Université Sultan Moulay Slimane
École Nationale des Sciences Appliquées – Khouribga

Filière : Informatique et Ingénierie des Données

Rapport du TP2 :

Projet Book Reading Tracker

Réalisé par :
Ezaakouni Selma

Année universitaire : **2025/2026**

Introduction

Le présent TP a pour objectif la création d'une application web permettant de suivre l'avancement de la lecture de différents livres. L'application repose sur une architecture moderne comprenant :

— un **frontend** simple en HTML/CSS (TailwindCSS),

— un **backend** développé en TypeScript avec Node.js et Express,

— une base de données **MongoDB** pour la persistance des livres,

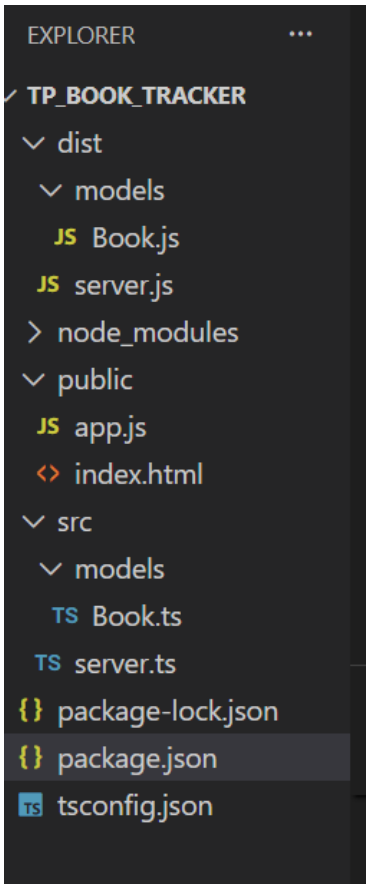
— une structure orientée objet avec une classe `Book`,

— l'utilisation de **modules TypeScript** et d'**Enums**.

Ce rapport présente les éléments essentiels du TP, les choix techniques réalisés ainsi que les explications des parties importantes du code.

Structure du projet

La structure générale du projet est la suivante :



Cette architecture sépare clairement :

— la **logique métier** dans `Book.ts`,

— le **serveur Express** dans `server.ts`,

— le **frontend** dans le dossier `public`.

Frontend : Formulaire et Interface

Le fichier `index.html` contient un formulaire permettant d'ajouter un livre et une table affichant les livres stockés dans MongoDB.

0.1 Formulaire

Le formulaire permet de saisir :

— titre,

— auteur,

— nombre de pages,

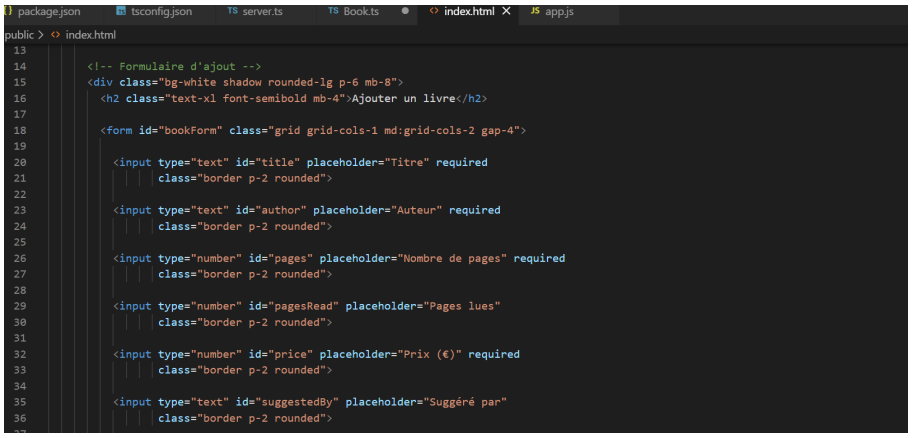
— nombre de pages lues,

— prix,

— statut (Enum),

— format (Enum),

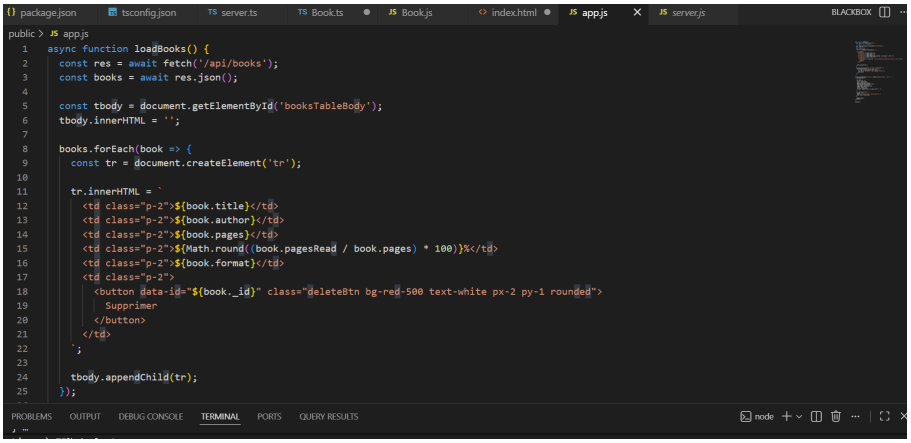
— suggéré par.



TailwindCSS est utilisé pour garantir un style moderne avec peu de code CSS.

0.2 Affichage des Livres

L'affichage se fait dans une table HTML remplie dynamiquement par `app.js`, utilisant la route `/api/books` du backend.



Backend : Modèle Book

Le fichier **Book.ts** contient :

— les **Enums** du statut et du format,

— l'**interface IBook**,

— le **schéma Mongoose** pour MongoDB,

— le **modèle BookModel**,

— la **classe Book** avec sa logique métier.

0.3 Enums

Les Enum permettent de restreindre les valeurs autorisées :

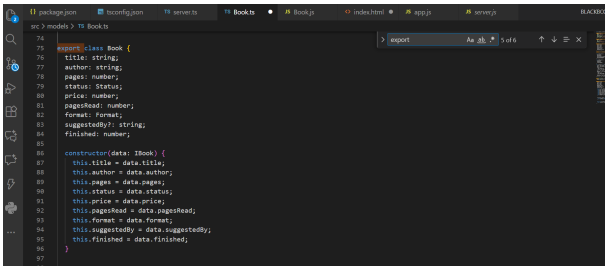
```
import mongoose, { Schema, Document, Model } from 'mongoose';

export enum Status {
  READ = 'Read',
  REREAD = 'Re-read',
  DNF = 'DNF',
  CURRENTLY_READING = 'Currently reading',
  RETURNED_UNREAD = 'Returned Unread',
  WANT_TO_READ = 'Want to read'
}
```

Schéma Mongoose

```
const bookSchema = new Schema<BookDocument>({
  title: { type: String, required: true },
  author: { type: String, required: true },
  pages: { type: Number, required: true },
  status: {
    type: String,
    enum: STATUS_VALUES,
    default: Status.WANT_TO_READ
  },
  price: { type: Number, required: true },
  pagesRead: { type: Number, default: 0 },
  format: {
    type: String,
    enum: FORMAT_VALUES,
    default: Format.PRINT
  },
  suggestedBy: { type: String, default: '' },
  finished: { type: Number, default: 0 }
});
```

Classe Book

La classe représente la logique métier du TP. Cette classe permet :



```
export class Book {
  title: string;
  author: string;
  pages: number;
  status: Status;
  price: number;
  pagesRead: number;
  format: format;
  suggestedBy?: string;
  finished: number;

  constructor(data: IBook) {
    this.title = data.title;
    this.author = data.author;
    this.pages = data.pages;
    this.status = data.status;
    this.price = data.price;
    this.pagesRead = data.pagesRead;
    this.format = data.format;
    this.suggestedBy = data.suggestedBy;
    this.finished = data.finished;
  }
```

— de calculer le pourcentage lu,

— de supprimer un livre,

— de séparer logique métier et logique de persistance.

Backend : Serveur Express

Le fichier **server.ts** initialise le serveur Node.js, configure les routes et gère l'accès à MongoDB.

0.4 Connexion à MongoDB

```
mongoose
  .connect('mongodb://127.0.0.1:27017/book_tracker')
  .then(() => console.log(' Connecté à MongoDB'))
  .catch((err) => console.error(' Erreur MongoDB :', err));
```

0.5 Route GET /api/books

Récupération de tous les livres depuis MongoDB :

```
app.get('/api/books', async (req: Request, res: Response) => {
  try {
    const books = await BookModel.find();
    res.json(books);
  } catch (err) {
    console.error('Erreur GET /api/books :', err);
    res.status(500).json({ message: 'Erreur serveur' });
  }
});
```

0.6 Route POST /api/books

Ajout d'un livre après validation et application de la logique métier :

```
app.post('/api/books', async (req: Request, res: Response) => {
  try {
    const {
      title,
      author,
      pages,
      status,
      price,
      pagesRead,
      format,
      suggestedBy
    } = req.body;

    const totalPages = Number(pages);
    const readPages = Number(pagesRead ?? 0);
```

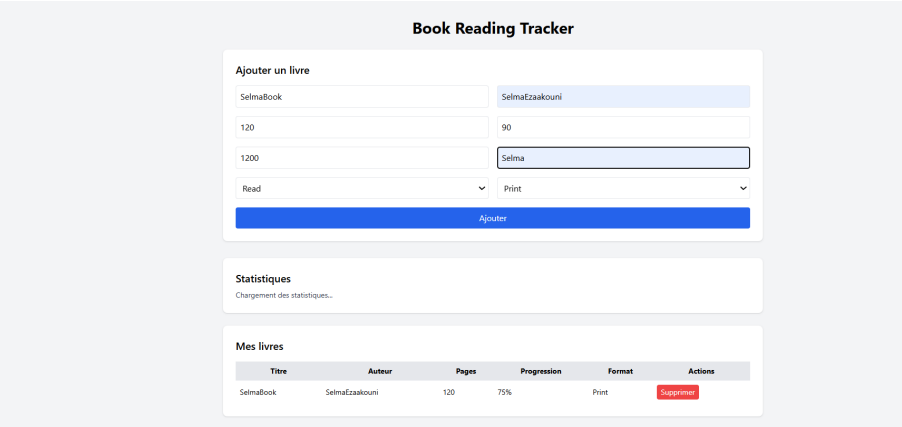
0.7 Route DELETE /api/books/ :id

```
app.delete('/api/books/:id', async (req: Request, res: Response) => {
  try {
    const id = req.params.id;
    await Book.deleteBook(id); // ★ utilisation de ta méthode statique
    res.json({ message: 'Livre supprimé' });
  } catch (err) {
    console.error('Erreur DELETE /api/books/:id :', err);
    res.status(400).json({ message: 'Erreur lors de la suppression' });
  }
});
```

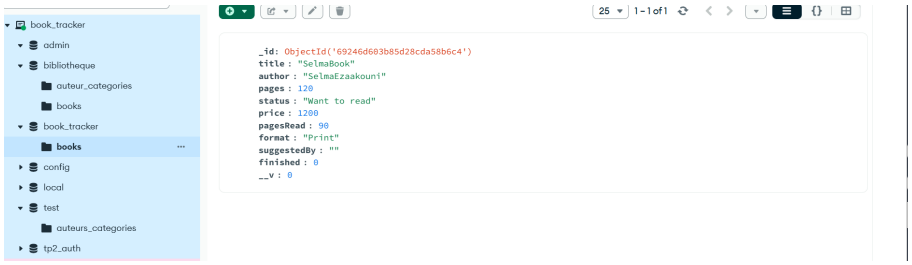
Résultat obtenu

L'application fonctionne comme suit :

1. On remplie le formulaire comme suit puis c'est ajouté au tableau :



2. Ses informations sont enregistrées dans MongoDB.



Conclusion

Ce TP m'a permis de manipuler plusieurs concepts importants du développement web moderne :

— création d'une API REST en TypeScript avec Express,
— utilisation d'une base de données NoSQL MongoDB,
— séparation entre logique métier et persistance des données,
— usage des Enums, interfaces et classes en TypeScript,
— compréhension du rôle d'un schéma Mongoose,
— interaction entre frontend et backend à travers des requêtes HTTP.