

**Projet SOS Immo**

**Sophie JOFFRE**

**Mémoire en vue de la certification**

**RNCP Concepteur développeur d’applications**

**14 février 2023**

****

**Sommaire**

[1 Contexte SOS Immo 5](#_Toc126144767)

[1.1 Fonctionnement des services immobiliers 5](#_Toc126144768)

[1.2 Transformation de la gestion de suivi d’incident 5](#_Toc126144769)

[1.3 Objectifs 5](#_Toc126144770)

[1.4 Besoins de l’application 6](#_Toc126144771)

[2 Conception 7](#_Toc126144772)

[2.1 Use Case 7](#_Toc126144773)

[2.2 Profils utilisateurs 7](#_Toc126144774)

[3 Architecture d’application 8](#_Toc126144775)

[3.1 Front End 9](#_Toc126144776)

[ Composant 9](#_Toc126144777)

[ UseState ou variable d’état 9](#_Toc126144778)

[ Dans notre application : mise en pratique 11](#_Toc126144779)

[3.2 Back End – structure à couches 12](#_Toc126144780)

[3.3 Base de données 13](#_Toc126144781)

[ Utilisateurs - Habilitations 14](#_Toc126144782)

[ Prestataires - Types d’incidents 15](#_Toc126144783)

[ Emplacements – Types d’emplacements 15](#_Toc126144784)

[ Incidents 16](#_Toc126144785)

[ Journaux 18](#_Toc126144786)

[4 Cinématiques utilisateur et écrans principaux 18](#_Toc126144787)

[4.1 Cinématique Usager 19](#_Toc126144788)

[4.2 Cinématique Technicien 22](#_Toc126144789)

[4.3 Cinématique Valideur 23](#_Toc126144790)

[4.4 Cinématique Admin 24](#_Toc126144791)

[ Gestion des données nécessaires à la création d’incidents 24](#_Toc126144792)

[ Focus gestion Utilisateurs 25](#_Toc126144793)

[5 Sécurité 27](#_Toc126144794)

[5.1 Utilisateur 27](#_Toc126144795)

[5.1.1 Création 27](#_Toc126144796)

[5.1.2 Authentification 28](#_Toc126144797)

[5.2 Gestion de mot de passe 29](#_Toc126144798)

[5.2.1 Hachage 29](#_Toc126144799)

[5.2.2 Salage 30](#_Toc126144800)

[5.3 Identifiants UUID 31](#_Toc126144801)

[5.4 Accès aux services et cookie session 32](#_Toc126144802)

[5.4.1 Besoin 32](#_Toc126144803)

[5.4.2 Principe 32](#_Toc126144804)

[5.4.3 Utilisation des cookies par SOS Immo 33](#_Toc126144805)

[5.5 Externalisation des paramètres de l’application 34](#_Toc126144806)

[5.6 Archivage 35](#_Toc126144807)

[5.7 Sauvegarde 36](#_Toc126144808)

[5.8 Utilisation de triggers 36](#_Toc126144809)

[6 Tests 38](#_Toc126144810)

[6.1 Tests unitaires 38](#_Toc126144811)

[6.2 Tests fonctionnels 39](#_Toc126144812)

[ Base de tests 39](#_Toc126144813)

[7 Organisation 40](#_Toc126144814)

[8 Perspectives d’évolutions 41](#_Toc126144815)

[8.1 Fonctionnelles 41](#_Toc126144816)

[8.2 Techniques 42](#_Toc126144817)

[9 Bilan 43](#_Toc126144818)

[9.1 Ce que je savais déjà 43](#_Toc126144819)

[9.2 Découvertes de la formation mises en pratique pour le projet 43](#_Toc126144820)

[9.3 Recherches personnelles. 44](#_Toc126144821)

[9.4 Point sur l’application à aujourd’hui 44](#_Toc126144822)

[9.5 Bilan personnel - remerciements 44](#_Toc126144823)

[ANNEXES 45](#_Toc126144824)

[Annexe 1 – Ecran du pipeline GitHub 45](#_Toc126144825)

[Annexe 2 – exemple de scénario complexe 46](#_Toc126144826)

[Annexe 3 - Tests Postman 48](#_Toc126144827)

[Annexe 4 - Collections Postman 49](#_Toc126144828)

[Annexe 5 - schéma déploiement 50](#_Toc126144829)

Introduction

Je m’appelle Sophie Joffre. Comptable de carrière, j’avais envie de changer de voie professionnelle. Etant titulaire d’un BTS Informatique de gestion, j’ai décidé de revenir à mes premières amours, l’informatique, la partie software : le développement et la réalisation d’applications.

Ainsi, et pendant près d’1 an, j’ai suivi un cursus de formation pour apprendre le métier de développeur en informatique. Cette formation m’a été proposée par mon employeur Société Générale, (en partenariat avec l’Ecole Efrei Paris), dans le cadre d’un reskilling.

L’investissement pour l’entreprise et pour moi-même a été énorme et je suis heureuse de pouvoir maintenant présenter mon projet de fin d’étude à l’examen du Titre Professionnel de « Concepteur développeur d’applications ».

Le projet s’appelle SOS Immo. C’est un outil de gestion de tickets, ces tickets étant des incidents techniques pouvant survenir dans un immeuble de bureau de grande taille.

* Chaque occupant de l’immeuble pourra signaler et suivre un incident.
* Un technicien pourra prendre en charge et signaler la fin de l’intervention.
* Les managers pourront savoir quels techniciens s’occupent de quels incidents.

Le choix d’une telle application s’est imposé pour plusieurs raisons :

* Elle correspond parfaitement à mes appétences : mettre en place des applications qui facilitent la vie des gens et les aident dans leur travail.
* Après 25 ans à la Défense dont environ 10 ans comme élue au CSSCT, anciennement CHSCT[[1]](#footnote-1), je connais le fonctionnement « technique » d’un immeuble de ce genre.
* Ce projet contient les éléments nécessaires au passage du RNCP.

Abstract

My name is Sophie. I was trained for one year to become a software developer.

This training was proposed to me by my employer Société Générale thanks to a partenership with the Effrei Paris school. For me, it’s the opportunity to change professional orientation.

This is a big investment for my company and myself and, now, I’m glad to present my end-of-study project for the Professional Title of « Designer and developer of applications » exam.

It is called SOS Immo. It consists of managing technical indicents that may occur in a tall office building.

With this tool, every occupant of the building can report an incident and follow up. After that, A technician can solve the issue and report the end of the intervention. The technicials manager can use the tool to know which technician takes care of which indicent.

# Contexte SOS Immo

## Fonctionnement des services immobiliers

Bien sûr, je ne peux pas affirmer que toutes les entreprises fonctionnent sur le même modèle.

Je prends ici l’exemple que je connais.

Les tours de bureaux sont de véritables fourmilières qui abritent de nombreux corps de métiers. Le service de gestion d’immeuble nécessite une organisation bien rodée afin qu’employés de bureau et ouvriers puissent effectuer leur travail sans se gêner mutuellement. Les uns, la plupart du temps, totalement ignorants de la présence des autres.

L’entreprise occupante (propriétaire ou locataire de l’immeuble) n’intervient pas elle-même lorsque survient un incident technique. Elle utilise les services d’entreprises prestataires spécialisées dans leur domaine (plomberie, ascensoriste, ménage, etc.). Des contrats sont passés à la suite d’appels d’offre. Ils sont signés pour plusieurs années. Ces entreprises retenues emploient des équipes qui ont leur propre ligne hiérarchique et rendent compte à l’entreprise occupante.

Quand l’immeuble est assez grand, les techniciens « prestataires » ont, sur place et à demeure, leurs propres vestiaires, bureaux, locaux pour pouvoir intervenir à tout moment dans un délai convenu.

## Transformation de la gestion de suivi d’incident

Historiquement la centralisation des incidents est assurée par un standard téléphonique, des cahiers, post’it, dash board, etc.

Dans un contexte de digitalisation des process et pour gagner en efficacité, on a imaginé une application qui remplacerait ce standard en allant plus loin encore : une plateforme unique pour tous les occupants de l’immeuble, qu’ils soient employés de bureau, intervenants extérieurs ou responsables d’immeuble. L’information serait alors disponible en temps réel à tous les acteurs concernés, sans intermédiaire.

Une plateforme de ce genre est en place aux services centraux de Société Générale depuis quelques années. Je m’en suis inspirée pour ce projet de fin d’étude

Pour construire ce projet, je suis allée rencontrer les premiers concernés : des responsables d’immeuble de grande taille[[2]](#footnote-2). Ils m’ont expliqué comment ils utilisent l’outil et leurs besoins en la matière.

## Objectifs

Objectif principal

Aider au suivi des incidents.

Objectifs secondaires

Aider les services techniques à identifier les points de vigilances : prestataires potentiellement en sous-effectif, étages ou types d’incidents particulièrement récurrents.

## Besoins de l’application

Les équipes prestataires sont autonomes mais le service de gestion d’immeuble reste maître et garde le dernier mot. Le service de gestion d’immeuble est également responsable des données et droits d’accès de l’application. Il a donc un profil Admin.

N’importe quel occupant de l’immeuble, quel que soit son profil, doit pouvoir faire un signalement et le suivre jusqu’à résolution. Un incident doit être résolu dans un délai convenu, ici 24 heures.

L’affectation d’un technicien à un incident peut se faire de 3 façons :

* Auto-affectation.
* Affectation par un responsable de l’équipe prestataire.
* Affectation par le service de gestion d’immeuble.

L’attribution d’un incident à une entreprise prestataire se fait de 2 façons :

* Au signalement, calcul par l’outil d’après le type d’incident.
* Réattribution par le service de gestion d’immeuble, si besoin.

À tout moment, de la création à la clôture d’un incident, un commentaire doit pouvoir être ajouté.

* Usager : sur ses signalements.
* Technicien : sur les incidents qui lui sont affectés.
* Valideur : sur les incidents affectés à son équipe.
* Admin : sur n’importe quel incident.

Seuls les incidents techniques courant sont suivis. Pas les gros travaux, ni les travaux planifiés.

L’outil ne doit contenir qu’un minimum d’informations personnelles. Il est administré par le service d’immeuble, pas le service d’achat ni les ressources humaines. Il ne s’agit donc pas de gérer les contrats.

Les occupants n’ont pas accès directement à leurs données via l’application. Ces données devront d’ailleurs être réduit au strict minimum pour le suivi des incidents. En amont, l’administrateur a connaissance de chaque nouvelle arrivée et créée les comptes avec droits d’accès selon le profil de l’utilisateur. Ce process n’est pas géré par l’application.

# Conception

## Use Case



## Profils utilisateurs

**2 profils internes** – salariés de l’entreprise occupantes

* Admin :
  + A la fois service de gestion d’immeuble et administrateur de l’application.
  + A tous les droits sur l’outil.
* Usager :
  + Tout autre employé interne.
  + Utilise l’outil pour signaler les incidents et valider la fin d’intervention.
  + Seul profil non « technique ».

**2 profils externes** – salariés des entreprises prestataires[[3]](#footnote-3)

* Techniciens
  + En charge de la résolution des incidents.
  + Utilise l’outil pour suivre les incidents non résolus le concernant.
* Valideur :
  + Responsable d’une équipe de techniciens.
  + Utilise l’outil pour la coordination de son équipe.

# Architecture d’application

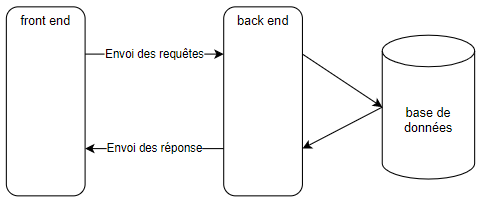
Une application moderne doit permettre à plusieurs utilisateurs, chacun sur leur propre machine, de communiquer entre eux et d’accéder aux mêmes données en même temps.

L’architecture SOS Immo est composée de 2 parties distinctes :

* Le **Front end** est la partie visible du programme, l’interface graphique, ou IHM[[4]](#footnote-4), qui permet de naviguer entre les différentes fonctionnalités.
* Le **Back end** est l’ensemble des règles métier qui alimente le front.

Il s’occupe de tous les calculs et du stockage des données.

Ainsi tous les utilisateurs ont le même niveau d’information.



La communication entre back et front se fait au moyen d’adresses URL, via un navigateur web au choix de l’utilisateur.

Les 2 projets sont développés en Java Script. Ils ont nécessité l’installation de packages, npm et node notamment. Ces packages permettent à leur tour l’installation et l’appel aux librairies JavaScript.

## Front End

Le Front de SOS Immo utilise la librairie React qui permet la création d’interfaces utilisateurs interactives en SPA[[5]](#footnote-5).



Le principe d’un SPA est un site avec 1 seule page et une multitude de composants pour l’habiller.

[Le but est d’éviter le chargement d’une nouvelle page à chaque action demandée.](https://fr.wikipedia.org/wiki/Application_web_monopage)

Pour fonctionner, React utilise certains éléments spécifiques, tels que les composants et les useStates. Par convention, la page est nommée « index.js ». Elle appelle un composant unique « App.js » qui servira de gare de triage pour les autres composants.

Index.js – page unique du front end

Une image contenant texte

Description générée automatiquement

#### Composant

Un composant est un élément de page. Il contient

* Du code html pour l’affichage.

Le rendu visuel peut être complexe ou très simple : un tableau, un bouton, un formulaire, etc.

* Des fonctions qui lui permettent d’interagir avec le Back.
* Des variables ou autres éléments propres à React, tels que des useStates.

Ces composants sont déclenchés si les conditions sont réunies et peuvent s’appeler les uns les autres. On parle alors de parents et d’enfants.

#### UseState ou variable d’état

Elément React composé d’une variable[[6]](#footnote-6) (l’état) et d’une fonction (pour la mise à jour de l’état). Il a la particularité d’être transmissibles d’un composant parent à un composant enfant, et sa valeur peut être mise à jour par un enfant. Quand un useState est mis à jour, le composant parent dans lequel il a été déclaré est recalculé.

L’état du useState est préservé tant que le composant est actif[[7]](#footnote-7).

Lors de l’appel à un enfant, le parent lui transmet des attributs (qui peuvent ou non être des useStates). L’ensemble de ces attributs est récupéré par l’enfant dans un seul objet « props ».

App.js – composant principal

1. Import : librairie React et bootstrap, fichier .css et extrait des composants enfants appelés par App.js

Une image contenant texte

Description générée automatiquement

1. Déclaration du useState. Pour des raisons pratiques j’ai créé un seul useState varGlob, de type objet, contenant les éléments conditionnant l’appel aux composants enfants et donc susceptibles d’être mis à jour par ces enfants.

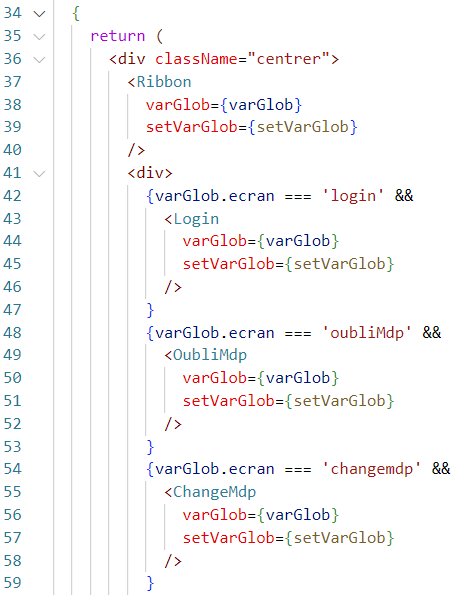
Une image contenant texte

Description générée automatiquement

1. Retour du code http : lancement de 2 composants à la fois

* « Ribbon.js » automatiquement
* Un autre composant conditionné aux valeurs de varGlob.

Tous les composants enfants reçoivent en attribut le useState varGlob. Sa mise à jour modifie les conditions de lancement du second composant.



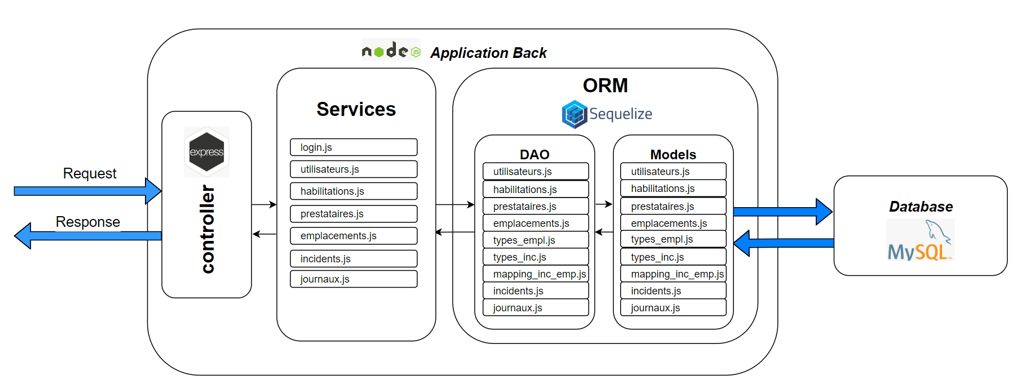
#### Dans notre application : mise en pratique

|  |  |  |  |
| --- | --- | --- | --- |
| Etape | varGlob | Composants actifs | Visuel |
| **Démarrage** | ecran : «login»  nom : «»  profil : «» | Ribbon.js  Login.js[[8]](#footnote-8) |  |
| varGlob.ecran est initialisé : « login ».  Déclenchement des composants Ribbon et Login. | | | |
| **Authentification réussie** | ecran : «Menu»  nom : «toto»  profil : «usager» | Ribbon.js  Menu.js[[9]](#footnote-9) |  |
| varGlobal.ecran est mis à jour : « Menu ».  Déclenchement des composants Ribbon et Menu, à la place de Login. | | | |

La sélection d’un bouton met de nouveau à jour varGlobal.ecran…

## Back End – structure à couches

Le back est un serveur NodeJS, contrôlé par la librairie Express. Il est conçu avec une structure à couches, chacune ayant son propre rôle dans l’organisation globale du Back.



**Fichier «controller»**

Le cerveau de l’application.

Son rôle est de recevoir les requêtes/demandes du Front (via des adresses url) et de les traiter avec les fonctions adéquates.

**Couche «services»**

Fonctions appelées par le controller. Il y en a donc 1 par adresse url. On parle aussi d’API[[10]](#footnote-10))

J’ai regroupé les services par catégorie (login, utilisateurs, incidents) et logé chaque catégorie dans un fichier .js distinct.

**Couche « persistance »**

Les services n’interrogent pas directement la base de données.

L’interaction entre les 2 est assurée par la librairie Sequelize, qui est un framework permettant de gérer l’accès à la donnée et la persistance dans une base[[11]](#footnote-11).

C’est un système ORM[[12]](#footnote-12). Il possède les caractéristiques suivantes :

* Conversion des tables de la base de données en autant de classes objets.

Chaque table est une classe d’objet. Chacun colonne est 1 attribut de l’objet.

Ces classes sont logées dans les fichiers Models. J’ai choisi de créer 1 fichier model par classe, donc par table.

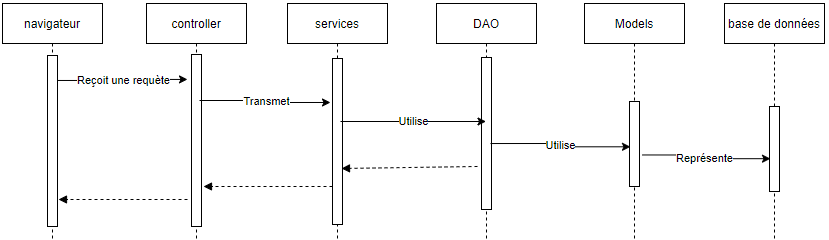
* Fonctions propres, dites DAO[[13]](#footnote-13), pour récupérer les données de la base, les convertir en objet et vice versa : un objet peut être converti en enregistrement. Les DAO évitent au maximum la saisie de requêtes SQL, ce qui rend le code plus lisible. Et, quand ces requêtes sont inévitables, elles sont paramétrées pour empêcher les problèmes d’injections SQL.

J’ai mis toutes ces fonctions DAO dans des fichiers distincts des Models, pour clarifier le code.

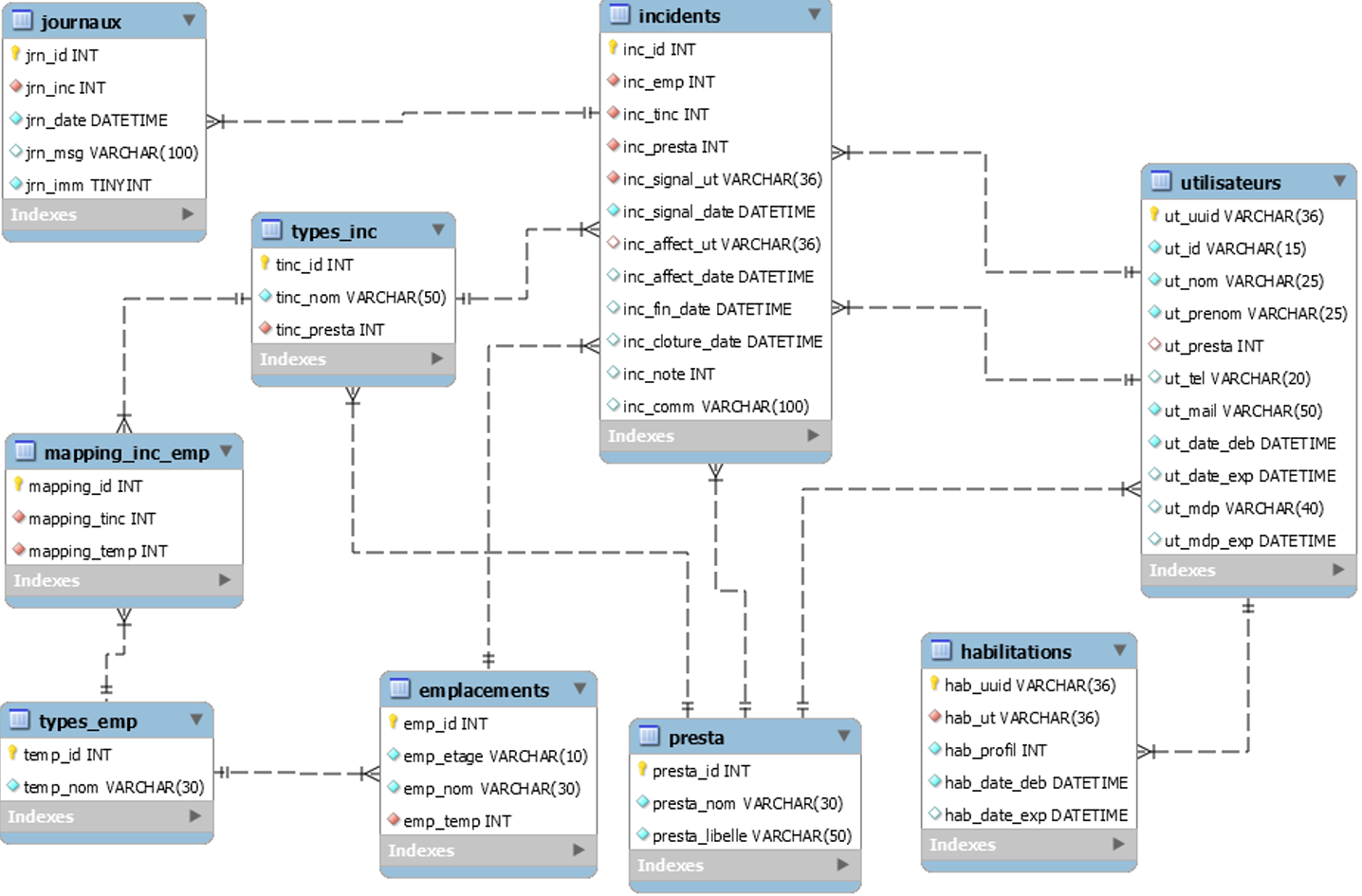
**Base de données**

Pour fonctionner, le Back a besoin d’une mémoire. Celle-ci est stockée dans une base de données MySQL.

**En conclusion**



## Base de données



Après avoir listé un premier dictionnaire de données, j’ai établi un modèle de base de données qui s’est étoffé avec le temps.

Puis j’ai créé une base de données MySQL nommée « sos\_immo » alimentée manuellement la 1ère fois.

Toutes les clés primaires sont des entiers auto-incrémentés sauf pour 2 d’entre-elles, pour lesquelles j’ai opté pour un UUID. Je reviendrai sur cette distinction dans le chapitre sur la [sécurité](#_Identifiants_UUID).

Par la suite la base a été étoffée par l’ajout de 4 tables « archives ».

En réalité, 3 bases de données identiques sont créées :

1. « sos\_immo » qui est la base de production finale.
2. « sos\_immo\_test » qui sert pour tester les mises à jour back et front avant leur mise en production.
3. « sos\_immo\_sauv » qui est le reflet de la base de production.

Tout cela sera détaillé dans les chapitres [Achivage et sauvegarde](#_Archivage_et_sauvegarde) et [Base de tests](#_Base_de_tests).

#### Utilisateurs - Habilitations

Tous les occupants de l’immeuble, internes ou externes, sont des utilisateurs. Dans la table Utilisateurs, la distinction se verra au niveau du champs ut\_presta, contenant l’identifiant de la table Presta (vide pour l’utilisateur interne).

Marginalement, le profil d’un utilisateur peut être amené à évoluer (un technicien devient valideur, un usager devient admin…). Le cas échéant, il faudra créer une nouvelle habilitation (en date jour) et renseigner la date d’expiration de l’habilitation précédente (même date).

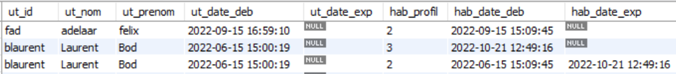
Pas de table pour les niveaux d’habilitations. La liste de ces niveaux est « en dur » dans le code du programme.

1. Utilisateur inactivé
2. Usager (externe)
3. Technicien (interne)
4. Valideur (externe)
5. Admin (interne)

Un utilisateur interne (champs ut\_presta vide) ne peut pas accéder aux profils 2 et 3.

De même, un utilisateur externe (champs ut\_presta rempli) n’a pas accès aux profils 1 et 4.

Extrait MySQL : Utilisateurs et historique d’habilitations



Fad est Technicien depuis le 15/09/2022

Blaurent a été Technicien du 15/06/2022 au 21/10/2022 et est Valideur depuis.

Un utilisateur ayant quitté l’entreprise peut être amené à revenir. Ce sera surtout le cas pour les prestataires.

Son compte ne sera donc pas supprimé immédiatement. Il sera dans un premier temps inactivé. Pour cela le profil 0 lui sera affecté et son mot de passe détruit. Les comptes inactivés devront être détruits au maximum 36 mois après inactivation.

À tout moment, un Admin ou valideur peut affecter n’importe quel technicien (de la spécialité) à un incident.

Un admin peut clôturer automatiquement un incident résolu à la place de l’usager.

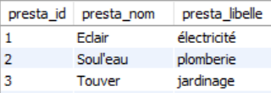
Ainsi, l’inactivation d’un utilisateur n’a pas d’impact sur le suivi des incidents. Il est tout de même nécessaire d’avoir au moins 1 compte Admin actif.

#### Prestataires - Types d’incidents

Comme on peut le constater, il n’y a pas de données d’entrée/sortie des entreprises prestataires. Si une entreprise perd le contrat, il suffira de remplacer son nom par un autre dans la table « presta ». Nous sommes dans un outil de gestion d’incident. Le nom exact de l’entreprise a assez peu d’importance. Ce qui compte, c’est qu’elle soit rattachée aux bons types d’incidents et que ses techniciens/valideurs soient bien identifiés[[14]](#footnote-14).

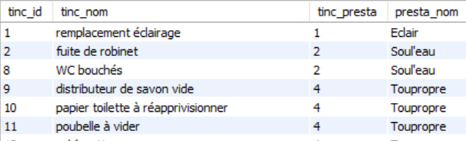
Extrait MySQL : Presta - utilisateurs

Une image contenant table

Description générée automatiquement 

Chaque type d’incidents ne peut être traité que par une entreprise prestataire en particulier, ces entreprises étant spécialisées.

Extrait MySQL : Presta – types d’incidents



#### Emplacements – Types d’emplacements

Chaque emplacement appartient à un type d’emplacement particulier. Il ne peut pas arriver n’importe quel type d’incident dans n’importe quel type d’emplacement.

Extrait MySQL : Emplacements - types d’emplacements

Une image contenant texte, capture d’écran, intérieur

Description générée automatiquement

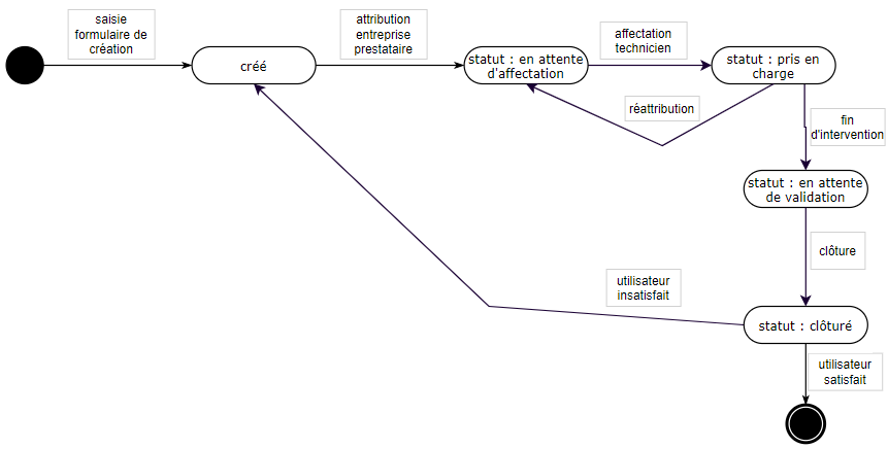
Extrait MySQL : Mapping types d’emplacements-types d’incidents

Une image contenant table

Description générée automatiquement

#### Incidents

Cycle de vie d’un incident



Tous les incidents suivent le même parcourt et passent par 4 états, ou statuts, successifs.

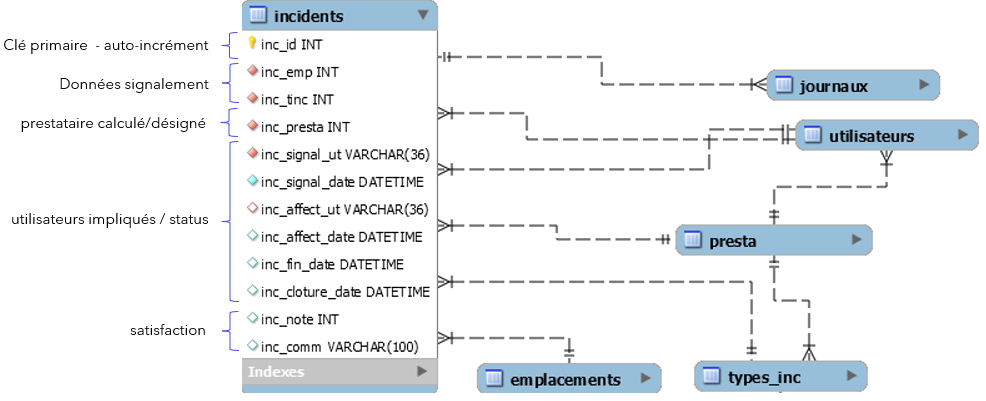
* A sa création, 1 incident est en attente d’affection. Personne ne s’occupe encore de lui.
* Puis un technicien lui est affecté. Il est donc pris en charge.
* A la fin de l’intervention, l’incident est en attente de validation par l’utilisateur auteur du signalement.
* Après validation, l’incident est clôturé.

Le process s’arrête là si l’utilisateur est satisfait du résultat de l’intervention. S’il est mécontent, il le signale et un nouvel incident est automatiquement généré avec les mêmes paramètres que le précédent, avec le motif d’insatisfaction en commentaire.

Le cycle de vie recommence avec ce nouvel incident jusqu’à résolution du problème.

Tout comme l’incident est le cœur de l’application, la table Incidents est le cœur de la base de données.

Focus table Incidents



* Données signalement

L’utilisateur renseigne un emplacement et un type d’incident, éventuellement un commentaire[[15]](#footnote-15).

* Prestataire

L’entreprise responsable (champ inc\_presta) est dans un premier temps calculé par le back d’après le type d’incident (inc\_tinc). S’il s’avère que le type d’incident a été mal renseigné ou mal évalué lors du signalement, un Admin a la possibilité de mettre ce champ à jour. Cependant, il a été arbitré de garder la trace du signalement original, d’où la nécessité de garder les 2 champs inc\_tinc et inc\_presta.

* Utilisateurs impliqués / statuts

Les 4 champs « Datetime » représentent les 4 statuts du cycle de vie.

On conserve en base 2 liens avec la table Utilisateurs.

* L’auteur du signalement (pour validation).
* Le technicien actuellement affecté (pour le pilotage de l’équipe).

Les noms des différents techniciens et entreprises prestataires qui se sont éventuellement succédé sur 1 même incident ne sont pas conservés.

* Satisfaction usager

Une note et un commentaire laissé par l’usager à la clôture de l’incident.

#### Journaux

Les utilisateurs « techniques » communiquent en laissant des messages et commentaires. Les usagers aussi dans une moindre mesure.

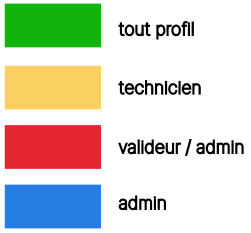
Un message ne concerne qu’un seul incident. Le choix a été arbitré de ne pas en mémoriser l’auteur dans un champs particulier mais plutôt, au besoin, dans le texte du message. Pour plusieurs raisons :

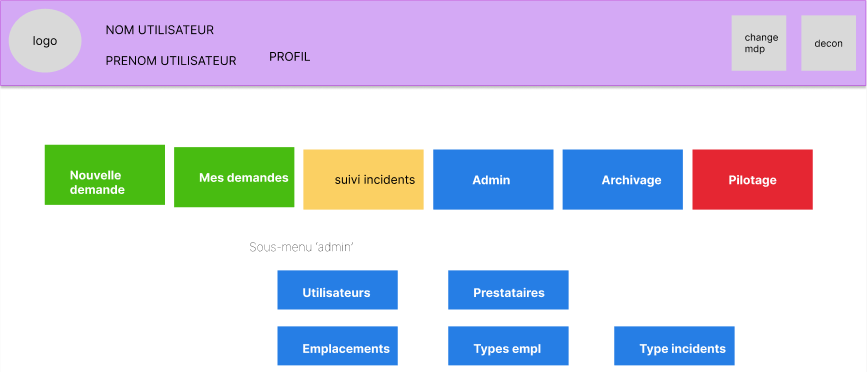
* L’auteur n’est pas forcément une information pertinente.
* Pas de statistique à faire sur cette donnée.
* Inutile de prévoir des espaces dédiés dans l’interface graphique et la base de données.
* Gain de temps : il faudrait interroger systématiquement la table Utilisateurs pour chaque commentaire.

Le booléen jrn\_imm indique si le message est destiné aux équipes techniques (true) ou peut être rendu public (false). Par exemple, l’usager n’a pas à connaitre le nom du technicien.

# Cinématiques utilisateur et écrans principaux

Voici le menu complet des actions possibles de l’application.





Aucun profil n’a accès à l’ensemble de ces actions.

Tous les profils peuvent créer et suivre leurs propres signalements.

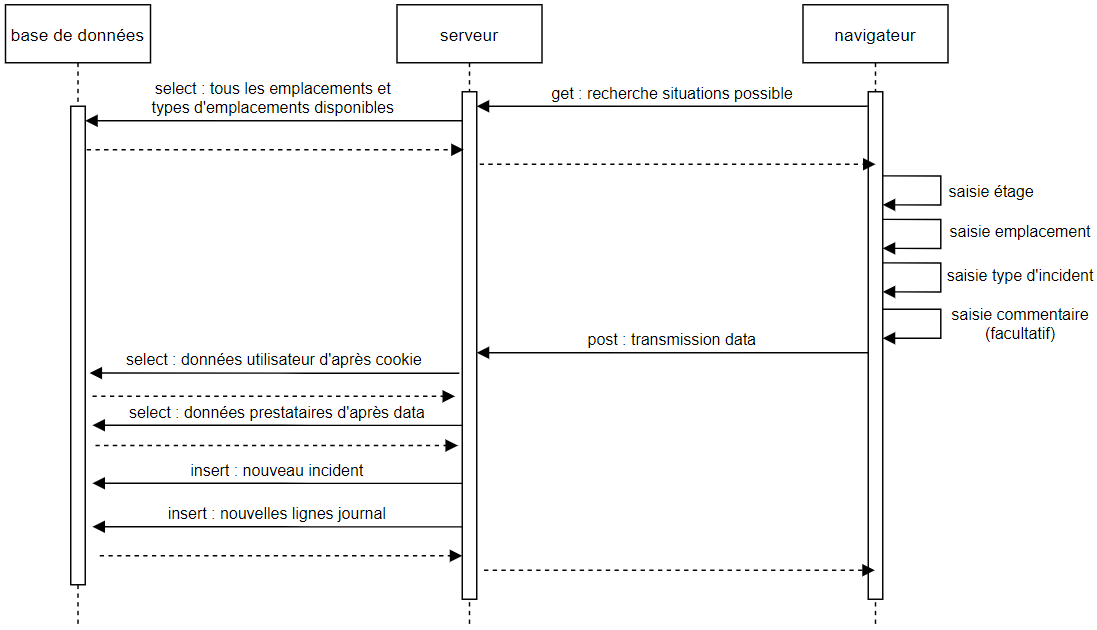
Le suivi d’incident est réservé aux techniciens. Il liste les interventions non terminées de l’entreprise prestataire en question.

L’écran pilotage qui contient tous les incidents listés par catégories.

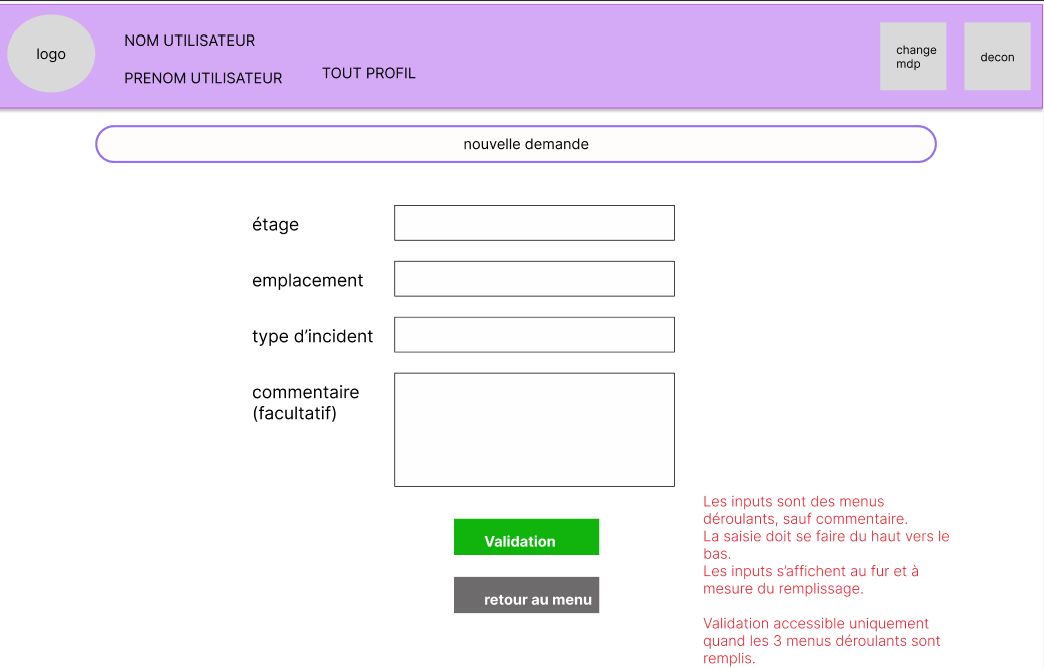
Les boutons bleus sont réservés au profil Admin. Ils donnent accès aux différents écrans de gestion, archivage et sauvegarde de données.

## Cinématique Usager

Diagramme de séquence : création d’un incident



Maquette nouvelle demande



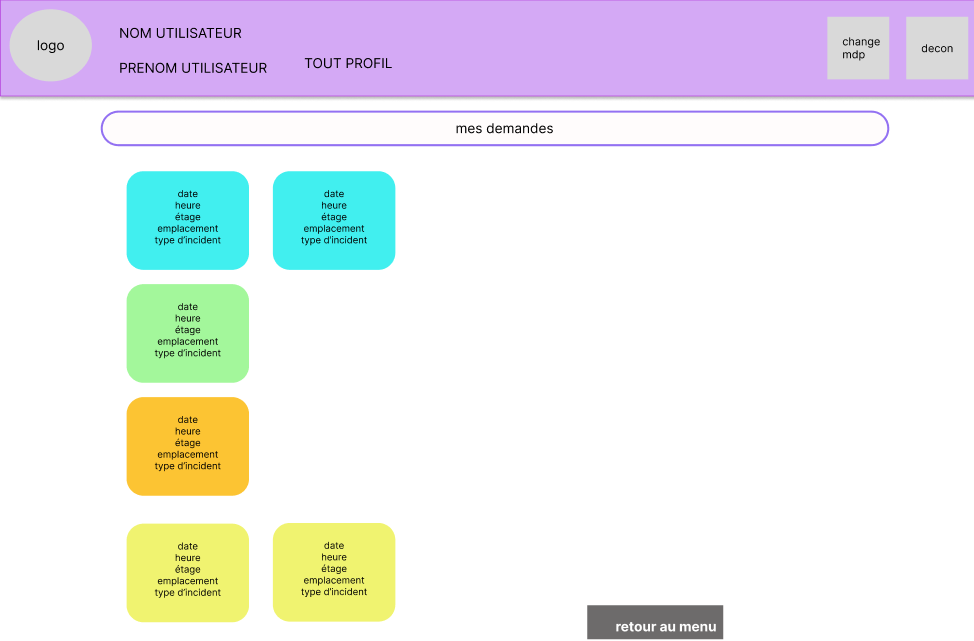
Au lancement de cet écran le back envoie la liste de tous les emplacements possibles dans l’immeuble avec tous les types d’incidents possible dans chacun de ces emplacements.

Le formulaire se compose de listes déroulantes interdépendantes. Ne seront proposés que les emplacements et types d’incidents correspondants à l’étage et à l’emplacement sélectionnés.

A la validation, l’emplacement, le type d’incident et le commentaire sont envoyés au Back. Celui-ci calcule l’entreprise prestataire attribuée à cet incident suivant le type renseigné. Puis sauvegarde le nouvel incident en base.

L’utilisateur est redirigé vers le suivi d’avancement de ses propres signalements.

Maquette mes demandes



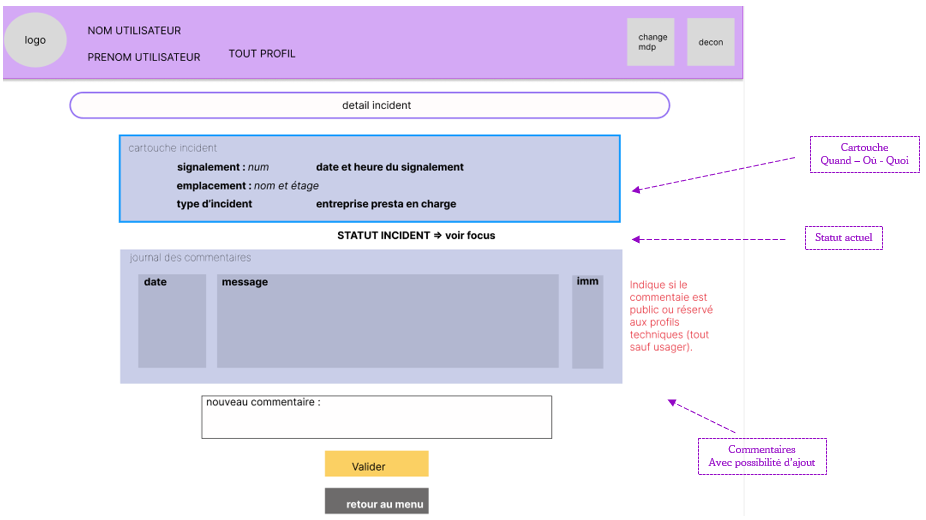
Une image contenant texte

Description générée automatiquement

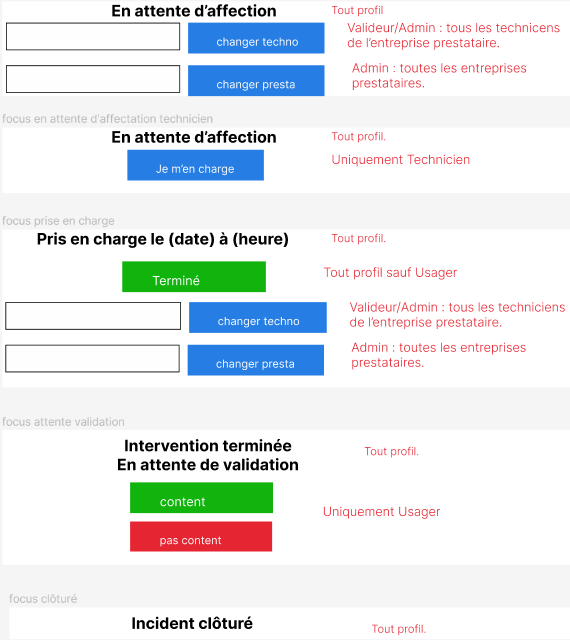
Visuellement j’ai opté pour un dash board de vignettes classées par date et code couleur, selon le statut actuel de l’incident. Cette présentation est adaptée à l’usager, qui fait assez peu de signalements individuellement.

L’utilisateur peut sélectionner une vignette pour consulter le détail de l’incident.

Maquette fiche incident



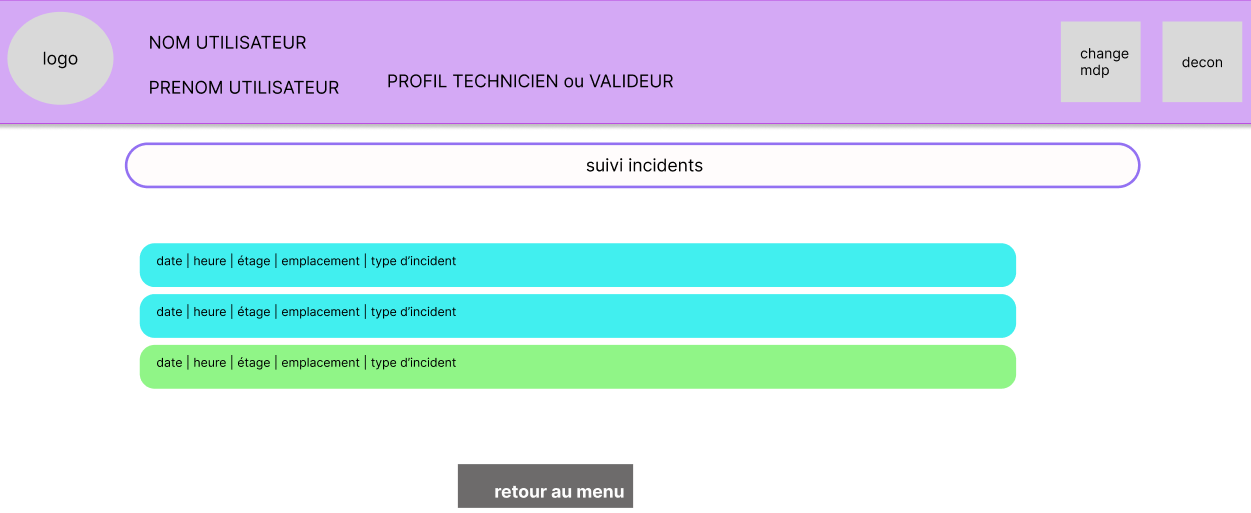
carte d’identité. Quand, où et quoi

Maquettes focus statut incident

Tout le monde peut connaitre le statut. Mais les fonctionnalités associées ne sont accessibles que selon le profil de d’utilisateur.

## Cinématique Technicien

Maquettes Suivi d’incidents



Sur le modèle de l’écran « mes demandes », des vignettes avec un résumé de l’incident et un code couleur pour le statut. De même, à la sélection, on bascule sur la fiche Incident.

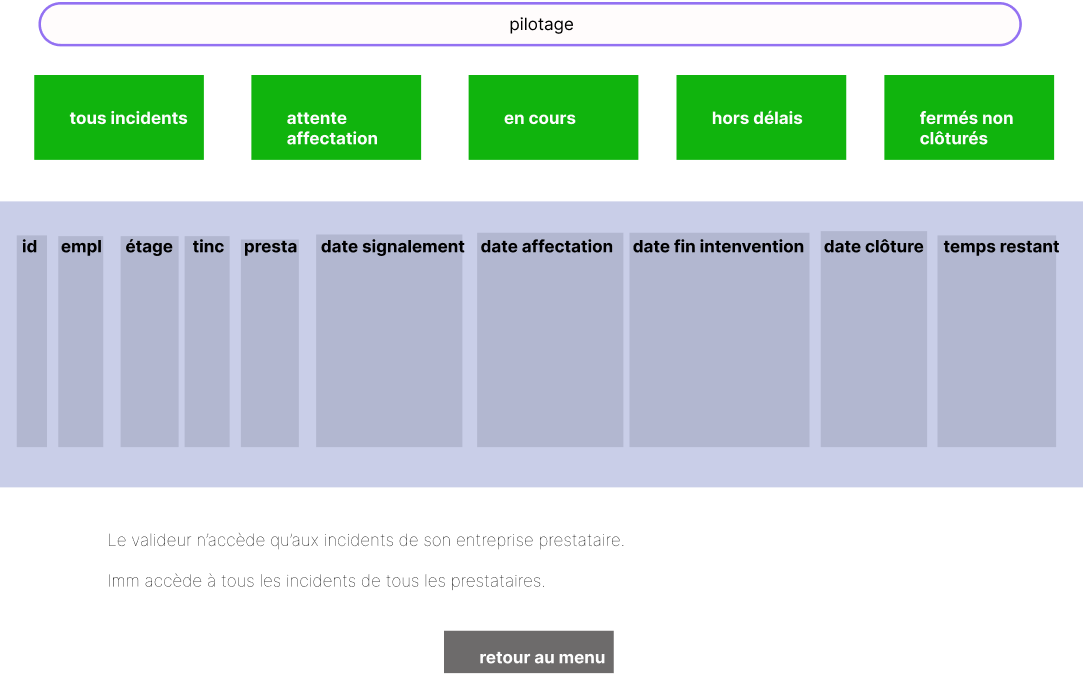
Accès à tous les incidents « En attente d’affectation » de l’entreprise prestataire concernée.

Accès aux incidents « Pris en charge » par le technicien.

Le visuel est un peu différent de « mes demandes ». Il est plus adapté à une utilisation de type « to do list ».

## Cinématique Valideur

Maquettes Pilotage



Incidents classés par catégorie. Ecran disponible pour les profils Valideurs et Admin.

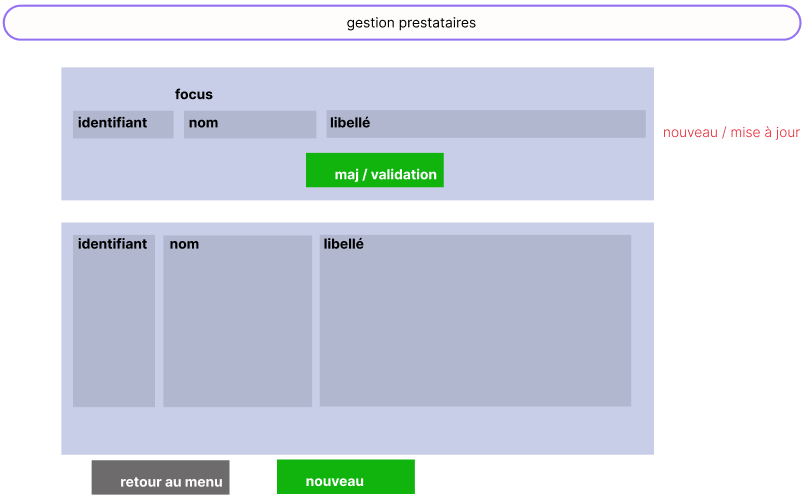
Les Valideurs n’ont accès qu’aux incidents attribués à leur propre entreprise.

## Cinématique Admin

Le profil Admin a tous les droits : il peut directement gérer les données de toutes les tables de la base, à l’exception d’Incidents et Journaux. Il peut intervenir sur les incidents et journaux, par l’intermédiaire de l’écran « Pilotage », qui lui donne accès à tous les incidents.

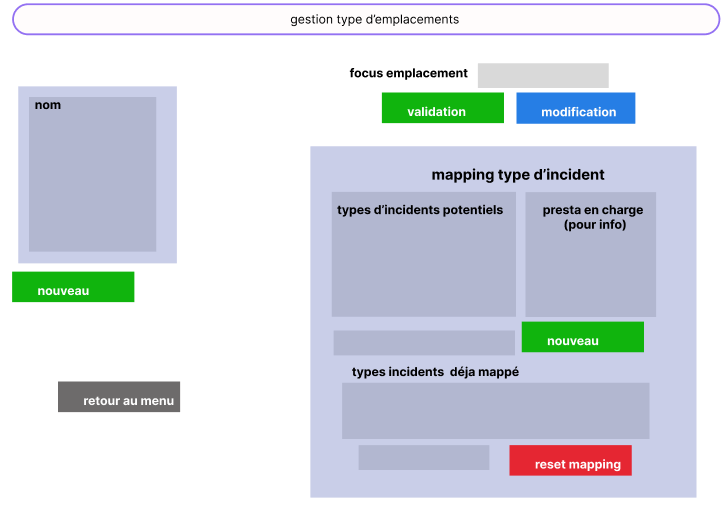
#### Gestion des données nécessaires à la création d’incidents

Maquette gestion prestataires – similaire pour gestion emplacements et types d’incidents



Liste des élément, focus et création d’un nouvel élément sur une même page.

Maquette gestion type d’emplacements

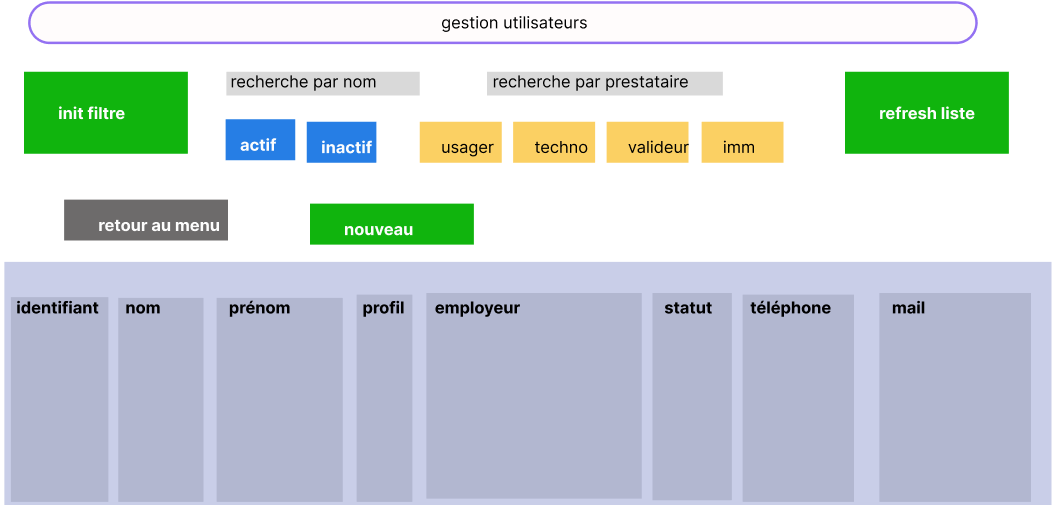


Gestion du mapping type d’emplacement/type d’incident sur la page, en plus du type d’emplacement.

#### Focus gestion Utilisateurs

Les données Utilisateurs nécessiteront de fréquentes mises à jour. Pour simplifier les recherches, on a installé un système de critères dans l’écran « gestion utilisateurs ».

Maquettes « Gestion utilisateurs »

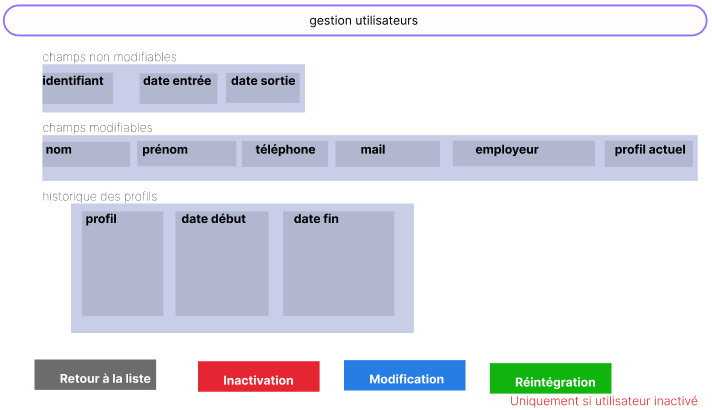


Critères :

* Statut (actif ou inactif)
* Profils
* Entreprise prestataires
* Recherche par mot-clé , présent dans les noms et/ou prénoms et/ou identifiants.

Ces critères peuvent se cumuler.

Maquette fiche utilisateur



Pour une évolution future : une recherche par étage et type d’emplacement peut être envisagée dans l’écran « gestion de emplacements ». Il y a peu de champs mais la liste des emplacements va rapidement être très importante.

# Sécurité

## Utilisateur

### Création

SOS Immo est conçu pour un usage professionnel au sein d’un immeuble de bureaux. Son accessibilité est régie par certaines règles :

* Tous les occupants de l’immeuble peuvent y accéder.
* La gestion des données dont les comptes utilisateurs, est assurée par les administrateurs.

Maquette nouvel utilisateur

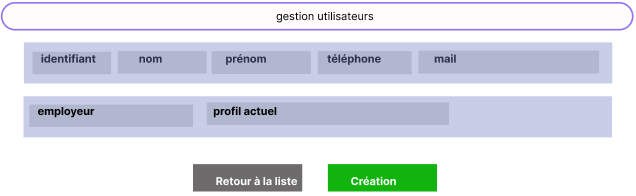
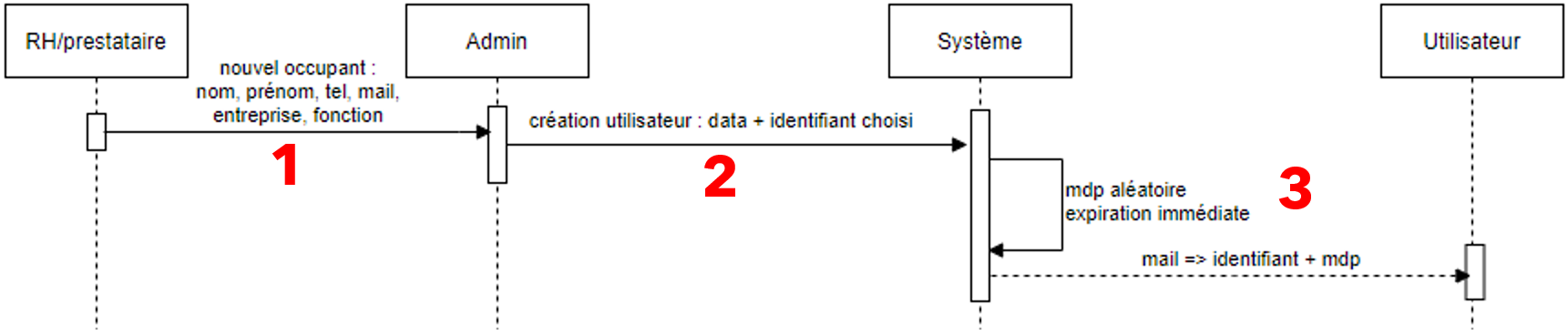


Diagramme de séquence : création d’un utilisateur



1. Un Admin a connaissance en amont d’une nouvelle arrivée. Ce process n’est pas géré dans SOS Immo.

Données obligatoire tous profils : nom, prénom, mail[[16]](#footnote-16), téléphone.

Données obligatoire profils prestataires : nom de l’entreprise, rôle hiérarchique.

1. L’Admin saisi ces données dans l’application.

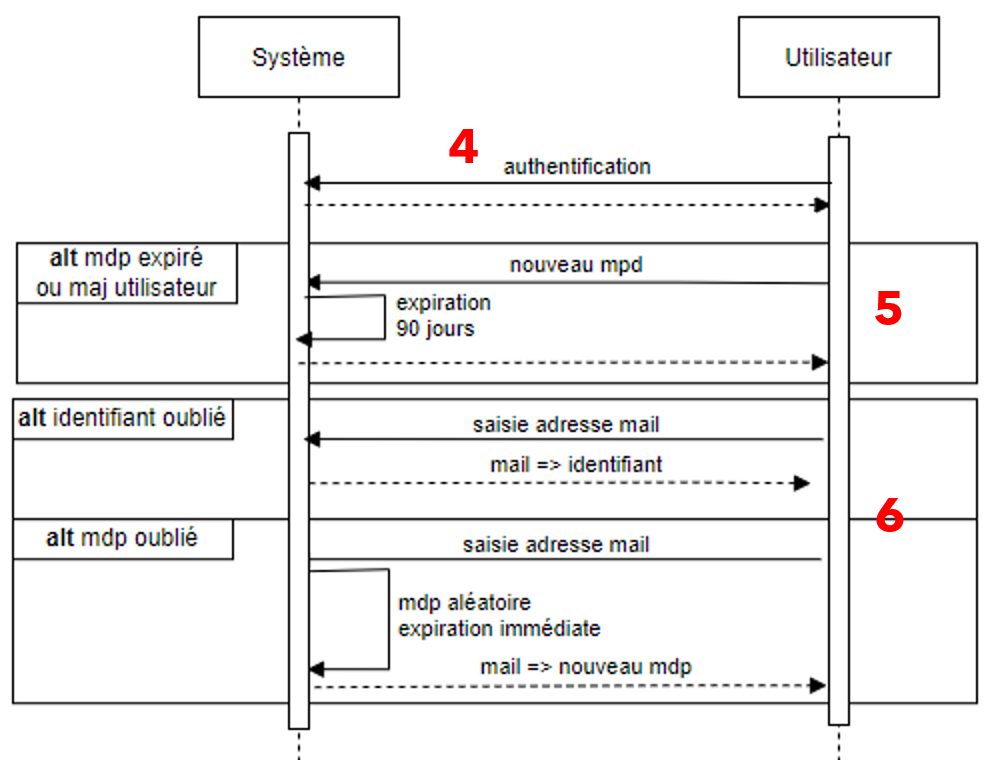
Il complète avec un identifiant unique et le profil, déterminé d’après le service (profil interne) et le rôle (profil externe).

1. Le back calcule un mot de passe aléatoire à expiration immédiate.

Il enregistre le nouvel utilisateur et sa première habilitation en base[[17]](#footnote-17) puis transmet les données d’identification (identifiant unique + mot de passe) au nouvel utilisateur par mail.

### Authentification

Diagramme de séquence : authentification



1. A présent l’utilisateur peut saisir ses données d’authentification.

Maquette authentification

Une image contenant texte

Description générée automatiquement

Le Back contrôle que l’identifiant et le mot de passe fournis appartiennent bien à un même utilisateur.

1. Après la réussite de chaque authentification, la date d’expiration du mot de passe est contrôlée.

Le cas échéant, l’utilisateur est redirigé vers l’écran de changement de mot de passe. Ce sera le cas systématiquement à la première connexion.

Maquette modification mot de passe

Une image contenant texte

Description générée automatiquement

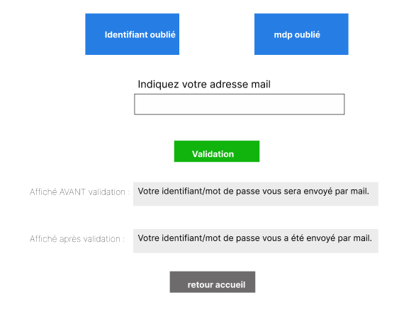
L’utilisateur doit saisir 3 champs :

* Mot de passe actuel.
* Nouveau mot de passe une première fois. Il ne sera valide que s’il contient certaines caractéristiques minimum[[18]](#footnote-18).
* Une seconde fois, validé que s’il est identique à la première saisie.

Le back reçoit les 2 mots de passe. Il enregistre le nouveau en base après avoir contrôlé que l’ancien appartient bien à l’auteur de la modification. Ce nouveau mot de passe aura une validité de 90 jours. L’utilisateur pourra aussi décider de le renouveler à tout moment.

1. L’utilisateur est un être humain, il peut oublier ses données d’authentification et demander à les récupérer.

Maquette identifiant oublié



Le back contrôle que le mail fourni correspond bien à un utilisateur de la base.

Ensuite il transmet par mail à ce même utilisateur, soit son identifiant unique, soit un nouveau mot de passe aléatoire qu’il faudra changer à la première connexion.

## Gestion de mot de passe

Le mot de passe, qu’il ait été généré par l’application ou déterminé par l’utilisateur lui-même n’est pas immédiatement sauvegardé en base. Auparavant il passe par 2 étapes :

### Hachage

Principe

Le hachage est un moyen de crypter un message.

On trouve des fonctions de hachage dans des librairies dédiées. Pour SOS Immo, j’ai utilisé la librairie « sha1 », en JavaScript.

* En entrée : une chaine de caractères – que ce soit une simple lettre ou un document entier.
* En sortie : un nombre hexadécimal de 32 caractères, appelée empreinte numérique.

Avantages

* L’empreinte est particulièrement difficile à retenir et donc adaptée aux données confidentielles. De plus, la taille du champ peut être facilement calibrée dans la base de données.
* Le cryptage est à sens unique. Il n’existe pas de fonction de dé-hachage.
* Quelle que soit la librairie et quel que soit le langage utilisé l’empreinte sera toujours la même et totalement spécifique. Deux documents (même très proche) produiront toujours les deux empreintes différentes.
* Le mot de passe est impossible à reconstituer. En cas d’oubli de l’utilisateur, le seul moyen d’entrer dans l’application est d’en générer un nouveau.

Limites

* Les humains sont assez prévisibles : un programme de craquage peut être utilisé pour retrouver les empreintes. De plus, des dictionnaires entiers de mots de passe vs empreintes sont disponibles plus ou moins facilement.
* Pour contrer ce défaut, des caractéristiques minimums pour les mots de passe sont exigées. Malgré tout, nous avons souvent tendance à utiliser les mêmes types de séquence (1 majuscule suivie de minuscules, le tout formant un mot, en terminant par les chiffres). Le hachage ralentira tout de même considérablement un programme de craquage.

### Salage

Pour ralentir encore le travail des hackers, on procède également au salage du mot de passe. C’est un procédé purement mécanique : on ajoute à la chaine à crypter une autre chaine (de préférence unique) avant le hachage.

J’ai choisi de concaténer l’identifiant unique avec le mot de passe avant de hacher le tout.

* Un programme de craquage devra connaître l’identifiant de l’utilisateur pour pouvoir travailler.
* Plusieurs utilisateurs peuvent utiliser un même mot passe, les empreintes sauvegardées dans la base seront forcément différentes.

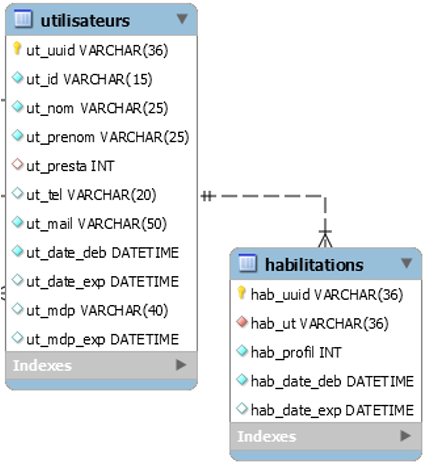
Exemples : chaînes à crypter et empreintes numérique calculées par la fonction « sha1 »



## Identifiants UUID

Il n’y a pas de données vraiment confidentielles dans la base, comme les coordonnées des prestataires par exemple. C’est pourquoi pour la plupart des tables ont comme clé primaire un entier auto-incrémenté, très simple à mettre en place.

Extrait de la base de données



Les quelques données un peu sensibles se trouvent dans les tables Utilisateurs (noms, coordonnées professionnelles, mots de passes) et Habilitions (profil).

Pour ces tables, j’ai choisi un UUID[[19]](#footnote-19) comme clé primaire.

Tout comme l’empreinte de hachage, l’UUID est un nombre hexadécimal de 32 caractères. A la différence de l’empreinte, il n’est pas calculé d’après une chaîne mais généré aléatoirement par une fonction ad oc. Ici j’ai utilisé la librairie « uuidv4 », en JavaScript. La complexité de l’UUID lui garantit, quasiment, d’être unique.

L’intérêt d’utiliser un UUID comme clé primaire est encore une fois de ralentir les hackers. Si un hacker découvre l’url nécessitant un identifiant utilisateur, il aura moins de chance d’arriver à ses fins avec « 06bad5fe-8173-47df-8831-77bccf20f3f9 » ou « 94de4d95-e933-46aa-b94a-99366e47b1a2 » qu’avec « 1 », « 2 », « 789 » ou « 42 ».

Extrait MySQL : habilitations et Utilisateurs

Une image contenant texte, table

Description générée automatiquement

## Accès aux services et cookie session

### Besoin

Sécuriser les données ne suffit pas. Il faut aussi sécuriser leur accès.

Un premier filtre est fait par le biais de l’interface graphique : c’est le profil de l’utilisateur qui lui donne accès ou non aux différentes fonctionnalités. Mais ce n’est pas suffisant car on peut facilement transmettre des requêtes au serveur sans passer par l’interface graphique.

Il faut qu’à chaque fois qu’un service est sollicité par le serveur, il contrôle : l’auteur de cette requête a-t-il le droit d’obtenir une réponse ?

### Principe

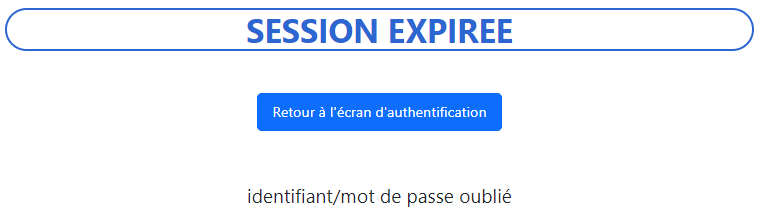
Pour cela, j’ai choisi d’utiliser un cookie de session. Ce cookie est un petit fichier créé par le serveur et stocké dans le navigateur. Ses création, destruction, utilisation doivent être définies dans le serveur. Il peut contenir toutes les informations que le développeur juge nécessaires.

Ici, le serveur crée un cookie de session après la réussite de l’authentification.

Y sont stockés les infos sur l’utilisateur qui seront nécessaires aux futurs contrôles :

* Un booléen[[20]](#footnote-20) qui signale la réussite de l’authentification.
* UUID
* Identifiant unique[[21]](#footnote-21)
* Profil
* Prestaire employeur[[22]](#footnote-22) (le cas échéant, null par défaut)

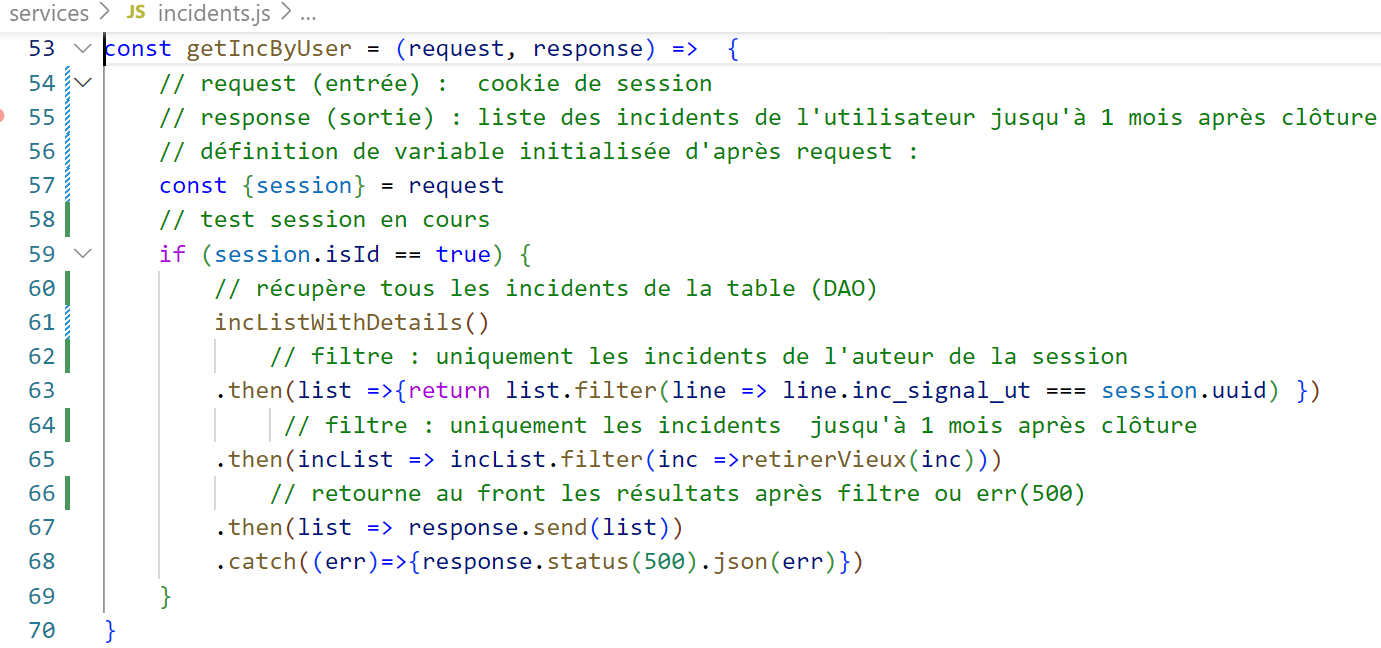
Quand un service est appelé par le serveur, il peut tester le cookie à tout moment pour contrôler la validité de la demande. Si la demande n’est pas valide, l’utilisateur est redirigé vers le message suivant :



Un cookie à une durée de vie limitée qu’il faut paramétrer avant sa création. Pour SOS Immo, j’ai décidé qu’il s’autodétruirait 10 minutes après la dernière connexion à la base.

### Utilisation des cookies par SOS Immo

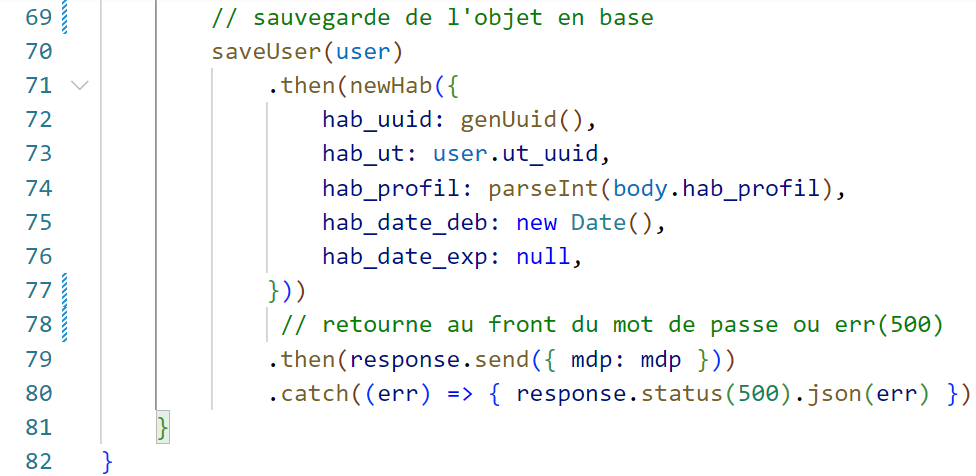
Extrait code : service « getIncByUser »



Cookie testé au début et au cours de process.

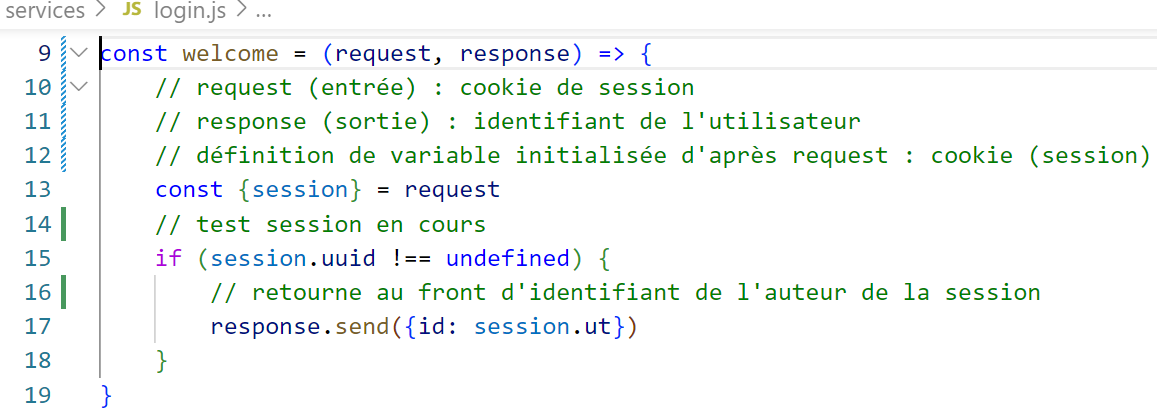
Extrait code : service « creaOneUser »





Cookie testé uniquement au début du process.

Extrait code : service « welcome » - appelé par l’écran d’authentification[[23]](#footnote-23)



Cookie testé au début puis retourné partiellement en fin de process.

## Externalisation des paramètres de l’application

Pour se connecter à une base de données, le back end a besoin de plusieurs informations : type de base (ici MySQL), localisation (IP, ici localhost), nom, utilisateur et mot de passe de la base.

Il est très dangereux d’écrire ces informations, surtout les 3 dernières, directement dans le code.

C’est pourquoi on utilise des variables d’environnement.

Une image contenant texte

Description générée automatiquement

Elles sont enregistrées directement dans les paramètres de la machine. Si un certain nombre de variables d’environnement est déjà défini, on peut autant que nécessaire pour nos propres besoins.

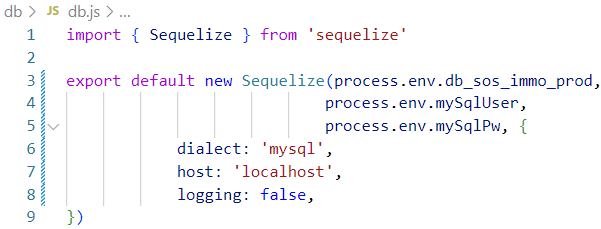
Pour faire fonctionner le back end SOS Immo, les variables suivantes devront être définies au préalable :

* db\_sos\_immo\_prod
* mySqlUser
* mySqlPw

D’autres éléments peuvent être externalisées de la même manière, comme l’adresse et le port du serveur…

En javaScript, une variable d’environnement est appelée par le mot-clé « process.env.nomVariable ».

Extrait code : définition de la base de données rattachée à l’ORM Sequelize



## Archivage

Les tables Incidents et Utilisateurs seront rapidement très lourdes.

* Les incidents clôturés n’ont plus vraiment d’utilité.
* Les comptes des utilisateurs sortis des effectifs (inactifs) ne sont pas supprimés immédiatement, au cas où ils seraient amenés à revenir bientôt (surtout dans le cas des prestataires). Mais au bout d’un moment, on peut considérer qu’ils n’ont plus beaucoup d’utilité non plus.

Il faut pouvoir archiver les incidents et utilisateurs et par conséquent les journaux et habilitations rattachés. C’est-à-dire, mettre de côté, sans pour autant supprimer, les incidents clôturés et les utilisateurs inactifs depuis un certain temps.

* Chaque fois qu’un enregistrement d’1 de ces 4 tables est supprimé, il est recopié à l’identique dans une table « \_arc ».

Les tables « \_arc » ont donc les mêmes champs que les tables d’origine plus 1 champs « \_arc » indiquant à quelle date a eu lieu l’archivage.

Ainsi, on dispose toujours des incidents (avec leurs journaux) et des utilisateurs (avec leur historique d’habilitations) en cas de besoin.

Bien sûr les tables \_arc auront finalement besoin d’être purgées à leur tour. Cette fois, on supprimera définitivement les données. L’archivage et les purges seront gérés par le profil Admin.

Page écran « Archivage »



Les 5 autres tables[[24]](#footnote-24) n’ont pas d’archivage. Il n’est pas pour l’instant prévu de supprimer ces données et un archivage serait de toute façon sans intérêt.

## Sauvegarde

En cas d’accident, il est toujours utile d’avoir une base de réplication, logée dans une autre machine que celle hébergeant la base de production. Pour cela, j’ai créé une base « sos\_immo\_sauv » qui est la réplication de la base de prod « sos\_immo ».

Tout comme « sos\_immo\_test\_sauv » qui est la réplication de la base de prod « sos\_immo\_test ».

Les données sont recopiées, mises à jour et même supprimées dans la base de sauvegarde au fur et à mesure qu’elles le sont dans la base de prod grâce à l’utilisateur de triggers.

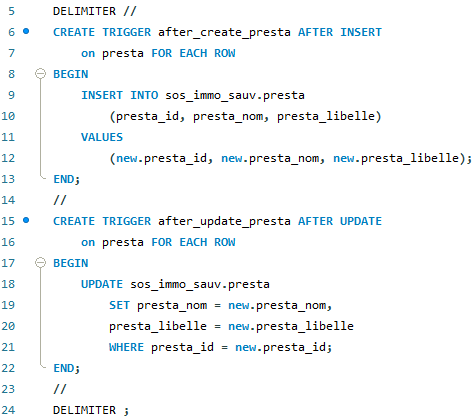
## Utilisation de triggers

Les triggers/déclencheurs sont des scripts SQL rattachés à une base de données et programmés pour faire des actions lorsque survient un évènement précis (insert, update, delete). Si la base venait à disparaitre, ils seraient détruits en même temps.

La gestion de l’archivage et de la sauvegarde est assurée par des triggers rattachés à la base de prod « sos\_immo ».



Script trigger : insert et update de la table Presta - similaires aux autres tables



1 trigger pour chaque évènement Insert et Update.

Pas de trigger pour Delete car il n’est pas prévu de supprimer ces données[[25]](#footnote-25).

Script trigger : delete de la table Utilisateurs- similaire pour les 4 tables avec archivage

Une image contenant texte

Description générée automatiquement

1. Un insert est joué dans la table \_arc.
2. Un autre insert est joué dans la même table de la base de sauvegarde.
3. L’enregistrement est supprimé de la table « utilisateurs » de la base de sauvegarde.

Pas de trigger « update » pour les tables \_arc : on ne modifie pas une donnée archivée.

Une image contenant texte

Description générée automatiquementScript trigger : delete de la table Utilisateurs\_arc – similaire pour les 4 tables \_arc

Une donnée supprimée d’une table d’archive l’est définitivement.

# Tests

Pour mon premier projet, je me suis retrouvée confrontée à un problème :

* Pour le front, comment tester les composants React ?
* Pour le back, comment définir des tests unitaires alors que toutes les fonctions sont des services qui interagissent avec la base de données ?

La réflexion a été intense, j’ai tout de même réussi à réaliser 2 sortes de tests : unitaires pour les fonctions front et des fonctionnels pour le back.

## Tests unitaires

Dans le métier de développeur, les tests unitaires sont réalisés au fur et à mesure que les fonctions sont mises en place.

La librairie Jest, en JavaScript, m’a permis de tester les fonctions front utilisées par les composants pour :

* Déterminer les « class » css du code html d’après certains paramètres.
* La mise en forme des dates.
* Les manipulations un peu fastidieuses et répétitives (ex. nom du profil d’après le numéro, retour Null si un champ est vide, etc.)

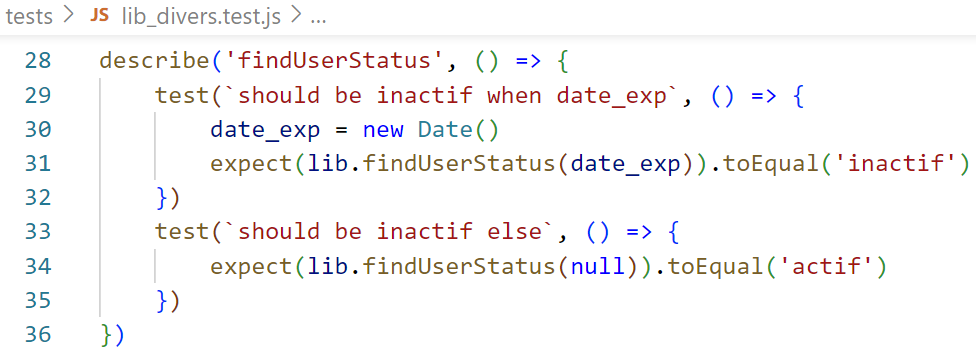
Exemple : la fonction « findUserStatus »

* La fonction retourne un string d’après la date entrée en argument[[26]](#footnote-26).

Une image contenant texte

Description générée automatiquement

* Le test unitaire consiste à contrôler que le string de sortie est bien le bon, suivant que l’argument d’entrée est Null ou pas.



Utilisation d’un Pipeline GitHub

Un pipeline est une série d’instructions lancée automatiquement à la suite d’un élément déclencheur.

L’outil GitHub possède un système de pipeline grâce auquel on peut, entre autres, lancer les tests unitaires après chaque push. Les instructions doivent être saisies en Linux dans un fichier .yml.

Le résultat est accessible dans le repo GitHub (menu Actions) après chaque push.

GitHub m’envoie un mail en cas d’échec.

Extrait code : pipeline Github .yml

Une image contenant texte

Description générée automatiquement

Dans un docker, le pipeline récupère les fichiers du repo GitHub, installe npm et lance les tests unitaires.

[Annexe 1 : écran du pipeline GitHub](#_Annexe_1_–)

## Tests fonctionnels

Côté back, il s’agit de contrôler l’enchaînement des services. Ces tests doivent être effectués dans la base de test. En effet, ils vont consister entre autres à insérer, supprimer et mettre à jour de nouvelles données.

#### Base de tests

Pour éviter les données fantômes et la corruption de données réelles, une base « sos\_immo\_test » est mise en place, avec une structure identique à celle de la base de prod « sos\_immo ».

Avant de lancer les tests fonctionnels :

* Réinitialiser entièrement cette base (tables et jeux de données).
* [Changer le nom de la base cible dans le code du back.](#_Externalisation_des_paramètres)

Un scénario est déterminé sur papier, avec des instructions à suivre à la lettre sur l’interface front.

[Annexe 2 - exemple de scénario complexe](#_Annexe_2_–)

Ce travail est fastidieux et l’interface front a des limites. J’ai utilisé l’outil PostMan qui permet de tester les API directement sans l’intermédiaire de l’interface graphique et d’établir de véritables scénarios d’enchaînement des services. Ces enchaînements sont nommés Collections.

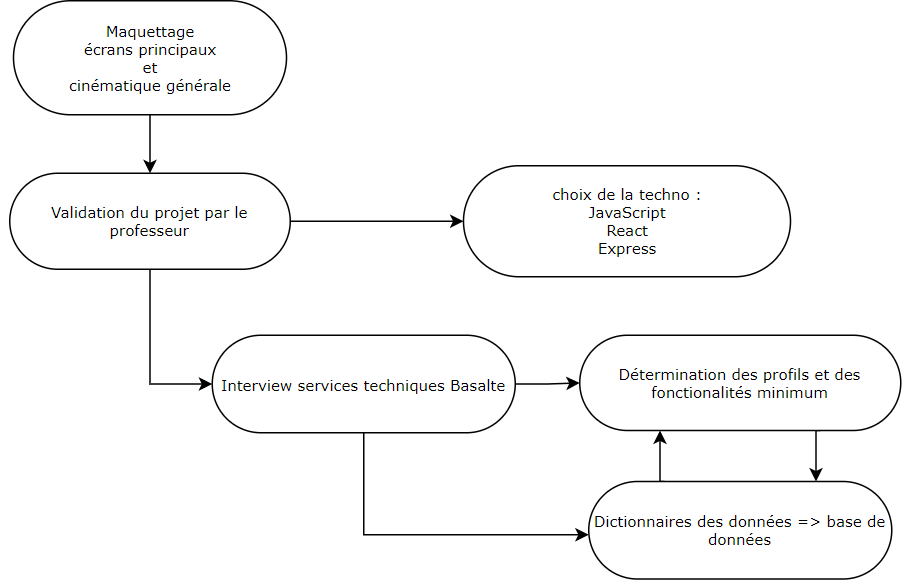
[Annexe 3 - exemples de test Postman](#_Annexe_3_-)

[Annexe 4 - exemples de collections Postman](#_Annexe_4_-_1)

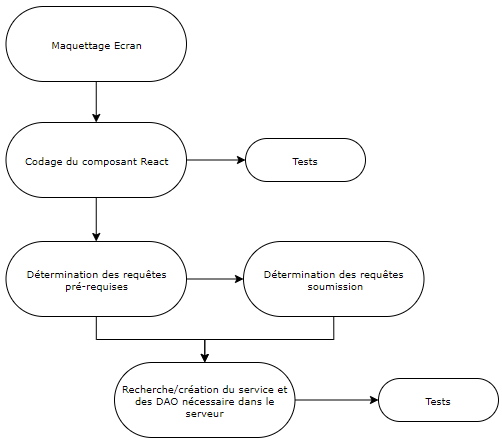
# Organisation

Un certain nombre d’outils ont été utilisés pour l’élaboration de ce projet. Il est inutile de tous les énumérer, voici les principaux logos.





Etapes préparatoires :



Création d’une fonctionnalité :

GitHub – outil de versionning

Conçu pour permettre à plusieurs développeurs de travailler sur les mêmes fichiers de code. Les modifications sont fusionnées (mergés) pour former le code définitif.

GitHub m’a servi de 2 façons :

* Comme sauvegarde du code et des fichiers annexes (présentations, diagrammes, etc.)
* Lancement automatique des tests unitaires via un système de pipeline.

J’ai créé 2 projets/repo distincts : back et front.

<https://github.com/mapette/sos_immo>

<https://github.com/mapette/sos_immo_backend>

# Perspectives d’évolutions

## Fonctionnelles

Certaines fonctionnalités pourraient être poussées ou étudiées :

Relance d’un incident non satisfaisant au lieu de la création d’un nouvel incident

C’est une demande tout à fait légitime qui permettrait un meilleur suivi des incidents. Après avoir étudié la question, j’en ai conclu qu’il faudrait scinder la table « incidents » en 2 tables « incidents » et « interventions », 1 incident pouvant contenir plusieurs interventions.

Réintégration d’un utilisateur inactif

Archivage des utilisateurs et sauvegarde incidents/utilisateurs archivés

Les boutons sont positionnés mais pas encore fonctionnels.

Récupération de données archivées

Réflexion en cours.

Arrivées groupées

Il peut être fastidieux de saisir chaque utilisateur à la main. Surtout en période de fort turn-over. On peut imaginer un tableau formalisé reçu en amont (probablement .csv). Celui-ci sera alors chargé par l’application qui procéderait aux créations. Un fichier log sera généré pour confirmer ou non la réussite du chargement de chaque ligne.

Gestion de plusieurs immeubles

Il suffirait d’ajouter un champs « immeuble » dans la table « emplacements ». A ce stade, une table d’immeubles ne me semble pas pertinente. Lors d’un signalement, il y aurait 1 liste déroulante supplémentaire en début de formulaire.

Enquête de satisfaction

2 champs sont prévus dans la table « incidents » pour que l’usager puisse entrer une note et un commentaire. Ces données pourraient être utilisées par les Admin et les Valideurs.

Statistiques

Pour l’instant l’écran pilotage produit des listes.

Par la suite, il pourrait également produire des graphes et des statistiques[[27]](#footnote-27).

Application mobile

La seule réelle utilité d’une version mobile (smartphone ou tablette) serait pour les techniciens, pour qu’ils n’aient pas besoin de retourner à leur base pour notifier une fin d’intervention et aller directement à la suivante. Les seuls écrans nécessaires seraient alors : login, suivi d’incident, fiche détail incidents, à la rigueur mot de passe oublié.

Le cookie de session aurait aussi besoin d’une durée de vie supérieur, au moins une demi-journée.

Ce serait une seconde appli front totalement distinctes de celle existante. J’aurais besoin d’en connaître davantage sur les librairies à utiliser et le déploiement web.

## Techniques

Pour être au plus proche des conditions d’utilisation réelles, SOS Immo devra être déployé sur une machine distante.

Chaque machine devra contenir :

* 2 conteneurs lancés en permanence qui contiendront respectivement la dernière version des applications back et front.
* La définition des variables d’environnement nécessaires au fonctionnement de l’application.
* La base de données de production.
* Le conteneur Back aura un lien (volume) vers la base de données. Si le conteneur est détruit, la base de données reste intacte.

Ainsi il ne sera pas nécessaire d’installer l’application front sur les ordinateurs de chaque utilisateur.

Pour un déploiement automatique, on utilisera des pipeline GitHub qui s’exécuteront à chaque push.

* Exportation de la nouvelle version du programme sur la machine distante.
* Reconstitution des images Docker si nécessaire.
* Destruction des dockers/conteneurs en cours et lancement de nouveaux conteneurs.

[Annexe 5 - schéma déploiement](#_Annexe_4_-)

# Bilan

## Ce que je savais déjà

Il y a maintenant un peu plus de 30 ans, j’ai entamé des études pour devenir ce qu’on appelait alors analyste programmeur. La vie a fait bifurquée mon parcours professionnel. Je suis maintenant fière de revenir à ce métier, qu’on appelle désormais développeuse[[28]](#footnote-28). En fait je n’ai jamais vraiment arrêté de programmer, que ce soit pour mon plaisir ou dans un cadre professionnel. Le plus souvent c’était des macro Excel et Access, certaines très complexes.

Aussi, j’avais déjà l’habitude

* D’organiser des projets.
  + Interroger les futurs utilisateurs pour recueillir leurs besoins
  + Etablir un dictionnaire de données et en faire une base.
  + Déterminer l’ordre des priorisations.
* De les développer.
  + Algorithmique.
  + Choix de la techno.
* De tester manuellement les fonctionnalités. Ou de les faire tester par les futurs utilisateurs.

A chaque fois j’étais déjà seule au développement. La grosse différence c’est qu’avec SOS Immo, il n’y a pas de responsable pour exprimer les besoins, valider et prioriser les fonctionnalités et finalement valider le projet. Ici j’ai assuré tous les rôles : j’ai porté, conçu, codé, testé, validé ce projet seule[[29]](#footnote-29).

Par curiosité, et dans l’optique d’un changement de carrière, j’avais appris à utiliser HTML et css pour créer une page web. J’avoue que, malgré bien des recherches, je n’avais pas trouvé quoi faire des données d’un formulaire. Ce qui est normal étant donné que je n’avais à ce moment-là aucune notion d’architecture front/back.

## Découvertes de la formation mises en pratique pour le projet

La formation m’a permis de découvrir beaucoup de nouveaux concepts. Certains m’ont servi spécifiquement dans ce projet.

* Utilisation des diagrammes UML.
* Le langage JavaScripts et approfondissement HTML et css.
* Tout ce qui concerne l’architecture client/serveur, front/back, l’utilisation des API et de React.
* La POO[[30]](#footnote-30), notion au départ très floue pour moi et complétement indispensable à n’importe quel développeur.

Le cours Devops m’a appris qu’il est possible d’automatiser les tests unitaires dans GitHub et d’utiliser les variables d’environnement. Il m’a aussi permis à commencer à étudier le déploiement à distance.

## Recherches personnelles.

Les cours que nous avons reçus étaient très complets, mais nous n’avons pas pu approfondir certaines notions qui m’ont manquées après coup.

La première mouture du projet était fonctionnelle mais mal conçue du point de vue architecture back.

* Les couches serveur/service/DAO n’étaient pas bien définies.
* La base de données était directement interrogée par les fonctions services via des requêtes SQL, sans utilisation d’ORM.

J’ai passé beaucoup de temps comprendre exactement le principe de l’ORM et son application. Maintenant le back est beaucoup mieux construit, bien plus clair.

Cela m’a également permis de bien mieux comprendre ce que je fais au quotidien dans mon nouveau travail.

J’ai passé aussi beaucoup de temps à étudier les tests. En théorie le concept est assez évident mais dans la pratique, on ne sait pas forcément quoi tester et comment le faire. Finalement j’ai pu faire des tests unitaires sur les fonctions front utilisées par les composants React, et des tests fonctionnels grâce à Postman.

J’ai découvert les triggers de base de données dans le cadre de mon travail. J’ai pensé qu’ils pourraient m’être très utiles pour l’archivage et la sauvegarde des données. Bien sûr, chez Société Générale, nous utilisons des outils qui génèrent automatiquement les scripts. Ici il m’a fallu donc apprendre à les écrire manuellement.

## Point sur l’application à aujourd’hui

SOS Immo est opérationnelle dans sa mission principale (suivi des incidents).

Elle possède un front end, un back end et une base de données.

La partie authentification est terminée.

La gestion des données « administrateur »[[31]](#footnote-31) aussi.

Il reste à finaliser l’archivage et la sauvegarde des données incidents et utilisateurs.

## Bilan personnel - remerciements

Ce projet est pour moi à la fois un aboutissement et un nouveau départ. J’ai appris plus de choses en 1 an et demi qu’au cours des 10 dernières années. Jamais je ne dénigrerais mon ancienne profession, mais je ne voudrais pas revenir en arrière.

Je tiens à remercier bien sûr les professeurs et les autres élèves de la cohorte, mes nouveaux managers et toutes leurs équipes. Pour leur confiance, leur patience et leur pédagogie.

Sans oublier les responsables du service immobilier de l’immeuble Basalte de la Défense qui m’ont beaucoup aidée lors de la conception du projet sans rien attendre en retour.

# ANNEXES

## Annexe 1 – Ecran du pipeline GitHub

Une image contenant texte

Description générée automatiquement

## Annexe 2 – exemple de scénario complexe

Suivre à la lettre les indications dans l’application front avec un environnement de test.

Prérequis :

La base test doit posséder deux utilisateurs Admin et Technicien avec les caractéristiques indiquées aux étapes A1 et T1.

Elle ne doit pas posséder d’utilisateurs avec comme id « nduval ».

Recommandation :

Pour gagner du temps et simplifier la procédure, utiliser 1 navigateur par profil d’utilisateur, soit 3 navigateurs différents (Chrome, Edge, Mozilla…).

Résumé du test :

Admin crée un nouvel utilisateur.

Il voit les incidents dans son écran pilotage.

Un nouvel utilisateur change son mot de passe à la première connexion.

Il fait un signalement et valide ou non l’intervention une fois terminée.

Technicien voit ce signalement dans son suivi.

Il prend en charge et termine une intervention.



## Annexe 3 - Tests Postman

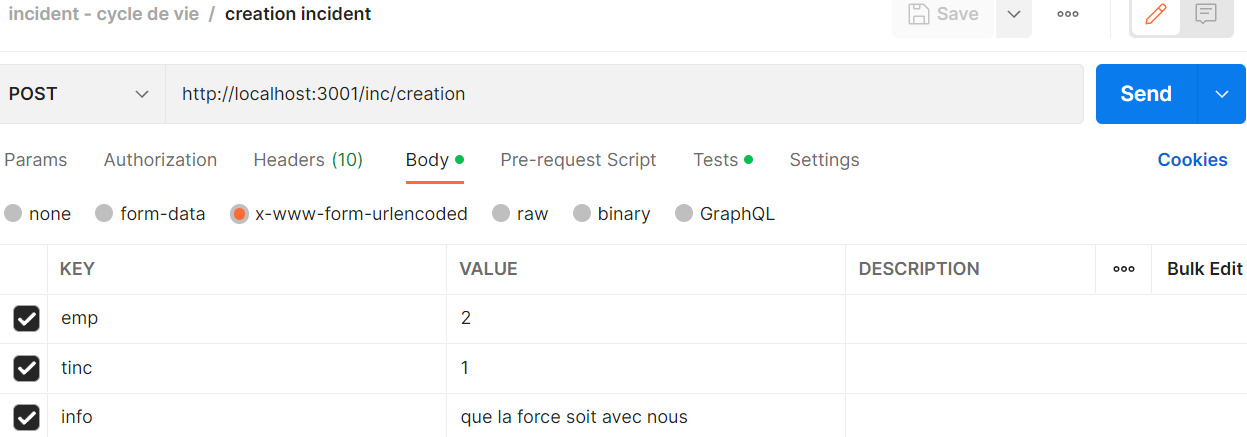
Saisir directement l’url et les données nécessaires le cas échéant :

* Données Body en cas de requête POST
* Paramètres en cas de requête GET

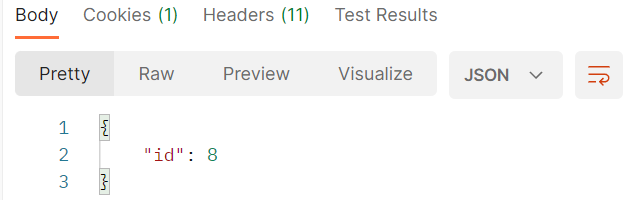
Les tests peuvent être nommés et sauvegardés pour réutilisation ultérieure.

Exemple : Création d’un incident

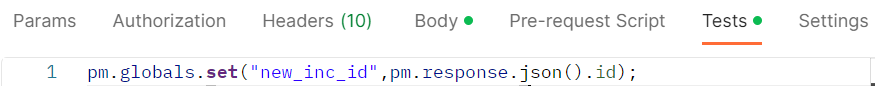
Ecrans Postman



Données nécessaires : emplacement, type d’incident, commentaire.



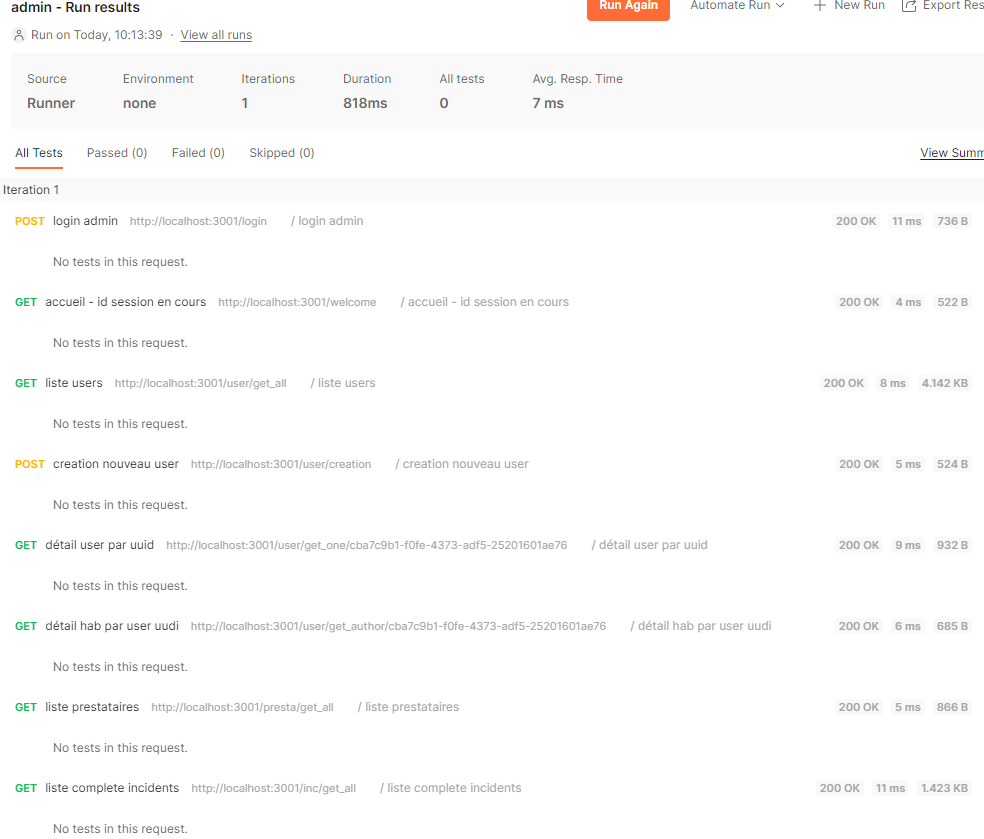
L’API retourne l’identifiant du nouvel incident.



Un système de variables permet de récupérer les retours et de les réutiliser pour un prochain test.

## Annexe 4 - Collections Postman

Les tests sont rassemblés dans l’ordre dans lequel ils seront exécutés.

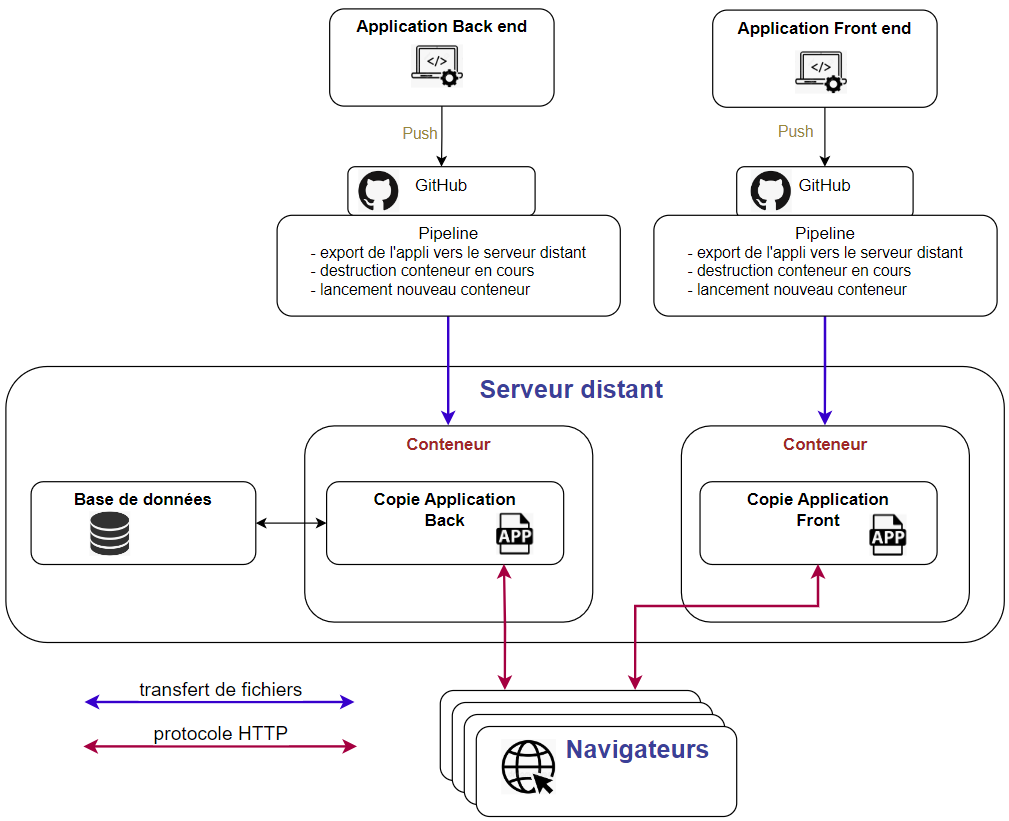
Une image contenant texte

Description générée automatiquementPour chaque collection, on peut cocher/décocher les services à contrôler.

L’enchaînement est déclenché par le bouton Run.

## 

## Annexe 5 - schéma déploiement



1. CHSCT Commission Hygiène, Sécurité et Conditions de Travail

   - CSSCT Commission Santé, Sécurité et Conditions de Travail [↑](#footnote-ref-1)
2. L’immeuble Basalte (Société Générale), à la Défense. [↑](#footnote-ref-2)
3. Un profil externe ne peut travailler que pour une seule entreprise prestataire. [↑](#footnote-ref-3)
4. IHM : Interface Homme-Machine [↑](#footnote-ref-4)
5. SPA : Sigle-Page Application, ou monopage [↑](#footnote-ref-5)
6. De n’importe quel type. [↑](#footnote-ref-6)
7. Une useState déclarée dans App.js ne sera jamais réinitialisée. [↑](#footnote-ref-7)
8. « Login.js » contient le formulaire d’authentification. [↑](#footnote-ref-8)
9. « Menu.js » contient toutes les fonctionnalités de l’application, accessibles par des boutons.

   L’affichage ou non de ces boutons est conditionné au profil de l’utilisateur. [↑](#footnote-ref-9)
10. API : Application Programming Interface [↑](#footnote-ref-10)
11. Ici nous avons une base MySQL. Sequelize peut interagir avec d’autres types de base, comme PostGreSQL. [↑](#footnote-ref-11)
12. ORM : Object Relational Mapping - interface entre le programme lui-même et la base de données [↑](#footnote-ref-12)
13. DAO :Data Access Object [↑](#footnote-ref-13)
14. On peut noter au passage que la nouvelle entreprise prestataire va reprendre la plupart des salariés de l’ancienne entreprise. Il n’y a donc que très peu d’impact sur le suivi des incidents. [↑](#footnote-ref-14)
15. Conservé dans la table Journaux. [↑](#footnote-ref-15)
16. Le mail doit être unique. Il servira en cas d’oubli de donnée d’authentification. [↑](#footnote-ref-16)
17. Les modalités de sauvegarde du mot de passe seront traitées dans le chapitre « [gestion du mot passe](#_Gestion_de_mot) ». [↑](#footnote-ref-17)
18. 12 caractères dont 1 majuscule, 1 minuscule, 1 chiffre et 1 caractère spécial => ()!@#$+-\*&\_. [↑](#footnote-ref-18)
19. Universally Unique Identifier [↑](#footnote-ref-19)
20. « true » après réussite de l’authentification – « false » en cas de déconnexion. [↑](#footnote-ref-20)
21. Identifiant peut être retrouvé grâce à l’UUID et vice versa. Conserver les 2 données permet tout de même de simplifier le programme. [↑](#footnote-ref-21)
22. Là aussi, on peut retrouver l’information par l’UUID. L’idée est de simplifier les recherches. [↑](#footnote-ref-22)
23. Fonctionnalité annexe : l’identifiant du dernier utilisateur s’affiche directement si le navigateur possède encore un cookie (dernière connexion depuis moins de 10 minutes). Sinon, pas de retour, ce qui est parfaitement transparent pour l’utilisateur : le formulaire d’authentification reste vierge. [↑](#footnote-ref-23)
24. Prestataires, emplacements, type d’emplacements, types d’incidents, mapping type d’emplacements/types d’incidents [↑](#footnote-ref-24)
25. Ce qui ne signifie pas que l’on ne peut pas les supprimer. Cela n’aurait juste aucune d’incidence sur la base « sos\_immo\_sauv ». [↑](#footnote-ref-25)
26. Utilisation : chapitre Cinématique-Admin [écran gestion Utilisateurs](#_Admin), colonne Statut. [↑](#footnote-ref-26)
27. Nombre d’incidents pour une période sur un étage, par prestataire, par type d’incident. Taux de satisfaction. [↑](#footnote-ref-27)
28. Contrairement au 20ème siècle, on peut féminiser le métier. [↑](#footnote-ref-28)
29. Sauf pour le recueil de besoin pour lequel je me suis fait aider par les services immobiliers Société Générale. [↑](#footnote-ref-29)
30. Programmation Orientée Objet [↑](#footnote-ref-30)
31. Utilisateurs et habilitations - prestataires, emplacements, type d’emplacements, types d’incidents, mapping [↑](#footnote-ref-31)