Documentation Plateforme

1- Structure du code

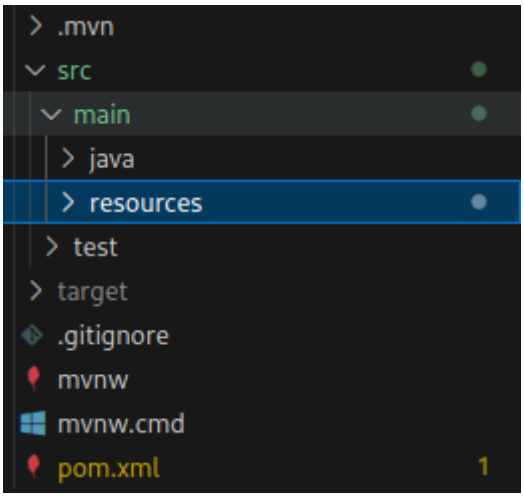
Frameworks

- **Spring Boot** : Framework Java qui permet la création d'applications java sécurisées et rapides. La structure du code sous ce framework se trouvera ci-dessous. Pour une formation de qualité sur ce Framework, je recommande la chaîne Dan Vega sur youtube.
- **ReactJS** : bibliothèque open source JavaScript pour créer des interfaces utilisateurs. Elle utilise une logique basées sur des composants (comme AngularJs et d'autres Frameworks) pour plus de visibilité de code. Chaque composant .jsx est une fonction de la forme:

```
import React from 'react'
import PropTypes from 'prop-types'

function test(props) {
    return (
        <div>test</div>
    )
}

test.propTypes = {}

export default test
```

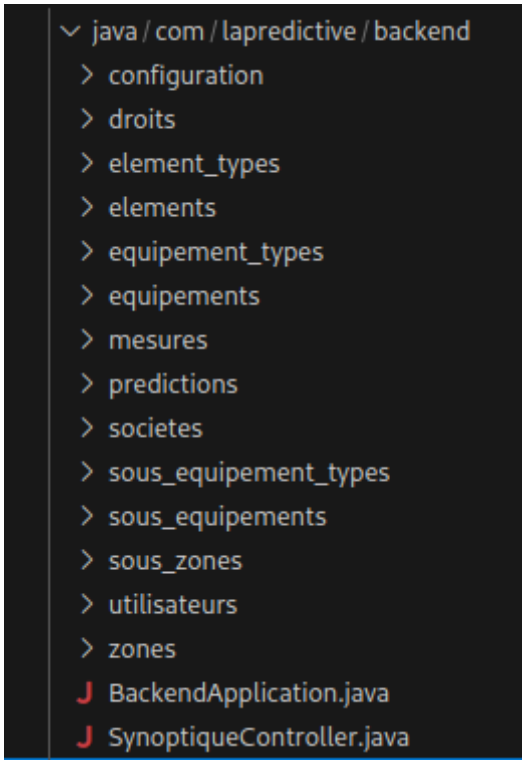
Donc une fonction où on défini la logique et comportement du composant en haut, et on retourne l'html. La structure par service du projet Front End est expliquée ci-dessous.

Structure BackEnd



Nous nous interesserons au dossier main et au fichier pom.xml:

- **pom.xml** : Ce fichier contient les différents packages utilisés dans la BackEnd et leurs versions. Pour faire un changement sur un package (dependency), on change les valeurs rentrées içi (version java, spring etc...).
- **main** : Ce dossier contient le code utilisé dans la Backend:
    - **resources**: Contient :
        - le fichier **application.properties** qui constitue des paramètres globals de l'application (paramètres base de données etc...).
        - le dossier **synoptiques** où nous retrouverons les synoptiques des différentes zones qu'on a chargé.
    - **java** : Ce dossier contient le code utilisé pour la backend. Il est structuré comme suit:



    Le dossier configuration contient des paramètres pour le fonctionnement de l'api.

    L'api suit un modèle par services, chaque service appel une table de notre base de données et contrôle comment l'utilisateur intéragit avec les tables de la base de données. Chaque service est composé au moins de:

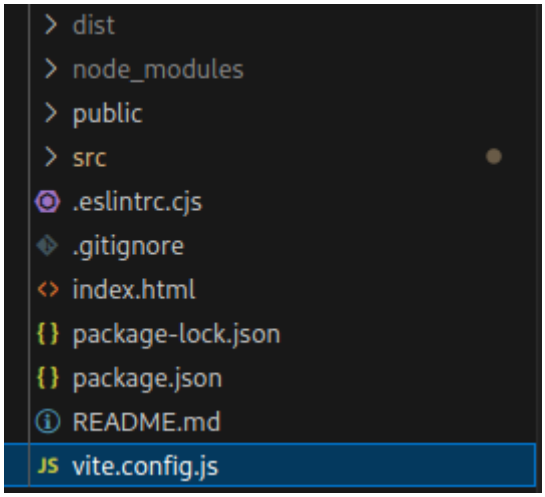
    - *Service*Repository.java: Qui lie l'api à la base de données.
    - *Service*.java: Qui définie la structure de la base dans l'api.
    - *Service*Service.java: Définie la logique des interactions de l'api avec les tables.
    - *Service*Controller.java: Map les fonctions du service à un endpoint (url).

    Les services peuvent contenir des fichiers DTO, ces fichiers servent à définir la logique de communication des objets (Exemple: Si on crée un utilisateur, son id est générée automatiquement. Ainsi, CreateUserDto.java définie les champs nécessaires pour la création d'un utilisateur).

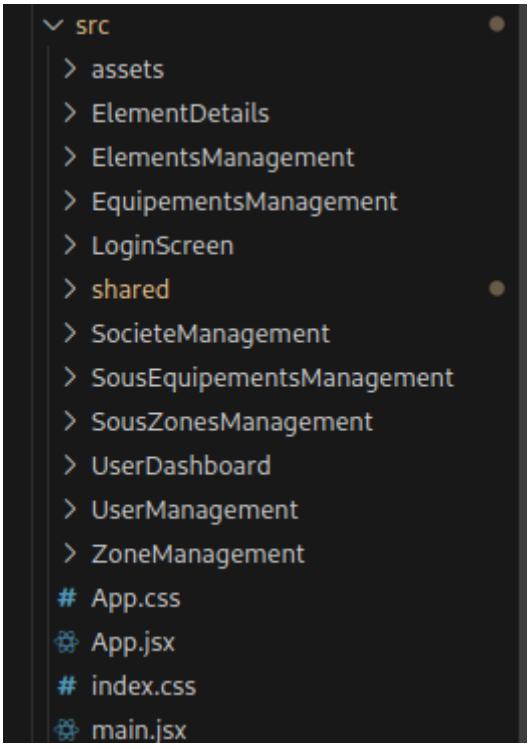
    - SynoptiqueController.java: Définit la logique de réception d'une synoptique de la Front End et sa sauvegarde en local.

Structure FrontEnd

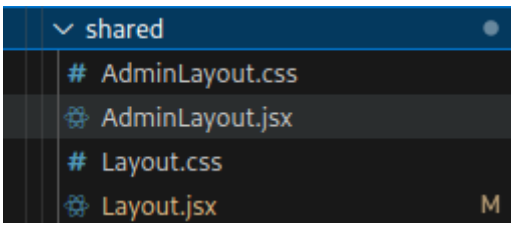
La front end est structurée comme suit:



- **vite.config.js** : Définit la configuration de la plateforme (Port etc...).
- **node_modules** : Contient les différents packages utilisés dans la plateforme. S'il n'existe pas, on lance **npm install** dans le dossier de la plateforme pour les installer.
- **src**: Contient le code de la plateforme:



- **App.jsx** : Définit le routage de notre plateforme.
- **main.jsx** : Constitue le programme qui lance la plateforme.
- **assets** : contient des ressources comme le logo de l'entreprise.
- ***Dossier*Management** : Ces dossiers contiennent les composants qui constituent les différentes pages de la plateforme admin.
- **UserDashboard** : C'est le code pour le composant de la page d'acceuil pour un client (Table des éléments et leurs classes).
- **Shared** : Contient des composants qui sont partagés par d'autres composants, on y trouve notamment Layout et AdminLayout, qui sont les code pour la sidebar et la topbar pour les plateformes client et administrateur respectivement:



- **LoginScreen** : Contient le code pour la page login.

2- Lancement de la plateforme sur LocalHost

BackEnd

- Pour essayer la backend, on exécute **java BackendApplication.java** dans le dossier java.com.lapredictive.backend.
- On pourra aussi lancer le backend après son build. Pour faire, il faut exécuter dans le dossier de la backend la commande **./mvnw clean package**. Cette commande créera un fichier .jar (Java ARchive), qui permet de déplacer la plateforme et de la lancer en utilisant: **java -jar application.jar**. Le nom est généré donc on remplace application.jar par le nom généré. (Opération à faire à chaque fois qu'on modifie le code)

FrontEnd

- Pour lancer la front end sur le local, on exécute d'abord **npm install** si on a pas installé les node_modules, puis **npm run dev**.

3- Ajouts discutés non réalisés

Liés qu'à la plateforme

- **Mot de passe oublié** : L'utilisateur doit être redirigé vers un formulaire où il rentre son email. Un mail de réinitialisation du mot de passe devrait être envoyé à cet email, qui permettera à l'utilisateur de changer son mot de passe.

Pour réaliser celà, je recommande:

-> Créer un dossier pour ce composant dans le dossier FrontEnd.

->Créer le code jsx et css pour le styler pour 2 pages: forme où l'utilisateur rentre son email et click sur bouton envoyer email de récupération, et forme en face de laquelle l'utilisateur se retrouve lorsqu'il click sur le lien qu'il recevra dans le message de l'email. Le bouton *envoyer mail de récupération* devrait être lié à un endpoint créé, et le bouton *Mettre à jour* devrait être lié à l'endpoint de modification d'utilisateur (Voir Mèthode Put sur le controller du service utilisateur).

-> Créer une partie backend qui permet de coder l'envoie automatiques de l'email avec un lien pour changer le mot de passe. Le lien qu'on enverrait avec le message de l'email devrait aussi contenir dans son corps l'id utilisateur afin de pouvoir l'utiliser en suite pour modifier ses informations.

- **Déclaration des pannes**: L'utilisateur doit pouvoir déclarer s'il constate une panne sur un des éléments dont il est en charge. Un email doit ainsi être envoyé à l'admin pour lui

annoncer la panne.

Comme pour le mot de passe oublié, on crée un dossier, où on code une forme constituée de choix de zones, sous zones... élément, pour savoir quel élément est tombé en panne, et une description que le client remplira. Un bouton déclarer qui sera lié à une endpoint dans la backend, qu'on codera, qui enverra un email à l'admin par l'adresse email qu'on paramètrera dans le fichier **application.properties** de la backend.

- **Vue utilisateur** : L'admin doit pouvoir partager la vue de l'utilisateur pour mieux s'informer sur sa panne (un click sur un utilisateur depuis la page admin).

Cette étape demande une compréhension du code **UserManagement** de la front end. On ajoute un **onRowClick** et on code une vue similaire à celle de **UserDashboard** de la front end.

- **Fichier personnels** : L'utilisateur doit pouvoir charger un document personnel sur chaque élément (une image / documentation élément).

On pourra s'inspirer du code **SynoptiqueController** pour coder la fonction qui map un endpoint à une fonction qui enregistre les fichiers localement.

Ajouts partagés avec les autres sujets

- Création des tables:
    - **Capteurs(id, id_element, n_serie, id_type_capteur)** : cette table servira pour sauvegarder les capteurs qu'on installe chez les clients.
    - **Type_capteur(id, type_mesure, reference)** : Pour garder les différents types de capteurs (température, vibratoire, acoustique...)
    - **Alarm(id, id_element, id_type_capteur, valeur_max)** : Cette table servira pour définir le seuil que le capteur ne doit pas dépasser, sinon un message d'alert est envoyé à l'admin.
    - **Personne_A_Prevenir(id, id_utilisateur, id_souszone)** : Cette table définira les personnes à prevenir au cas où un capteur dépasse la valeur max de sa mesure programmée.
    - **Alarm_log(id, date, capteur, mesure, client, societe, zone, sous_zone,...)** : Cette table servira pour sauvegarder les différentes alarmes envoyées à l'admin. Elle devra contenir le maximum d'information pour que l'admin ait une bonne visibilité sur les alarmes.
    - **Coffret(id, id_Zone/sousZone)** : Fait référence au coffret qu'on installe chez les clients.
    - **Materiel_coffret(id, reference, type, num_serie, autres_champs)** : Référencie les différents matériaux dans le coffret. Les autres champs peuvent être le login et mdp si c'est un RevPi...
    - **type_materiel (id, type, autres_champs)**: Fait référence aux types du matériel du coffret (RevPi, Modem...)

4- Comptes pour tester la plateforme

Admin

utilisateur: admin@maintenance-predictive.com

mdp: admin

Client

utilisateur: demonstrateur@maintenance-predictive.com

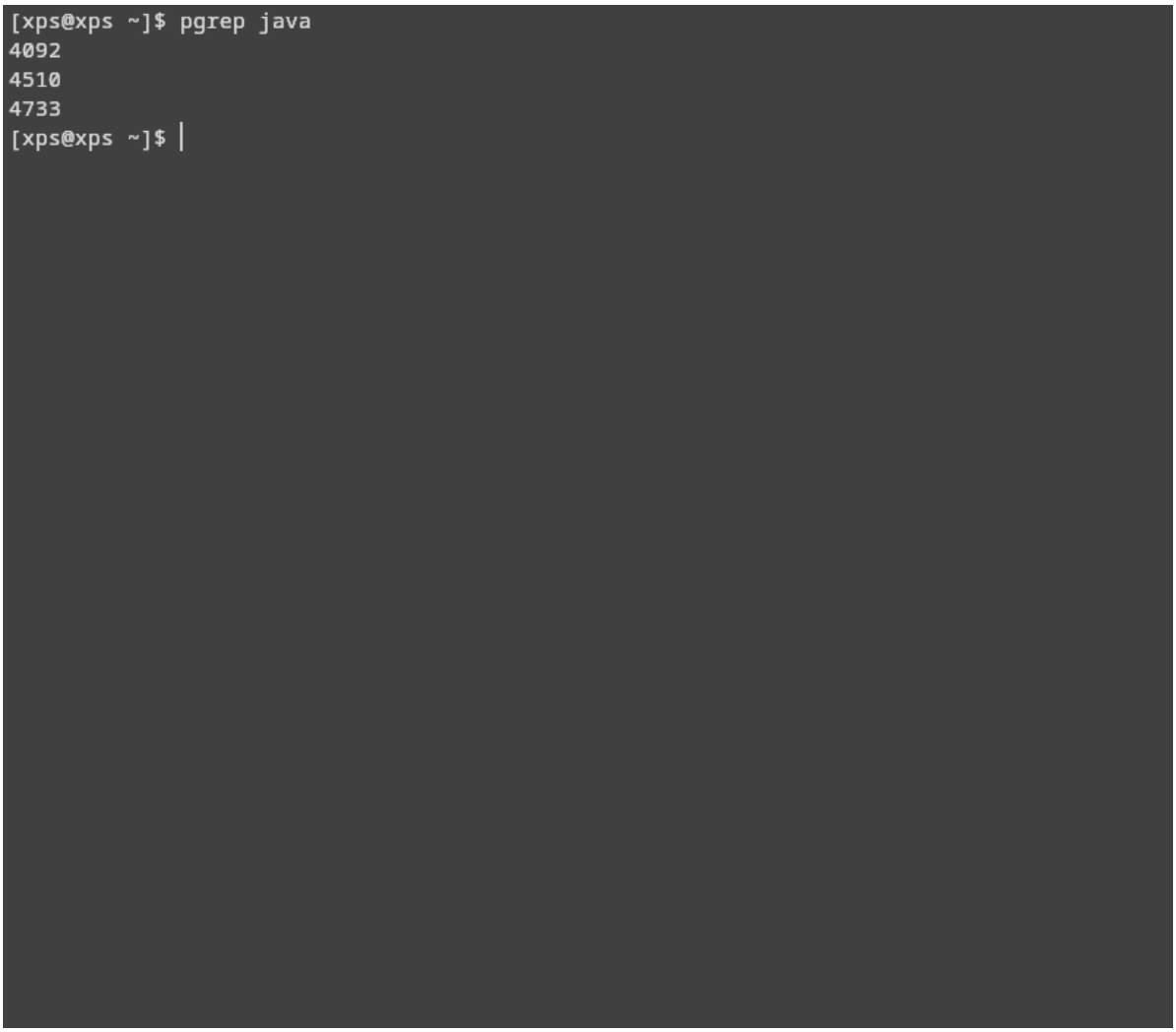
mdp: demonstrateur

5- Comment relancer le serveur Backend

On cherche les processes java

On utilise **pgrep java**

On reçoit un résultat comme suit



On utilise la commande **kill {pid}** sur les ids pour tuer les processes.

On relance le serveur en utilisant **java -jar Application.jar** (On change Application par le nom de notre jar).