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.

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.

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

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

FixedIncome

Pour rationnaliser les coûts et le traitement des transactions, les équipes SG ont entrepris le commissionnement d’une vieille application, Eole.

Les transactions Eole seront à moyen terme gérées par une autre application déjà existante et puissante, XOne.

Déjà les obligations (ou bonds) sont d’ores et déjà gérés par Xone. Cependant, Eole continue à générer les flux comptables.

Le service FixedIncome a la charge de cette partie du décommissionnement.

Il doit implémenter XOne pour qu’il puisse générer des flux comptables identiques à ceux générés par Eole actuellement.

FixedIncome est également responsable 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.

poste

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

Je développe des outils de tests sur ces flux.

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.).

L’outil est 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.

J’ai choisi moi-même de la technologie pour construire ce projet.

Le front est en React JavaScript.

Le back est développé en Node JS avec une base de données MySql.

Ce sont les technologies dans lesquelles je me sens le plus à l’aise avec les cours que nous avons reçu.

Use case diagram

SOS Immo s’adresse donc à plusieurs profils d’utilisateurs :

* Usager. Personnel interne travaillant dans l’immeuble. Il signale les incidents et valide la fin de l’intervention.
* Technicien. Personnel prestataire chargé de la résolution des incidents. Il peut s’assigner un incident.
* Valideur. Personnel prestataire responsable de l’équipe de techniciens de son entreprise. Il peut consulter le suivi des incidents en temps réel et assigner un incident à un de ses techniciens.
* Imm. Personnel interne appartenant aux service immobiliers, chargé de la gestion de l’application au quotidien. C’est donc à la fois l’administrateur des bases de données et le « client » des entreprises prestataires. Ayant tous les droits, il peut réattribuer un incident à un technicien précis ou une autre entreprise prestataire en cas d’erreur d’attribution.

Pour déterminer ces rôles, je suis allé interviewer un responsable d’immeuble de la Société Général. Il m’a confirmé que chacun de ces profils à la possibilité de faire des signalements.

Le service immobilier a absolument tous les droits sur l’incident durant son cycle de vie.

Par ailleurs, les utilisateurs communiquent beaucoup en ajoutant des commentaires, commentaires pas forcément visibles par l’usager auteur du signalement.

Front

Est géré grâce à la librairie ReactJS.

L’application React lance le fichier index.js.

Ce fichier contient l’unique page du site.

Dans cette page se trouvent les composants React qui seront déclenchés selon condition.

Un composant est élément de page. Il contient du code html, des fonctions et des variables.

Certaines de ces variables ont la particularité d’être transmissibles et les valeurs modifiables d’un composant à l’autre.

On les appelle useState.

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 aux valeurs des autres useState déclarées dans App.js.

Si « ecran » === « login », « Bandeau.js » n’affichera que le logo de l’application.

Sinon, 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.

Ces données sont stockées dans des useState et permettront l’affichage des composants à venir.

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

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

« Accueil.js » contient tous les boutons d’actions de l’application.

L’affichage ou non de ces boutons est conditionné à la valeur du useState « profil » de l’utilisateur.

Back est codé en Node.js.

Il utilise un serveur de la librairie Express.

Ce serveur reçoit les requêtes du Front, qui sont traitées par des fonctions, dites de services.

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

L’interaction entre les services et la base de données est assuré par un ORM. J’ai choisi 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.

L’ORM implémente aussi des fonctions qui font interagir les objets et les tables, sans forcément avoir à passer par des requêtes SQL.

Cela simplifie le code et sa maintenance.

Cela évite aussi les problèmes d’injections SQL.

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 et les useState
* 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 unitaires, j’ai dû suivre des tuto.

La notion de test unitaire ne posait pas de problème, c’était le codage en JavaScript.

Mes collègues m’ont beaucoup aidé. Nous en faisons très souvent.

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

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

Malgré de nombreuses heures avec Google, je ne suis pas parvenue à automatiser les tests d’intégration.

Il faudra que j’y arrive avant février prochain.

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