AP 2 Ged

Contexte :

L'objectif de cet AP2 est de créer une gestion électronique de documents (GED) en ligne, reliée à une base de données MongoDB. Cette GED permettra de créer des comptes de connexion afin de sécuriser l'accès aux documents. De plus, un lien vers un serveur de fichiers sera établi pour garantir une sauvegarde sécurisée des documents. Au sein de l'entreprise JSB, disposer d'une bonne GED est essentiel pour protéger et retrouver ses documents, surtout lorsque ces derniers contiennent des informations sensibles concernant des médicaments ou des patients.

Les technologies que j'ai choisies pour ce projet sont : React, MongoDB/Compass, Postman, Amazon Simple Storage Service et Node.js.

Voici ci-dessous quelques étapes importante a la création de mon projet

Rapport sur le Projet GED (Gestion Électronique de Documents) - Partie 1

Dans cette première partie du rapport, je vais décrire les éléments initiaux de notre projet GED en ligne, en me concentrant sur la mise en place de l'interface utilisateur.

Introduction

Le projet GED est une application web de gestion électronique de documents conçue pour simplifier la gestion des fichiers et documents. Dans cette section, je vais me concentrer sur la mise en place de l'interface utilisateur initiale de l'application.

Interface Utilisateur

L'interface utilisateur est un aspect clé de notre application, car elle permet aux utilisateurs de naviguer et d'interagir avec leurs documents. Nous avons choisi d'utiliser React, une bibliothèque JavaScript populaire, pour la création de l'interface utilisateur interactive. Voici une vue d'ensemble des composants principaux de notre interface utilisateur que j'ai développé :

1. Sidebar (Barre latérale) : La barre latérale affiche une liste de dossiers et de fichiers, permettant aux utilisateurs de naviguer dans la hiérarchie des documents. Chaque élément de la barre latérale est un lien cliquable qui permet d'accéder rapidement à un dossier ou à un fichier spécifique. Pour créer cette barre latérale, nous avons utilisé un composant React de base avec une structure HTML simple et des icônes pour représenter visuellement les dossiers et les fichiers.

```
// Sidebar.js

import React from 'react';
import './Sidebar.css';
import folderIcon from '../svg/folder-solid.svg';
import fileIcon from '../svg/file-solid.svg';

const Sidebar = () => {
    return (
        <div className="sidebar">
            <ul>
                <li>
                    <img src={folderIcon} alt="Dossier" />
                    Dossier 1
                </li>
                <li>
                    <img src={fileIcon} alt="Fichier" />
                    Fichier 1
                </li>
                {/* Ajoutez d'autres éléments ici */}
            </ul>
        </div>
    );
}

export default Sidebar;
```

Exemple de code de la barre latérale :

Dans cet exemple, folderIcon et fileIcon sont des images SVG utilisées comme icônes.

2. FileExplorer (Explorateur de fichiers) : Ce composant affiche la structure des fichiers sous forme d'arborescence. Les dossiers peuvent être développés ou repliés pour afficher leur contenu. Nous avons utilisé React pour rendre cette arborescence dynamique et interactive.

Exemple de code partiel de l'explorateur de fichiers :

```
// FileExplorer.js

import React, { useState } from 'react';

const FileExplorer = ({ fileStructure }) => {
    // État local pour suivre les nœuds déroulés
    const [expandedNodes, setExpandedNodes] = useState({});

    // ...
}
```

Dans ce composant, fileStructure est passé en tant que propriété, représentant la structure initiale des fichiers. Nous utilisons également l'état local pour suivre les nœuds déroulés.

3. Breadcrumb (Fil d'Ariane) : Le fil d'Ariane est utilisé pour indiquer la hiérarchie actuelle de la navigation dans les dossiers. Il affiche les noms des dossiers dans l'ordre depuis la racine jusqu'au dossier actuel.

Exemple de code du fil d'Ariane :

```
// Breadcrumb.js

import React from 'react';
import './Breadcrumb.css';

const Breadcrumb = () => {
    return (
        <div className="breadcrumb">
            Dossier1 {'>'} Sous-dossier1 {'>'} Fichier1.txt
        </div>
    );
}
```

4. ActionBar (Barre d'actions) : La barre d'actions permet aux utilisateurs d'ajouter des fichiers ou de télécharger des dossiers. Pour ajouter un fichier, nous avons ouvert un explorateur de fichiers en utilisant un composant d'entrée de fichier.

Exemple de code partiel de la barre d'actions :

```
// ActionBar.js

import React, { useState } from 'react';
import './ActionBar.css';
import folderIcon from '../svg/plus-solid.svg';
import uploadIcon from '../svg/upload-solid.svg';

const ActionBar = () => {
    // ...
}
```

Dans ce composant, nous utilisons un état local pour suivre le fichier sélectionné à télécharger.

Partie 2

Dans cette deuxième partie du rapport, je vais poursuivre la description de notre projet GED en ligne en mettant l'accent sur la gestion des événements utilisateur, la navigation, et l'interaction avec la base de données.

Gestion des Événements Utilisateur

Une application web interactive comme la nôtre repose sur une gestion efficace des événements utilisateur. Nous utilisons React pour gérer ces événements de manière réactive et structurée.

Gestion des Clics sur les Nœuds de l'Explorateur de Fichiers :

Lorsqu'un utilisateur clique sur un nœud (dossier ou fichier) dans l'explorateur de fichiers, nous devons réagir à cet événement pour afficher ou masquer les sous-dossiers, le cas échéant. Voici comment nous gérons cela :

javascript

Copy code

// FileExplorer.js

```
const handleNodeClick = (nodeName) => {
  setExpandedNodes((prevState) => ({
    ...prevState,
    [nodeName]: !prevState[nodeName],
  }));
};
```

Lorsque l'utilisateur clique sur un nœud, cette fonction est appelée, mettant à jour l'état local pour indiquer si le nœud est déroulé ou replié.

Navigation avec React Router :

Pour gérer la navigation entre différentes pages de l'application, nous utilisons React Router. Cela nous permet de définir des routes pour chaque page et d'afficher dynamiquement le contenu en fonction de l'URL. Voici comment nous utilisons React Router dans notre application :

javascript

Copy code

```
// App.js

import { BrowserRouter as Router, Routes, Route, Link } from 'react-router-dom';

function App() {
  return (
    <Router>
      <div className="App">
        {/* ... */}
        <Link to="/inscription">Inscrivez-vous</Link>

        <Routes>
```

```
            <Route path="/inscription" element={<RegistrationForm />} />
            {/* ... autres routes */}
          </Routes>
        </div>
      </Router>
    );
}
```

Dans cet extrait de code, nous définissons une route vers la page d'inscription et utilisons des liens <Link> pour naviguer entre les pages.

Interaction avec la Base de Données

L'interaction avec la base de données est une partie cruciale de notre application GED. Nous utilisons MongoDB, une base de données NoSQL, pour stocker les informations sur les dossiers et les fichiers.

Connexion à la Base de Données :

La première étape est d'établir une connexion à la base de données MongoDB. Voici comment nous faisons cela :

javascript

Copy code

```
// database.js

const { MongoClient } = require('mongodb');

const uri = 'mongodb://localhost:27017/ged';
const client = new MongoClient(uri);

async function connectToMongoDB() {
```

```
    try {
        await client.connect();
        console.log('Connected to MongoDB');
    } catch (error) {
        console.error('Error connecting to MongoDB', error);
    }
}

module.exports = { connectToMongoDB };
```

Cette connexion est établie au démarrage de l'application.

Opérations CRUD :

Nous effectuons des opérations CRUD (Créer, Lire, Mettre à jour, Supprimer) pour interagir avec la base de données. Par exemple, pour créer un nouveau dossier, nous insérons un document dans la collection MongoDB correspondante :

javascript

Copy code

```
// Exemple de création d'un dossier dans la base de données

const folder = {
    name: 'Nouveau Dossier',
    parentFolderId: '12345', // ID du dossier parent
};

const result = await client.db('ged').collection('folders').insertOne(folder);
```

De même, nous effectuons des requêtes pour lire et afficher la structure des fichiers et dossiers à l'utilisateur.

Technologies Utilisées :

Node.js : Nous avons choisi Node.js comme environnement d'exécution pour notre serveur backend. Node.js est rapide, événementiel, et parfaitement adapté pour les applications réseau.

Exemple de code - Initialisation d'un serveur Node.js :

javascript

Copy code

```
const express = require('express');

const app = express();

const PORT = 5000;


app.listen(PORT, () => {
  console.log(`Serveur en écoute sur le port ${PORT}`);
});
```

Express.js : Express.js est un framework web minimal pour Node.js. Il simplifie la création d'API RESTful et de routes.

Exemple de code - Configuration d'une route avec Express :

javascript

Copy code

```
const express = require('express');

const app = express();


app.get('/api/users', (req, res) => {
  // Logique de récupération des utilisateurs depuis la base de données
  res.json(users);
```

});

MongoDB : Nous avons utilisé MongoDB, une base de données NoSQL, pour stocker nos données. MongoDB est extensible, flexible et adapté aux données non structurées.

Exemple de code - Connexion à MongoDB avec Mongoose :

javascript

Copy code

```
const mongoose = require('mongoose');

const dbUrl = 'mongodb://localhost:27017/monapp';


mongoose.connect(dbUrl, { useNewUrlParser: true, useUnifiedTopology: true })
  .then(() => {
    console.log('Connecté à MongoDB');
  })
  .catch((err) => {
    console.error('Impossible de se connecter à MongoDB:', err);
  });
```

Mongoose : Mongoose est une bibliothèque Node.js pour interagir avec MongoDB. Elle offre un schéma de données, des validations et des fonctionnalités puissantes pour faciliter la gestion des données.

Exemple de code - Création d'un modèle utilisateur avec Mongoose :

javascript

Copy code

```
const mongoose = require('mongoose');


const userSchema = new mongoose.Schema({
  name: String,
```

```
  email: String,
  password: String,
});

const User = mongoose.model('User', userSchema);
```

Fonctionnalités Implémentées :

Routes API RESTful : Nous avons créé plusieurs routes API pour gérer les opérations CRUD (Create, Read, Update, Delete) sur les ressources telles que les utilisateurs.

Exemple de code - Routage API pour la création d'un utilisateur :

```
javascript
Copy code
app.post('/api/users', (req, res) => {
  const newUser = new User(req.body);
  newUser.save()
    .then(() => {
      res.status(201).json(newUser);
    })
    .catch((err) => {
      res.status(400).json({ error: 'Erreur lors de la création de l\'utilisateur' });
    });
});
```

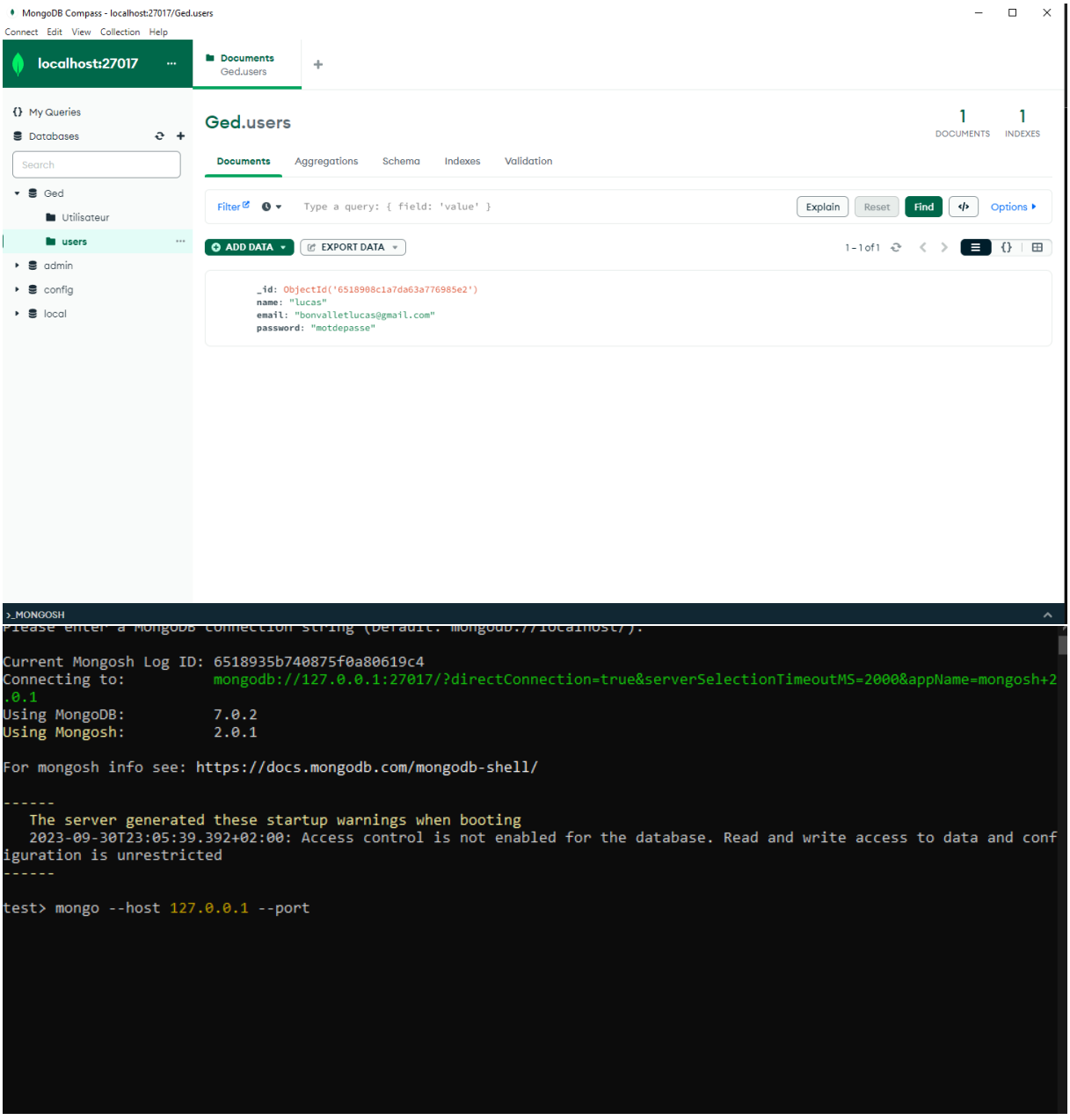
Gestion des erreurs : Nous avons mis en place une gestion des erreurs pour gérer les cas où quelque chose ne se passe pas comme prévu, par exemple lors de la validation des données.

Exemple de code - Middleware de gestion des erreurs :

javascript

Copy code

```
app.use((err, req, res, next) => {
  console.error(err.stack);
  res.status(500).send('Quelque chose s\'est mal passé!');
});
```



Conclusion :

Ce projet ne représente pas, à mes yeux, une grande réussite. J'ai perdu beaucoup de temps à cause du passage d'un ordinateur à un autre, ce qui a engendré de nombreuses erreurs. Toutefois, je ne dirais pas que je suis insatisfait : j'ai appris de nombreuses nouvelles technologies et j'ai pu renouer avec le Backend. Côté Frontend, les choses se compliquent. Mon sens du design, comparable à celui d'un enfant de 8 ans, ne m'aide pas à créer des éléments esthétiquement plaisants. Même en disposant d'une base de travail pré-existante avant le projet, j'ai eu du mal à concrétiser mes idées. Le temps a également été un adversaire redoutable. Malgré les deux mois qui nous ont été accordés, je n'ai pas pu atteindre les objectifs du projet. Cependant, je compte poursuivre, améliorer, voire même finaliser ce projet pour le présenter lors de ma soutenance. Je suis convaincu de son potentiel et je souhaite le mener à terme. Je vous remercie d'avoir pris le temps de lire mon rapport. Bien qu'il soit court et critique, je l'ai rédigé à la dernière minute afin de privilégier la continuation de mon projet.