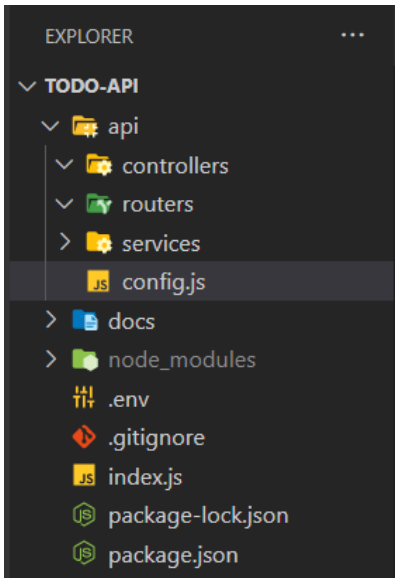
React, Node et MySql Login (Partie 3)

Hachage des mots de passe (pincode)

Étape 5 : Modification de la config

Remplacer la config stockée dans dev.config.json par un unique fichier config à la racine du dossier api et supprimer ensuite le dossier configs

Nous en profitons pour ajouter dans la config le prefix de l'algorithme de hachage que nous allons utiliser (bcrypt)




```
api > config.js > ...
1   const dotenv = require('dotenv');
2   dotenv.config();
3   const config = {
4       dev : {
5           db : {
6               host: "localhost",
7               port: "3306",
8               user: "root",
9               password: "",
10              database: "todo_db"
11          },
12          hash : {
13              prefix: "$2b$08$"
14          },
15      },
16
17      prod : {
18          db : {
19              host: "",
20              port: "",
21              user: "",
22              password: "",
23              database: ""
24          },
25          hash : {
26              prefix: "$2b$08$"
27          },
28      }
29
30  }
31
32  module.exports = config[process.env.NODE_ENV];
```

Dans ce nouveau fichier de config, nous exportons la config en fonction de l'environnement choisi dans le fichier .env à la racine de l'application

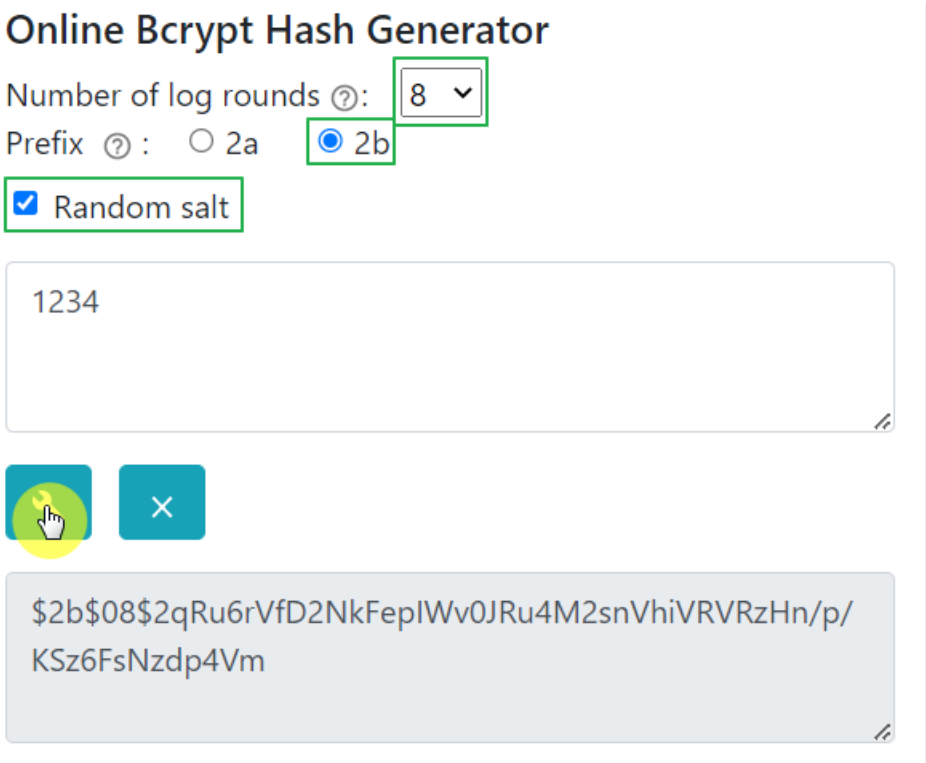
```
.env
1   NODE_ENV=dev
```

Nous devons légèrement modifier le database.service pour utiliser la nouvelle config

```
api > services > database.service.js > ...
1   const mysql = require("mysql2/promise");
2   // const dotenv = require('dotenv');
3   // dotenv.config();
4   // const config = require(`../configs/${process.env.NODE_ENV}.config`)
5   const config = require("../config");
6
7   let db;
8   async function connect(){
9       if(!db){
10          const {host, port, database, user, password} = config.db;
11          db = await mysql.createConnection({
```

Les lignes 2, 3 et 4 sont remplacées par la ligne 5.

Étape 6 : Hachage du mot de passe manuellement en base de données
A l'aide de l'outil https://bcrypthashgenerator.tool-kit.dev/
Hasher le pincode 1234 avec les paramètres suivants :



Puis remplacer 1234 par la valeur générée (sans le préfix) 2qR……4Vm dans phpMyAdmin

Étape 7 : Installation de bcrypt et modification de la route POST/login de l'API

Après avoir installer bcrypt : https://www.npmjs.com/package/bcrypt

Nous modifions la route POST/login dans index.js

```
6   app.use(cors());
7   const bcrypt = require("bcrypt");
8   const config = require("./api/config");
9
10  app.post("/login", async (req, res) => {
11    const { body } = req;
12    //console.log(body);
13    const sql = `SELECT * FROM customer
14                 WHERE is_deleted = 0
15                 AND email = '${body.email}'`;
16    await query(sql)
17      .then(async (json) => {
18        //console.log(json);
19        const user = json.length === 1 ? json.pop() : null;
20        //console.log(user);
21        if (user) {
22          const pincodesMatch = await bcrypt.compare(body.pincode, config.hash.prefix + user.pincode);
23          if(!pincodesMatch){
24              throw new Error("Bad Login");
25          }
26          const { id, email } = user;
27          const data = { id, email };
28          res.json({ data, result: true, message: `Login OK` });
29        } else {
30          throw new Error("Bad Login");
31        }
32      })
33      .catch((err) => {
34        res.json({ data: null, result: false, message: err.message });
35      });
36  });
37
```

Ln 7 et 8 : import du module bcrypt et de la config

Ln 21 et 22 : Si nous récupérons un user en DB (bon email), nous comparons le mot de passe en clair reçu dans le body de la requête HTTP avec le mot de passe hashé en DB auquel il ne faut pas oublier de rajouter le préfix (stocké dans la config)

Ln 23 à 25 : Si les mots de passe ne correspondent pas, nous déclenchons une erreur "Bad Login"

Re tester tous les cas possibles (idem étape 4) depuis le formulaire