📚 Guided Project: Building a Book Management System with Express.js & Frontend Filtering

This project will **refactor the frontend** and **build a full backend** using **Express.js**, replacing the previous JSON server. Trainees will apply **backend filtering, sorting, and handling CORS**, while reusing the frontend they previously built.

🚀 Project Objectives

1. **Build a backend with Express.js** to serve book data via API.
2. **Enable CORS** for frontend access.
3. **Implement filtering, sorting, and pagination** on the backend.
4. **Refactor the frontend** to fetch data from the new backend instead of the JSON server.
5. **Use frontend JavaScript** to apply dynamic UI filters and sorting.

📂 Project Structure

pgsql
CopyEdit

```
/book-management
|── /backend
|   ├── server.ts (Express backend)
|   ├── books.ts (Book data and filtering logic)
|   ├── routes.ts (Routes for handling API requests)
|   ├── middleware.ts (CORS & error handling)
|   ├── package.json
|   ├── tsconfig.json
|── /frontend
|   ├── index.html (UI for book display)
|   ├── style.css (Basic styles)
|   ├── index.ts (Handles DOM manipulation)
|   ├── api.ts (Handles API requests)
```

```
|    ├── bookRenderer.ts (Formats and displays books)
|── README.md
```

📌 Step 1: Setting Up the Backend

🛠️ Backend Setup

- Install **Express.js, TypeScript, and CORS**.
- Create a **server** that serves book data via API.

👨‍💻 Trainee Tasks

- Set up an **Express.js server** with TypeScript.
- Create an **API route (`/api/books`)** to serve book data.

📌 Step 2: Implement Backend Filtering & Sorting

🔄 Filtering Books

- Allow filtering by:
  - **Genre** → `/api/books?genre=Fantasy`
  - **Year Published** → `/api/books?year=1950`
  - **Page Count** → `/api/books?pages=500`
- Combine filters dynamically.

📊 Sorting Books

- **Sort by year (ascending/descending).**
- **Sort by number of pages.**

👨‍💻 Trainee Tasks

- Implement **query parameters (`req.query`)** for filtering and sorting.
- Ensure the backend returns **filtered & sorted results** dynamically.

📌 Step 3: Handling CORS & Error Management

✅ Enable CORS

- Allow **frontend requests** (`http://localhost:5173`).
- Use **middleware** to apply CORS policies.

🚨 Error Handling

- Handle **invalid requests** (e.g., `400 Bad Request`).
- Handle **server errors** (e.g., `500 Internal Server Error`).

👨‍💻 Trainee Tasks

- Set up **CORS middleware**.
- Implement **error handling** middleware.

📌 Step 4: Refactor the Frontend

🔄 Update API Calls

- Replace **JSON server calls** with **Express API requests**.

🎨 Frontend Filtering & Sorting

- Implement **dropdowns for filtering**.
- Add **buttons for sorting**.
- Use **.map(), .filter(), and .sort()** on fetched data.

👨‍💻 Trainee Tasks

- Update **API requests** in `api.ts`.
- Implement **event listeners** for dynamic filtering/sorting.

📌 Step 5: Dynamic UI Interactions

⚡ Live Book Filtering

- **User selects "Fantasy" → Only Fantasy books appear.**
- **User selects "Before 1950" → Only older books appear.**

📢 Special Book Messages

- **If pages > 500 → Display “Long Read” badge.**
- **If year < 1900 → Display “Classic Literature” alert.**

👨‍💻 Trainee Tasks

- Implement **event-driven UI updates**.
- Add **badges for special books**.

🎯 Final Task: Building a Powerful Book Searching System

✅ **Backend** serves **filtered & sorted book data**.
✅ **Frontend** dynamically filters **books based on user input**.
✅ **CORS enabled** to allow frontend-backend interaction.
✅ **Books sorted & formatted with UI enhancements**.

💡 Stretch Goals

- **Implement pagination** → `/api/books?page=2&limit=5`
- **Allow multiple filters at once** (e.g., `?genre=Fiction&year=1950`).
- **Enhance UI with animations** when filtering.

🚀 Expected Outcome

By the end of this project, trainees will: ✔ **Build & manage an Express.js backend**
✔ **Filter & sort books on the backend**
✔ **Handle CORS & API requests correctly**
✔ **Implement dynamic UI filtering on the frontend**

🔥 **Now they have a complete backend-powered book management system!** 🚀