Rapport de TP TypeScript

Présenté par:
Inane Salma

1- Introduction et Objectifs du TP

Ce TP a pour but de comprendre la programmation avec TypeScript à travers un petit projet pratique : un "Book Reading Tracker".L'idée est d'écrire un serveur Express typé en TypeScript, de manipuler une base MongoDB avec Mongoose, et d'exposer quelques routes simples pour créer, lister, mettre à jour et supprimer des livres.
Parmi les objectifs :

- Installer et configurer TypeScript : créer la configuration du compilateur (tsconfig) et les scripts NPM afin de compiler un projet Node en JavaScript.
- Mettre en place un serveur Express typé : initialiser un serveur HTTP en TypeScript, activer les middlewares essentiels (JSON, CORS) et structurer les routes.
- Modéliser les livres avec Mongoose : définir un schéma strict (types, enums, validations) et établir la connexion à une base MongoDB.
- Développer les opérations de gestion des livres : implémenter l'ajout, la consultation, la mise à jour de la progression de lecture et la suppression, avec gestion d'erreurs claire.

2 - backend

2.1 tsconfig.json

```json
{
  "compilerOptions": {
    "target": "ES2020",
    "module": "commonjs",
    "rootDir": "./Backend",
    "outDir": "./Backend/dist",
    "strict": true,
    "esModuleInterop": true,
    "forceConsistentCasingInFileNames": true,
    "skipLibCheck": true,
    "types": ["node"],
    "resolveJsonModule": true
  },
  "include": ["Backend/**/*.ts"],
  "exclude": ["Backend/dist", "Frontend"]
}
```

J'ai créé ce fichier pour configurer la compilation TypeScript du backend.Je définis module: "commonjs" et moduleResolution: "node" pour un projet Express/Node, je sépare le code source (rootDir: Backend) et le JavaScript généré (outDir: Backend/dist), Les types Node sont inclus (types: ["node"]) pour que les APIs natives soient reconnues par le compilateur.

2.2 tsconfig.json

```typescript
import { Schema, model, Document } from 'mongoose';

export type BookStatus =
  | 'Read' | 'Re-read' | 'DNF' | 'Currently reading' | 'Returned Unread' | 'Want to read';
export type BookFormat = 'Print' | 'PDF' | 'Ebook' | 'AudioBook';

export interface IBook extends Document {
  title: string;
  author: string;
  pages: number;
  status: BookStatus;
  price: number;
  pagesRead: number;
  format: BookFormat;
  suggestedBy?: string;
  finished: boolean;
}
```

```ts
const bookSchema = new Schema<IBook>({
  title: { type: String, required: true },
  author: { type: String, required: true },
  pages: { type: Number, required: true, min: 1 },
  status: {
    type: String,
    enum: ['Read','Re-read','DNF','Currently reading','Returned Unread','Want to read'],
    default: 'Want to read'
  },
  price: { type: Number, default: 0 },
  pagesRead: { type: Number, default: 0, min: 0 },
  format: { type: String, enum: ['Print','PDF','Ebook','AudioBook'], default: 'Print' },
  suggestedBy: { type: String, default: '' },
  finished: { type: Boolean, default: false }
});

bookSchema.pre('save', function (next) {
  this.finished = this.pagesRead >= this.pages;
  if (this.pagesRead > this.pages) this.pagesRead = this.pages;
  next();
});

export default model<IBook>('Book', bookSchema);
```

J'ai créé ce modèle Mongoose pour traduire les exigences de l'énoncé en schéma MongoDB. On y déclare les champs demandés avec deux enums pour status et format. Le hook pre('save') applique la règle métier : finished passe à true lorsque pagesRead >= pages.

2.3 bookRoutes.ts

```ts
import { Router, Request, Response } from 'express';
import Book, { IBook } from '../models/Book';

const router = Router();

router.post('/', async (req: Request, res: Response) => {
  try {
    const book = new Book(req.body as Partial<IBook>);
    await book.save();
    res.status(201).json(book);
  } catch (error: unknown) {
    const msg = error instanceof Error ? error.message : String(error);
    res.status(400).json({ error: msg });
  }
});

router.get('/', async (_req: Request, res: Response) => {
  const books = await Book.find().sort({ _id: -1 });
  res.json(books);
});
```

```
router.put('/:id', async (req: Request, res: Response) => {
  const { pagesRead } = req.body as { pagesRead: number };
  const book = await Book.findById(req.params.id);
  if (!book) return res.status(404).json({ message: 'Book not found' });

  const value = Math.max(0, Number(pagesRead));
  book.pagesRead = Math.min(value, book.pages);
  book.finished = book.pagesRead >= book.pages;
  await book.save();

  res.json(book);
});

router.delete('/:id', async (req: Request, res: Response) => {
  await Book.findByIdAndDelete(req.params.id);
  res.json({ message: 'Book deleted' });
});

export default router;
```

J'ai créé routeur pour exposer l'apis qui correspond aux opérations demandées :

- POST /api/books pour ajouter un livre
- GET /api/books pour lister les livres
- PUT /api/books/:id pour mettre à jour la progression
- DELETE /api/books/:id pour supprimer les

2.4 server.ts

```
1  import express from 'express';
2  import mongoose from 'mongoose';
3  import cors from 'cors';
4  import dotenv from 'dotenv';
5  import path from 'path';
6  import bookRoutes from './routes/bookRoutes';
7
8  dotenv.config({ path: path.resolve(__dirname, '../.env') });
9
10 const app = express();
11 app.use(cors());
12 app.use(express.json());
13
14 app.use('/api/books', bookRoutes);
15
16 mongoose
17   .connect(process.env.MONGO_URI as string)
18   .then(() => console.log('✅ MongoDB connected'))
19   .catch((err: unknown) => {
20     const msg = err instanceof Error ? err.message : String(err);
21     console.error('❌ MongoDB error:', msg);
22   });
```

J’ai écrit ce point d’entrée pour initialiser l’application : création d’un serveur Express avec les middlewares requis, montage des routes /api/books,connexion à MongoDB avec Mongoose, écoute sur 5000.Ce fichier relie toutes les pièces et met en œuvre l’objectif du tp: un mini-serveur Express en TypeScript, connecté à MongoDB, prêt à recevoir les opérations de suivi de lecture.

```
> start
> node Backend/dist/server.js

[dotenv@17.2.3] injecting env (2) from Backend\.env -- tip: ⚙ specify custom .env file path with { path: '/custom/path/.env' }
🚀 Server running on port 5000
✅ MongoDB connected
```

3 - Frontend

3.1 tsconfig.json

```json
{
  "compilerOptions": {
    "target": "ES2020",
    "module": "ES2020",
    "moduleResolution": "bundler",
    "lib": ["DOM", "ES2020"],
    "rootDir": "./Frontend/js",
    "outDir": "./Frontend/dist",
    "strict": true,
    "esModuleInterop": true,
    "skipLibCheck": true
  },
  "include": ["Frontend/js/**/*.ts"],
  "exclude": ["Frontend/dist", "Backend"]
}
```

J’ai ajouté un tsconfig dédié au front pour compiler le TypeScript destiné au navigateur. je configure module: "ES2020" et moduleResolution: "bundler" pour générer des modules ES chargés directement dans le navigateur. j’active lib: ["DOM","ES2020"] pour typer le DOM, et je sépare les dossiers (rootDir: Frontend/js → outDir: Frontend/dist) afin d’obtenir un build propre.

3.2 book.ts

```ts
export enum BookStatus {
  Read = 'Read',
  Reread = 're-read',
  DNF = 'DNF',
  CurrentlyReading = 'Currently reading',
  ReturnedUnread = 'Returned Unread',
  WantToRead = 'Want to read'
}
export enum BookFormat {
  Print = 'Print',
  PDF = 'PDF',
  Ebook = 'Ebook',
  AudioBook = 'AudioBook'
}
export interface BookDTO {
  _id?: string;
  title: string;
  author: string;
  pages: number;
  status: BookStatus;
  price: number;
  pagesRead: number;
  format: BookFormat;
  suggestedBy?: string;
  finished: boolean;
}
export default class Book implements BookDTO {
  _id?: string;
  title: string;
  author: string;
  pages: number;
  status: BookStatus;
  price: number;
  pagesRead: number;
  format: BookFormat;
  suggestedBy?: string;
  finished: boolean;
```

```ts
constructor(data: Partial<BookDTO> & Pick<BookDTO, 'title' | 'author' | 'pages'>) {
  this._id = data._id;
  this.title = data.title;
  this.author = data.author;
  this.pages = Number(data.pages);

  this.status = (data.status ?? BookStatus.WantToRead) as BookStatus;
  this.price = Number(data.price ?? 0);
  this.pagesRead = Math.min(Number(data.pagesRead ?? 0), this.pages);
  this.format = (data.format ?? BookFormat.Print) as BookFormat;
  this.suggestedBy = data.suggestedBy ?? '';
  this.finished = Boolean(data.finished ?? this.pagesRead >= this.pages);
}

currentlyAt(): { page: number; total: number; percent: number } {
  const percent = this.pages ? Math.round((this.pagesRead / this.pages) * 100) : 0;
  return { page: this.pagesRead, total: this.pages, percent };
}

async deleteBook(apiDelete: (id: string) => Promise<unknown>): Promise<void> {
  if (!this._id) throw new Error('No _id on book');
  await apiDelete(this._id);
}
```

J’ai écrit ce point d’entrée pour initialiser l’application :chargement du .env (chemin robuste via path.resolve),création d’un serveur Express avec les middlewares requis (express.json pour lire le corps JSON, cors pour autoriser le front),montage des routes /api/books,connexion à MongoDB avec Mongoose et logs clairs en cas d’erreur,écoute sur PORT.Ce fichier relie toutes les pièces et met en œuvre l’objectif du TP : un mini-serveur Express en TypeScript, connecté à MongoDB, prêt à recevoir les opérations de suivi de lecture.

3.3 api.ts

```ts
import type { BookDTO } from './book';

const API_URL = 'http://localhost:5000/api/books';

async function jsonFetch<T>(input: RequestInfo, init?: RequestInit): Promise<T> {
  const res = await fetch(input, init);
  if (!res.ok) {
    const body = await res.text().catch(() => '');
    throw new Error(`${res.status} ${res.statusText} ${body}`);
  }
  return res.json() as Promise<T>;
}

export function apiList(): Promise<BookDTO[]> {
  return jsonFetch<BookDTO[]>(API_URL);
}

export function apiCreate(payload: Partial<BookDTO>): Promise<BookDTO> {
  return jsonFetch<BookDTO>(API_URL, {
    method: 'POST',
    headers: { 'Content-Type': 'application/json' },
    body: JSON.stringify(payload)
  });
}

export function apiDelete(id: string): Promise<{ message: string }> {
  return jsonFetch<{ message: string }>(`${API_URL}/${id}`, { method: 'DELETE' });
}

export function apiUpdateProgress(id: string, pagesRead: number): Promise<BookDTO> {
  return jsonFetch<BookDTO>(`${API_URL}/${id}`, {
    method: 'PUT',
    headers: { 'Content-Type': 'application/json' },
    body: JSON.stringify({ pagesRead })
  });
}
```

J’ai isolé les appels HTTP dans un module api.ts pour séparer la communication réseau de la logique ui. chaque fonction est générique et typée :

- apiList() retourne un tableau de BookDTO,
- apiCreate() envoie le payload typé pour créer un livre,
- apiUpdateProgress() met à jour pagesRead,
- apiDelete() supprime un livre.

3.4 app.ts

```ts
import Book, { BookDTO, BookFormat, BookStatus } from './book.js';
import { apiList, apiCreate, apiDelete, apiUpdateProgress } from './api.js';

// Sélecteurs DOM typés
const bookListEl = document.getElementById('bookList') as HTMLDivElement | null;
const form = document.getElementById('bookForm') as HTMLFormElement | null;
const statsEl = document.getElementById('globalStats') as HTMLDivElement | null;

if (!bookListEl || !form || !statsEl) {
  throw new Error('Missing required DOM elements (bookList, bookForm, globalStats).');
}

function badge(text: string | number): string {
  return `<span class="text-xs px-2 py-1 rounded-full bg-slate-100 border">${text}</span>`;
}

function progressBar(percent: number): string {
  return `
    <div class="w-full bg-slate-200 rounded h-2 mt-1">
      <div class="h-2 rounded ${percent === 100 ? 'bg-emerald-500' : 'bg-blue-500'}" style="width:${percent}%;"></div>
    </div>
  `;
}

function renderStats(books: Book[]): void {
  const totalPages = books.reduce((a, b) => a + b.pages, 0);
  const totalRead  = books.reduce((a, b) => a + b.pagesRead, 0);
  const finished   = books.filter(b => b.finished).length;
  const pct = totalPages ? Math.round((totalRead / totalPages) * 100) : 0;

  statsEl!.classList.remove('hidden');
  statsEl!.innerHTML = `
    <h2 class="text-xl font-semibold mb-2">Global stats</h2>
    <div class="grid grid-cols-2 md:grid-cols-4 gap-4">
      <div class="bg-slate-50 border rounded p-4">
        <div class="text-3xl font-bold">${books.length}</div>
        <div class="text-slate-500">Books</div>
      </div>
      <div class="bg-slate-50 border rounded p-4">
        <div class="text-3xl font-bold">${finished}</div>
        <div class="text-slate-500">Finished</div>
      </div>
      <div class="bg-slate-50 border rounded p-4">
        <div class="text-3xl font-bold">${totalRead}/${totalPages}</div>
        <div class="text-slate-500">Pages read</div>
      </div>
      <div class="bg-slate-50 border rounded p-4">
        <div class="text-3xl font-bold">${pct}%</div>
        <div class="text-slate-500">Completion</div>
      </div>
    </div>
  `;
}
```

```ts
function renderBookCard(b: Book): string {
  const { page, total, percent } = b.currentlyAt();

  return `
    <article class="bg-white rounded-lg shadow p-5" data-id="${b._id ?? ''}">
      <div class="flex items-start justify-between gap-3">
        <div>
          <h3 class="text-lg font-bold">${b.title}</h3>
          <p class="text-slate-600">Author: ${b.author}</p>
          <div class="flex items-center gap-2 mt-1">
            ${badge(b.status)}
            ${badge(b.format)}
            ${b.finished ? '<span class="text-xs px-2 py-1 rounded-full bg-emerald-100 text-emerald-700 border border-emer
          </div>
        </div>
        <button class="btn-delete bg-red-500 text-white px-3 py-1 rounded hover:bg-red-600">Delete</button>
      </div>

      <div class="mt-3 text-sm">
        <div>Progress: <span class="font-medium">${page}/${total}</span> pages (${percent}%)</div>
        ${progressBar(percent)}
      </div>

      <div class="mt-4 flex items-center gap-2">
        <input type="number" min="0" max="${b.pages}" value="${b.pagesRead}" class="inp-progress border rounded p-2 w-32"
        <button class="btn-save bg-blue-600 text-white px-3 py-2 rounded hover:bg-blue-700">Save progress</button>
      </div>

      ${b.suggestedBy ? `<div class="mt-3 text-xs text-slate-500">Suggested by: ${b.suggestedBy}</div>` : ''}
    </article>
  `;
}
```

```ts
async function loadAndRender(): Promise<void> {
  const data: BookDTO[] = await apiList();
  const books = data.map(d => new Book(d));
  renderStats(books);
  bookListEl!.innerHTML = books.map(renderBookCard).join('');

  // Event delegation par carte
  bookListEl!.querySelectorAll('article').forEach(card => {
    const id = (card as HTMLElement).dataset.id ?? '';

    const btnDelete = card.querySelector<HTMLButtonElement>('.btn-delete');
    const btnSave   = card.querySelector<HTMLButtonElement>('.btn-save');
    const inp       = card.querySelector<HTMLInputElement>('.inp-progress');

    if (btnDelete) {
      btnDelete.addEventListener('click', async () => {
        if (!id) return;
        await apiDelete(id);
        await loadAndRender();
      });
    }

    if (btnSave && inp) {
      btnSave.addEventListener('click', async () => {
        if (!id) return;
        const value = Number(inp.value);
        await apiUpdateProgress(id, value);
        await loadAndRender();
      });
    }
  });
}
```

```ts
const toNum = (v: FormDataEntryValue | null | undefined, fallback = 0): number => {
  const n = Number(v ?? fallback);
  return Number.isFinite(n) ? n : fallback;
};

form.addEventListener('submit', async (e: SubmitEvent) => {
  e.preventDefault();

  const fd = new FormData(form);
  // Record<string, FormDataEntryValue> → on construit un payload typé
  const payload: Partial<BookDTO> = {
    title: String(fd.get('title') ?? ''),
    author: String(fd.get('author') ?? ''),
    pages: toNum(fd.get('pages'), 0),
    price: toNum(fd.get('price'), 0),
    pagesRead: toNum(fd.get('pagesRead'), 0),
    status: (fd.get('status') as BookStatus) ?? BookStatus.WantToRead,
    format: (fd.get('format') as BookFormat) ?? BookFormat.Print,
    suggestedBy: String(fd.get('suggestedBy') ?? ''),
    finished: false
  };

  if (!payload.title || !payload.author || !payload.pages || payload.pages <= 0) {
    alert('Title, author and pages are required.');
    return;
  }

  if ((payload.pagesRead ?? 0) > (payload.pages ?? 0)) {
    alert('Pages read must be <= total pages');
    return;
  }

  await apiCreate(payload);
  form.reset();
  await loadAndRender();
});
```

J'ai centralisé l'orchestration de l'interface dans app.ts. le fichier sélectionne les éléments du DOM avec des types explicites, charge les livres via apiList(), puis rend chaque carte (titre, auteur, badges statut/format, barre de progression). Les boutons save progress et delete déclenchent des appels API typés (apiUpdateProgress, apiDelete) et rechargent l'affichage. le formulaire d'ajout valide les entrées avant d'appeler apiCreate(). une section statistiques globales (total de livres, nombre terminés, pages lues, pourcentage) est recalculée à chaque rendu.

3.5 app.ts

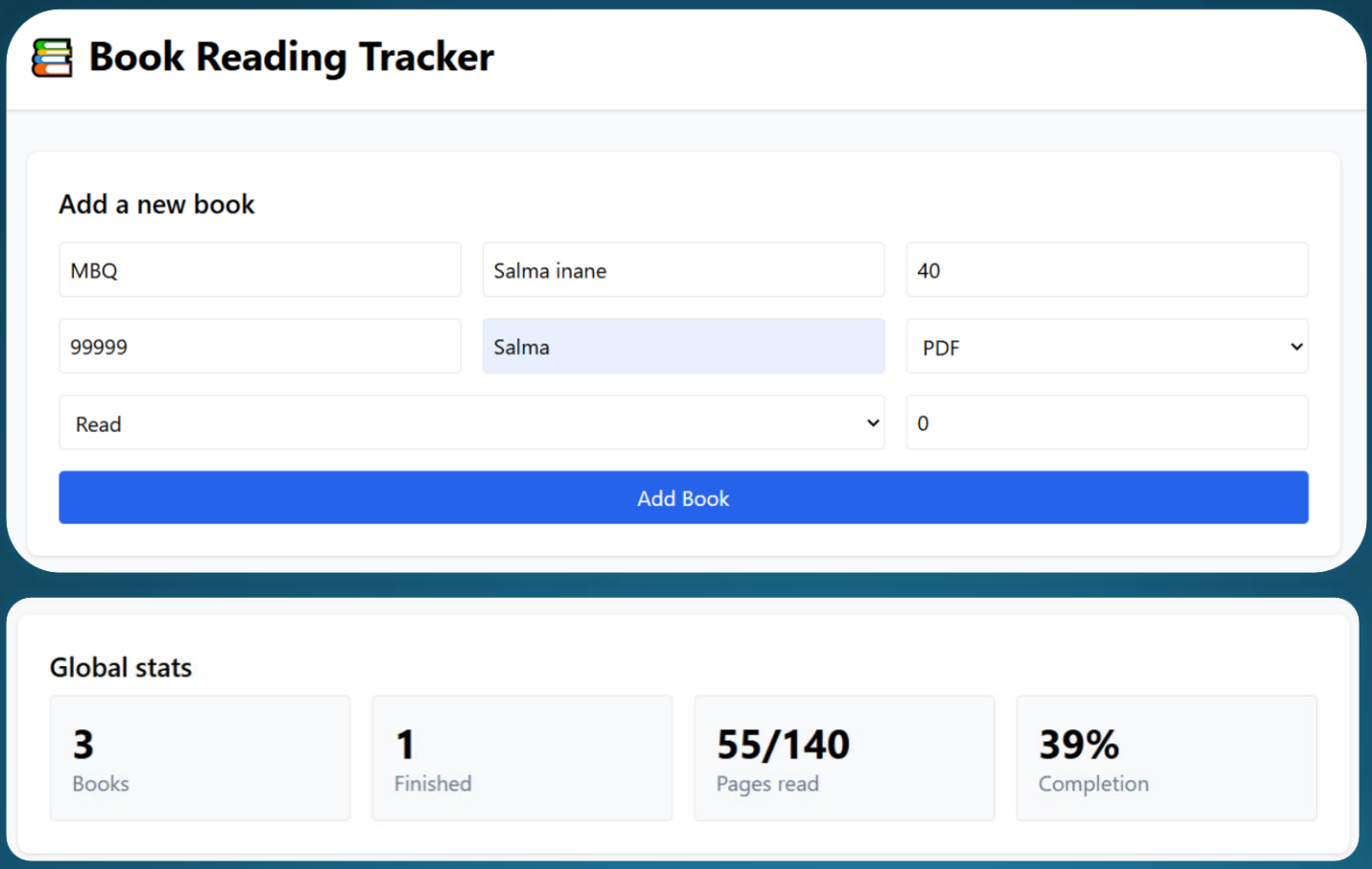
```html
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8" />
  <title>Book Reading Tracker</title>
  <meta name="viewport" content="width=device-width, initial-scale=1" />
  <script src="https://cdn.tailwindcss.com"></script>
</head>
<body class="bg-slate-50 min-h-screen">
  <header class="bg-white shadow">
    <div class="max-w-5xl mx-auto px-4 py-6">
      <h1 class="text-3xl font-bold">📚 Book Reading Tracker</h1>
    </div>
  </header>

  <main class="max-w-5xl mx-auto px-4 py-8 space-y-10">

    <!-- Form -->
    <section class="bg-white rounded-lg shadow p-6">
      <h2 class="text-xl font-semibold mb-4">Add a new book</h2>
      <form id="bookForm" class="grid grid-cols-1 md:grid-cols-3 gap-4">
        <input class="border rounded p-2" name="title" placeholder="Title" required />
        <input class="border rounded p-2" name="author" placeholder="Author" required />
        <input class="border rounded p-2" type="number" name="pages" placeholder="Total pages" min="1" required
        <input class="border rounded p-2" type="number" name="price" placeholder="Price (optional)" min="0" />
        <input class="border rounded p-2" name="suggestedBy" placeholder="Suggested by (optional)" />

        <select name="format" class="border rounded p-2">
          <option>Print</option>
          <option>PDF</option>
          <option>Ebook</option>
          <option>AudioBook</option>
        </select>
```

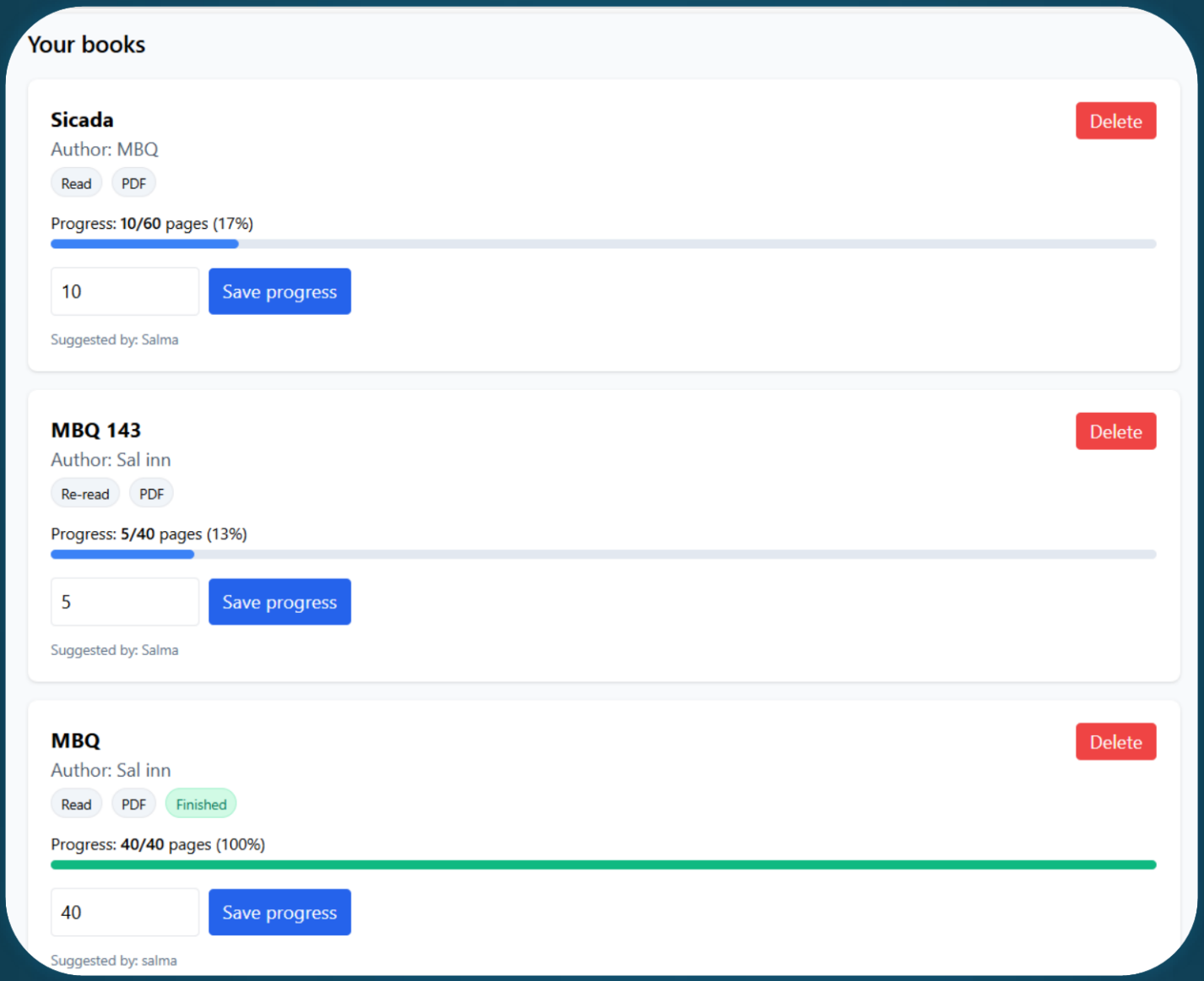
```html
  <select name="format" class="border rounded p-2">
    <option>Print</option>
    <option>PDF</option>
    <option>Ebook</option>
    <option>AudioBook</option>
  </select>

  <select name="status" class="border rounded p-2 md:col-span-2">
    <option>Want to read</option>
    <option>Currently reading</option>
    <option>Read</option>
    <option>Re-read</option>
    <option>DNF</option>
    <option>Returned Unread</option>
  </select>

  <!-- pagesRead initial -->
  <input class="border rounded p-2" type="number" name="pagesRead" placeholder="Pages read (start)" value="0"

  <button class="bg-blue-600 text-white rounded p-2 md:col-span-3 hover:bg-blue-700 transition">
    Add Book
  </button>
</form>
</section>

<!-- Global stats -->
<section id="globalStats" class="bg-white rounded-lg shadow p-6 hidden"></section>
```

```
    <!-- Book list -->
    <section>
      <h2 class="text-xl font-semibold mb-4">Your books</h2>
      <div id="bookList" class="space-y-4"></div>
    </section>
  </main>

  <footer class="text-center text-slate-400 py-6">
    Built by Salma Inane IID3
  </footer>

  <script type="module" src="./dist/app.js"></script>
</body>
</html>
```

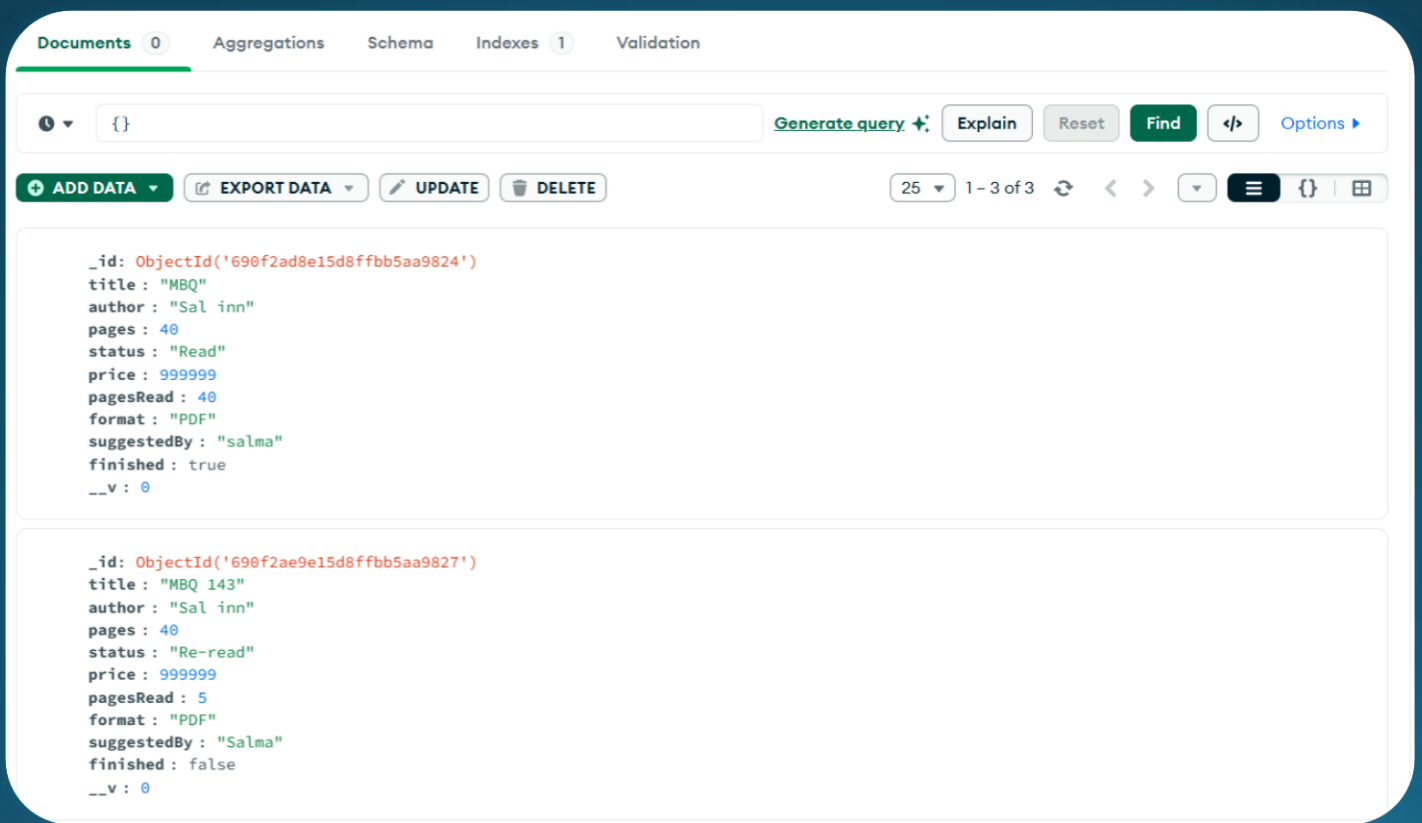
Cette page fournit le gabarit HTML stylé avec TailwindCSS. elle contient le formulaire d'ajout, une zone pour la liste des livres et une section de statistiques. elle charge le JavaScript compilé (./dist/app.js) en module ES :

4 - Resultat




Your books
Sicada
Author: MBQ
Read
PDF
Progress: 10/60 pages (17%)
10
Save progress
Suggested by: Salma
Delete
MBQ 143
Author: Sal inn
Re-read
PDF
Progress: 5/40 pages (13%)
5
Save progress
Suggested by: Salma
Delete
MBQ
Author: Sal inn
Read
PDF
Finished
Progress: 40/40 pages (100%)
40
Save progress
Suggested by: salma
Delete

MongoDB


Documents 0
Aggregations
Schema
Indexes 1
Validation
{}
Generate query
Explain
Reset
Find
Options
ADD DATA
EXPORT DATA
UPDATE
DELETE
25
1 - 3 of 3
_id: ObjectId('690f2ad8e15d8ffbb5aa9824')
title : "MBQ"
author : "Sal inn"
pages : 40
status : "Read"
price : 999999
pagesRead : 40
format : "PDF"
suggestedBy : "salma"
finished : true
__v : 0
_id: ObjectId('690f2ae9e15d8ffbb5aa9827')
title : "MBQ 143"
author : "Sal inn"
pages : 40
status : "Re-read"
price : 999999
pagesRead : 5
format : "PDF"
suggestedBy : "Salma"
finished : false
__v : 0

Conclusion

Ce TP m’a permis de comprendre et pratiquer typeScript dans un contexte concret. en réalisant un petit projet complet, j’ai vu comment typeScript s’utilise vraiment au quotidien : typer les données, organiser le code de façon claire, j’ai aussi mieux saisi le cycle complet d’une application avec une logique simple et compréhensible .Au-delà du résultat, cette expérience m’a surtout comprendre le rôle des types pour guider le développement. Je me sens désormais plus à l’aise pour aborder d’autres projets avec typeScript et appliquer les mêmes réflexes de clarté et de fiabilité.