React, Node et MySql Login (Partie 7)

Envoi du token vers l'API à chaque refresh de l'application React

Nous n'avons pas encore résolu le problème de perte des infos de login lors d'un refresh. Pour cela nous allons créer une route GET/auth sur l'API node qui sera chargée de vérifier le token envoyé par l'application React et qui renverra les infos de login si le token est vérifié.

Étape 14 : Création de la route GET/auth sur l'API node

index.js > ...

```
12  const { query } = require("./api/services/database.service");
13  const bcrypt = require("bcrypt");
14  const config = require("./api/config");
15  const jwt = require("jsonwebtoken");
16
17  app.get("/auth", async (req, res) => {
18    const auth = req?.cookies?.auth;
19    if(auth){
20      const data = jwt.verify(auth, config.token.secret)
21      if(data){
22        res.json({ data, result: true, message: `Auth OK` });
23      }
24      else{
25        res.json({ data: null, result: false, message: `BAD Auth` });
26      }
27    }
28    else{
29      res.json({ data: null, result: false, message: `BAD Auth` });
30    }
31  });
32
33  app.post("/login", async (req, res) => {
34    const { body } = req;
35    const sql = `SELECT * FROM customer
```

Ln 18 : Nous récupérons le cookie auth (qui contient le token) s'il existe, puis nous le vérifions. S'il s'agit d'un token correct nous renvoyons alors les infos qu'il contient (Ln 22) vers le front. Sinon nous renvoyons une réponse "BAD Auth" (Ln 25 et 29)

…

Étape 15 : Ajout d'un useEffect dans le contexte d'authentification pour envoyer une requête GET/auth vers l'API à chaque chargement du composant (et donc chaque refresh)

src > contexts > AuthContext.jsx > AuthProvider

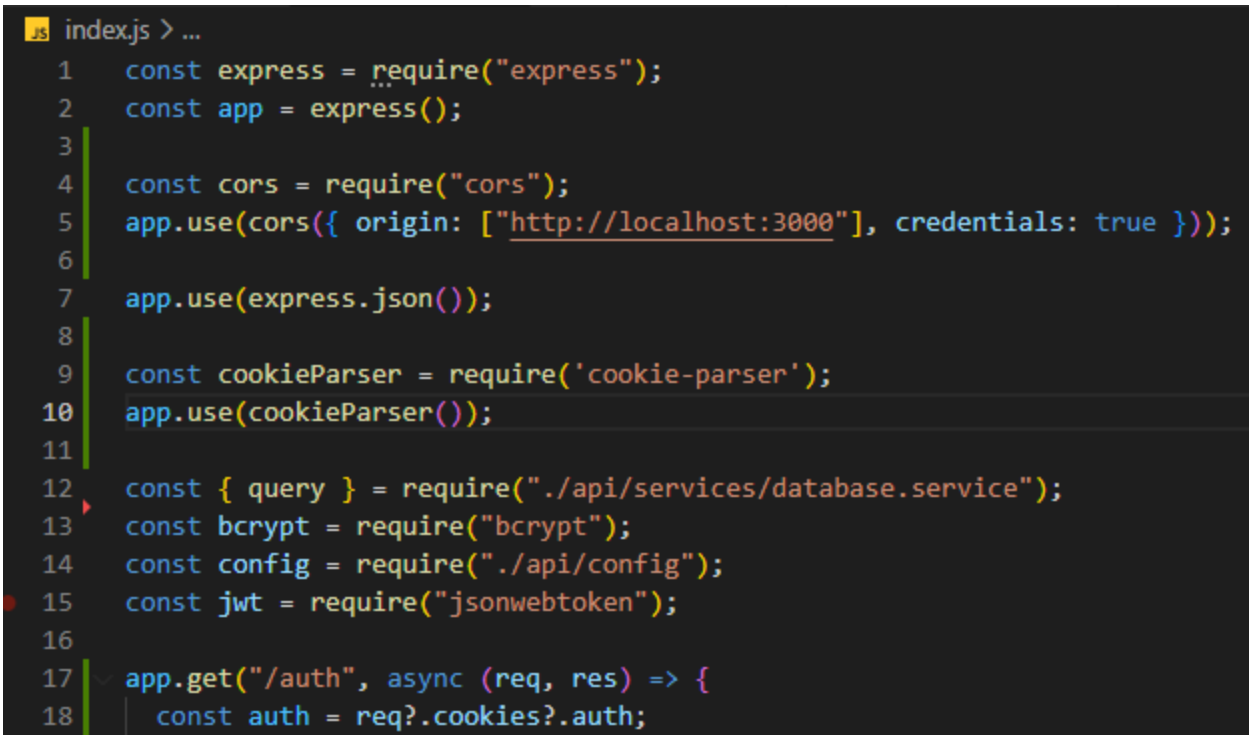
```
1  import { createContext, useState, useMemo, useEffect } from "react";
2
3  const AuthContext = createContext();
4
5  const AuthProvider = ({ children }) => {
6    const [auth, setAuth] = useState(null);
7    const authMemo = useMemo(() => ({ auth, setAuth }), [auth]);
8
9    useEffect(() => {
10
11     fetch("http://localhost:5000/auth", {credentials: "include"})
12       .then((resp) => resp.json())
13       .then((json) => {
14         console.log(json);
15         setAuth(json.data);
16       })
17       .catch((err) => console.log(err));
18
19   }, []);
20
21   return (
22     <AuthContext.Provider value={authMemo}>{children}</AuthContext.Provider>
23   );
24 };
25
26 export { AuthContext, AuthProvider };
```

Ln 11 : credentials include permet d'envoyer les cookies avec la requête
Ln 15 : mise à jour des infos de login avec ce qui est renvoyé par l'API node

Si vous testez en l'état, (faire un refresh suite à un login réussi) ça ne fonctionne pas, il manque 2 choses sur l'API node

…

Étape 16 : Ajout d'options à cors et installation de cookie-parser pour récupérer les cookies côté API



```
index.js > ...
1   const express = require("express");
2   const app = express();
3
4   const cors = require("cors");
5   app.use(cors({ origin: ["http://localhost:3000"], credentials: true }));
6
7   app.use(express.json());
8
9   const cookieParser = require('cookie-parser');
10  app.use(cookieParser());
11
12  const { query } = require("./api/services/database.service");
13  const bcrypt = require("bcrypt");
14  const config = require("./api/config");
15  const jwt = require("jsonwebtoken");
16
17  app.get("/auth", async (req, res) => {
18    const auth = req?.cookies?.auth;
```

Ln 5 : ajout des options origin et credentials à cors
Ln 9 et 10 : ajout de cookie-parser (voir https://www.npmjs.com/package/cookie-parser)

Vous pouvez tester un refresh suite à un login réussi, les infos ne sont plus perdues par le contexte.

Mettez des points d'arrêt des 2 côtés pour comprendre ce qui se passe