Université Sultan Moulay Slimane
École Nationale des Sciences Appliquées – Khouribga

Filière : Informatique et Ingénierie des Données

Rapport du TP2 :

Développement d'une application d'inscription et d'authentification avec Express

Réalisé par :
Ezaakouni Selma

Année universitaire : **2025/2026**

Introduction

Ce TP avait pour objectif de mettre en pratique plusieurs technologies du développement web côté serveur :

— **Express.js** pour créer un serveur HTTP Node.js modulaire,

— **MongoDB** pour stocker les utilisateurs,

— **Passport.js** pour gérer l'authentification,

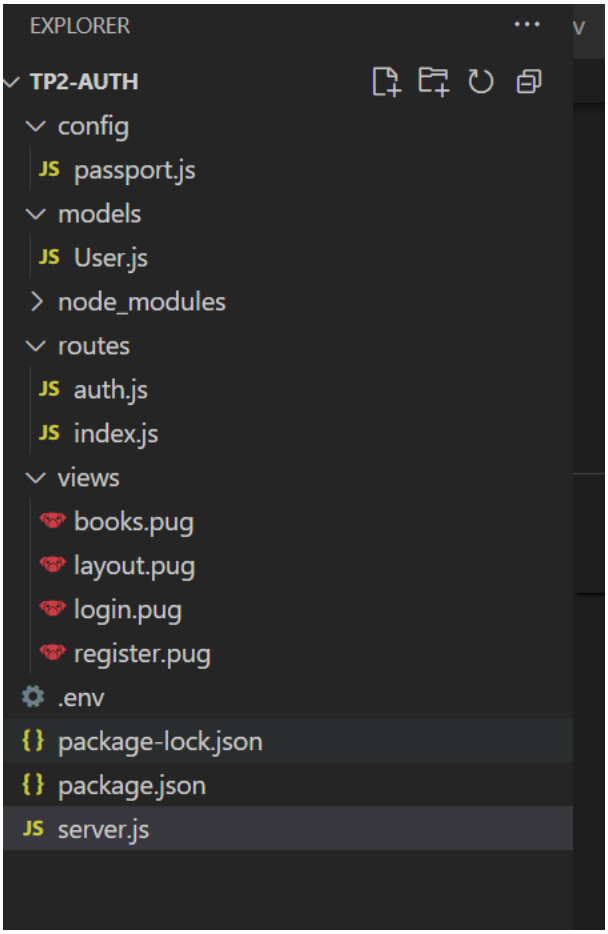
— **Pug** comme moteur de template,

— **Tailwind CSS** pour le style des pages.

L'application finale permet à un utilisateur de s'inscrire, de se connecter, et d'accéder à une page protégée affichant une liste de livres. Cette page n'est accessible qu'après authentification réussie.

Structure du projet

La structure générale du projet est la suivante :



Chaque dossier joue un rôle précis :

— `config/` contient la configuration de Passport.js ;

— `models/` définit le modèle Mongoose des utilisateurs ;

— `routes/` gère les routes de navigation ;

— `views/` contient les templates Pug pour l'interface ;

— `server.js` initialise le serveur et assemble tous les modules.

Connexion à la base de données MongoDB

Le fichier `server.js` établit la connexion à MongoDB à l'aide de Mongoose. L'URI est définie dans un fichier `.env` :

Dans le code, la connexion se fait ainsi :

```
15  console.log('BOOT: connecting MongoDB →', process.env.MONGO_URI);
16  mongoose.connect(process.env.MONGO_URI)
17    .then(() => console.log(' MongoDB connecté'))
18    .catch(err => console.error(' MongoDB error:', err));
19
```

Cette étape permet de garantir que toutes les opérations d'inscription et d'authentification auront accès à la base.

Modèle utilisateur : `User.js`

Le modèle définit la structure d'un utilisateur dans MongoDB. Il contient deux champs :

— `email` : identifiant unique ;

— `password` : mot de passe haché.

```
const UserSchema = new mongoose.Schema({
  email: {
    type: String,
    required: [true, 'L'email est requis.'],
    unique: true, // un seul compte par email
    lowercase: true,
    trim: true
  },
  password: {
    type: String,
    required: [true, 'Le mot de passe est requis.'],
    minlength: [6, 'Le mot de passe doit contenir au moins 6 caractères.']
  }
}, { timestamps: true });
```

Avant chaque sauvegarde, un hook Mongoose (`pre('save')`) hache automatiquement le mot de passe avec **bcrypt**. Cela garantit qu'aucun mot de passe en clair n'est stocké.

Configuration de Passport.js

Le module `Passport.js` facilite l'authentification. Nous avons utilisé la stratégie **locale**, basée sur l'email et le mot de passe. Passport sérialise ensuite l'utilisateur dans la session, afin de le reconnaître sur chaque requête suivante.

```
config > passport.js > <unknown> > exports
module.exports = function (passport) {
  passport.use(new LocalStrategy(
    { usernameField: 'email' },
    async (email, password, done) => {
      try {
        // Normalisation simple
        const mail = (email ?? '').toString().trim().toLowerCase();
        const pwd  = (password ?? '').toString();

        if (!mail || !pwd) {
          return done(null, false, { message: 'Email et mot de passe requis.' });
        }

        const user = await User.findOne({ email: mail });
        if (!user) return done(null, false, { message: 'Email inconnu.' });

        const ok = await user.matchPassword(pwd);
        if (!ok) return done(null, false, { message: 'Mot de passe invalide.' });

        return done(null, user);
      } catch (err) {
```

Le serveur principal : `server.js`

Ce fichier assemble tous les composants :

1. Initialisation d'Express et des middlewares ;
2. Connexion à MongoDB ;
3. Configuration de la session et de `connect-flash` ;
4. Initialisation de Passport ;
5. Déclaration des routes ;
6. Démarrage du serveur.

Le code est ordonné pour que les middlewares (session, flash, passport) soient disponibles avant les routes.

```
app.use(express.json());
app.use(express.urlencoded({ extended: true }));
app.use(session({ secret: process.env.SESSION_SECRET, resave: false, saveUniniti
app.use(flash());
require('./config/passport')(passport);
app.use(passport.initialize());
app.use(passport.session());
```

Routes d'inscription et de connexion : `auth.js`

Ce fichier gère :

— **/register** : création d'un nouvel utilisateur avec validations ;

— **/login** : vérification via Passport.js ;

— **/logout** : fermeture de session.

Le code utilise un callback personnalisé pour la route de connexion, afin d'éviter les erreurs quand `req.flash` est absent.

```
router.post('/login', (req, res, next) => {
  passport.authenticate('local', (err, user, info) => {
    if (err) return next(err);

    if (!user) {
      if (typeof req.flash === 'function') {
        req.flash('error', info?.message || 'Identifiants invalides.');
      }
      return res.redirect('/login');
    }

    req.logIn(user, (err) => {
      if (err) return next(err);
      return res.redirect('/books');
    });
  })(req, res, next);
});
```

Protection des routes : `index.js`

La route '/books' est protégée par un middleware `ensureAuthenticated` :

```
function ensureAuthenticated(req, res, next) {
  if (req.isAuthenticated && req.isAuthenticated()) return next();
  req.flash('error', 'Tu dois être connecté pour accéder à cette page.');
  return res.redirect('/login');
}
```

Les vues Pug

Les pages sont créées avec le moteur **Pug**. Chaque page hérite du template principal `layout.pug`, qui contient la barre de navigation et l'inclusion de Tailwind CSS.

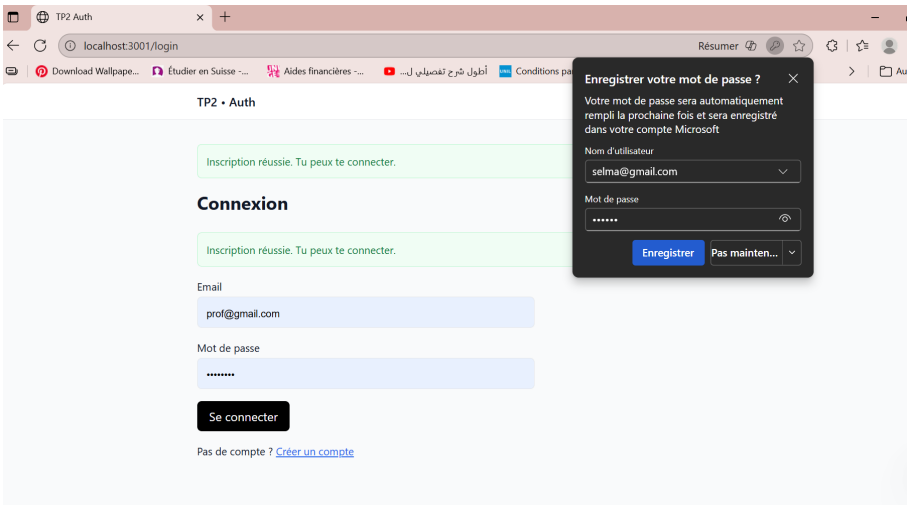
Exemple : page de connexion

```
form(method="POST" action="/login")
  div
    label(for="email") Email
    input(type="email" name="email" required)
  div
    label(for="password") Mot de passe
    input(type="password" name="password" required)
  button(type="submit") Se connecter
```

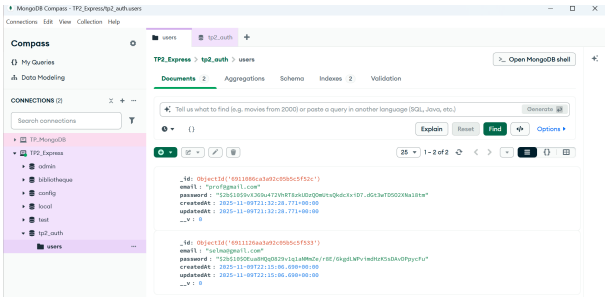
Résultat obtenu

L'application fonctionne comme suit :

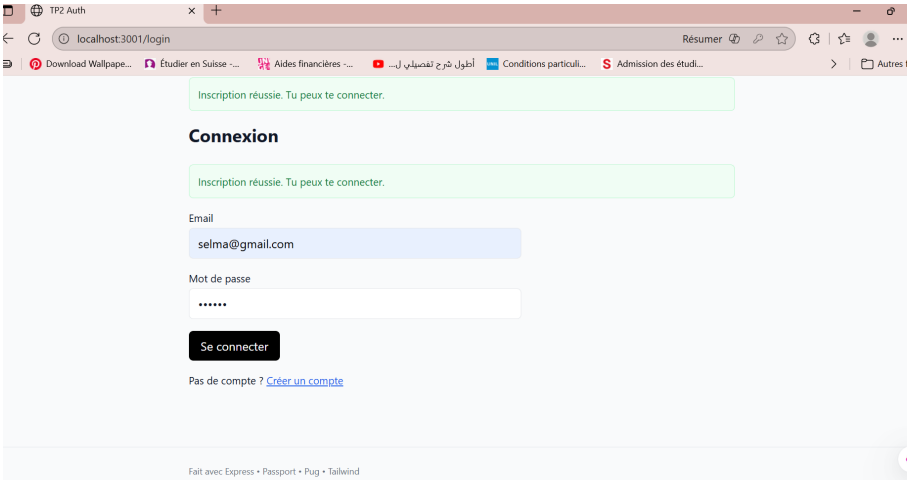
1. L'utilisateur s'inscrit depuis `créer un compte`.



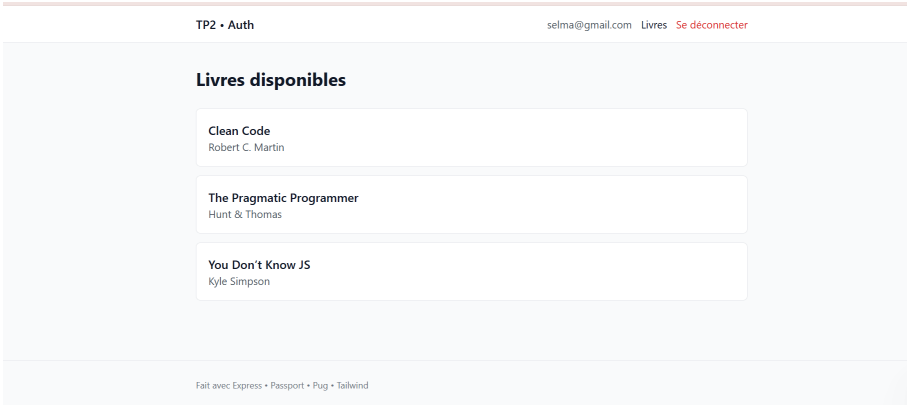
2. Ses informations sont enregistrées dans MongoDB avec un mot de passe haché.



3. Il peut ensuite se connecter via `/se connecter`.



4. Après authentification réussie, il est redirigé vers la page `/books`.



5. La déconnexion se fait via le lien `/se Déconnecter`.

Conclusion

Ce TP a permis de mettre en œuvre l'ensemble du cycle d'authentification d'un site web moderne :

- la persistance des utilisateurs avec MongoDB,
- la sécurité des mots de passe avec `bcrypt`,
- la gestion de session et de cookies avec `express-session`,
- l'authentification centralisée via `passport.js`.