Plan

Sg – reskilling

Présentation SOS Immo

*Les différents besoins*

Use case

Architecture du projet

Focus front

Focus back

Présentation des fonctionnalités/cinématiques

Focus sur l’incident

Cycle de vie

Ecrans relatifs

Base de données

Focus sur la table incidents

Sécurité

Mot de passe

Authentification

Données

Cookie de session

Organisation

GitHub

Trello

tests

click

2

SG

Société Générale est une banque de stature internationale qui, comme toutes les banques d’investissements, a un besoin grandissant de savoir-faire en informatique.

click

Parallèlement, d’autres métiers de plus en plus automatisés nécessitent moins de personnel.

Elle a donc eu l’idée de plan de formation d’ampleur afin de s’adapter à ses nouveaux besoins.

click

Les formations de reconversion, ou reskilling, permettent à des salariés de changer de parcourt professionnel.

Au prix d’un important investissement personnel, ils ont la possibilité de découvrir un nouveau métier et de passer un diplôme d’Etat.

C’est à ce titre que j’ai rejoins il y quelques mois une équipe de développeurs.

click

FixedIncome

Plusieurs missions :

* Décommissionnement Eole, qui gère entre autres les obligations (bonds).
  + Comme d’autres produits, les bonds doivent être entièrement pris en charge par une autre plateforme : XOne.
  + Pour l’instant, le booking et la partie paiement des bonds et déjà gérée dans Xone. Mais la gestion des flux comptables reste à implémenter.
* Responsabilité de la maintenance de XOne sur la partie bonds.
* Contrôle du booking des transactions
* Gestion de leur cycle de vie
* Calcul des positions (stocks)
* Respect des contraintes règlementaires

*Les nouveaux développements font l’objet d’une release toutes les 6 semaines.*

*La semaine précédente est consacrée aux tests de non-régression.*

*De plus, pour les petits développements « en urgence », il existe la possibilité de patchs 2 fois par semaines.*

*Le développeur est alors responsable des tests.*

click

poste

En tant que débutante, mon poste consiste à participer aux tests de non-régression et à l’implémentation des flux comptables dans XOne.

Je développe des outils de tests sur ces flux.

3 Présentation SOS Immo

SOS Immo est un outil de gestion de tickets, ces tickets étant les incidents techniques usuels pouvant survenir dans un immeuble de bureau de grande taille.

Dans un tel immeuble, des techniciens appartenant à toutes sortes d’entreprises prestataires sont sur place, à demeure, pour pouvoir intervenir à tout moment, dans un délai convenu. Chaque entreprise prestataire est spécialisée (plomberie, ascensoriste, ménage, etc.).

Jusqu’ici, la centralisation des incidents était assurée par un standard téléphonique, des cahiers et des post’it.

Ce qui pouvait entrainer malentendus, retards, appels dans le vide…

click

L’idée était de créer une plateforme unique pour tous les occupants de l’immeuble, qu’ils soient employés de bureau, intervenants extérieur ou responsables d’immeuble.

L’objectif principal de l’application est bien sûr d’aider au suivi des incidents.

Un autre objectif est d’aider Imm à identifier les points de vigilances : prestaires potentiellement en sous-effectif, étages ou types d’incidents particulièrement récurrents, satisfaction des usagers.

Pour construire ce projet, je suis allée rencontrer les responsables de l’immeubles Basalte, à la Défense, qui appartient à SG.

Ils m’ont été d’une grande aide pour l’identification, la confirmation ou non de telle ou telle fonctionnalité.

click

4 diagramme de Use case

Après avoir défini les besoins, j’en suis arrivée à la conclusion que 4 profils différents interviennent dans le process.

click

2 internes et 2 externes

Interne => entreprise occupant (proprio ou locataire de l’immeuble)

Externe => entreprise prestataire sous contrat

click

Usager dont le rôle se borne à signaler l’incident et valider la fin d’intervention.

click

Imm, service immobilier, responsable d’immeuble qui est l’administrateur de l’appli et qui a donc tous les droits.

click

Technicien qui résout les problèmes

click

Valideur qui est le chef du technicien. Il utilise l’outil pour coordonner son équipe.

J’ai conçu ce projet avec une architecture web client-serveur classique.

click

SOS Immo est développé comme 2 projets indépendants.

Le Front end qui est l’interface utilisateur et qui permet de naviguer entre les différentes fonctionnalités.

La partie visible du programme.

Les Back end en est partie invisible.

Il fait les calculs et garde la mémoire.

Front

Est géré par la librairie ReactJS.

Le concept en est assez simple : 1 page pour le site et une multitude de composants pour l’habiller qui sont déclenchés selon certaines conditions.

Un composant est donc un élément de page. Il contient

* du code html qui lui-même peut appeler d composants
* des fonctions qui lui permettent d’envoyer des requêtes au Back
* des variables.

Certaines de ces variables ont la particularité d’être transmissibles d’un composant parent à un composant enfant, et leur valeur mises par un enfant.

On les appelle useState.

Quand un useState est mise à jour par un compo enfant, le compo parent dans lequel il a été déclaré est recalculé.

click

6

Par convention, le seul composant de index.js est « App.js »

Intermédiaire entre index.js et tous les autres composants.

Dans App.js, j’ai déclaré plusieurs useState dont la première, nommée « ecran » initialisée « login ».

Le code de Apps.js lance systématiquement 2 composants :

* 1 composant « Bandeau.js »
* 1 autre composant qui dépendra de la valeur du useState « ecran ».

1/ Le contenu de « Bandeau.js » est lui-même conditionné à « ecran ».

Systématiquement « Bandeau.js » affiche le logo de l’application.

Si « ecran » !== « login », authentification réussie, il affichera aussi d’autres éléments, nom et profil de l’utilisateur.

2/ Le 2nd composant est donc conditionné à la valeur « ecran ».

Au démarrage, « Login.js ».

Contient le formulaire d’authentification et le bouton de soumission.

Il contient aussi la fonction qui va communiquer ces éléments au Back et récupérer les données utilisateurs.

Quand l’authentification est validée, la valeur du useState « ecran » est mise à jour à « menu ».

Certaines données de l’utilisateurs sont aussi stockées dans des useState et permettront l’affichage des composants à venir.

Le composant « Login.js » n’est plus afficher. A la place, le composant « Accueil.js ».

« Accueil.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.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Pour ce qui est du Back

Retour 5 - architecture

Back est codé en Node.js.

Il a une structure de plusieurs couches qui se partagent le travail.

click

7

Le cerveau de l’application, c’est le serveur, ou controller. Fichier distinct.

Il est géré par la librairie Express.

Ce serveur reçoit les requêtes du Front.

Ces requêtes sont traitées par des fonctions, dites de services.

Pour la clarté du code, ces services ont été dispatchés par catégorie.

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

L’interaction entre les services et la base de données est assuré par un ORM, ici Sequelize.

L’ORM reçoit les options de connexion à la base de données et transpose les différentes tables en autant de classes objet.

Chaque colonne de la table est représentée par un attribut de l’objet.

Ainsi les services utilisent des objets et sans savoir à connaître leur provenance.

L’ORM paramétré pour éviter au maximum les requêtes SQL. Et, même si parfois cette requête est inévitable, elle empêche les problèmes d’injections SQL.

Le code et sa maintenance en est donc simplifié.

Passons aux fonctionnalités.

click

8

Voici le menu complet des actions.

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

Tous les profils peuvent faire des signalements et en suivre l’état d’avancement.

Seuls les prestataires ont accès au suivi d’incident, qui liste les interventions non terminées du prestataire en question.

Les chefs (imm et valideur) peuvent consulter le pilotage. Qui contient les incidents regroupés par catégories.

Seul imm peut administré les éléments de maintenance des données.

Je vous propose la cinématique d’un utilisateur imm.

click

9

Maintenant la raison de cette appli, c’est l’indicent.

D’où focus.

click

10 diagramme état-transition

Tous les incidents suivent le même parcourt.

Ils passent linéairement par 4 états, ou statut.

A sa création, 1 inc 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’inc est en attente de validation par l’utilisateur qui a fait le signalement.

Puis l’utilisateur valide la clôture.

Après clôture, l’incident n’a plus lieu d’être. On pourrait le retirer de la base.

Bien entendu on ne retire rien, il sera archivé.

Statistique

Si l’utilisateur est mécontent de l’intervention, il le signale et un nouvel inc est automatiquement généré à l’identique du premier, avec le motif d’insatisfaction.

Le cycle de vie recommence avec ce nouvel inc.

Les écrans relatifs à ce cycle.

click

11 signalement

Au lancement de cet écran, il y a échange front/back.

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.

Sachant que tout ne peut pas arriver n’importe où.

Un premier menu déroulant s’affiche pour sélectionner l’étage.

Une fois l’étage renseigné, le deuxième menu déroulant propose les emplacements possibles de cet étage précis.

Ensuite, le dernier menu déroulant propose tous les types d’incidents pouvant survenir dans cet emplacement.

Chaque emplacement appartenant à un type particulier.

Le commentaire est facultatif.

A la validation, une requête est envoyée au Back.

Celui-ci détermine quelle entreprise prestataire doit être attribuée à cet incident suivant le type d’incident renseigné.

L’utilisateur est redirigé vers l’écran de suivant d’avancement de ses propres signalements.

Click suivi d’avancement

Chaque signalement est une vignette.

L’ordre de ces vignettes et leur couleur indique le statut actuel de l’incident.

Bien sûr l’utilisateur peut consulter le détail de chaque incident.

click

12 détail

Une fiche incident est partagée en 3 parties.

1er/ Cartouche, sorte de carte d’identité. C’est quand, où et quoi.

Dernier/ Commentaire associé à cet indicent précis.

*A Basalte, on m’a expliqué que les prestataires et imm communiquent énormément par ce biais.*

Au milieu/ le status

Tout le monde peut le connaitre. Mais les fonctionnalités qui le concerne ne sont accessibles que selon le profil.

Par exemple, l’usager ne peut agir qu’après la fin de l’intervention.

Le technicien peut prendre en charge et terminer une intervention.

Pour développer le projet, j’ai commencé par faire un diagramme de useCase. Ensuite, j’ai constitué un dictionnaire de données. Je les ai trié par catégorie, ce qui m’a permis de concevoir la base de données de mon application.

click

13 bd

Cette base est composée de 9 tables. Avec clés primaires et pas mal de clés étrangères.

* Le cœur de cette base est la table incidents.
* Identifiant en auto-incrément
* Emp et tinc fourni par l’auteur du signalement.
* Le presta est calculé. Cela pourrait faire doublon avec le tinc.

Mais à Basalte, on m’a expliqué que bien souvent, l’utilisateur se trompe de tinc, voir écrit n’importe quoi.

Il arrive donc que l’incident doive être attribué à un autre prestataire que celui calculé en premier lieu.

J’ai choisi de garder les infos fournies par l’utilisateur et de pouvoir changer le prestataire en charge.

* Identifiant de l’auteur du signalement. Indispensable car c’est lui qui devra faire la clôture.
* Identifiant du technicien. Pour savoir qui fait quoi.
* Dates et heures de chaque étape/statut. A fin de pilotage et de statistiques.
* Enquête de satisfaction.

Cœur => incidents

Poumon => utilisateurs

click

14 diagramme de séquence

Nouvelle arrivée est signalée par RH (int) ou par presta directement à Imm.

Nom prenom mail tel – presta : fonction

*Imm a la responsabilité de maintenir la base à jour avec les données qui lui sont fournis.*

Imm saisi ces données + identifiant unique.

L’application se charge de générer un mot de passe aléatoire et de le transmettre par mail à ce nouvel l’utilisateur.

Ce mot de passe est à expiration immédiate.

A chaque connexion, la date d’expiration du mot de passe est contrôlée.

L’utilisateur est redirigé vers l’écran de changement de mot de passe le cas échéant.

Le nouveau mdp aura lui une validité de 90 jours.

L’utilisateur a la possibilité d’avoir un trou de mémoire.

Il peut demander à récupérer son identifiant par mail.

Si c’est le mdp qui est demandé, un nouveau mdp aléatoire à expiration immédiate en envoyé par mail.

Voici les écrans

click

15 authentification

Le Back contrôle que l’identifiant + mdp appartiennent à 1 même utilisateur.

Click oubli

En cas d’oubli, l’utilisateur est invité à renseigner son mail et ce qu’il souhaite récupérer.

L’appli contrôle que l’adresse existe en base et envoie l’identifiant/mdp à cette adresse.

Click changement mdp

N’importe quand, l’utilisateur changer son mdp. Il le doit de toute façon en cas d’expiration.

Mdp actuel

Nouveau 1 – différent mdp actuel et satisfaire aux caractéristiques minimums requises.

Nouveau 2 – identique nouveau 1

Le Back contrôle que l’ancien mdp est associé à l’utilisateur en cours.

Je vais revenir sur le diagramme de séquence de tout à l’heure

click

16 diagramme

On voit que Imm connaît l’identifiant.

Le programme génère un mdp et l’envoie le tout à l’utilisateur.

Il ne garde pas ce mdp en mémoire.

click hashage

Avant la sauvegarde, ce mdp passe à travers une fonction de hashage..

Le hachage est un moyen de crypter un message.

C’est une fonction transforme une chaine de caractères en un nombre hexadécimal de 32 caractères.

Ce nombre a 2 particularités :

Sens unique – Il ne peut pas être décrypté.

Une même chaîne ou document ne produira qu’1 seul résultat de hash et ce résultat lui est totalement spécifique.

*Librairie « sha1 ».*

Mais c’est pas tout.

click salage

Pour encore plus de sécurité, on procéde au salage du mdp.

C’est-à-dire qu’avant de le hashé, on lui adjoint une autre chaine de caractère. J’ai choisi l’identifiant, puisqu’il est unique.

Le programme concatène mdp et identifiant et hash le tout.

C’est ce hash qui est sauvegardé en base.

Ainsi le mdp original est inaccessible, même pour un opérateur imm.

click fin

17 données

Pas de données vraiment confidentielles

Num ss, adresse, siret, âge…

Mais tout de même sensibles

Nom mail telephone, niveau d’habilitation

7 des 9 tables ont pour clé primaire un entier en auto incrément.

Pour Utilisateurs et Habilitations, pour lesquels j’ai choisi une clé plus sécurisée, une Uuid.

Universally Unique Identifier

Fonction qui permet de générer aléatoire un nombre hexadécimal de 32 caractères.

Ce nombre a la garantie d’être tout à fait unique.

Il n’apparait sur aucun écran de l’interface.

*Librairie « uuidv4 ».*

J’ai choisi d’utiliser un Uuid comme clé de la table Utilisateurs en plus de l’identifiant unique. Cet identifiant ne me semblait pas assez sûr puisqu’il est précisément conçu pour être facile à retenir au moment de l’authentification.

Utiliser des uuid pour les tables les plus sensibles et hasher/saler les mdp contribuent à la sécurisation des données.

Mais il faut aussi sécuriser l’accès à ces données.

click

18 cookie

Un premier filtre est fait par l’appli web, puisque que c’est le profil de l’utilisateur qui lui donne accès ou non aux différentes fonctionnalités.

Mais c’est pas suffisant.

Il faut qu’à chaque service sollicité par le serveur, contrôle soit fait : le demandeur est-il autorisé à demander ?

Pour cela, j’ai choisi d’utiliser un cookie de session.

Ce cookie est un petit fichier stocké par le serveur dans le navigateur.

Il est donc créé par le serveur après la réussite de l’authentification.

Il contient…

Quand un service est appelé par le serveur, il va commencer par contrôler la validité de la demande.

Exemple

* Je veux créer un nouvel utilisateur
  + Le composant React avait auparavant vérifier les données avant d’envoyer requête POST adéquat.
  + Le serveur appelle le service de création d’ut.
  + La première chose que fait ce service, c’est de contrôler grâce au cookie
    - Que l’auteur la requête est bien identifié et connecté
    - Qu’il a le bon profil
* Je veux la liste des incidents que j’ai moi-même déclarés.
  + Le serveur appelle le service de sélection.
  + Le service contrôle que l’auteur la requête est bien identifié et connecté.
    - Le profil n’a pas d’importance.
  + Après l’extraction de la liste, le service fait un filtre et ne que ceux déclaré par l’utilisateur identifié par le cookie.

DEMO

Ouvrir un mozilla et un chrome

Nouvel utilisateur : nono duval – technicien chez Toupropre

Utiliser le mail efrei

Mail avec id + mdp

Lisbeth est employée, elle veut signaler un incident.

Elle se connecte – elle se trompe et puis elle y arrive

Elle n’a pas encore signalé d’incident

Elle en signale un chez Toupropre

Nono travaille pour Toupropre

Il s’authentifie avec son mail tout neuf.

Change mdp

Suivi d’incidents

bilan

Pour mon projet de fin d’étude, ma hiérarchie et moi-même n’avons pas trouvé de sujet purement Société Générale. Nous y avons longuement réfléchi. Les différents sujets que nous avons étudiés étaient impossibles à transférer sur un ordinateur tiers. Et aucun ne nécessitait la conception d’une base de données ni d’identification d’utilisateur.

J’ai donc choisi un sujet personnel.

La partie conception n’a pas posé trop de difficultés car j’ai l’habitude de faire ce genre d’analyse, surtout pour des projets personnels. Même si aucun de ces projets n’a jamais été aussi complexe.

Il y a eu de nombreux essais et tâtonnements bien sûr.

Mais j’ai l’habitude de jouer tous les rôles (product owner, scrum, developpeur…), d’organiser mon temps et de gérer les priorités. Revenir en arrière si nécessaire.

Pour la réalisation du projet, j’ai choisi le JavaScript car je m’y sens à l’aise.

J’estimais avoir suffisamment assimilé de notions pour pouvoir commencer.

* Pour le front, le système de composants React.
* Pour le back, le serveur Express et le cookie de session

Par contre je partais de zéro pour ce qui est de l’ORM.

Si j’avais compris le principe de POO, je n’avais pas du tout compris comment l’utiliser dans le cadre d’une application client/serveur.

J’ai suivi plusieurs tuto avant de poser mon choix sur la librairie Sequelize.

Maintenant j’ai compris l’intérêt de l’ORM et de représenter les éléments des tables par des objets.

De même pour les tests, que ce soient les tests unitaires du front que les tests d’intégration du back.

J’ai passé de très déprimantes heures avec Google.

Le concept ne posait pas de problème : mes profs et mes collègues m’en avait déjà beaucoup parlé.

J’avais juste un souci de codage.

Pour d’autres sujets annexes il a aussi fallu visionner pas mal de tuto et de pages web :

* préparation de mails
* librairie react-hook-form pour simplifier la gestion des formulaires

Il me reste encore à m’occuper de la date d’expiration du mot de passe, à terminer la gestion admin et à améliorer les retours back vers front, notamment en cas d’interruption de connexion.

D’autres évolutions doivent être envisagées comme l’archivage des incidents clôturés et les productions de kpi.

Grand merci à mon responsable, qui m’a fait confiance, et à toute l’équipe FixedIncome pour son soutien. Grace à eux, j’ai découvert que l’informatique c’est beaucoup de pédagogie et de patience.

Merci beaucoup également aux responsables du service immobilier de l’immeuble Basalte pour le temps qu’ils m’ont consacrés et dont l’aide m’a été très précieuse pour ce projet. Je connais depuis des années leur gentillesse et leur disponibilité, maintenant je leur dois la suite de ma carrière.

Une image contenant texte

Description générée automatiquement

Une image contenant texte

Description générée automatiquement

Une image contenant texte

Description générée automatiquement

Une image contenant texte

Description générée automatiquement

Une image contenant texte

Description générée automatiquement