ENSA KHOURIBGA
ÉCOLE NATIONALE DES SCIENCES APPLIQUÉES

جامعة السلطان مولاي سليمان
Université Sultan Moulay Slimane

ExpressJS

Présenté par:
Inane Salma

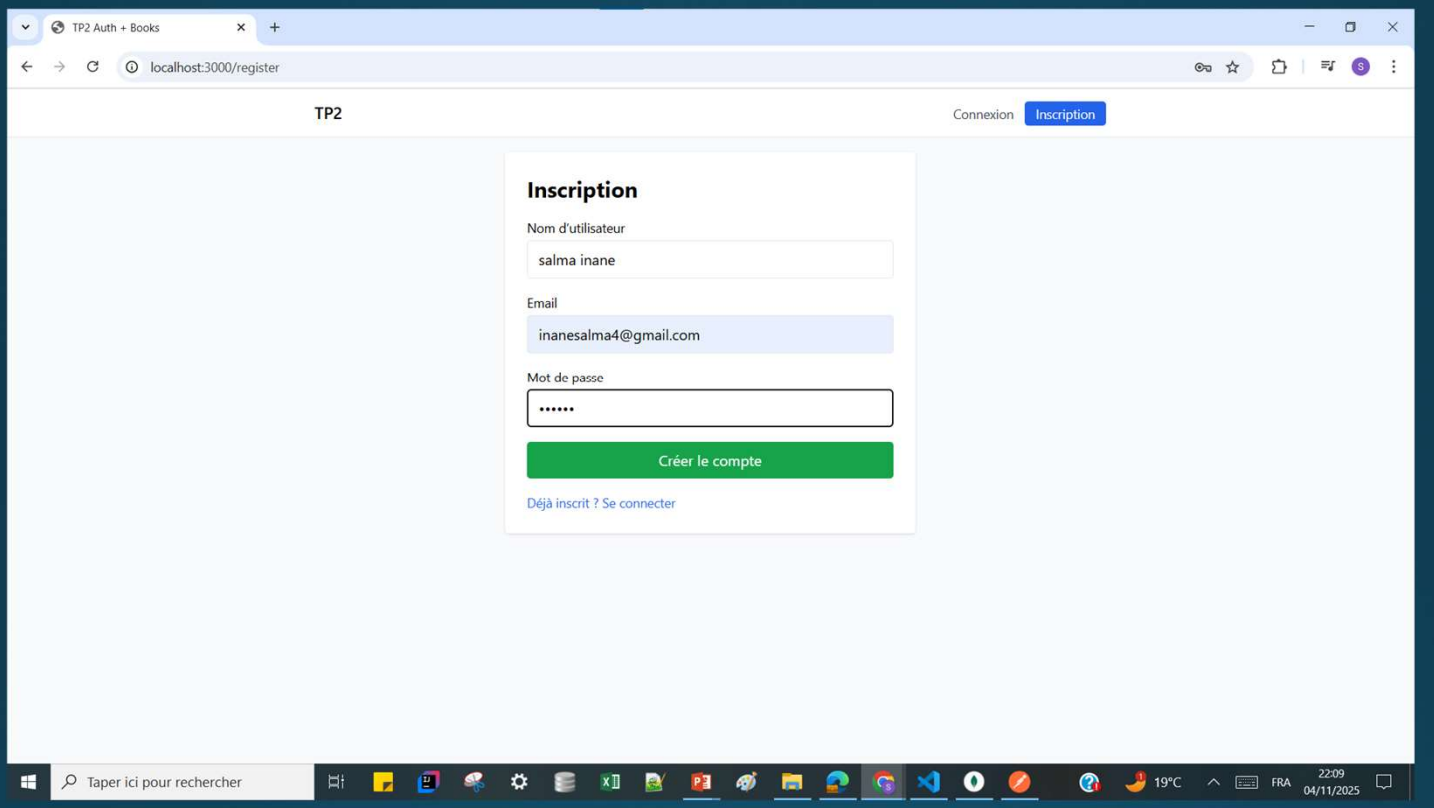
Introduction

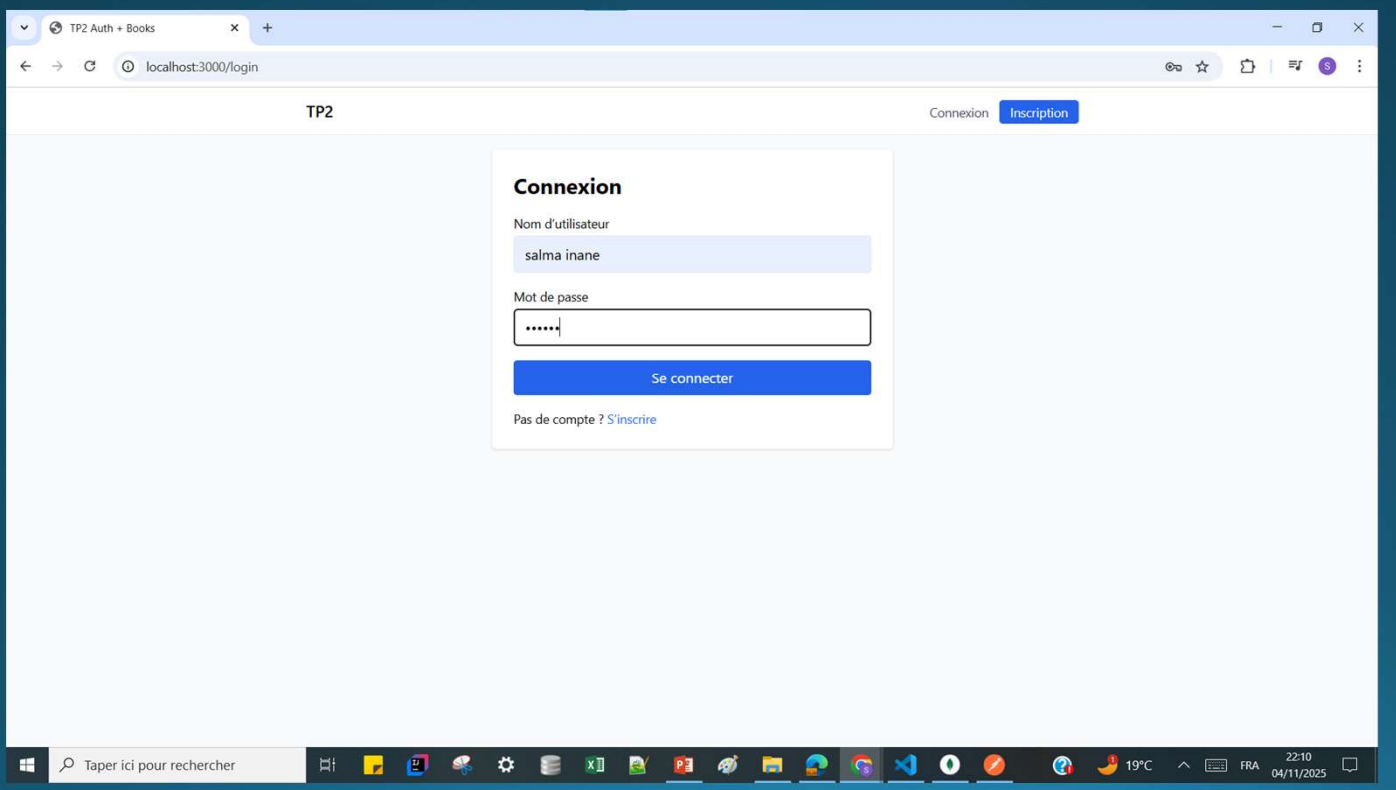
Dans ce TP, j'ai construit une petite application web d'authentification avec Node.js/Express en suivant un fil conducteur très concret : une page d'inscription et de connexion en Pug, des comptes stockés dans MongoDB, et Passport.js pour gérer la stratégie locale et la session. L'idée était de comprendre "le vrai flux" d'une app : depuis le formulaire jusqu'au stockage, puis la vérification du mot de passe et la création de la session. Après connexion, l'utilisateur est redirigé vers la page *Books*, dont les données sont volontairement gardées dans une variable locale pour illustrer la différence entre persistance (utilisateurs en base) et données volatiles (livres).

Cette page n'est accessible qu'aux personnes authentifiées grâce à un middleware simple qui protège la route. Enfin, j'ai utilisé Tailwind CSS pour obtenir une mise en forme propre et responsive sans passer trop de temps sur le design. Mon objectif, en tant qu'étudiante, était d'assembler ces briques, de clarifier le rôle de chacune et de poser des bases solides pour la suite.

- Create a registration and authentication web page using Pug

Les pages d'inscription et de connexion sont réalisées avec Pug dans src/views/register.pug et src/views/login.pug, toutes deux héritant de src/views/layout.pug. Les formulaires form(method="post") collectent username, email (pour l'inscription) et password, puis envoient les données aux routes Express.





```
extends layout

block content
  .mx-auto.max-w-md.bg-white.shadow.rounded.p-6
    h1.text-2xl.font-bold.mb-4 Connexion
    form(action="/login" method="post" class="space-y-4")
      div
        label.block.text-sm.mb-1(for="username") Nom d'utilisateur
        input#username(type="text" name="username" required class="w-full border rounded px-3 py-2")
      div
        label.block.text-sm.mb-1(for="password") Mot de passe
        input#password(type="password" name="password" required class="w-full border rounded px-3 py-2")
      button(type="submit" class="w-full bg-blue-600 text-white py-2 rounded") Se connecter
    p.mt-4.text-sm
      | Pas de compte ?
      a.text-blue-600.ml-1(href="/register") S'inscrire
```

books.js | passport.js | login.pug | register.pug

tp2-auth-books > src > views > register.pug

```
extends layout

block content
  .mx-auto.max-w-md.bg-white.shadow.rounded.p-6
    h1.text-2xl.font-bold.mb-4 Inscription
    if error
      p.text-red-600.mb-3= error
    form(action="/register" method="post" class="space-y-4")
      div
        label.block.text-sm.mb-1(for="username") Nom d'utilisateur
        input#username(type="text" name="username" required class="w-full border rounded px-3 py-2")
      div
        label.block.text-sm.mb-1(for="email") Email
        input#email(type="email" name="email" required class="w-full border rounded px-3 py-2")
      div
        label.block.text-sm.mb-1(for="password") Mot de passe
        input#password(type="password" name="password" required class="w-full border rounded px-3 py-2")
      button(type="submit" class="w-full bg-green-600 text-white py-2 rounded") Créer le compte
    p.mt-4.text-sm
      a.text-blue-600(href="/login") Déjà inscrit ? Se connecter

```

- The user's information should be stored in mongoDB

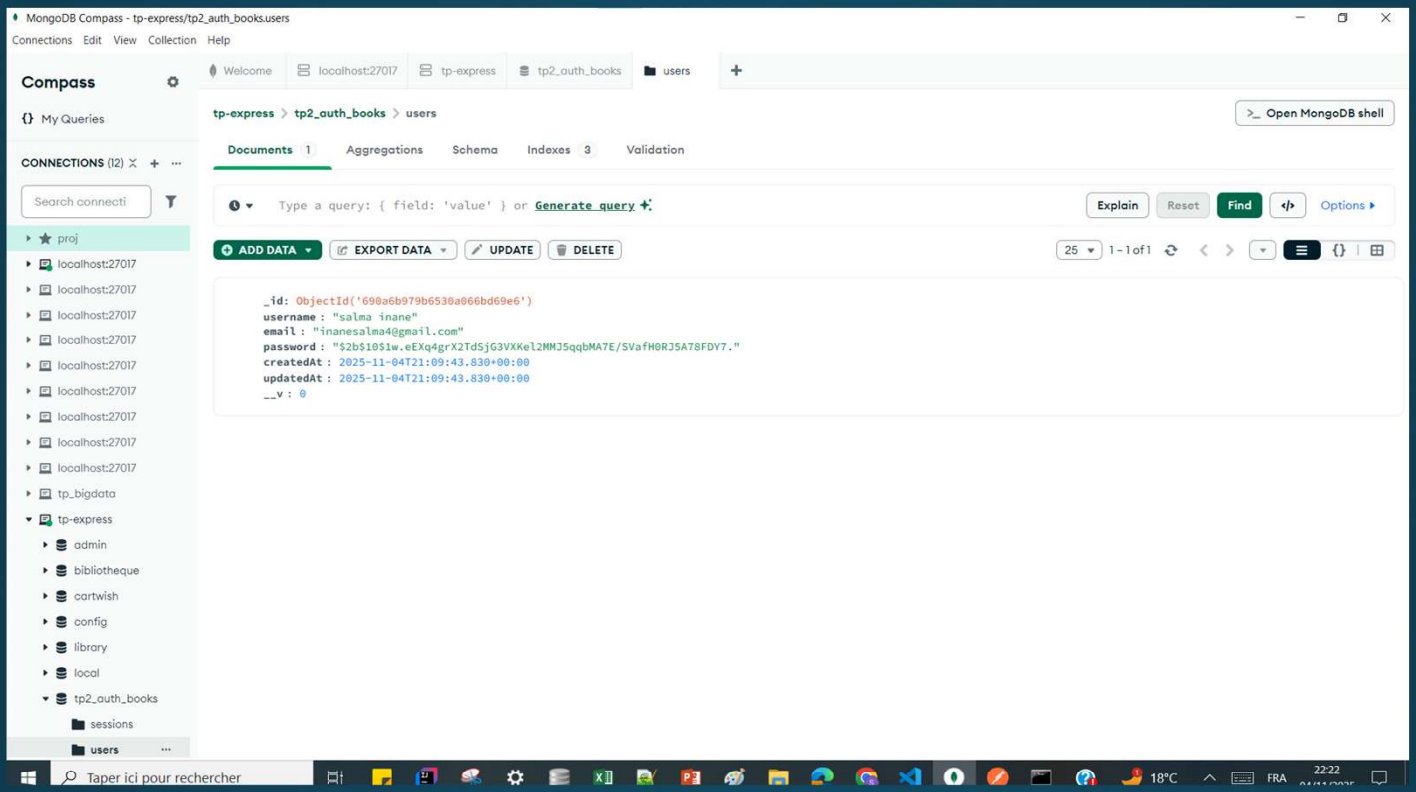
Le modèle Mongoose src/models/User.js définit le schéma User et hache le mot de passe avec bcrypt dans un hook pre("save").

```
const UserSchema = new mongoose.Schema({
  username: { type: String, required: true, unique: true, trim: true },
  email:    { type: String, required: true, unique: true, lowercase: true, trim: true },
  password: { type: String, required: true }
}, { timestamps: true });

UserSchema.pre("save", async function(next) {
  if (!this.isModified("password")) return next();
  const salt = await bcrypt.genSalt(10);
  this.password = await bcrypt.hash(this.password, salt);
  next();
});

UserSchema.methods.comparePassword = function(candidate) {
  return bcrypt.compare(candidate, this.password);
};

module.exports = mongoose.model("User", UserSchema);
```



- Use passport.js for handling authentication

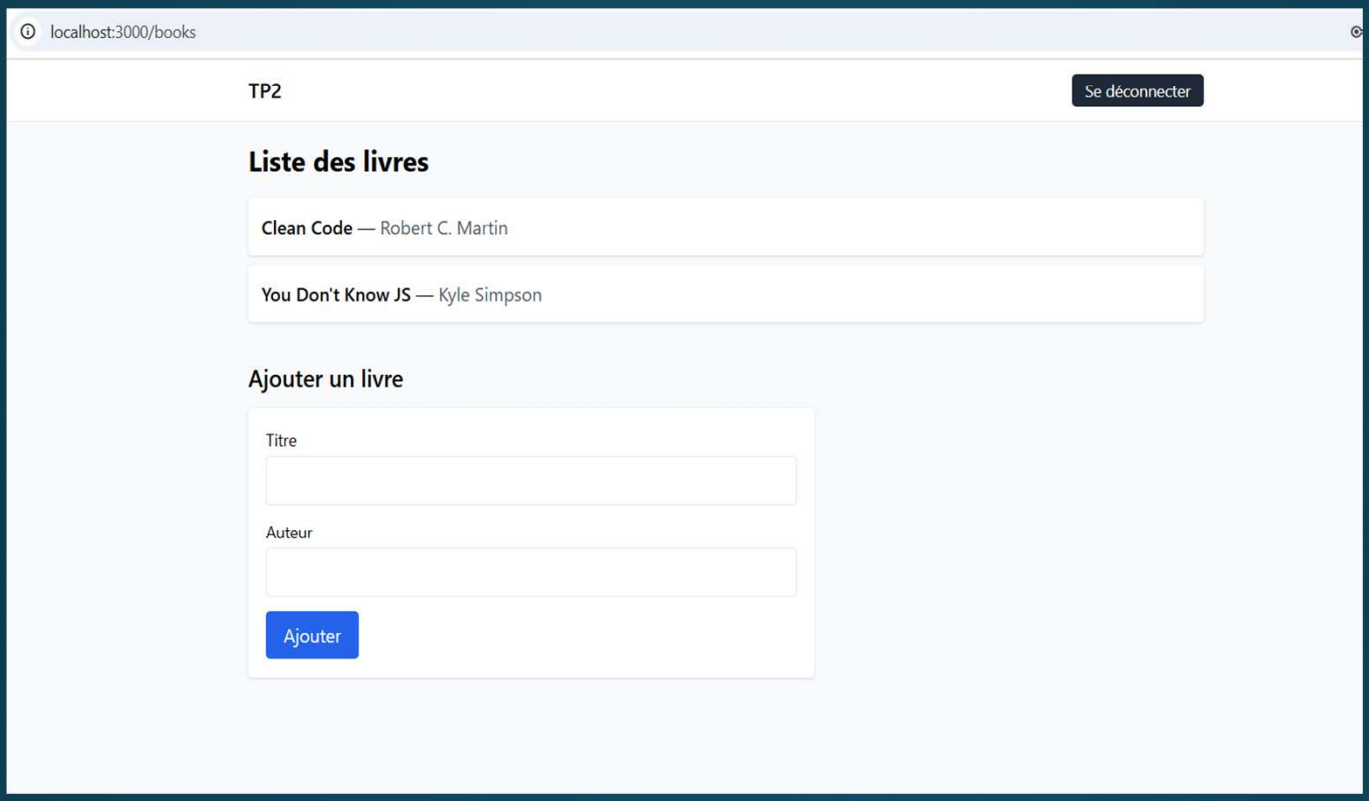
Dans passport.js, on configure Passport pour gérer l'authentification via une **LocalStrategy** : lors d'un POST /login, Passport récupère username et password, cherche l'utilisateur correspondant en MongoDB (User.findOne({ username })) puis vérifie le mot de passe en comparant la valeur saisie avec le **hash** stocké (user.comparePassword, basé sur bcrypt). Si l'utilisateur n'existe pas ou si le mot de passe est incorrect, la stratégie renvoie done(null, false) avec un message d'erreur ; sinon, elle renvoie done(null, user) pour signaler le succès. Après une connexion réussie, serializeUser indique à Passport **quoi stocker en session**

```
const LocalStrategy = require("passport-local").Strategy;
const User = require("./models/User");      Pin selection to current chat prompt (Ctrl

module.exports = function(passport) {
  passport.use(new LocalStrategy(async (username, password, done) => {
    try {
      const user = await User.findOne({ username });
      if (!user) return done(null, false, { message: "Utilisateur inconnu" });
      const ok = await user.comparePassword(password);
      if (!ok) return done(null, false, { message: "Mot de passe invalide" });
      return done(null, user);
    } catch (err) {
      return done(err);
    }
  }));

  passport.serializeUser((user, done) => done(null, user.id));
  passport.deserializeUser(async (id, done) => {
    try {
      const user = await User.findById(id).lean();
      done(null, user);
    } catch (err) {
      done(err);
    }
  });
```

- After authenticating successfully redirect to the books page

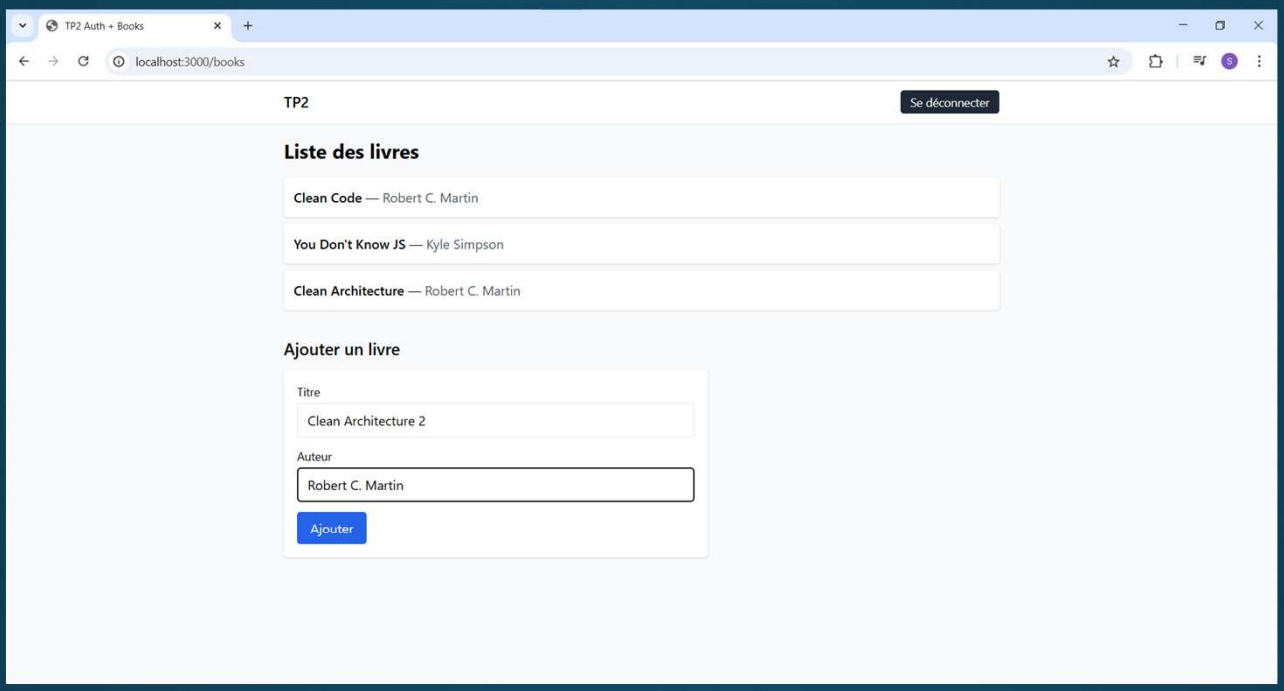


La redirection après authentification est gérée dans la route POST de connexion (`src/routes/auth.js`) via Passport : `passport.authenticate("local", { successRedirect: "/books", failureRedirect: "/login" })`.

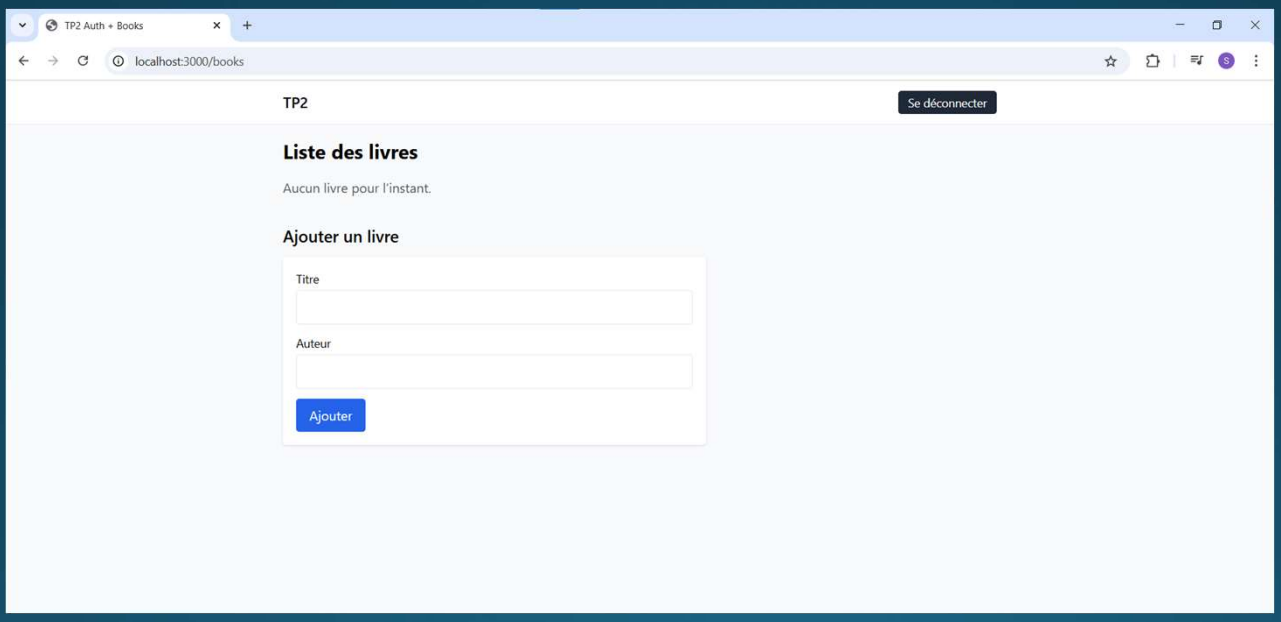
Quand la stratégie locale valide le couple username/mot de passe, Passport appelle `done(null, user)` puis envoie automatiquement une redirection HTTP 302 vers `/books`. En cas d'échec, l'utilisateur est renvoyé vers `/login`.

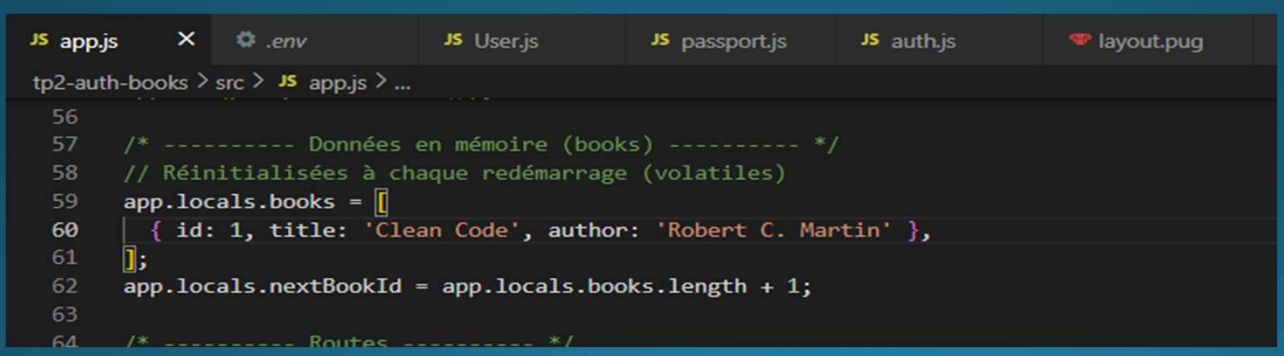
- The books are stored in a local variable

Les livres ne sont pas en base mais en mémoire, dans le module des routes des livres (`src/routes/books.js`). On y déclare un tableau local, par ex. `const books = [ ... ]`, utilisé pour l'affichage et l'ajout. Comme cette structure est volatile, toute donnée ajoutée disparaît au redémarrage du serveur (comportement attendu pour un stockage local/éphémère).



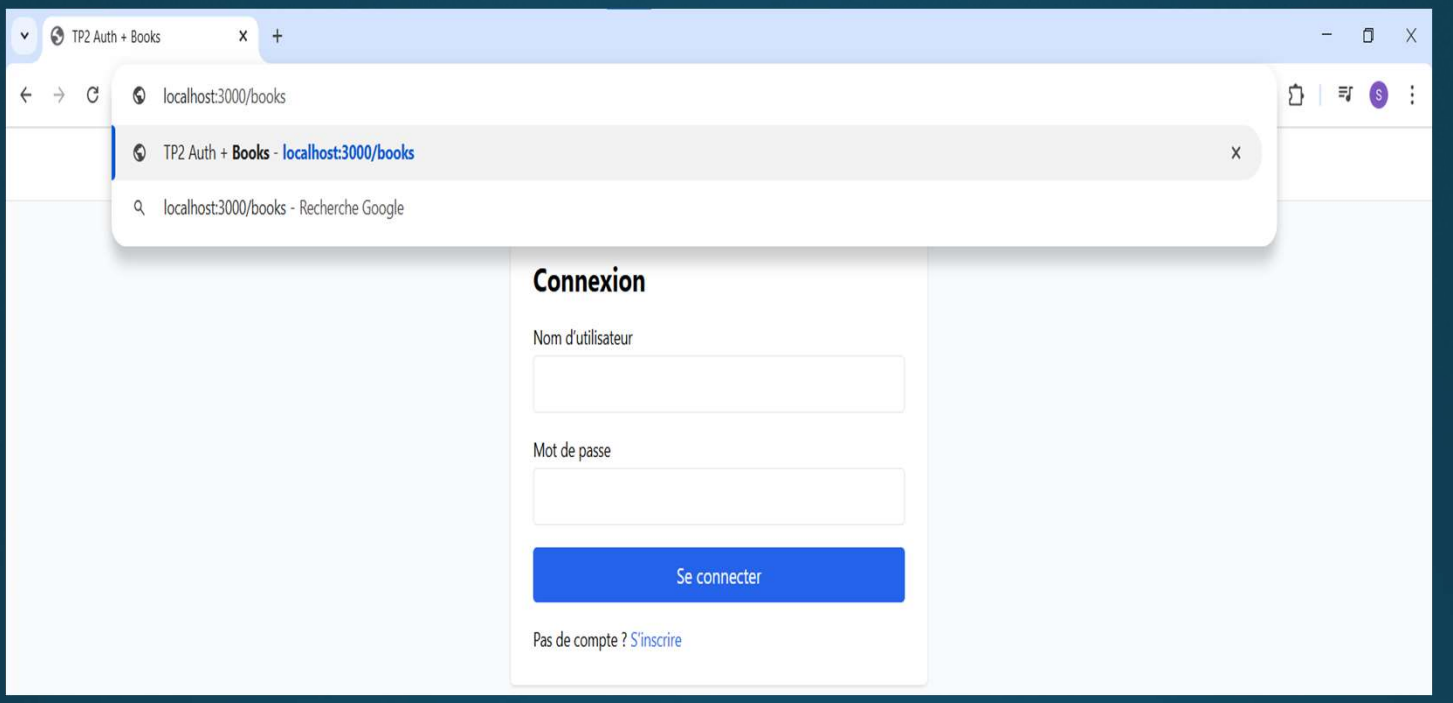
Après le redémarrage les livres ont disparu

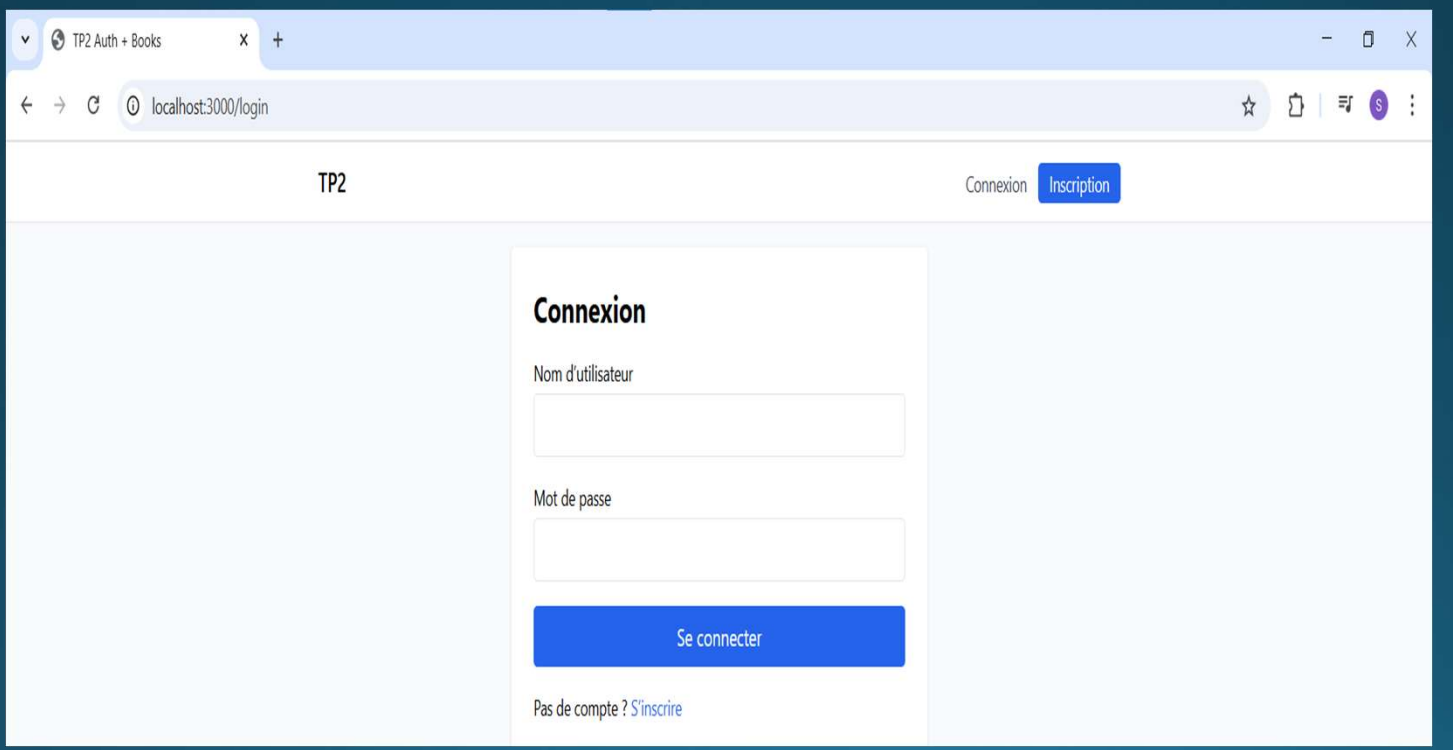




The books page should not be accessible unless the user is authenticated

La protection d'accès repose sur un middleware `ensureAuth` défini dans `src/routes/books.js` qui vérifie `req.isAuthenticated()`. Si l'utilisateur n'est pas connecté, il est redirigé vers `/login` (`res.redirect("/login")`). Les routes `/books` (GET pour l'affichage et POST pour l'ajout) utilisent ce middleware, garantissant que la page et ses actions ne sont disponibles que pour une session authentifiée.





- Use Tailwind CSS to handle the styling part of the application

Le style est assuré par Tailwind CSS injecté dans la mise en page Pug (`src/views/layout.pug`) via le CDN : `script(src="https://cdn.tailwindcss.com")`. Les vues (`login.pug`, `register.pug`, `books.pug`) appliquent ensuite les classes utilitaires Tailwind (ex. `bg-white`, `rounded`, `shadow`, `px-4`, `text-blue-600`) pour la mise en forme. Cette intégration via CDN évite toute étape de build et permet un design cohérent et réactif sur l'ensemble des pages.

```
doctype html
html(lang="fr")
  head
    meta(charset="utf-8")
    meta(name="viewport" content="width=device-width, initial-scale=1")
    title= title || 'Auth + Books'
    script(src="https://cdn.tailwindcss.com")
```

Conclusion

Au final, l'application fonctionne comme prévu : inscription et connexion sécurisées, session active, redirection vers *Books* et accès restreint aux seules utilisatrices et aux seuls utilisateurs connectés, avec une interface soignée grâce à Tailwind. Ce TP m'a aidée à démystifier l'enchaînement Express → Pug → MongoDB → Passport, à mieux organiser les routes et les vues, et à adopter de bons réflexes (hachage des mots de passe, middlewares, gestion basique des erreurs).