Rapport de TP EXPRESS JS

IID3 2025

Realise par :

- Abdelmoiz kadouaoui

Encadre par :

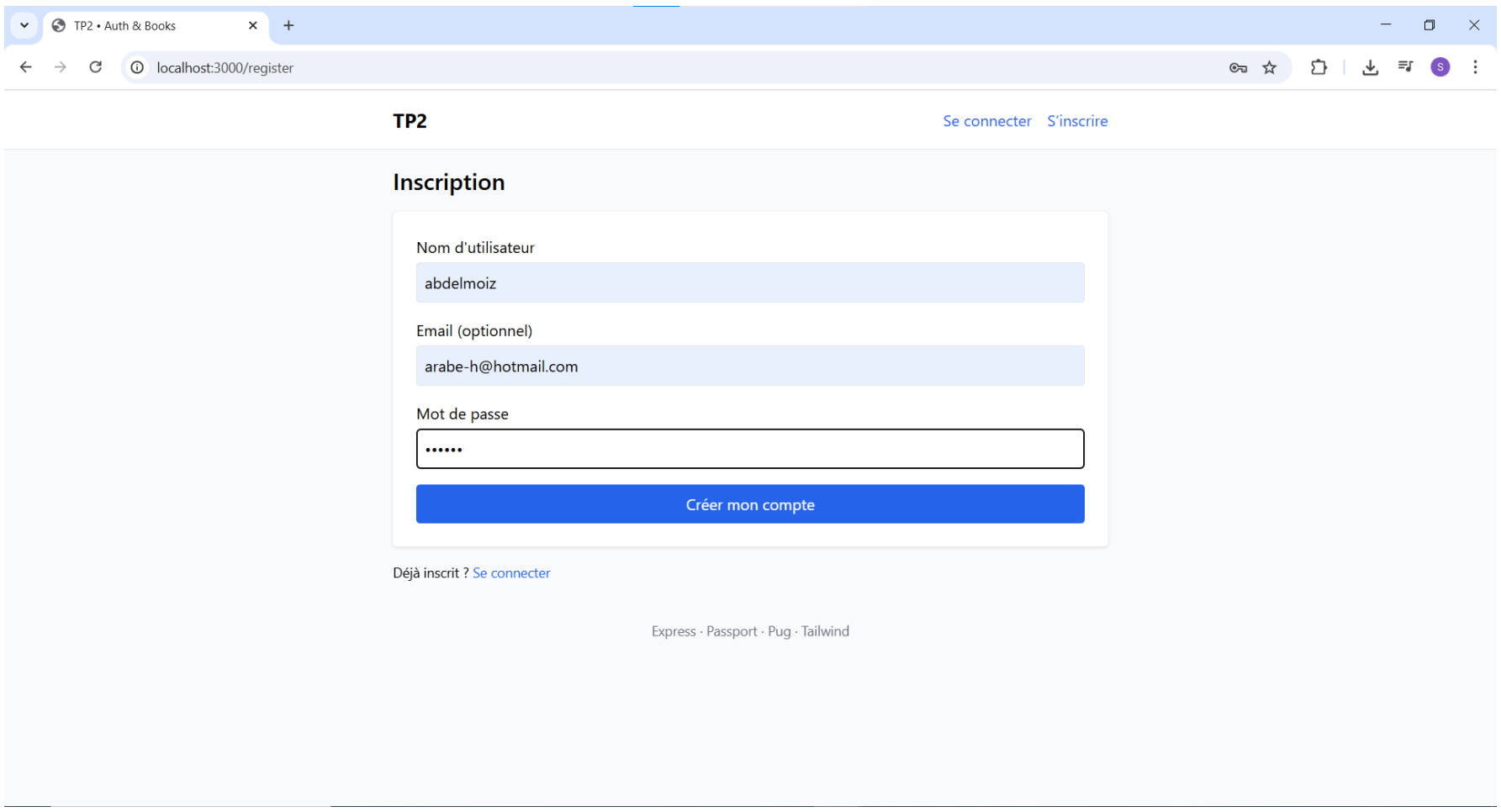
- Pr. Amal ourdou

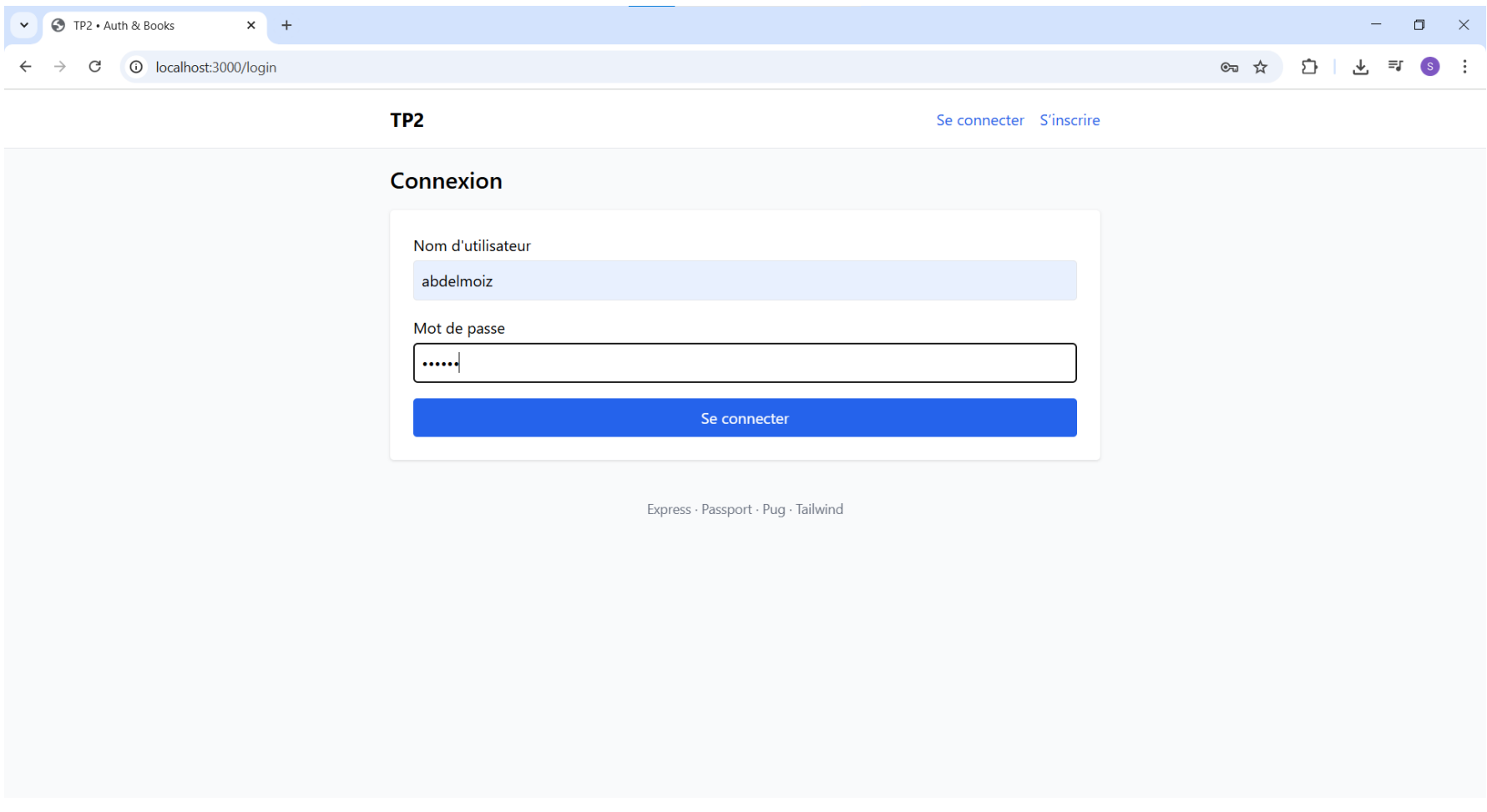
Introduction :

Ce travail a pour objectif de concevoir une petite application web complète illustrant un parcours d'authentification moderne avec la pile Node.js/Express. L'interface est réalisée en **Pug**, la persistance des comptes dans **MongoDB** (via Mongoose), et l'authentification gérée par **Passport.js** (stratégie locale, sessions). Après connexion réussie, l'utilisateur est redirigé vers la page **/books**, dont les données sont conservées en mémoire locale, et l'accès demeure strictement réservé aux utilisateurs authentifiés. La présentation visuelle s'appuie sur **Tailwind CSS** afin de structurer rapidement des formulaires et des listes propres, sans feuille de style dédiée. Ce TP met ainsi en pratique la séparation des responsabilités (vues, routes, middleware), la gestion sécurisée des mots de passe, et le contrôle d'accès par session, conformément au cahier des charges récapitulé dans le rapport.

Create a registration and authentication web page using Pug

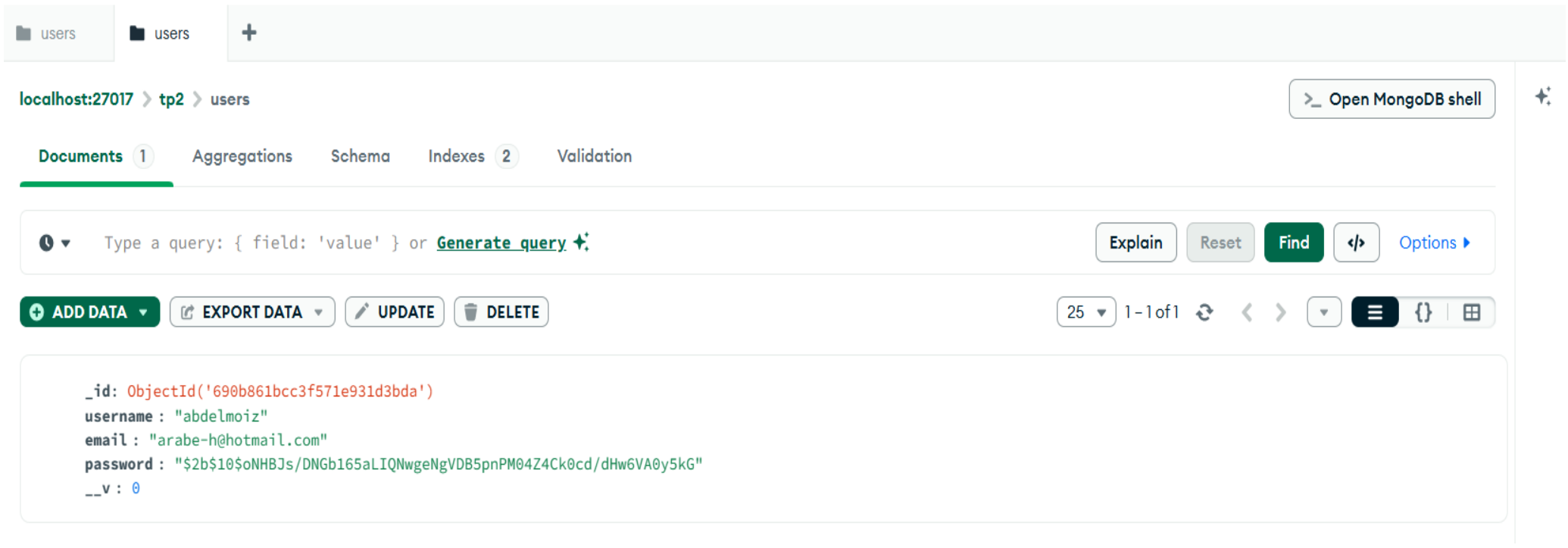
Les pages **Pug** sont src/views/register.pug (inscription) et src/views/login.pug (connexion), toutes deux héritant de src/views/layout.pug. Elles affichent des formulaires HTML simples qui envoient leurs données vers /register (POST) et /login (POST).





The user's information should be stored in mongoDB

La persistance se fait via Mongoose : src/models/User.js définit le schéma (username, email, password). La connexion à MongoDB est établie dans src/server.js avec mongoose.connect(process.env.MONGO_URL).



Use passport.js for handling authentication

L'authentification est assurée par Passport.js avec la LocalStrategy déclarée dans src/config/passport.js. Au login, on récupère l'utilisateur par username, puis on vérifie le mot de passe avec bcrypt.compare.

Passport sérialise l'ID utilisateur en session (serializeUser) et désérialise l'objet utilisateur à chaque requête (deserializeUser).

Dans src/server.js, passport.initialize() et passport.session() branchent Passport sur la session Express (configurée via express-session).

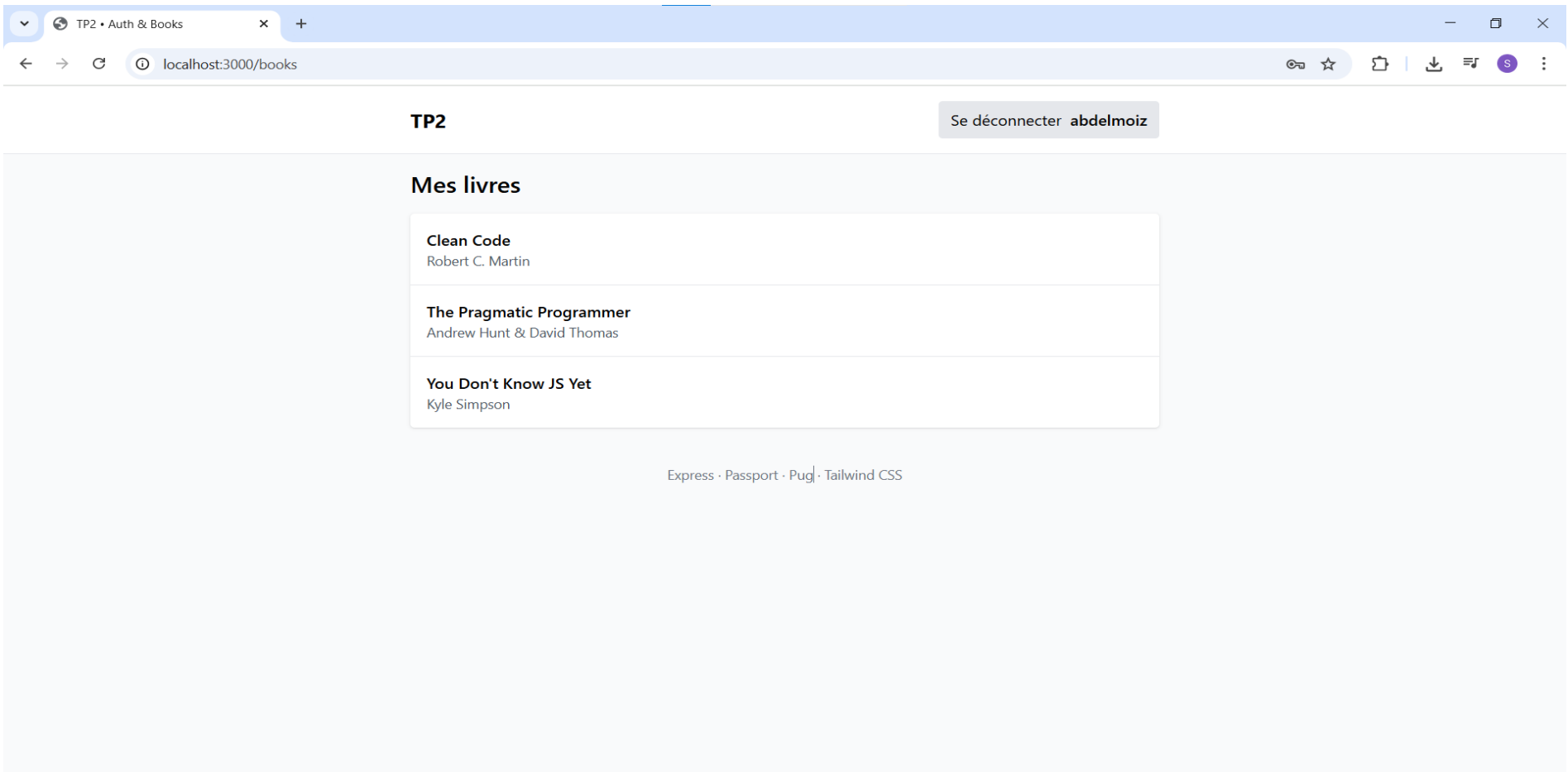
Résultat : si les identifiants sont valides, Passport marque la requête comme authentifiée pour l'accès aux routes protégées.

```js
const LocalStrategy = require('passport-local').Strategy;
const bcrypt = require('bcrypt');
const User = require('../models/User'); // <-- un seul ..

module.exports = function initPassport(passport) {
  passport.use(
    new LocalStrategy(async (username, password, done) => {
      try {
        const user = await User.findOne({ username });
        if (!user) return done(null, false);
        const ok = await bcrypt.compare(password, user.password);
        if (!ok) return done(null, false);
        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);
      done(null, user);
    } catch (err) {
      done(err);
    }
  });
};
```

After authenticating successfully redirect to the books page



Dans src/routes/auth.js, la route POST /login utilise passport.authenticate('local', { successRedirect: '/books', failureRedirect: '/login?error=1' }). En cas de succès, l'utilisateur est redirigé automatiquement vers /books, ce qui répond exactement au critère.

The books are stored in a local variable

server.js

src > server.js > ...

```
42
43  // variable globale: livres en mémoire
44  app.locals.books = [
45    { id: 1, title: 'Clean Code', author: 'Robert C. Martin' },
46    { id: 2, title: 'The Pragmatic Programmer', author: 'Andrew Hunt & David Thomas' },
47    { id: 3, title: "You Don't Know JS Yet", author: 'Kyle Simpson' },
48  ];
49
```

La liste des livres est conservée en mémoire dans src/server.js via app.locals.books = [...]. Ce « local » d'application est accessible partout ; la route /books la lit et la passe à la vue books.pug pour l'affichage.

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

La protection d'accès repose sur le middleware ensureAuth (src/middleware/auth.js) qui vérifie req.isAuthenticated(). La route /books applique ce garde-fou : router.get('/', ensureAuth, ...). Si l'utilisateur n'est pas connecté, il est redirigé vers /login ; sinon, la page s'affiche. Cela garantit que /books reste strictement réservé aux sessions authentifiées.

```
const express = require('express');
const bcrypt = require('bcrypt');
const passport = require('passport');
const User = require('../models/User');
const router = express.Router();
router.get('/register', (req, res) => res.render('register'));
router.post('/register', async (req, res) => {
  const { username, email, password } = req.body;
  try {
    if (!username || !password) {
      return res.status(400).render('register', { error: 'Username et mot de passe obligatoires.' });
    }
    const exists = await User.findOne({ username });
    if (exists) {
      return res.render('register', { error: "Ce nom d'utilisateur est déjà pris." });
    }
    const hash = await bcrypt.hash(password, 10);
    await User.create({ username, email, password: hash });
    return res.redirect('/login?registered=1');
  } catch (e) {
    console.error(e);
    return res.render('register', { error: 'Erreur serveur.' });
  }
});
```

```
router.post(
  '/login',
  passport.authenticate('local', {
    successRedirect: '/books',
    failureRedirect: '/login?error=1',
  })
);

// --- Logout ---
router.post('/logout', (req, res, next) => {
  req.logout((err) => {
    if (err) return next(err);
    res.redirect('/login');
  });
});

module.exports = router;
```

Use Tailwind CSS to handle the styling part of the application

Le style est géré par Tailwind CSS inclus dans le layout : script(src='https://cdn.tailwindcss.com') dans src/views/layout.pug. Toutes les vues utilisent ensuite des classes utilitaires Tailwind (bg-white, rounded, shadow, text-blue-600, etc.) pour mettre en forme formulaires, boutons.

layout.pug

src > views > layout.pug

```
doctype html
html(lang='fr')
  head
    meta(charset='utf-8')
    meta(name='viewport', content='width=device-width, initial-scale=1')
    title TP2 • Auth & Books
    // Tailwind CDN
    script(src='https://cdn.tailwindcss.com')
  body(class='min-h-screen bg-gray-50')
    header(class='bg-white border-b')
      nav(class='max-w-3xl mx-auto flex items-center justify-between p-4')
        a(href='/' class='text-xl font-bold') TP2
        if currentUser
          form(action='/logout', method='POST')
            button(type='submit' class='px-3 py-2 rounded bg-gray-200 hover:bg-gray-300')
              | Se déconnecter
              span(class='font-semibold ml-1')= currentUser.username
        else
          div
            a(href='/login' class='mr-4 text-blue-600 hover:underline') Se connecter
            a(href='/register' class='text-blue-600 hover:underline') S’inscrire
    main(class='max-w-3xl mx-auto p-4')
      block content
    footer(class='text-center text-sm text-gray-500 p-6')
      | Express · Passport · Pug · Tailwind CSS

```

Conclusion :

L’application livrée répond à l’ensemble des exigences : inscription/connexion en Pug, stockage des utilisateurs dans MongoDB, authentification par Passport.js avec redirection vers /books, protection de la route par middleware, et mise en forme avec Tailwind. Au-delà de la conformité fonctionnelle, ce TP consolide des compétences clés en JS côté serveur, en templating, et en sécurisation par sessions.

Fin