

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

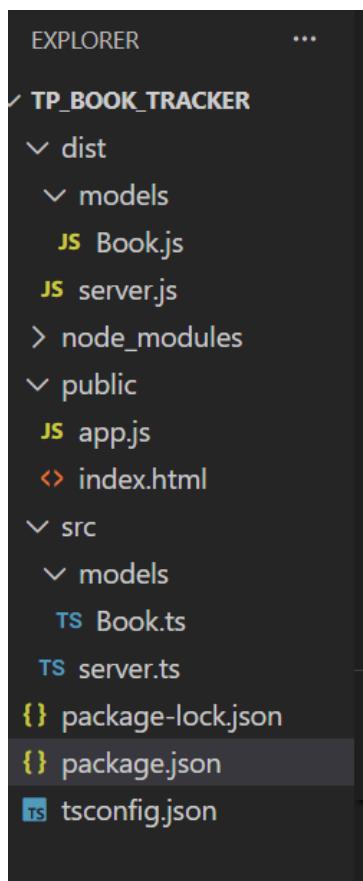
Le présent TP a pour objectif la création d'une application web permettant de suivre l'avancement de la lecture de différents livres. L'application repose sur une architecture moderne comprenant :

- 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.

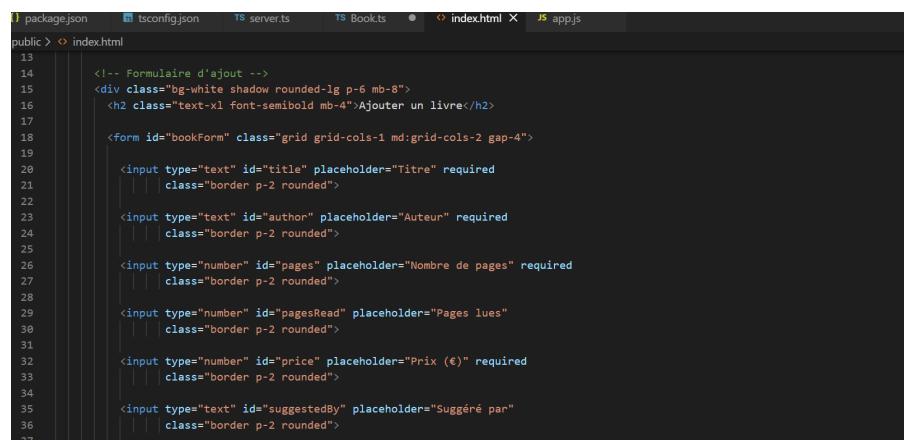
## Frontend : Formulaire et Interface

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.



```
1 package.json  2 tsconfig.json  3 server.ts  4 Book.ts  5 index.html  6 app.js
public > index.html
13
14    <!-- Formulaire d'ajout -->
15    <div class="bg-white shadow rounded-lg p-6 mb-8">
16        <h2 class="text-xl font-semibold mb-4">Ajouter un livre</h2>
17
18        <form id="bookForm" class="grid grid-cols-1 md:grid-cols-2 gap-4">
19
20            <input type="text" id="title" placeholder="Titre" required
21                class="border p-2 rounded">
22
23            <input type="text" id="author" placeholder="Auteur" required
24                class="border p-2 rounded">
25
26            <input type="number" id="pages" placeholder="Nombre de pages" required
27                class="border p-2 rounded">
28
29            <input type="number" id="pagesRead" placeholder="Pages lues"
30                class="border p-2 rounded">
31
32            <input type="number" id="price" placeholder="Prix (€)" required
33                class="border p-2 rounded">
34
35            <input type="text" id="suggestedBy" placeholder="Suggéré par"
36                class="border p-2 rounded">
37
38
```

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.

```

1  import mongoose, { Schema, Document, Model } from 'mongoose';
2
3  export enum Status {
4    READ = 'Read',
5    REREAD = 'Re-read',
6    DNF = 'DNF',
7    CURRENTLY_READING = 'Currently reading',
8    RETURNED_UNREAD = 'Returned Unread',
9    WANT_TO_READ = 'Want to read'
10   }

```

## 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

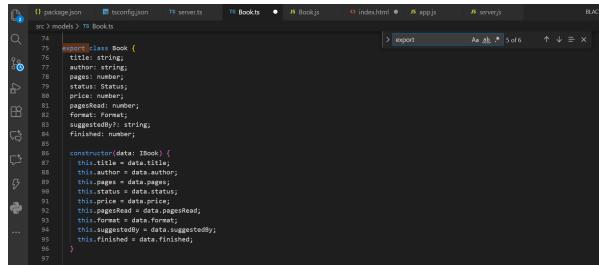
```

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 :



```
74  export class Book {
75    title: string;
76    author: string;
77    pages: number;
78    status: Status;
79    price: number;
80    format: Format;
81    pagesRead: number;
82    suggestedBy: string;
83    finished: number;
84  }
85
86  constructor(data: IBook) {
87    this.title = data.title;
88    this.author = data.author;
89    this.pages = data.pages;
90    this.status = data.status;
91    this.price = data.price;
92    this.format = data.format;
93    this.pagesRead = data.pagesRead;
94    this.suggestedBy = data.suggestedBy;
95    this.finished = data.finished;
96  }
97}
```

- 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 :

```
34 app.post('/api/books', async (req: Request, res: Response) => {
35   try {
36     const {
37       title,
38       author,
39       pages,
40       status,
41       price,
42       pagesRead,
43       format,
44       suggestedBy
45     } = req.body;
46
47     const totalPages = Number(pages);
48     const readPages = Number(pagesRead ?? 0);
```

## 0.7 Route DELETE /api/books/ :id

```
81 app.delete('/api/books/:id', async (req: Request, res: Response) => {
82   try {
83     const id = req.params.id;
84     await Book.deleteBook(id); // * utilisation de ta méthode statique
85     res.json({ message: 'Livre supprimé' });
86   } catch (err) {
87     console.error('Erreur DELETE /api/books/:id :', err);
88     res.status(400).json({ message: 'Erreur lors de la suppression' });
89   }
90 });
91
```

## Résultat obtenu

L'application fonctionne comme suit :

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

The screenshot shows the 'Book Reading Tracker' application interface. At the top, there's a modal window titled 'Ajouter un livre' (Add a book) with fields for Title ('SelmaBook'), Author ('SelmaEzaakouni'), Pages Read ('120'), Total Pages ('1200'), Progression ('90'), Format ('Selma'), and Status ('Read'). A 'Ajouter' (Add) button is at the bottom of the modal. Below the modal, there's a 'Statistiques' (Statistics) section with a link 'Changement des statistiques...'. At the bottom, there's a table titled 'Mes livres' (My books) with columns: Titre, Auteur, Pages, Progression, Format, and Actions. One row is visible: SelmaBook, SelmaEzaakouni, 120, 75%, Print, and a red 'Supprimer' (Delete) button.

## 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.