



---

## Rapport Du TP

---

**Réalisé Par :**

ARI CHAYMAA

2024/2025

# Description

## I. Introduction :

Dans un monde où le rythme de vie est de plus en plus rapide, la lecture est souvent reléguée au second plan au milieu de diverses responsabilités et distractions. Un Book Reading Tracker constitue un outil précieux pour les lecteurs passionnés, les aidant à s'organiser et à rester motivés dans leur parcours de lecture.

Ce TP vise à créer une application web conviviale permettant aux utilisateurs d'enregistrer et de gérer leurs livres, de suivre leur progression en lecture et de visualiser leurs réalisations au fil du temps.

## II. Objectifs :

Les principaux objectifs du projet Book Reading Tracker sont les suivants :

### 1. Formulaire d'enregistrement de livres :

Développement d'un formulaire HTML pour permettre d'enregistrer de nouveaux livres avec les attributs suivants nécessaires.

### 2. Implémentation de la classe Livre :

Création d'une classe Book qui encapsule les attributs et les fonctionnalités du livre, y compris :

- Un constructeur pour initialiser les propriétés du livre.
- Une méthode pour vérifier la progression actuelle de la lecture
- Une méthode pour supprimer un livre de la liste (`deleteBook`).

### 3. Suivi de la progression en lecture :

Mettre en œuvre une fonctionnalité pour suivre et afficher la progression de la lecture de chaque livre, montrant le pourcentage de pages lues par rapport au nombre total de pages.

#### **4. Aperçus globaux de la lecture :**

Création d'une section de tableau de bord qui résume les statistiques de lecture de l'utilisateur, y compris le nombre total de livres lus et le nombre total de pages lues dans l'ensemble des livres.

#### **5. Persistance des données avec MongoDB :**

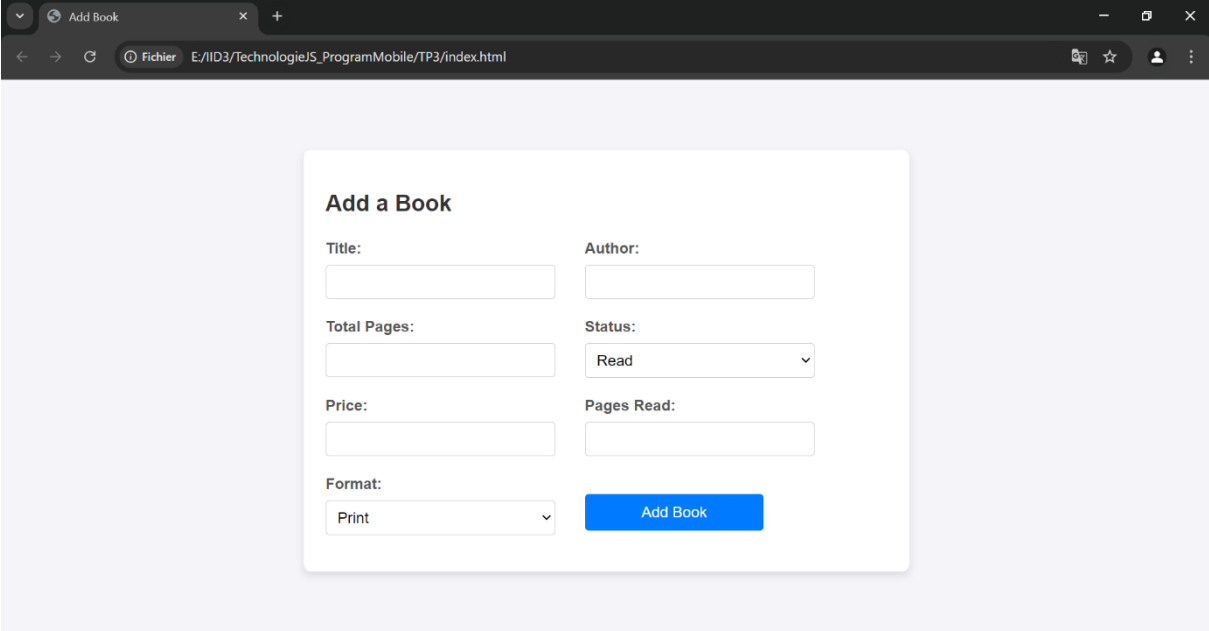
Stocker les livres enregistrés dans une base de données MongoDB pour garantir la persistance et la récupération des données pour l'application.

### **III. Technologies Utilisées**

- **Node.js** : Environnement d'exécution JavaScript côté serveur.
- **Express** : Framework web pour Node.js, facilitant la gestion des requêtes HTTP.
- **Mongoose** : Bibliothèque pour interagir avec MongoDB, simplifiant la modélisation des données.
- **TypeScript** : Superset de JavaScript qui ajoute des types statiques, améliorant la robustesse du code.
- **HTML/CSS** : Langages pour la création de l'interface utilisateur.
- **JavaScript** : Langage de programmation utilisé pour la logique côté client.

# Réalisation

## I. Création d'un formulaire d'ajout de livres :



The screenshot shows a web browser window with a single tab titled 'Add Book'. The address bar displays the file path 'E://ID3/TechnologieJS\_ProgramMobile/TP3/index.html'. The main content area features a light purple background with a white form titled 'Add a Book'. The form contains several input fields and a submit button:

- Title:** A text input field.
- Author:** A text input field.
- Total Pages:** A text input field.
- Status:** A dropdown menu with 'Read' selected.
- Price:** A text input field.
- Pages Read:** A text input field.
- Format:** A dropdown menu with 'Print' selected.
- Add Book:** A blue button to submit the form.

## II. Création de la classe Book :

Le code présente une classe **Book** qui modélise un livre avec des propriétés telles que l'identifiant, le titre, l'auteur, le nombre total de pages, le statut de lecture, le prix, les pages lues, et le format (imprimé, PDF, ebook ou audiobook). Les énumérations **BookFormat** et **BookStatus** définissent les formats disponibles et les différents états d'un livre (par exemple, lu, en cours de lecture, ou non lu). Le constructeur initialise ces propriétés et détermine si le livre est terminé en comparant les pages lues au total. La méthode **currentlyAt()** retourne une chaîne indiquant la progression actuelle de la lecture, tandis que la méthode **delete()** permet de supprimer le livre de la base de données via une requête HTTP DELETE, en affichant un message de succès ou d'erreur selon le résultat de l'opération.

```
enum BookFormat {
    PRINT = "Print",
    PDF = "PDF",
    EBOOK = "Ebook",
    AUDIOBOOK = "AudioBook",
}

enum BookStatus {
    READ = "Read",
    RE_READ = "Re-read",
    DNF = "DNF",
    CURRENTLY_READING = "Currently Reading",
    UNREAD = "Unread",
    WANT_TO_READ = "Want to Read",
}
```

```
class Book {
    id: number; // Auto-incremented ID
    title: string;
    author: string;
    totalPages: number;
    status: BookStatus;
    price: number;
    readingPages: number;
    format: BookFormat;
    finished: boolean;

    constructor(
        id: number, title: string, author: string, totalPages: number, status: BookStatus,
        price: number, readingPages: number, format: BookFormat,
    ) {
        this.id = id; // Set the ID
        this.title = title;
        this.author = author;
        this.totalPages = totalPages;
        this.status = status;
        this.price = price;
        this.readingPages = readingPages;
        this.format = format;
        this.finished = readingPages >= totalPages;
    }
}
```

```

currentlyAt(): string {
  return `You are currently at page ${this.readingPages} out of ${this.totalPages}.`;
}

async delete(): Promise<void> {
  const response = await fetch(`http://localhost:3000/books/${this.id}`, {
    method: 'DELETE',
  });

  if (response.ok) {
    alert(`Book "${this.title}" deleted successfully!`);
  } else {
    const error = await response.json();
    alert(`Error deleting book: ${error.message}`);
  }
}
}

```

### III. Configuration de la base de données MongoDB :

```

// Connect to MongoDB
mongoose.connect('mongodb://localhost:27017/Book')
  .then(() => console.log("Connected to MongoDB"))
  .catch(error => console.error("MongoDB connection error:", error));

```

Ce code établit une connexion à une base de données MongoDB nommée "Book" en utilisant la bibliothèque Mongoose, qui est un ODM (Object Data Modeling) pour MongoDB et Node.js.

La méthode **mongoose.connect()** prend l'URL de la base de données comme argument. Si la connexion réussit, un message "Connected to MongoDB" est affiché dans la console. En cas d'échec, une erreur est capturée et un message d'erreur est affiché, indiquant le problème rencontré lors de la tentative de connexion. Cela permet de gérer les erreurs de manière efficace et de s'assurer que l'application est informée de l'état de la connexion à la base de données.

## IV. Création de Book Schéma :

```
// Book Schema
const bookSchema = new mongoose.Schema({
  id: { type: Number, unique: true }, // Auto-incremented id as a number
  title: String,
  author: String,
  totalPages: Number,
  status: String,
  price: Number,
  readingPages: Number,
  format: String,
  finished: Boolean,
});

// Define the Book model
const Book = mongoose.model('Book', bookSchema);
```

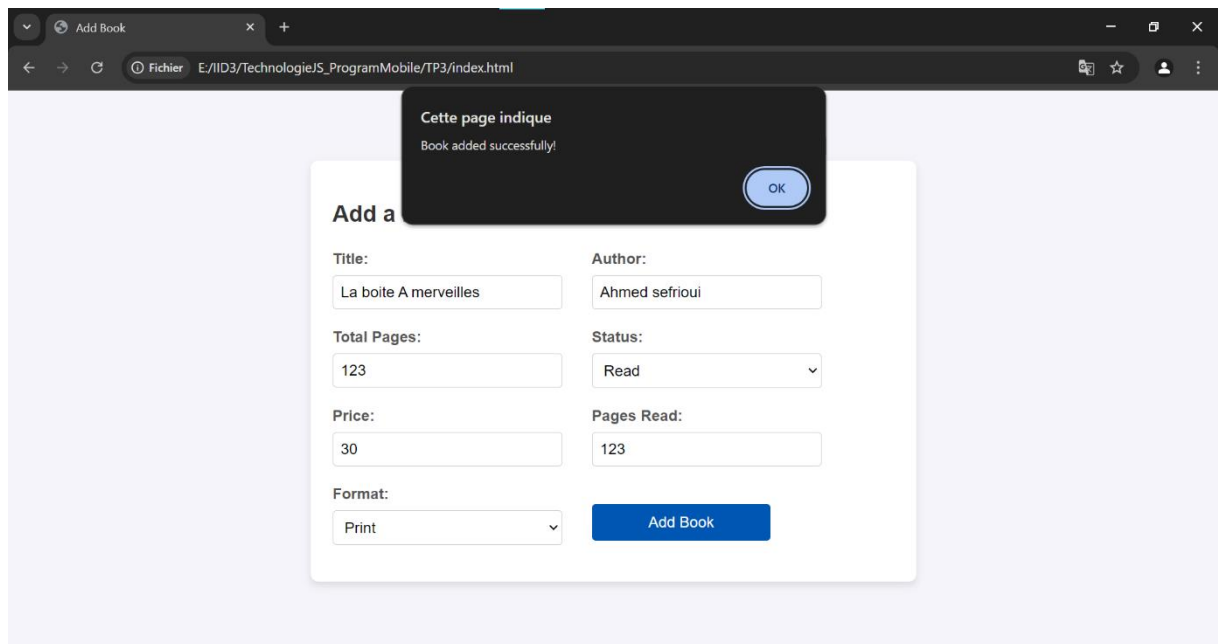
On définit un schéma Mongoose pour les livres, spécifiant les propriétés et types de données de chaque livre ainsi qu'une contrainte d'unicité pour l'`id`. Ensuite, il crée un modèle `Book` basé sur ce schéma, permettant d'interagir avec la collection de livres dans la base de données MongoDB (par exemple, pour créer, lire, mettre à jour ou supprimer des documents).

## V. Endpoint :

- Ajouter :

```
// Route to add a new book
app.post('/books', async (req: Request, res: Response) => {
  const bookData = req.body;
  const book = new Book({
    ...bookData,
    id: await getNextSequenceValue('bookId'), // Get the next sequence value for id
  });

  try {
    const savedBook = await book.save();
    res.status(201).json(savedBook);
  } catch (error) {
    res.status(500).send(error);
  }
});
```



On définit une route HTTP POST pour ajouter un nouveau livre à la collection dans la base de données. Lorsqu'une requête est effectuée sur « /books », On crée une nouvelle instance de « Book » en utilisant les données fournies dans le corps de la requête et en assignant un identifiant unique (`id`). Ensuite, On tente de sauvegarder le livre dans la base de données. Si l'opération réussit, il renvoie une réponse avec le livre enregistré et un code de statut 201 (Créé) ; sinon, il renvoie une erreur avec le code de statut 500 (Erreur interne du serveur).

- **Supprimer :**

```
app.delete('/books/:id', async (req: Request, res: Response): Promise<any> => {
  const bookId = parseInt(req.params.id); // Convert the ID from string to number

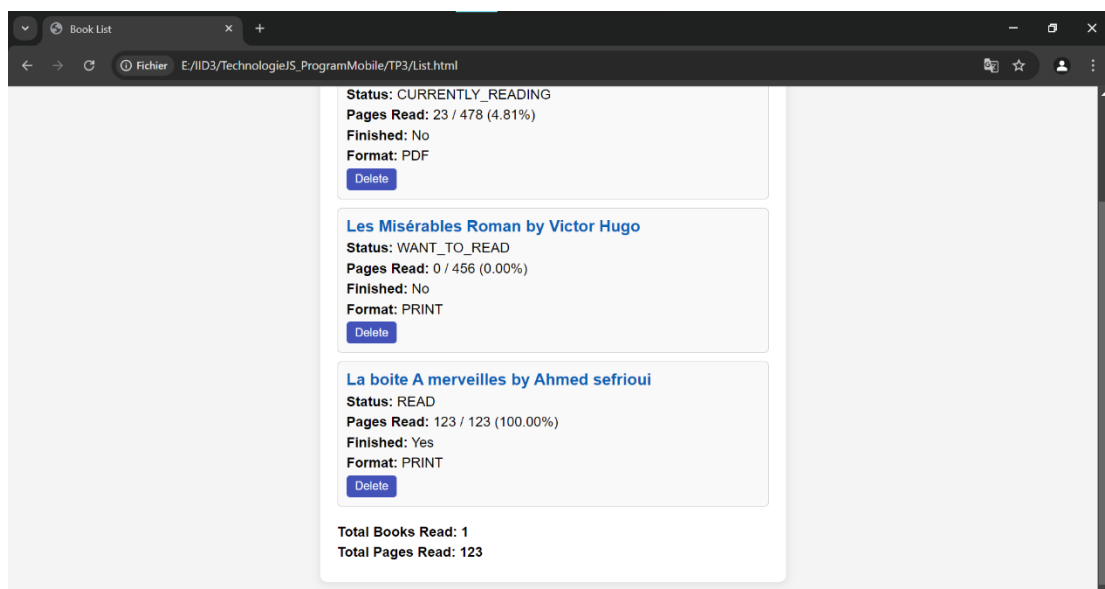
  try {
    const result = await Book.findOneAndDelete({ id: bookId }); // Using 'id' for the
    if (result) {
      return res.status(200).json({ message: 'Book deleted successfully' });
    } else {
      return res.status(404).json({ message: 'Book not found' });
    }
  } catch (error) {
    return res.status(500).json({ message: 'Error deleting book', error });
  }
});
```



- Listes des livres :

```
// Route to get all books
app.get('/books', async (req: Request, res: Response) => {
  try {
    const books = await Book.find({});
    res.json(books);
  } catch (error) {
    res.status(500).send(error);
  }
});
```

#### IV. Affichage de la liste des livres :



L'interface « BookList » est conçue pour résumer les informations d'une collection de livres, incluant le nombre total de livres lus (`totalReadBooks`), le total de pages lues (`totalPages`), et le pourcentage de lecture (`readingPercentage`). Le champ `totalReadBooks` indique combien de livres ont été classés comme "lus", tandis que `totalPages` représente la somme des pages de tous ces livres. Le `readingPercentage` est calculé en divisant le nombre total de pages lues par le total des pages des livres lus, multiplié par 100 pour obtenir un pourcentage. Cette interface permet ainsi aux utilisateurs de visualiser facilement leurs progrès et de gérer leurs statistiques de lecture.