Yvon Point : yvon.point@etu.utc.fr
Alberic Preveraud De Vaumas : alberic.preveraud-de-vaumas@etu.utc.fr

RAPPORT projet SR10

création d'un site web

"pôle emploi" : site d'annonces d'emploi

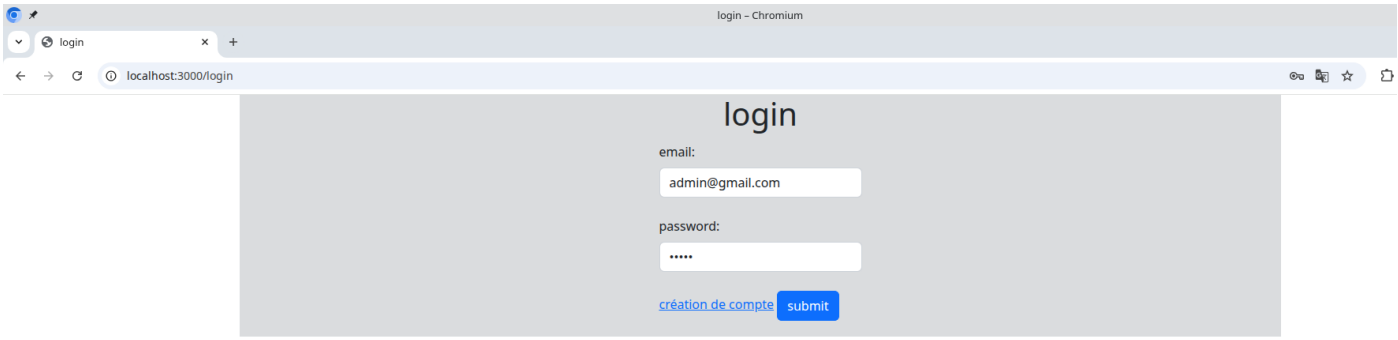
introduction

Le but de ce TP est d'apprendre à créer un site web fonctionnel selon un cahier des charges en utilisant essentiellement les frameworks Bootstrap (frontend), Node.js et express (backend).

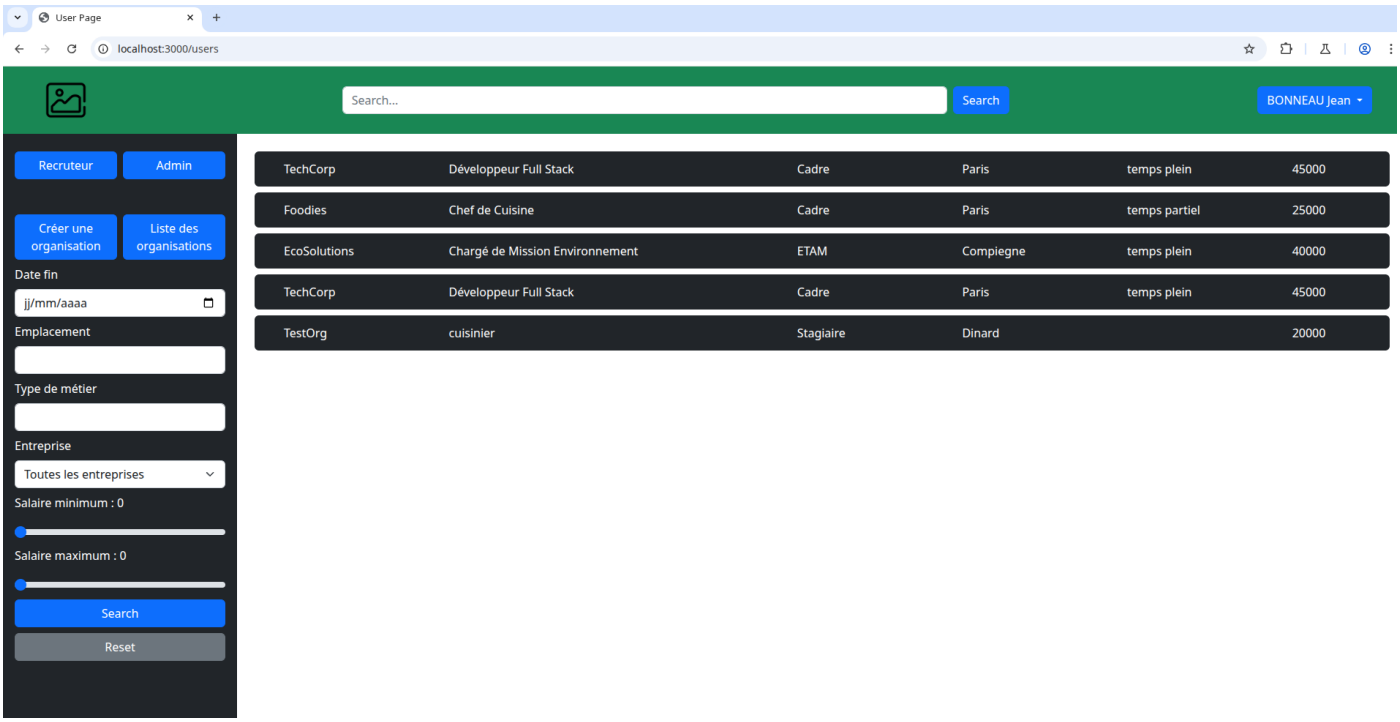
I. présentation rapide

Nous avons choisi de faire une page principale par rôle avec des filtres pour éviter d'avoir trop de pages. Par exemple l'affichage des demandes d'administreur et de la liste de tous les utilisateurs est géré par une unique page administrateur.

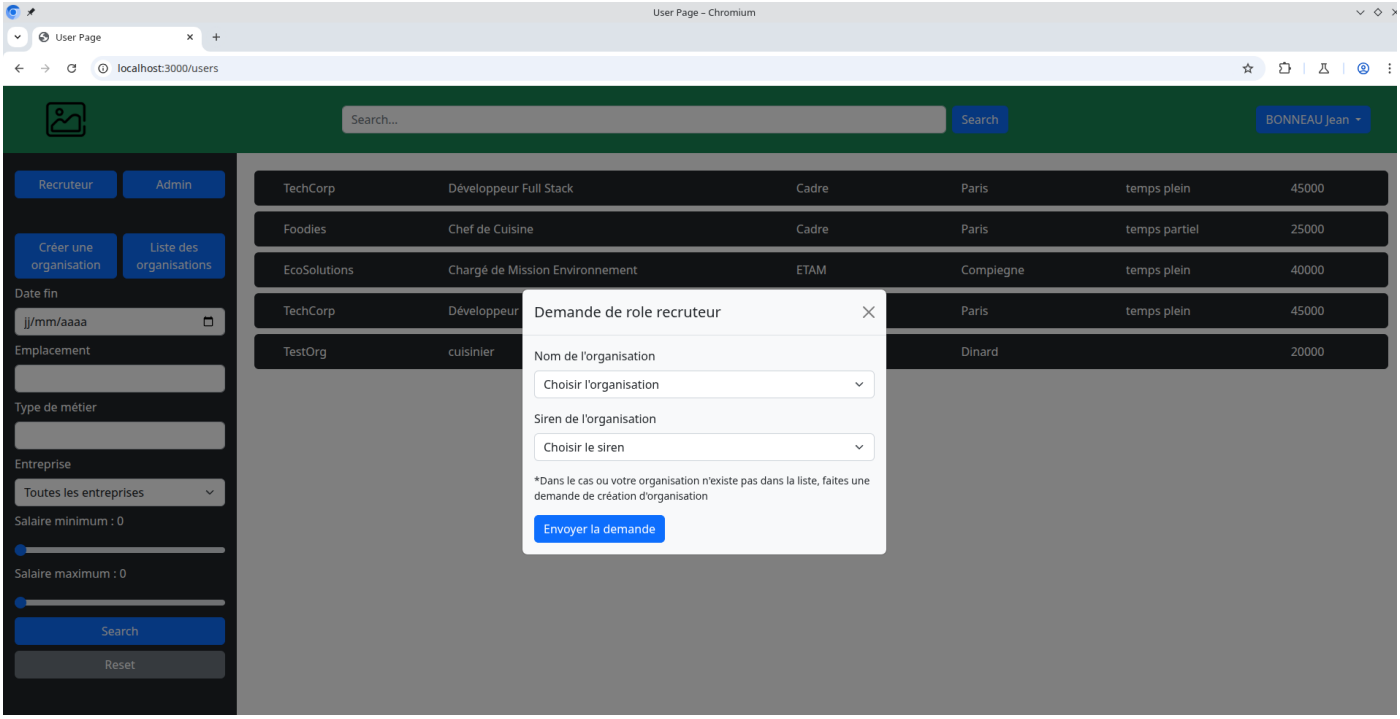
Par défaut on arrive sur la page "login".



1.page utilisateur



- si on est simple utilisateur le bouton "Admin" (respectivement "Recruteur") permet de faire une demande pour devenir administrateur (ou recruteur).



demande droits recruteur (après avoir crée une entreprise)

```
1 • SELECT * FROM sr10p081.UTILISATEUR;
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

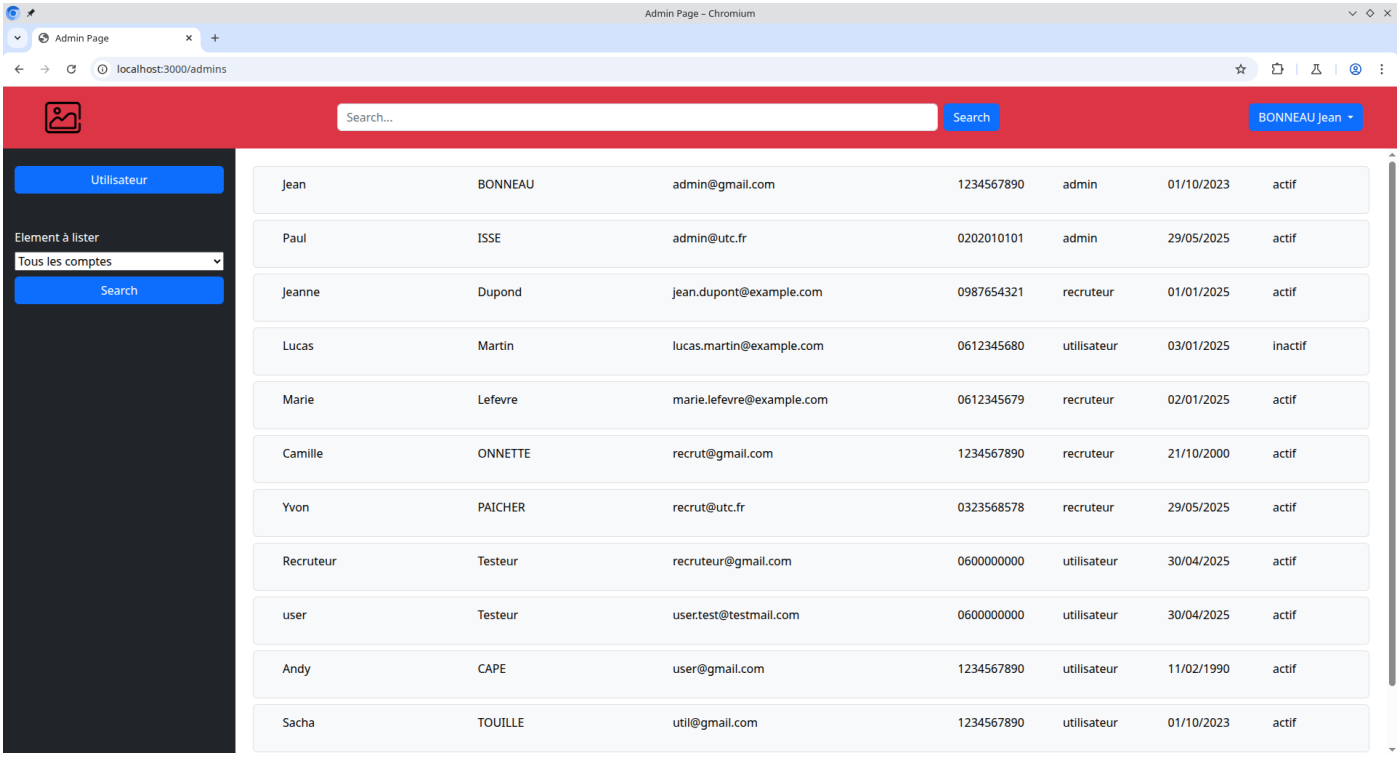
#	email	mdp	nom	prenom	tel	date_creation	statut_compte	organisation	role_utilisateur	demande
1	admin@gmail.com	admin	BONNEAU	Jean	1234567890	2023-10-01	actif	123456123	admin	recruteur

La demande est mémorisée dans la table Utilisateur de la Base de donnée.

- sinon on accède à la page administrateur ou recruteur selon notre rôle.

Par exemple après avoir cliqué sur le bouton "Admin" :

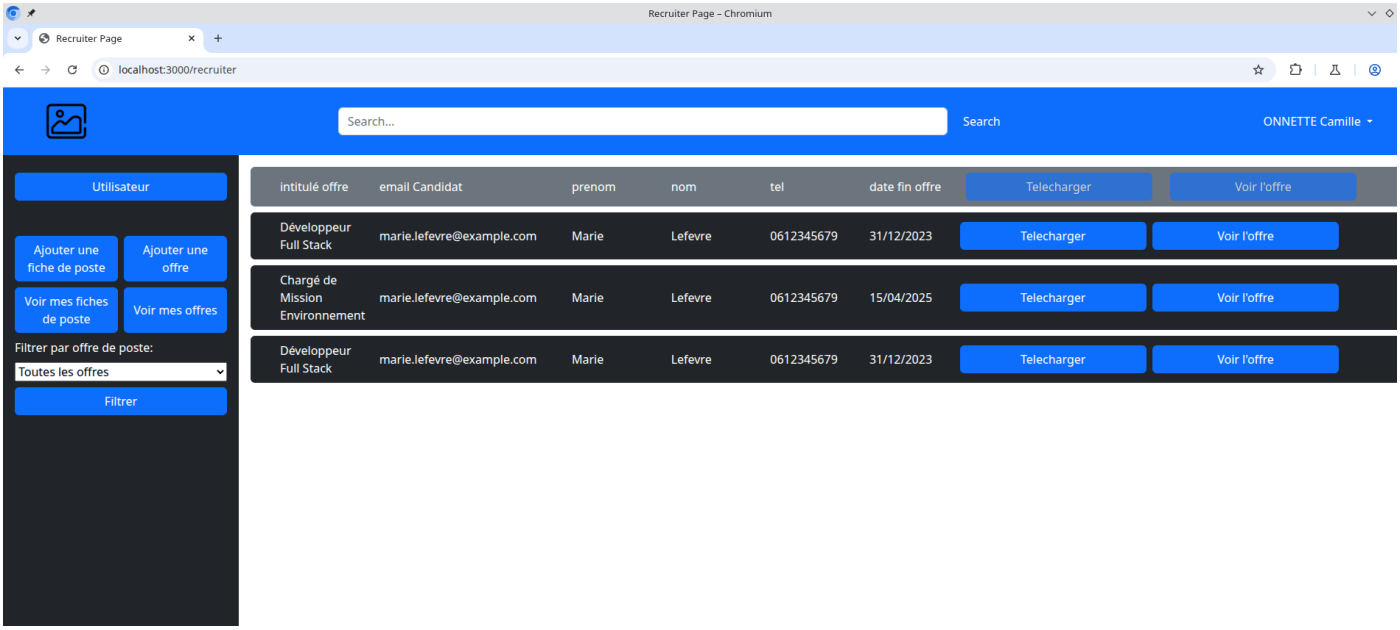
2.page administrateur



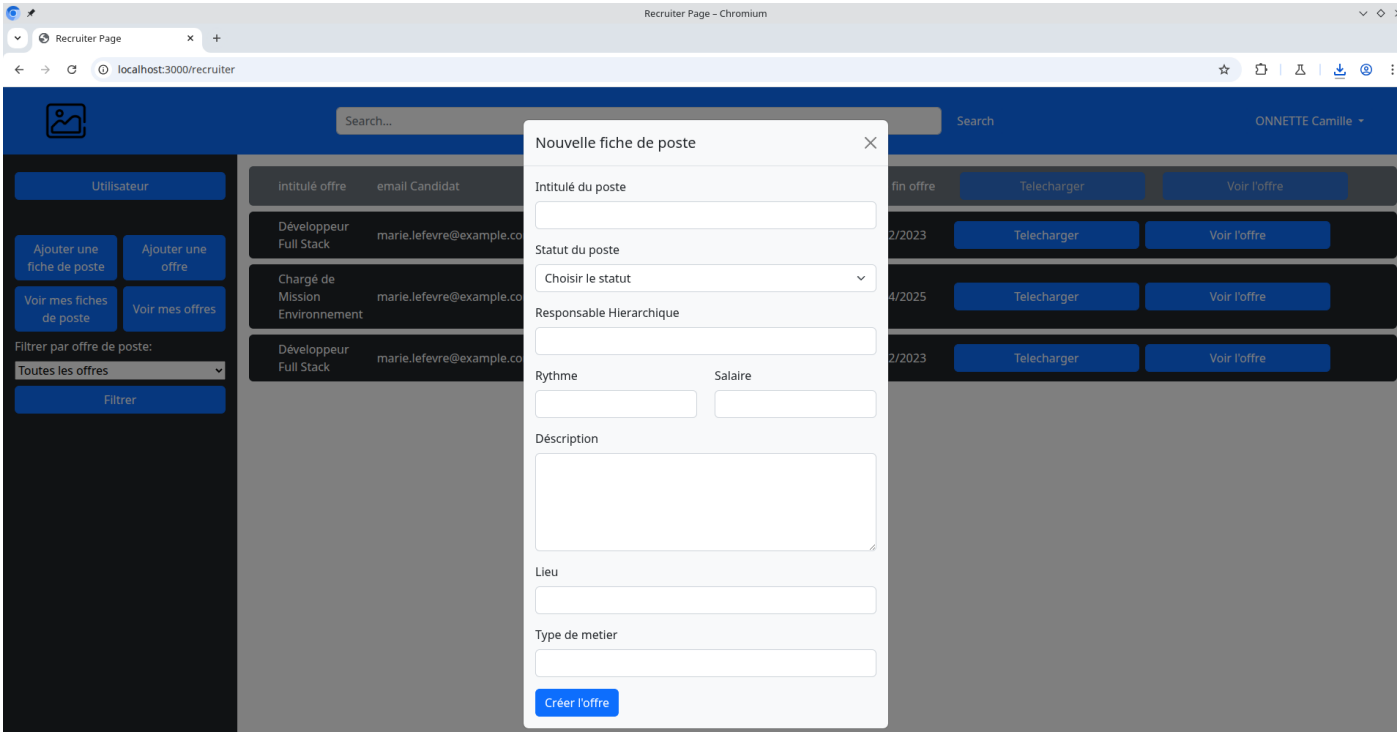
page admin

le bouton "Utilisateur " permet de revenir sur la page utilisateur.

3.page recruteur



page recruteur



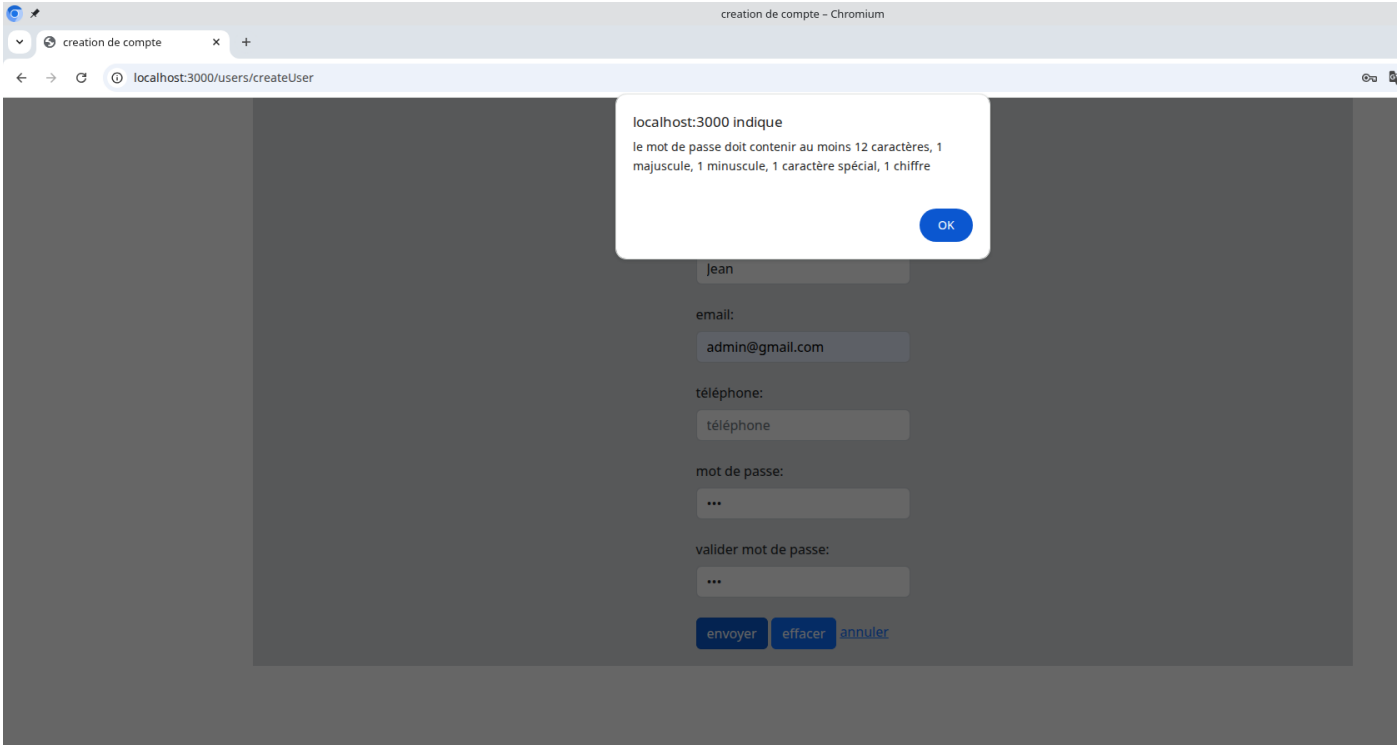
page recruteur création de fiche de poste

II.sécurité

1.mot de passe

recommandations.

Pour éviter les attaques par force brute (avec des listes de mot de passe) il est recommandé (imposé) d'utiliser un mot de passe de longueur minimale 12 caractères et difficile à trouver (caractères spéciaux, etc...).
Il serait aussi utile de bloquer le compte pendant un temps limité après 3 tentatives infructueuses.



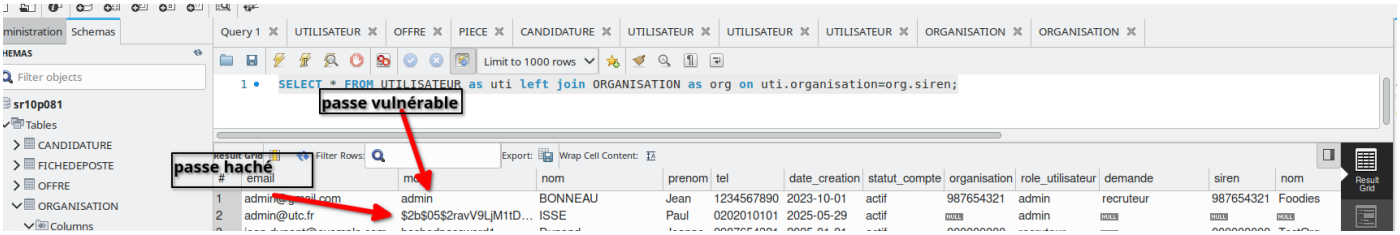
création utilisateur : mot de passe fort

```
JS scriptCreateUser.js M ×
public > javascripts > JS scriptCreateUser.js > verifForm
 7
 8
 9      if (password.value != password_confirm.value) {
10          alert("erreur:le mot de passe et mot de passe confirmé sont différents")
11          return false;
12      }
13
14      if (!(
15          password.value.match(/[0-9]/g) &&
16          password.value.match(/[A-Z]/g) &&
17          password.value.match(/[a-z]/g) &&
18          password.value.match(/[^a-zA-Z\d]/g)
19      )) {
20          alert("le mot de passe doit contenir au moins " + TAILLE_PASSW + " caractères, 1 majuscule, 1 minuscule, 1 caractère
21          return false;
22      }
23
24
25      if (password.value.length < TAILLE_PASSW) {
26          alert("le mot de passe doit contenir au moins " + TAILLE_PASSW + " caractères")
27          return false;
28      }
29
30
31      hashPwd = md5(password.value)
32
33      console.log('passe haché ',hashPwd);
34      password.value=hashPwd;
35
36      console.log('password.value ',password.value);
37
38      // alert("debug bloqué");
39      // return false;
40
41      return true
42  }
43
44
```

création utilisateur : code de vérification de la validité du mot de passe.

hachage.

Au niveau du serveur, le mot de passe est haché avant d'être stocké dans base de données.
Il serait nécessaire en plus d'avoir une connexion sécurisée (https) pour que le mot de passe ne soit pas envoyé en clair (http).



base de donnée utilisateur : mot de passe haché (ou pas).

code :

```
createUser: function (email, nom, prenom, tel, mdp, callback) {
    const saltRounds = 5;
    bcrypt.hash(mdp, saltRounds).then((res) => {
        hashMdp = res;
        createUserSimple(email, nom, prenom, tel, hashMdp, callback);
    }).catch((err) => {
        console.log(err);
    });
},
```

code model/user.js : utilisation de **bcrypt** pour hacher le mot de passe

On utilise la librairie bcrypt lors de la création d'un utilisateur et lors du login. Tout est centralisé dans un seul fichier de type model_js (modèle MVC).
Bcrypt fonctionne aussi si les mots de passe ont été préalablement stockés en clair.

2.protection injection sql.

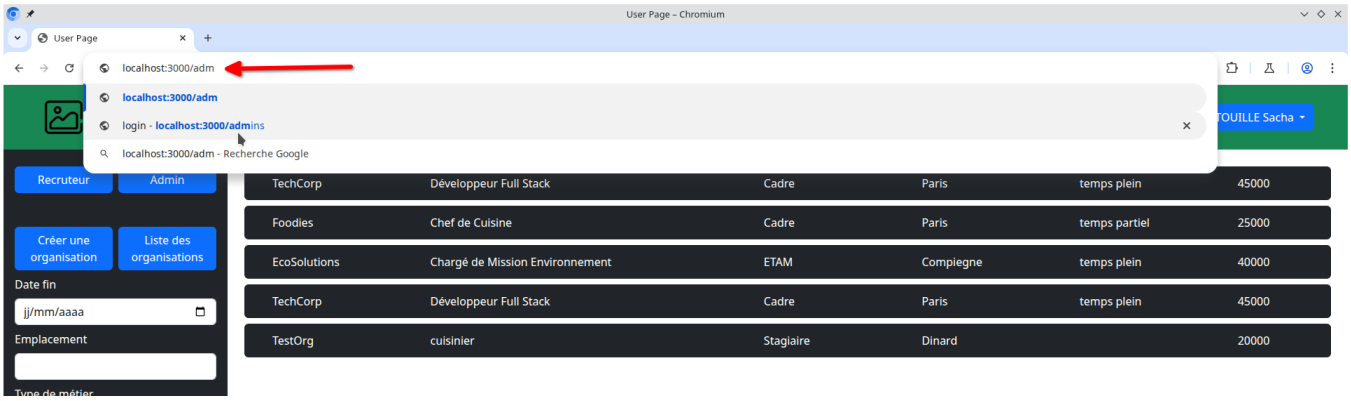
- principe de l'attaque : du code sql est écrit dans des champs d'un formulaire afin qu'il soit exécuté sur le serveur.
- remède : sql préparées ('?' remplacé par des paramètres) .

```
sql = `INSERT INTO UTILISATEUR (
email ,
mdp ,
nom ,
prenom ,
tel ,
date_creation ,
statut_compte ,

role_utilisateur
)
VALUES ( ?,?,?,?,?,?,?,?);`


db.query(sql, [
    email,
    mdp,
    nom,
    prenom,
    tel,
    date_creation,
    statut_compte,
    role_utilisateur

],
```

code model/user.js : utilisation de **requêtes préparées**

3. contrôle des accès.

- En entrant l'url dans la barre d'adresse du navigateur, un pirate pourrait accéder à des ressources ou fonctionnalités reservées à l'administrateur.



tentative d'accès à la page administrateur => ici renvoie à l'accueil (login)

- grace à la bibliothèque session on enregistre le rôle (donc les privilèges) de l'utilisateur connecté et on interdit les accès non autorisés.

```
// check user before app.use (path, router)
app.all("*", function (req, res, next) {
  // console.log("debug : app.all()")
  // console.log(req.session)
  if(req.path.startsWith("/api/")) {
    return next();
  }

  const nonSecurePaths = ["/login", "/signup","/logout","/users/createUser"];
  const adminPaths = ["/admin","/admins"]; //list des urls admin
  const recruterPaths = ["/recruiter"]; //list des urls recruteur
  if (nonSecurePaths.includes(req.path)) {
    console.log("debug non secure path")
    return next();}
  //authenticate user
  if (adminPaths.includes(req.path)) {
    // console.log("debug admin path");
    if (session.isConnected(req.session, "admin")) return next();
    else res.redirect("/login");
      // res
      //   .status(403)
      //   .render("error", { message: " Unauthorized access", error: {} });
  }else if (recruterPaths.includes(req.path)) {
    // console.log("debug recruter path");
    if (session.isConnected(req.session, "recruteur")) return next();
    // else  res.status(403).render("error", { message: " Unauthorized access", er
    else res.redirect("/login");

  }
  else {
    if (session.isConnected(req.session)) return next();
    // not authenticated
    else res.redirect("/login");
  }
});
```

conclusion sécurité.

La sécurité joue un rôle majeur dans la création de site web de nos jours. Il y a 40 ans les premiers sites web n'étaient *quasiment* pas protégés car les attaques étaient peu nombreuses : elles se sont développées véritablement à partir des années 1990.
Les techniques d'attaques sont de plus en plus nombreuses et complexes, un développeur web doit connaître toutes les méthodes pour se protéger, et se former continuellement.

III. conclusion

Ce projet nous a permis de :

- nous former sur la création de site web (HTML,CSS+Javascript) grace à :
    - Bootstrap
    - Node.js + express

qui sont des frameworks très utilisés.

- D'apprendre à gérer les sessions et la sécurité.
- découvrir les API REST
- coté serveur : gérer la base de données.
- Choisir entre un modèle "M.V.C" ou bien "single-page application (SPA) avec *Vue.js*