Aufgabe: Erstelle eine Buchverwaltung mit NodeJS

Wir haben uns angesehen, wie wir mit NodeJS, EJS, Express,... einen funktionierenden Webserver erstellen und dort mit Hilfe einer Datenbank ein sicheres Login zu einem Backend ermöglichen. Zur Vertiefung des Gelernten soll nun eine Buchverwaltung mittels NodeJS erstellt werden. Gehe dazu die folgenden Seiten durch und präsentiere am Ende dein Ergebnis. Materialien und Hilfsmittel findest du auf der letzten Seite.

Erstelle ein neues Node.js-Projekt und richte die grundlegende Ordnerstruktur ein. Implementiere einen einfachen Express-Server und integriere EJS als Templating-Engine.

Schritte:

1. **Projekt initialisieren:**
   - Führe `npm init` aus und installiere die benötigten Pakete (`express`, `ejs`,...).
   - Richte ein Git-Repository für eine Versionskontrolle ein (optional).(Heinweis: Gitignore Datei)
2. **Ordnerstruktur:**
   - Erstelle Ordner für `views`, `public` (für statische Dateien) und `routes` (optional).
3. **Express-Server aufsetzen:**
   - Erstelle eine `server.js` (oder `app.js`) Datei.
   - Implementiere einen einfachen Express-Server, der EJS als View-Engine verwendet und statische Dateien aus dem `public`-Verzeichnis bedient.
   - Lege die Route für die Startseite fest (z. B. GET `/`), die eine einfache EJS-View rendert.

Am Ende solltest du einen funktionierenden Express-Server haben, der eine EJS-Seite (z. B. "Willkommen zur Buchverwaltung!") ausgeliefert.

Integration der Datenbank mit Prisma

Binde Prisma in das Projekt ein und konfiguriere eine SQLite-Datenbank (oder eine andere relationale Datenbank, falls gewünscht, bzw vorhanden). Definiere ein einfaches Datenmodell (z. B. "Book" mit Feldern wie `id`, `title`, `author`, `publishedDate`).

Schritte:

1. **Installation und Setup von Prisma:**
   - Installiere Prisma (`npm install prisma`)
   - Führe `npx prisma init` aus, um die Konfigurationsdateien zu erstellen.

2. **Definiere das Datenmodell:**

Bearbeite die Datei `prisma/schema.prisma` und definiere ein Modell z. B.:

```
model Book {
  id            Int      @id @default(autoincrement())
  title         String
  author        String
  publishedDate DateTime
}
```

   -
3. **Migration und Datenbankerstellung:**
   - Führe `npx prisma db push , npx primsa generate` aus, um die Datenbank basierend auf dem Modell zu erstellen.
4. **Testen der Datenbankverbindung:**
   - Erstelle ein paar Dummydaten mit hilfe vom Prisma Studio `npx prisma studio`
   - Schreibe ein kleines Script, das mithilfe des Prisma-Clients einen Datensatz erstellt und abruft.

Implementierung der CRUD-Funktionalitäten

Erweitere deine Anwendung um Routen und Views, um die grundlegenden CRUD-Operationen (Erstellen, Lesen, Aktualisieren, Löschen) für das Datenmodell umzusetzen.

Schritte:

1. **Erstellen (Create):**
   - Implementiere eine Route (`GET /books/new`) zur Anzeige eines Formulars, um ein neues Buch anzulegen.
   - Implementiere die zugehörige POST-Route (`POST /books`) zum Abspeichern des neuen Eintrags in der Datenbank.
2. **Lesen (Read):**
   - Erstelle eine Route (`GET /books`), die eine Liste aller Bücher aus der Datenbank abruft und mittels einer EJS-View darstellt.
   - Implementiere weine Detailseite (`GET /books/:id`) für einzelne Datensätze.
3. **Aktualisieren (Update):**
   - Erstelle eine Route (`GET /books/:id/edit`), die ein Formular zur Bearbeitung eines Buchs anzeigt.

- Implementiere die zugehörige POST-/PUT-Route (`POST /books/:id` bzw. `PUT /books/:id`), um die Änderungen zu speichern.

  https://www.w3schools.com/Tags/ref_httpmethods.asp

4. **Löschen (Delete):**
   - Füge in der Listen- oder Detailansicht einen Button zum Löschen eines Buchs ein.
   - Implementiere die DELETE-Route (`DELETE /books/:id`).

     https://www.w3schools.com/Tags/ref_httpmethods.asp

5. **Integration in die Views:**
   - Nutze EJS, um dynamisch die Daten anzuzeigen und Formulare korrekt zu rendern.
   - Achte auf sinnvolles Error-Handling und Rückmeldungen an den Benutzer (z. B. Erfolgsmeldungen, Fehlermeldungen).

Die fertige CRUD-Anwendung ermöglicht es, Bücher anzulegen, aufzulisten, zu bearbeiten und zu löschen. Die Anwendung kommuniziert mit der Datenbank über Prisma und zeigt die Daten mithilfe von EJS an.

Erweiterungen & optionale Aufgaben

Diese folgenden Aufgaben dient zur Vertiefung und optionalen Erweiterung der bereits umgesetzten Funktionalitäten.

Optionale Erweiterungsaufgaben:

1. **Validierung und Error-Handling:**
   - Integriere serverseitige Validierung der Formulareingaben (z. B. mittels eines Validierungspakets oder eigener Logik).
   - Implementiere aussagekräftige Fehlermeldungen und nutze Express-Middleware zur Fehlerbehandlung.
2. **API-Endpunkte:**
   - Erstelle zusätzlich zu den klassischen HTML-Routen eine RESTful API, die JSON-Daten liefert (z. B. GET `/api/books`, POST `/api/books` etc.).
   - Dokumentiere kurz, wie diese Endpunkte konsumiert/genutzt werden können.
3. **Frontend-Erweiterungen:**
   - Integriere clientseitiges JavaScript, um beispielsweise Löschoperationen asynchron durchzuführen oder Formulare dynamisch zu validieren.
   - Optional: Nutze ein Frontend-Framework (z. B. jQuery) für einfache Interaktivität.
4. **Sicherheit:**
   - Implementiere grundlegende Sicherheitsmaßnahmen wie das Verhindern von SQL-Injections (Prisma hilft hier schon) oder Cross-Site-Scripting (XSS) in den EJS-Views.

5. **Backend:**
    - Implementiere ein Backend. Es soll möglich sein, sich einzuloggen. Nur wenn der Benutzer eingeloggt ist, kann ein Buch hinzugefügt oder gelöscht werden.

Hinweis:
Ihr könnt hier je nach Interesse und Zeit eigenständig entscheiden, welche Erweiterungen Ihr implementieren möchtet. Auch wenn nicht alle optionalen Aufgaben vollständig umgesetzt werden, zählt/hilft die Auseinandersetzung mit den Themen.

Abschluss, Testing und Reflexion

Testet eure Anwendung gründlich, behebt gefundene Fehler und bereitet eine kurze Präsentation bzw. einen schriftlichen Bericht vor, in dem ihr folgende Punkte behandelt:

- **Herausforderungen und Lösungen:**
    - Welche Schwierigkeiten gab es bei der Implementierung der einzelnen Komponenten?
    - Wie habt ihr diese Probleme gelöst?
- **Reflexion:**
    - Welche neuen Erkenntnisse habt ihr gewonnen?
    - Wo seht ihr noch Verbesserungsbedarf?
- **Demo:**
    - Führt eine kurze Demonstration eurer Anwendung durch

Materialien und Hilfsmittel

- **Dokumentation:**
    - Offizielle Dokumentationen von Express, EJS, Prisma
- **Beispielprojekte:**
    - Im GitRepo: https://github.com/ChristopherSchwerdt/FIAE/tree/main/Unterrichtsmaterialien/Web/NodeJS%20Backend
- **Tools:**
    - Code-Editor (VS Code,WebStorm,...)
    - Node.js und NPM installiert
    - Git (optional, aber empfohlen)
    - Browser für Tests