My name is Sophie. for one year i was trained in order to change my professional way and become an IT developer.

This training was proposed to me by my employer Société Générale. It was a partenership with the Effrei Paris school.

This is a big investment for the company and I and now I’m glad to present you my end-of -study project.

The project, called SOS Immo, is a ticket manager tool, these tickets are technical incidents that may append in a tall office building.

With this tool, every occupants of the building will be able to report an incident and comment it.

The technicians will be able to take care and report the resolutions.

The managers will be able to know which technicians take care which indicents.

I will start by explain the project and its technical aspects (architecture and database). Next, the features, with focus on most important elements (incidents, users). Focus on security, testing and, to finish, a little demonstration.

click

1. SOS Immo

Donc sos\_immo gère les incidents dans un immeuble de bureau de grande taille.

Dans immeuble de ce genre, l’entreprise occupante (proprio ou locataire de l’immeuble) ne s’occupe pas elle-même de ce genre de choses. Elle passe des contrats avec d’autres entreprises prestataires spécialisées dans leur domaine (plomberie, ascensoriste, ménage, etc.).

Ces entreprises emploient des techniciens sur place, à demeure, pour pouvoir intervenir à tout moment, dans un délai convenu.

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

click

Dans un contexte de digitalisation des process, l’idée est venue d’une plateforme unique pour tous les occupants de l’immeuble, qu’ils soient employés de bureau, intervenants extérieur ou responsables d’immeuble.

Je dois dire qu’une plateforme de ce genre a été mise en place SG. Je reprends l’idée à mon compte.

Pour construire ce projet, je suis allée rencontrer des responsables d’immeuble Société Générale, à la Défense.

Ils m’ont expliqué comment ils utilisent l’outil et leurs besoins en la matière.

click

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

Les objectifs secondaires : aider les services techniques à identifier les points de vigilances : prestaires potentiellement en sous-effectif, étages ou types d’incidents particulièrement récurrents

J’en ai déterminé les fonctionnalités suivantes.

Click

1. diagramme de Use case

Après avoir défini les besoins, j’en suis arrivée à définir 4 profils différents.

click

2 internes et 2 externes

Interne => entreprise occupante

Externe => entreprise prestataire sous contrat

click

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

click

Admin, c’est à la fois le service immobilier, responsable d’immeuble, et 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.

Ces 2 profils externes ne travaillent que pour une seule entreprise prestataire chacun.

**Passons à la partie « technique » de la présentation.**

click

J’ai conçu SOS Immo avec une architecture web client-serveur classique.

Il est développé comme 2 projets indépendants que je vais détailler tout de suite.

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.

**Le tout est codé en JavaScript.**

Front

Est géré par la librairie React.

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

Ils peuvent même s’appeler les uns les autres (parents => enfants)

Un composant est donc un élément de page. Il contient

* du code html  :
  + visuellement : tableau, bouton, formulaire, etc.
* des fonctions qui lui permettent d’interagir avec le Back
* d’autres éléments propres à React à mi-chemin entre variables et fonctions qu’on appelle useState.

Variables pour simplifier de n’importe quel type. Ont la particularité d’être transmissibles d’un composant parent à un composant enfant, et leur valeur mises à jour par un enfant.

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

Et c’est ça qui fait fonctionner le tout.

**Par exemple**

click

1. 5

Voici la page et son seul composant. Par convention « App.js »

Gare de triage pour tous les autres composants.

Dans App.js, j’ai déclaré plusieurs useState dont le premier, nommé « ecran », string initialisé « login ».

Le code HTML 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 – id + mdp.

Il contient aussi la fonction qui va communiquer ces éléments au Back.

Quand l’authentification est validée, certaines données utilisateurs sont récupérées et la valeur d’« ecran » est mise à jour à « menu ».

Comme « ecran » est un useState, le composant dans lequel il a été déclaré (App.js donc) est recalculé.

Le composant « Login.js » n’est plus affiché. 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

click

1. 6

Le back est un serveur NodeJS contrôlé par la librairie Express.

Le controller c’est 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.

Ces fonctions sont des services que j’ai regroupés par catégorie (login, utilisateurs, incidents).

Chaque catégorie de services est logée dans un fichier .js distinct.

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

Les services ne vont pas interroger directement la base de données.

L’interaction entre les 2 est assuré par un ORM, Object Relational Mapping.

C’est un système qui converti les 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.

Ce sont les fichiers Models.

J’ai choisi la librairie Sequelize de NodeJS.

Sequelize possède ces propres fonctions pour récupérer les données de la base, les convertir en objet et vice versa.

Il évite au maximum les requêtes SQL. Et, quand ces requêtes sont inévitables, il empêche les problèmes d’injections SQL.

J’ai mis ces fonctions dans des fichiers DAO (Data Access Object) distincts des Models, pour clarifier le code.

Ce sont les fonctions DAO qui sont utilisées par les services.

Ainsi, chaque couche a son propre rôle dans l’organisation global du Back.

Le controller reçoit les demandes, qu’il transmet aux services, qui utilisent les fonctions d’ORM, qui utilisent les objets, qui représentent les tables de la bd.

click

1. database

**Pour en finir avec la technique**, j’ai réparti toutes les données nécessaires pour ce projet en 9 catégories.

Le résultat est une base de données de 9 tables.

Chaque table a sa clé primaire.

Les plus importantes sont les tables Incidents, Utilisateurs.

J’en parlerais plus un peu plus tout à l’heure.

Passons aux fonctionnalités.

click

1. menu

Voici le menu complet des actions.

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. Ils y trouvent la liste des interventions non terminées qui les concernent.

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

Les boutons bleus réservés au profil « Admin » : accès aux différents écrans de gestion et d’archivage de données.

click

1. diagramme état-transition

L’incident est donc le cœur de l’appli.

Tous les incidents suivent le même parcourt (workflow) => 4 états, ou statuts.

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 auteur du signalement.

Puis l’utilisateur valide la clôture.

Après clôture, l’incident est mort.

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.

click

1. Les écrans relatifs à ce cycle.

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.

Puisque 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.

Le commentaire est facultatif.

A la validation, le tout est envoyé au Back.

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

Puis sauvegarde en base.

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

Click suivi d’avancement

Chaque signalement est une vignette.

La couleur de ces vignettes indique le statut actuel de l’incident.

Bien sûr l’utilisateur peut consulter le détail d’une de ces vignettes puisqu’il s’agit de ses propres incidents.

click

1. détail

Une fiche incident est partagée en 3 parties.

En haut/ Cartouche, sorte de carte d’identité. Quand, où et quoi.

En bas/ Commentaire associé à cet incident précis.

Au milieu/ le status

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

click

1. Focus table incident

click

1. diagramme de séquence

**A quel moment est créé un compte utilisateur ?**

SOS Immo est conçu pour un usage professionnel au sein d’1 immeuble de bureaux.

Ce n’est ni un réseau social ni pas un site marchand.

Tous les occupant doivent pouvoir y accéder mais ce sont les services immobiliers qui administrent les comptes.

click

L’Admin a connaissance en amont d’une nouvelle arrivée. Ce procès n’est pas géré dans SOS Immo.

Pour créer 1 compte : Nom prenom mail tel – presta : fonction

click

L’admin saisi ces données + identifiant unique dans l’application.

Et celle-ci se charge de générer un mot de passe aléatoire à expiration immédiate.

Elle transmet l’id+mdp par mail à ce nouvel l’utilisateur.

Tout à l’heure, focus mdp

Click

A la demande de connexion, le Back contrôle que l’identifiant + mdp fournis appartiennent à 1 même utilisateur.

Click

A chaque connexion, 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.

Le nouveau mdp aura lui une validité de 90 jours.

Par la suite, l’utilisateur peut changer son mdp quand ça lui chante.

L’utilisateur a aussi le droit d’avoir des trous de mémoire.

Click

En cas d’oubli d’id/mdp, l’utilisateur est invité à renseigner son mail.

L’appli contrôle que l’adresse existe en base.

S’il s’agit de l’identifiant, il sera juste transmis par mail.

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

Voici les écrans

click

1. authentification

Click oubli

Click changement mdp

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 et fait la maj en base.

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

click

1. Mot de passe

On voit qu’admin connaît l’identifiant.

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

Il ne garde pas ce mdp en base.

click hashage

Auparavant ce mdp est haché.

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 système a 3 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.

Universel – On peut trouver des librairies proposant des fonctions de hachage pour n’

*Librairie « sha1 ».*

Mais c’est pas tout.

click salage

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

C’est un procédé purement mécanique : on ajoute à la chaine à crypter une autre chaine (de préférence unique) avant le hashage.

J’ai concaténé l l’identifiant avec le mdp + haché le tout.

click fin

C’est ce hash qui est sauvegardé en base.

Le mdp original est totalement inaccessible, personne ne peut pas le reconstituer.

click

1. UUID

Pas de données très sensibles.

click

Clé primaire => entiers en auto //// Sauf pour Utilisateurs et Habilisations => UUID

*Nombre hexadécimale de 32 caractères – très compliqué*

*Universally Unique Identifier*

Librairie « uuidv4 »

Sécuriser les données, c’est bien, mais il faut aussi songer à sécuriser leur accès.

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 fois qu’un service est sollicité par le serveur, il contrôle si la demande est-il licite ?

click

1. cookie

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.

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

Je l’ai paramétré pour stocker certaines infos sur l’utilisateur.

isId – signale réussite de la connexion…

Exemples d’utilisations différentes

* Je veux créer un nouvel utilisateur
  + 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 adéquat.
  + Ce service contrôle que l’auteur la requête est bien connecté.
    - Le profil n’a pas d’importance.
  + Ensuite, le service récupère la liste des incidents (via DAO) et fait un filtre. Ne garde que ceux déclaré par l’utilisateur identifié par le cookie.

Cookie peut être appelé à tout moment.

Durée de vie limitée.

Passons à la partie tests

click

1. Tests unitaires

Un TU est une fonction qui tester le résultat d’une autre fonction.

Dans un monde parfait, il est conçu pendant, voir avant, le codage de la fonction à tester.

Le principe est de faire tourner la fonction et de comparer le résultat calculé au résultat attendu.

Il existe des librairies permettant de construire facilement des TU. J’ai choisi Jest, en javaScript.

Là aussi je vais vous donner un exemple.

Dans l’écran de gestion Utilisateurs, on trouve les utilisateurs affichés dans un tableau.

Dans ce tableau, le status actif/inactif indique si l’utilisateur fait encore partie du personnel.

Dans la table Utilisateurs, on ne trouve pas actif/inactif, mais un champ date, qui est initialisé à null par défaut.

On y trouve la date de départ le cas échéant.

La fonction findUserStatus prend en paramètre une date et calcule un string actif ou inactif.

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.

J’ai utilisé un pipeline GitHub pour automatiser le lancement des tests unitaires à chaque push.

click

1. Tests fonctionnels

Ces tests contrôlent, l’enchaînement de différentes fonctions.

Typiquement, un incident, une fois créé, est-il bien accessible au technicien concerné ?

On va donc être amené à manipuler les données de la base.

Pour éviter les accidents et de polluer la base de données, on fait ces tests sur une base de données dédiée, jumelle de la base de prod.

On peut faire le test manuellement : le testeur suit des instructions précises et contrôle à chaque fois que le résultat est celui attendu.

Juste besoin d’un tableau

Comme ce travail est fastidieux et que l’interface front a des limites. J’ai utilisé le nagivateur PostMan qui permet de tester directement les url sans passer par l’appli front.

Il permet aussi d’établir de véritables scénarios d‘enchaînent des services.

click

click

16 organisation

Pour ce qui est des outils utilisés pour l’élaboration de ce projet

J’ai créé 2 dépôts sur outil de versioning GitHub.

Pour le suivi d’avancement du projet j’ai utilisé Trello, surtout au début.

Lister et trier les tâches à effectuer.

Eviter de se disperser.

Pour tester le code, j’ai utilisé la librairie Jest, pour le javaScript.

Mise en page css

Pour tester les services Back, j’ai utilisé l’outil PostMan qui permet de faire des scenario qui enchaînent les services.

Démo

Il est temps de vous présenter de visu SOS Immo.

Je ne vais pas pouvoir vous montrer toutes les fonctionnalités.

Je vais donc me concentrer sur le cycle de vie de l’incident.

click

17 bref

Année particulièrement enrichissante, éprouvante certes, mais passionnante.

click

De par ma formation initiale et mon expérience professionnelle et personnelle, j’avais déjà des notions d’informatique :

Interroger les futurs utilisateurs pour recueillir leurs besoins

Etablir un dictionnaire de données et en faire une base.

Déterminer l’ordre des priorisations. Plutôt avec un responsable.

Ici c’est moi qui ai assuré à peu près tous les rôles : je l’ai porté, conçu, codé, testé, validé.

click

Au cours de l’année écoulée, j’ai appris énormément de choses que j’ai pu mettre en pratique pour ce projet :

Web => quelques notions HTLM => Application client/serveur

La POO : indispensable à n’importe quel développeur.

Langages

Devops – qui m’a permis d’automatiser les tests unitaires et qui va me servir au déploiement sur un serveur distant

click

J’ai bien sûr dû compléter par des recherches personnelles.

Principalement pour comprendre le principe de l’ORM. Vaste sujet.

Beaucoup de temps aussi notamment pour les tests.

Mais là c’était beaucoup plus un problème de codage que compréhension du concept.

Et quelques fonctionnalités secondaires.

click

Aujourd’hui SOS Immo est opérationnel.

La mission principale de (suivi des incidents) est remplie.

Et la partie authentification est terminée.

click

Il me reste à finaliser l’administration de certaines données.

Je souhaite également déployer l’appli sur une machine distante.

On peut aussi envisager d’autres évolutions comme l’archivage des incidents clôturés, utilisateurs inactivés, la production de kpi, etc.

click

18 enfin

Je tiens à remercier bien sûr les professeurs et les autres élèves de la cohorte, mon nouveau manager et toute son équipe.

Sans oublier les responsables du service immobilier de l’immeuble Basalte de la Défense.