Crud Nodejs, Reactjs, MySql

Ce projet se fera sur Google Chrome car Mozilla Firefox n'apprécie pas les requêtes multiserveurs.

Dans ce projet vous allez concevoir une application web avec un crud complet en nodejs pour le backend, et React.js pour le front le tout avec la base de données en MySQL.

Vous allez créer un dossier (moi je l'ai appelé crudNodeReact), et à l'intérieur un dossier nommé backend. Vous allez ouvrir un terminal et vous allez devoir générer le fichier **package.json** afin de préparer votre environnement avec la commande **npm init -y**. Vous aurez le **package.json** qui va s'afficher, il doit être dans le dossier si ce n'est pas le cas mettez-vous dans votre dossier et relancer la commande puis effacer le premier.

Vous installerez ensuite deux packages Express et MySQL, avec la commande **npm install mysql express**, express c'est un framework js, qui permet de construire des applications basées sur node.js, il créera la partie serveur.

Vous avez désormais le dossier **Node Modules** ainsi que le fichier **package-lock.json**, vous allez maintenant créer un fichier **server.js** dans le dossier backend, nous allons mettre dans ce fichier notre code node. Mais avant cela vous allez créer votre application en **React**, pour cela votre terminal doit être sur le dossier racine et non plus dans le dossier backend, ensuite vous taperez la commande **npx create-react-app frontend**, comme cela ce sera déjà dans un dossier frontend, oui on sépare bien le front et le back. Cela va créer toute la base de votre appli React. Ensuite vous aurez besoin de cors, enfait votre backend ne permettra pas à votre frontend d'accéder à son api donc le cors va résoudre cela, tapez la commande **npm i cors**, cors sera désormais visible dans le node module de votre backend.

Allez sur server.js et commençons à coder,

```
const express = require("express"); //création de const express qui permet d'acceder à express
const app = express(); //permet  d'utiliser les méthodes de l'objet express dans la variable "app"

app.listen(8081, () => { //attribue le port  8081 au serveur et exécute une fonction anonyme listen
    console.log('Server is running on port 8081');
})
```

Ensuite vous pouvez exécuter votre backend en console en tapant **node server.js**, et vérifier qu'il restitue votre message. Super vous allez pouvoir créez votre api sur votre server ainsi que la bdd, vous allez pouvoir lancer MySQL (phpMyAdmin). Ensuite vous allez ajouter les const de connexion pour cors et mysql,

```
const cors = require("cors");//permet aux différents serveurs d'échanger des données entre eux.
const mysql = require("mysql")
```

Vous allez créer les options de cors pour vous assurer un bon fonctionnement de cors et des requêtes multiserveurs, je n'invente rien tout est dans la doc de cors.

```
const corsOptions = {
    origin: [
        'http://localhost:3000',
        'http://localhost:8081',
    ],
    optionsSuccessStatus: 200, // some legacy browsers (IE11, various SmartTVs) choke on 204
    methods: 'GET,HEAD,PUT,PATCH,POST,DELETE',
    headers: 'Content-Type,Authorization',
    credentials: true, // allow cookies to be sent with requests
};
```

Ensuite on va créer notre connexion avec la base de données, vous allez créer une base de données que je vais appeler **crudnode** et créer une table étudiant (**student** pour moi), je laisserai 3 colonnes dans la table, **id** en auto-increment, ensuite le **name** en varchar et le nombre que vous voulez, ensuite **email** en varchar et pareil pour la longueur. Super créons donc la connexion à la bdd,

```
mysql.createConnection({
    host: 'localhost',
    user: 'root',
    password: '',
    database: 'crudnode'
})
```

Ensuite créons un endPoint (une route),

```
app.get("/", (req, res) => {  //crée un endpoint a ladresse "/" et ensuite il attend une requete et une reponse
    res.json("Salut a toi de puis le backend"); //en reponse on renvoie le message
})
```

Vérifier que votre server tourne en terminal puis lancer l'url localhost :8081 vous aurez accès a votre message, du endPoint, ok parfait on peut continuer. Vous pouvez couper le serveur et lancer la commande **npm start** il lancera votre backend, et recharger votre page du navigateur afin de vérifier que tout fonctionne toujours.

Parfait on va pouvoir gérer les donnée depuis la bdd, avant cela vous allez attribuer une variable a votre connexion,

```
const database = mysql.createConnection({
    host: 'localhost',
    user: 'root',
    password: '',
    database: 'crudnode'
})
```

Alors on va faire en sorte que cette route nous affiche tous nos étudiants en créant la requête,

```
app.get("/", (req, res) => {  //crée un endpoint a ladresse "/" et ensuite il attend une requete et une reponse
    //res.json("Salut a toi de puis le backend"); //en reponse on renvoie le message
    const sql = "SELECT  * FROM student"; //crée la  requête SQL pour récupérer toutes les informations du tablea
    database.query(sql, (err, data) => { //fonction de rappel qui prend deux arguments err pour les error et data
      if(err) return res.json("Error"); // la condition  est vraie, on execute ce qu'il y a après le "if", sinon
      return res.json(data);
    })
```

Vous pouvez sauvegarder et vous allez passer au code frontend, allez dans le app.js, vous allez configurer votre routeur **React**, **BrowserRouter** est un bibliothèques qui permet de transformer une app **React** en spa, cela permet l'affichage des différentes pages.

```
import './App.css';
import 'bootstrap/dist/css/bootstrap.min.css';
import {BrowserRouter, Routes, Route} from  "react-router-dom";
```

Après avoir fait les imports, vous allez utiliser les routes du router du navigateur et avec le path définir le chemin qui sera '/' et qui donnera le composant Student.

```
function App() {
  return (
    <div className="App">
      <BrowserRouter>
        <Routes>
          <Route path='/' element={<Student />}></Route>
        </Routes>
      </BrowserRouter>
    </div>
  );
}
```

Vous allez ensuite créer le component Student, dans le folder src, vous allez créer la fonction, le return ainsi que l'export. Ensuite vous importez le component, dans le app.js. Ensuite vous allez pouvoir exécuter votre frontend. Dans votre terminal vous allez pointer sur le frontend et vous allez lancer la commande **npm start**. Si jamais vous avez des erreurs à vous de les résoudre seul, elles sont très simples et vous y serez souvent confronté. Bon tout est réglé et tout fonctionne très bien, vous allez pouvoir avancer, vous arriver à afficher le mot étudiant (c'est un test d'affichage).

Vous allez enlever les fichiers inutile pour le moment, donc appTest.js, et logo.svg également, ensuite on s'occupe du component d'affichage des students, vous allez faire un table pour l'affichage vous avez l'habitude maintenant.

```
import React from 'react'

function Student() {
  return (
    <div className='d-flex vh-10 bg-primary justify-content-center align-items-center'>
        <div className='w-50 bg-white rounded'>
            <button className='btn btn-success'>Ajouter</button>
            <table className='table'>
                <thead>
                    <tr>
                        <th>Nom </th>
                        <th>Email</th>
                    </tr>
                </thead>
                <tbody>

                </tbody>
            </table>
        </div>
```

Dans le tbody vous utiliserez la méthode map, mais avant cela, il va falloir récupérer les données de votre backend, vous allez utiliser le useEffect(), et axios (qui n'est surement pas installé), pour qu'il obtienne les données de l'Api,

```
function Student() {

    useEffect(() => { // This useEffect hook is used to fetch data from the server when the component is mounted.
        axios.get('http://localhost:8081/')
        .then(res => console.log(res))  //Log the response
        .catch(err => console.log(err)); //Log any errors
    }, [])  // Empty dependency array, so it runs once on mount
```

Voilà qui est mieux, pensez à tester à chaque étape afin d'être sûr, et cela s'actualise automatiquement. Vous allez pouvoir désormais entrer quelques données dans votre table en dur afin de pouvoir les afficher et avancer dans votre code, vous allez maintenant les afficher. Vous allez les définir avec leurs setter dans une const,

```
function Student() {

    const [student, setStudent] = useState([])

    useEffect(() => { // This useEffect hook is
```

Et bien sûr les retourner en tant que réponse,

```
useEffect(() => { // This useEffect hook is used to fetch d
    axios.get('http://localhost:8081/')
    .then(res => setStudent(res.data))  //Log the response
```

Ensuite les afficher dans votre tbody avec la méthode **map**, essayer seul.

Bravo à ceux qui ont réussi pour les autre voici la correction,

```
<tbody>
    {
        student.map((data, i) => (
            <tr key={i}>
                <td>{data.name}</td>
                <td>{data.email}</td>
            </tr>
        ))
    }
</tbody>
```

On appelle donc notre tableau student et on applique la méthode map, ensuite on appelle les data et le i qui est l'index de chaque élément, le key est une façon d'appeler les index de chaque élément, et on affiche. Tout le monde a un résultat ?? Super on va passer à la prochaine étape, elle va être de créer une nouvelle colonne action et d'insérer deux boutons qui seront modifier et supprimer bien sûr. Faites cela seul, on voit ensemble la correction.

```
<tr key={i}>
    <td>{data.name}</td>
    <td>{data.email}</td>
    <td>
        <button className='btn btn-primary'>Modifier</button>
        <button className='btn btn-danger'>Supprimer</button>
    </td>
</tr>
```

Voilà pour nos deux boutons, bon maintenant lors du click sur le bouton ajouter on devra bien atterrir sur un formulaire, vous allez donc gérer cela. Vous allez transformer cela en lien avec **Link to**,

```
<Link to="/create" className='btn btn-success'>Ajouter</Link>
```

Et bien sûr faire l'import si ce n'est pas déjà fait, on n'oublie pas de rajouter la route également.

```
<Route path='/' element={<Student />}></Route>
<Route path='/create' element={<CreateStudent />}></Route>
```

Il reste à créer ce component désormais, donc new file, donc après cela vous devriez lors du click sur formulaire.

```
import React from 'react'

function CreateStudent() {
  return (
    <div>
      Create Student
    </div>
  )
}

export default CreateStudent
```

Et vous allez créer le formulaire d'inscription,

```
function CreateStudent() {
  return (
    <div className='d-flex vh-100 bg-primary justify-content-center align-items-center'>
        <div className='w-50 bg-white rounded p-3'>
            <form>
                <h2>Ajouter Etudiant</h2>
                <div className='mb-2'>
                    <label htmlFor="">Nom </label>
                    <input type="text" placeholder='Entrez votre nom' className='form-control' />
                </div>
                <div className='mb-2'>
                    <label htmlFor="">Email </label>
                    <input type="email" placeholder='Entrez votre email' className='form-control' />
                </div>
                <button className='btn btn-success'>Soumettre</button>
            </form>
        </div>
    </div>
```

C'est quelque chose que vous connaissez bien il n'y a pas de surprise ici, vérifier si le lien fonctionne et si cela affiche bien votre formulaire d'inscription. Passez ensuite à la gestion des données qui seront rentré dans les champs du formulaire, vous allez créer des const qui géreront le changement d'état pour ces deux champs à vous de jouer,

```
function CreateStudent() {

    const [name, setName] = useState('')
    const [email, setEmail] = useState('')

  return (
```

Super, maintenant obtenez ces valeurs lorsque vous entrerez la valeur dans le formulaire, vous devrez donc mettre à jour cela. Vous allez simplement utiliser et modifier l'évènement,

```
<h2>Ajouter Etudiant</h2>
<div className='mb-2'>
    <label htmlFor="">Nom </label>
    <input type="text" placeholder='Entrez votre nom' className='form-control' />
    onChange = {e => setName(e.target.value)}
```

Vous allez donc faire de même pour l'email, après cela vous allez devoir gérer le click sur le submit avec une fonction, avec le handle de l'évènement, et transmettre ces données au backend.

```
<form onSubmit={handleSubmit}>
```

Ensuite donc vous créez cette fonction,

```
function handleSubmit(event) {
    event.preventDefault();
    axios.post('http://localhost:8081/create', {name, email})
    .then(res => {
        console.log("Created Student", res)
    })
}
```

Si la fonction a réussi cela affichera le résultat en console, et vous ramener à la page d'accueil, pour cela vous allez devoir utiliser, le router de **react navigate**.

```
const [name, setName] = useState('')
const [email, setEmail] = useState('')
const navigate = useNavigate();

function handleSubmit(event) {
    event.preventDefault();
    axios.post('http://localhost:8081/create', {name, email})
    .then(res => {
        console.log("Created Student", res);
        navigate('/');
    })
}
```

Bravo, vous allez pouvoir passer à la création du **endPoint create** et votre logique backend, donc rendez-vous sur **server.js**, vous allez devoir préciser à votre app que vous envoyez du json en premier lieu,

```
app.use(express.json());
app.use(cors());
```

Ensuite vous allez créer le endPoint ainsi que la logique de transfert des données, (rappelez-vous le endPoint de page d'accueil) à vous de jouer et d'essayer cela seul.

```
app.post('/create', (req, res) => {
    const sql = "INSERT INTO student (`name`, `email`) VALUES (?)";
    const values = [
        req.body.name,
        req.body.email
    ]
    database.query(sql, [values], (err, data) => {
        if(err) return res.json("Error");
        return res.json(data);
    })

})
```

Aucune surprise ici non plus, parfait a ceux qui ont réussi, pour les autres voici la solution mais bravo à vous aussi. Désormais il reste le test du formulaire, ok pour moi, cela ajoute bien et me ramène à la page d'accueil et en bdd cela a ajouté mon nouvel étudiant.

Vous allez pouvoir passer à l'update, donc au click sur le bouton vous devriez aller sur un nouveau form, donc on le passe en Link to et vous faites l'adresse seul.

```
<Link to={`update/${data.id}`} className='btn btn-primary'>Modifier</Link>
<button className='btn btn-danger ms-2'>Supprimer</button>
```

Voici la route de la page update,

```
<Route path='/update/:id' element={<CreateStudent />}></Route>
```

Et vous importez comme toujours, vous testez votre bouton pour voir si cela fonctionne, tout fonctionne, super. Vous allez pouvoir passer au formulaire de modification alors, encore une fois il n'y aura pas beaucoup de différence avec la création vous pouvez copier-coller le fichier **create** dans **l'update**. Pensez à changer ce qui doit l'être, je vous laisse faire cela, pensez également à la méthode qui sera en **PUT** et non plus en **POST**.

```
function handleSubmit(event) { //handle submit function for t
    event.preventDefault();
    axios.put('http://localhost:8081/create', {name, email})
```

Vous passerez bien le **name** ainsi que **l'email** mais il sera judicieux de passer **l'id** également, pour obtenir **l'id** à partir des values d'url, vous utiliserez la fonction **useParams()**.

```
const [name, setName] = useState('')
const [email, setEmail] = useState('')
const {id} = useParams();
const navigate = useNavigate();
```

OK maintenant que vous pouvez obtenir **l'id**, vous allez pouvoir le transmettre à votre requête.

```
function handleSubmit(event) { //handle submit function for the f
    event.preventDefault();
    axios.put('http://localhost:8081/create', {id, name, email})
```

OK parfait, penser à modifier ce qui ne va pas sur cette ligne je vous laisse trouver. Vous me dites et on voit cela ensemble donc ne recopiez pas bêtement.

```
function handleSubmit(event) { //handle submit function for the form
    event.preventDefault();
    const updatedStudent = {name, email};
    axios.put(`http://localhost:8081/update/${id}`, updatedStudent)
   .then(res => {
        console.log(res);
        navigate('/'); //back to home if ok
    }).catch(err => console.log(err)); //if error print on conole
```

Super tout est prêt vous allez devoir vous occuper du back sur le fichier server.js, vous allez pouvoir copier-coller le code du **endPoint create** et effectuer les modifications pour **l'update**, d'après vous quelles sont-elles ??

La méthode, la route, la requête, ainsi que la récup de l'id, et donc la values.

```
app.put('/update/:id', (req, res) => {
    const sql = "update student set `name` = ?, `email` = ? WHERE id = ?";
    const values = [
        req.body.name,
        req.body.email
    ]
    const id = req.params.id;
    database.query(sql, [...values, id], (err, data) => {
        if(err) return res.json("Error");
        return res.json(data);
    })

})
```

Evidemment après avoir mis à jour un student vous devrez retourner à la page d'accueil, ensuite vous pouvez tester cela. Il est possible que vous deviez redémarrez les serveurs, mais pas obligatoire, à vérifier après le test si cela fonctionne parfaitement ou pas. Ok super tout fonctionne parfaitement, il vous reste désormais à créer le **delete** qui sera le dernière étape et encore une fois la plus simple.

Allez donc sur le fichier Student.js et modifier légèrement le button qui gère le delete, il vous faudra ajouter la méthode onClick et donc la fonction handleDelete en arrow function à vous d'essayer de la faire seul, je sais que vous pouvez le faire. Créer également la fonction handleDelete qui sera une fonction asynchrone (je vous aide là).

Correction :

```
<button className='btn btn-danger ms-2' onClick={ e => handleDelete(data.id)}>
```

```
const handleDelete = async (id) => {
    try {
        await axios.delete('http://localhost:8081/student/'+id)
        window.location.reload()
    } catch(err) {
        console.log(err);
}
```

Ok super, il reste plus qu'à créer le endpoint sur le fichier Server.js, vous pouvez copier-coller le endpoint de l'update et le modifier pour gagner du temps. Essayer de le faire seul, on se met des défi hein !!!

Correction :

```
app.delete('/student/:id', (req, res) => {
    const sql = "DELETE FROM student WHERE id =?";
    const id = req.params.id;

    database.query(sql, [id], (err, data) => {
        if(err) return res.json("Error");
        return res.json(data);
    })
})
```

Et voilà, on relance le serveur backend est on test, tout fonctionne parfaitement Bravo à tous et toutes vous avez réussi à créer un projet Crud complet en backend Node.js et frontend React, je vous félicite tous et toutes.