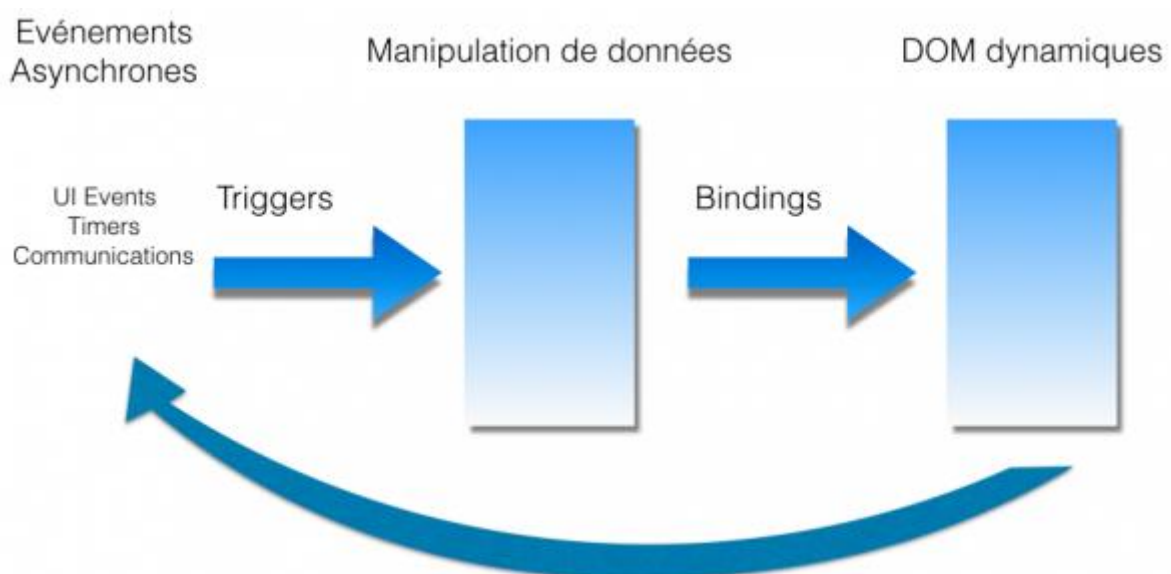
En cas d'erreur, un message d'avertissement est affiché à l'utilisateur et une entrée est ajoutée aux logs de l'application.

Les vues

Les vues sont la représentation graphique de l'application. Elles sont réalisées en HTML et CSS.

Ionic intègre une grande quantité de composants graphiques prêts à l'emploi, permettant de passer plus de temps sur le développement de la logique métier.

Étant sous Angular 5, il est possible d'utiliser le « Data Binding » qui est un élément essentiel dans les Frameworks de « Single Page Application ». Il permet de synchroniser la vue au modèle JavaScript sous-jacent. Voici un schéma général du fonctionnement du « Data Binding » :



Angular a défini quatre sortes de « Data Binding » pour synchroniser le template et le composant. Il est ainsi possible de propager une donnée du composant vers le DOM, du DOM vers le composant et dans les deux sens. Ces formes de « Data Binding » sont communément nommées :

Interpolation : Ce mécanisme permet de modifier le DOM à partir du modèle, si un changement est intervenu sur une valeur de ce dernier.

Property Binding : Ce mécanisme permet de valoriser une propriété d'un composant ou d'une directive à partir du modèle, si un changement est intervenu sur une valeur de ce dernier.

Event Binding : Ce mécanisme permet d'exécuter une fonction portée par un composant par suite d'un événement émis par un élément du DOM.

« Two-way » Data Binding : C'est une combinaison du Property Binding et du Event Binding sous une unique annotation. Dans ce cas-là, le composant se charge d'impacter le DOM en cas de changement du modèle ET le DOM avertit le composant d'un changement via l'émission d'un événement.

Génération d'un PDF en JavaScript

Afin de proposer aux utilisateurs d'enregistrer et/ou d'envoyer ainsi que de pouvoir le consulter, un état des lieux au format PDF directement depuis l'application, je me suis tourné sur la librairie js PDFMake.

Celle-ci est correctement documentée et la communauté a mis à disposition quelques tutoriels.

Je n'ai pas d'autres expériences en création de PDF mis à part cette application. Cependant, j'ai vite réalisé que cela pouvait être compliqué.

PDFMake fonctionne sur la base d'un tableau, dans lequel il faut organiser les données. Il supporte plus ou moins bien le CSS. En somme, il ne faut pas trop en demander et faire avec ce que l'on a.

Afin d'organiser correctement mon code, sachant que le PDF doit être composé d'un header avec les informations de l'état des lieux et de l'agence, d'une partie avec les informations des locataires et de même pour les propriétaires, puis la plus grande partie dédiée aux pièces composé d'éléments, eux même composés de texte et image et enfin pour finir, un footer composé de zone de signature. J'ai décomposé chaque partie dans des fonctions que je regroupe plus tard afin de générer le PDF.

Une fois le PDF généré, non seulement il est proposé à la lecture/enregistrement/envoie par mail, mais il est également enregistré au format Base 64 si le client a opté pour la version qui synchronise les données avec le logiciel Gestion Locative.

Cette version Base 64 sera mise en ligne sur l'API une fois la synchronisation effectuée puis téléchargé sur le poste du client et reconverti en fichier dans le logiciel Gestion Locative et stocké en local.

Création des jeux de test

Afin de pouvoir tester tous les cas de figure, j'ai mis en place plusieurs états des lieux :

- Un état des lieux sans toutes les informations propriétaires.
- Un état des lieux sans toutes les informations locataires.
- Un état des lieux avec des informations manquantes sur le lot.
- Un état des lieux complet.

Afin de pouvoir tester les requêtes sur l'API, j'ai utilisé INSOMNIA qui est un logiciel pour tester les API. Pour cela j'ai mis en place un espace de travail composé de dossiers triés par méthode (GET, POST, PUT, DELETE). Dans chaque dossier, un ensemble de requêtes pour effectuer les tests avec des jeux de données qu'il est facilement éditable car il s'agit de JSON.

De plus le workspace est exportable, une fois le jeu de test en place, je l'ai exporté dans le projet afin qu'il soit directement disponible depuis git.

GET : Une requête fournissant la liste des états des lieux en excluant ceux portant le statut « terminé » et ceux dont l'agence ne fait pas partie des droits de l'utilisateur.

POST : Une requête permettant de sauvegarder (INSERT) un état des lieux à effectuer, avec le lot, la liste des locataires et la liste des propriétaires .

PUT : Une requête permettant de modifier (UPDATE) une état des lieux qui vient d'être terminé.

Les Tests

Tests unitaires

Afin d'assurer une non régression de mon code, et une validation de certaines fonctionnalités lors du passage en production, j'ai mis en place des tests unitaires. Ces tests sont effectués sur toutes les pages et services.

Pour les pages, il s'agit dans un premier temps de vérifier que la page s'affiche correctement puis dans un second temps d'envoyer des paramètres et de vérifier que l'information affichée correspond bien aux paramètres.

En ce qui concerne les services, pour celui communiquant avec l'API, une vérification de l'envoi de données sur une route de test qui permet de tester le bon fonctionnement et de renvoyer vrai ou faux. Puis une vérification du mappage des données avec les modèles d'objet.

Pour le service qui s'occupe des logs, une simple vérification d'écriture avec une comparaison du nombre d'entrée dans les logs avant et après.

Tests fonctionnels

Avant la commercialisation / livraison, la phase de recettage interne intervient. Cette phase a pour but de vérifier le respect des règles de gestion listées dans le cahier des charges et le bon fonctionnement de la solution sur les différentes plateformes. La recette est ensuite répétée par mon tuteur à partir d'un document vierge dans lequel seules les actions/résultats attendus sont renseignées. Pour ce faire, j'ai réalisé des plans de tests se présentant sous la forme d'un tableau où chaque ligne représente une fonctionnalité à valider.

Une ligne se compose :

- Une action
- Un résultat attendu
- Une case OK/KO
- Un commentaire

Action		Résultat attendu	Android		IOS	Commentaire
			OK	38		
			KO	0		
			N/A	0		
Ajout d'un modèle de lot		Nouveau modèle ajouté à la liste		OK	KO	
Dupliquer un modèle de lot		Nouveau modèle ajouté à la liste avec les pièces et éléments du modèle dupliqué		OK	KO	
Modifier le libellé d'un modèle		Les modifications sont enregistrées		OK	KO	
Ajouter une pièce à un modèle		Nouvelle pièce dans la liste		OK	KO	
Dupliquer une pièce		La pièce est ajoutée au modèle avec les éléments de la pièce dupliquée		OK	KO	
Modifier une pièce		Les modifications sont enregistrées		OK	KO	
Ajouter un élément à une pièce		L'élément est ajouté à la liste		OK	KO	
Modifier un élément		Les modifications sont enregistrées		OK	KO	
Dupliquer un élément		Les attributs de l'élément sont identiques		OK	KO	
Ajouter un locataire		Le locataire s'affiche dans la liste		OK	KO	
Modifier un locataire		Les modifications sont enregistrées		OK	KO	
Afficher les informations d'un locataire		Les informations sont correctes		OK	KO	
Supprimer un locataire		La liste est mise à jour		OK	KO	
Ajouter un propriétaire		Le propriétaire s'affiche dans la liste		OK	KO	
Modifier un propriétaire		Les modifications sont enregistrées		OK	KO	
Afficher les informations d'un propriétaire		Les informations sont correctes		OK	KO	

Fiche de tests

Pour le suivi des versions de correctif et de toutes les anomalies ou demandes autres que celles listées dans le cahier des charges, j'utilise Trello avec un tableau dédié à l'application EDL.

Une liste est dédiée aux anomalies avec pour chaque problème une carte comportant la date et une description et un statut permettant de définir le type d'anomalie (problème applicatif, une fonctionnalité à ajouter, une mauvaise utilisation de l'application, ...).

Une liste comporte toutes les versions de l'application avec les dates de publication afin d'avoir un suivi sur la progression.

Maintenabilité

J'utilise GIT pour stocker et versionner mes développements avec GitKraken. Il y a une interface simple d'utilisation et plaisante à utiliser, la possibilité de gérer plusieurs serveurs de stockage, une parfaite comptabilité sur Windows.

Cependant, travaillant seul sur mes projets, je n'exploite aucune des fonctionnalités relatives au travail collaboratif.

Pour garantir une facilité de configuration et d'évolution de l'application, j'ai créé un fichier de « properties » permettant d'externaliser :

- Les différents liens pointant vers l'API
- Les différentes clés pour IndexedDB

Une vérification de la version de l'application installée s'effectue à chaque démarrage si la connexion internet est active, pointant sur un fichier situé sur le serveur de l'API. Fichier au format XML contenant le numéro de la dernière version ainsi qu'une URL pour télécharger le fichier d'installation.

Si le numéro de version de l'application installée est inférieur à celui du fichier XML, le téléchargement et l'installation s'effectuent automatiquement à condition que l'utilisateur l'accepte.

Afin de garantir la maintenabilité, j'ai mis en place une externalisation de l'ensemble des textes de l'application.

L'externalisation des textes est composée de plusieurs parties. Une variable dans le fichier de configuration globale indique le langage de l'application. Cela pourra directement être sélectionné depuis l'IHM dans le futur.

Une classe nommée `i18n` récupère cette variable dans une méthode « `setLocal` » qui permet de retourner une classe correspondant à la langue via un switch.

Pour l'instant il n'y a qu'une seule classe nommée « `Fr` ». Cette dernière est composée de variable static.

```
14         case 'FR':
15             return Fr;
16         default:
17             return Fr;
18     }
19 }
20 }
21
22 @Injectable()
23 export class Fr {
24     /**
25      * Type d'état des lieux
26      */
27     static EDL_ENTER: string = 'Entrée';
28     static EDL_EXIT: string = 'Sortie';
29     /**
30      * Texte pour la gestion des erreurs enregistré dans les logs
31      */
32     static ERROR_TITLE: string = 'Erreur !';
33     static ERROR_RETRY: string = 'Veuillez réessayer';
34 }
```

Déploiement

Le déploiement de l'application s'effectue via un fichier («.apk » pour Android ou un fichier «.ipa » pour IOS) installé directement sur l'appareil du client. Pour chaque nouvelle installation, l'application crée un modèle d'exemple au démarrage. L'utilisateur n'a plus qu'à entrer ses identifiants de connexion à l'API s'il en dispose. Dans le cas contraire, il doit renseigner les informations de son agence.

Affichage des messages utilisateurs

Pour l'affichage des messages à destination de l'utilisateur dans l'application, j'ai mis en place des « toast », sorte de notification s'affichant pour une durée limitée à l'écran afin d'avertir l'utilisateur, soit du bon déroulé des actions, soit des erreurs rencontrées. Les notifications proviennent du même service que les logs.

Gestions des logs

Pour améliorer le suivi des erreurs j'ai développé un service qui me permet de mettre en place un logger dans chaque pages et services. Dans le service « log » un méthode nommé « write » prend en paramètre :

Le niveau du log (VERBOSE, INFO, WARNINGS, ERRORS)

Le nom de la page

Le nom de la méthode

Un message de description

Les paramètres fournis à la méthode / L'objet de retour prévu de la méthode

```
write(level: string, component: string, methode: string, message: string,  
params?: { name: string, param: string }[])
```

La méthode « write » s'occupe de dater l'entrer puis d'ajouter une ligne composée de tous les paramètres dans un tableau qui est ensuite enregistré dans le localStorage.

Concernant les paramètres, il y a plusieurs façons de les ajouter. La première consiste à créer un objet anonyme composé d'un attribut name, et param tout deux au format string.

La seconde consiste à utiliser une méthode que j'ai mis en place dans le provider Log :

```
/**
 * @param name Nom du paramètre
 * @param param Valeur du paramètre
 */
public paramConstructor(name: string, param: string) {
    let params: { name: string, param: string } = {
        'name': name, 'param': param
    };
    return params;
}
```

Les paramètres sont ensuite à placer dans un tableau pour être fournis à la méthode « write ».

Si le tableau de paramètres qui est optionnel est présent, il est passé dans une boucle pour n'en faire qu'un seul string :

```
if (params !== null && params !== undefined && params.length > 0) {
    for (let i = 0; i < params.length; i++) {
        parameters += ' - Paramètre ' + i + ' ' + params[i].name + ' = ' +
params[i].param;
    }
}
```

Puis le tout est ajouté au tableau des entrées de log déjà présent :

```
logs.push(date.toLocaleString() + ' - ' + level + ' - ' + component + ' - ' +
methode + ' - ' + message + ' - ' + parameters);
this.storage.set('logs', logs);
```

Il est possible d'afficher les logs en passant par la page de support :

Console :

```
07/09/2018 à 11:54:34 - INFO - Liste Lot - constructor() - Chargement des données -- Aucun paramètres
07/09/2018 à 11:54:36 - INFO - Add Edit Lot - constructor() - Chargement des données -- Aucun paramètres
07/09/2018 à 11:54:37 - INFO - Add Edit Lot - saveLot() - Affichage d'une modale pour assigner un propriétaire au lot - - Paramètre 0 LOT ID = 1
07/09/2018 à 11:54:37 - INFO - Select Propriétaire - removeExistingPropriétaire() - Retire un propriétaire de la liste si il est déjà rattaché au lot - - Paramètre 0 Propriétaire ID = undefined
07/09/2018 à 11:54:37 - INFO - Select Propriétaire - dismiss() - Ferme la modale de sélection d'un propriétaire - - Paramètre 0 Propriétaire = 1
07/09/2018 à 11:54:38 - INFO - Add Edit Lot - saveLot() - Affichage d'une modale pour assigner un propriétaire au lot - - Paramètre 0 LOT ID = 1 - Paramètre 1 PROPRIETAIRE ID = 1
07/09/2018 à 11:54:38 - INFO - Add Edit Lot - saveLot() - Affichage d'une modale pour assigner un propriétaire au lot - - Paramètre 0 LOT ID = 1
07/09/2018 à 11:54:38 - INFO - Select Propriétaire - removeExistingPropriétaire() - Retire un propriétaire de la liste si il est déjà rattaché au lot - - Paramètre 0 Propriétaire ID = undefined
07/09/2018 à 11:54:39 - INFO - Select Propriétaire - dismiss() - Ferme la modale de sélection d'un propriétaire - - Paramètre 0 Propriétaire = 2
07/09/2018 à 11:54:39 - INFO - Add Edit Lot - saveLot() - Affichage d'une modale pour assigner un propriétaire au lot - - Paramètre 0 LOT ID = 1 - Paramètre 1 PROPRIETAIRE ID = 2
07/09/2018 à 11:54:40 - INFO - Add Edit Lot - selectLocataire() - Affichage d'une modale pour assigner un locataire au lot - - Paramètre 0 LOT ID = 1
07/09/2018 à 11:54:40 - INFO - Select Locataire - removeExistingLocataire() - Retire de la liste des locataires ceux déjà rattaché au lot -- Aucun paramètres
07/09/2018 à 11:54:41 - INFO - Select Locataire - dismiss() - Fermeture de la modal avec le locataire sélectionné - - Paramètre 0 Locataire Id = 2
07/09/2018 à 11:54:41 - INFO - Add Edit Lot - selectLocataire() - Locataire assigné au lot - - Paramètre 0 LOT ID = 1 - Paramètre 1 LOCATAIRE ID = 2
07/09/2018 à 11:54:42 - INFO - Add Edit Lot - selectLocataire() - Affichage d'une modale pour assigner un locataire au lot - - Paramètre 0 LOT ID = 1
07/09/2018 à 11:54:42 - INFO - Select Locataire - removeExistingLocataire() - Retire de la liste des locataires ceux déjà rattaché au lot -- Aucun paramètres
```

EXECUTER EFFACER

Gestion des logs :

AFFICHER LES LOGS DANS LA CONSOLE NETTOYER LES LOGS

Actuellement l'application log toutes les actions des utilisateurs afin d'avoir un retour sur son utilisation. La limite est de 300 entrées. Passé cette limite les 100 premières entrées sont supprimées.

Documentation

Documentation technique

Etant l'un des seuls développeurs à utiliser TypeScript et plus généralement les langages web, j'ai mis en place une documentation technique afin de fournir un support fiable et efficace pour maintenir l'application.

La documentation technique est composée de la liste des outils à installer, de l'environnement, des commandes, des pages et des points importants de l'application.

Voir Annexe 2

Conclusion

L'alternance fut pour moi une grande réussite sur le plan personnel. Je remercie vivement l'ENI de l'avoir mis en place. C'est une plongée dans le monde professionnel qui m'a permis de connaître mes limites, mes faiblesses mais aussi mes points fort. J'ai intégré une entreprise en plein développement et j'ai pu voir les besoins en temps réel.

L'expérience d'alternance m'a permis de conforter mon choix de reconversion professionnelle et m'a d'autant plus motivé à poursuivre des études dans ce domaine qui évolue chaque jour.

Le développement de diverses applications, avec plusieurs langages de programmation et divers Framework m'ont permis d'apprécier les différents aspects du développement.

J'ai apprécié développer sous JAVA, cependant, avec les Frameworks JavaScript tel Angular, il devient difficile de ne pas résister au développement d'une application pour tous les supports. Avec Ionic pour le mobile, Electron pour le desktop et simplement Angular pour le web, il n'y a qu'une seule application à développer et des surcouches à configurer, voire parfois à adapter.

Les performances du JavaScript sont honorables, il faut faire attention à l'architecture de l'application, effectuer des micros optimisations et savoir comment fonctionne la gestion de la mémoire pour en apprécier le potentiel.

Annexes

1. Etude préalable

Cette étude a permis de présenter le projet au dirigeant de l'entreprise. L'étude a été rédigée avec des bribes du cahier des charges.

2. Documentation technique

La documentation technique est toujours en cours de rédaction. Elle met en place les processus pour poursuivre le développement de l'application et son déploiement. Il est possible que certains éléments soient manquants ou mal documentés.

3. Maquettage de l'application

L'état actuel de l'application et la maquette de base n'ont plus grand-chose en commun. Les différentes réunion, présentations et démonstrations ont conduit à un projet au plus proche de l'attente des clients.

4. Méthodes de l'utilitaire pour le module Export Tier

Cette méthode a pour but de présenter des requêtes SQL dans le cadre du développement d'un module pour Gestion Locative.

5. MCD

Réalisation d'un MCD sous MySQL Workbench qui est selon moi un très bon outil. Il m'a permis de mettre en place visuellement la base de données de l'API.

6. Exemple d'état des lieux au format PDF

Le PDF est toujours en cours de réalisation, mais les bases sont posées.