BookHub Project Documentation

Overview

The **BookHub** project is a Node.js application that allows managing a collection of books and authors. It uses MongoDB to store data, with features such as author and book CRUD operations, pagination for listing authors and books, and relationships between books and their authors.

Technologies Used

- **Node.js**: Backend runtime environment
- **Express**: Web framework for routing and middleware
- **MongoDB**: NoSQL database for storing authors and books
- **Mongoose**: ODM (Object Document Mapping) for interacting with MongoDB
- **ES6+ JavaScript**: Modern JavaScript for implementing logic

---

Project Structure

```
BookHub/
├── db/
│   └── models/
│       ├── author.model.js
│       └── book.model.js
├── src/
│   ├── modules/
│   │   ├── author/
│   │   │   ├── author.controller.js
│   │   │   └── author.routes.js
│   │   ├── book/
│   │   │   ├── book.controller.js
│   │   │   └── book.routes.js
├── app.js
└── db/
    └── connection.js
```

How It Works

1. Database Connection (`connection.js`)

The database connection is established using Mongoose, connecting to a local MongoDB instance.

```
JS connection.js ×
db > JS connection.js > ...
1   import mongoose from 'mongoose';
2   import dotenv from 'dotenv';
3
4   dotenv.config(); // Load environment variables
5
    Codeium: Refactor | Explain | Generate JSDoc | ×
6   const connectiontDB = async () => {
7       try {
8           const uri = process.env.MONGO_URI || 'mongodb://localhost:27017/bookhub'; // Default to local if env var is not set
9
10          if (!uri) {
11              throw new Error('MongoDB URI is undefined');
12          }
13
14          await mongoose.connect(uri, { useNewUrlParser: true, useUnifiedTopology: true });
15          console.log("Database connected successfully");
16      } catch (error) {
17          console.error("Database connection failed:", error);
18      }
19  };
20
21  export default connectiontDB;
22
```

- **Mongoose connection**: Connects to MongoDB at `localhost:27017/collectionOfBooks`.
- **Error handling**: Logs any errors that occur during connection.

---

2. Models

2.1 Author Model (`author.model.js`)

The Author model defines the structure of the author document in MongoDB.

- **Fields**: The `Author` schema includes a name, biography (`bio`), birthdate (`birthDate`), and an array of books (referencing the `Book` model).
- **Timestamps**: Automatically generates `createdAt` and `updatedAt` fields.

2.2 Book Model (`book.model.js`)

The Book model defines the structure of the book document in MongoDB.

- **Fields**: The `Book` schema includes a title, content, a reference to an `Author`, and the published date.
- **Reference**: The `author` field stores the `ObjectId` of the associated `Author`.

---

3. Controllers

3.1 Author Controller (`author.controller.js`)

The controller handles the CRUD operations for authors, including pagination.

JS author.controller.js ×

src > modules > auhor > JS author.controller.js > ...

```
72
73  // Pagination request for authors
    Codeium: Refactor | Explain | ×
74  export const pagination = async (req, res, next) => {
75      try {
76          // Destructure the page and limit from the query string, and set default values
77          const { page = 1, limit = 10 } = req.query;
78
79          const pageNumber = parseInt(page, 10);
80          const limitNumber = parseInt(limit, 10);
81
82          // Query MongoDB to get authors, limit the results and skip based on the page
83          const authors = await Author.find({})
84              .limit(limitNumber) // Limit the number of results per page
85              .skip((pageNumber - 1) * limitNumber)
86              .sort({ createdAt: -1 }); // Sort by creation date in descending order (newest first)
87
88          // Count the total number of authors in the database
89          const count = await Author.countDocuments();
90
91          // Calculate the total number of pages based on the count of authors and limit
92          return res.status(200).json({
93              authors, // The paginated list of authors
94              totalPages: Math.ceil(count / limitNumber),
95              currentPage: pageNumber,
96          });
97      } catch (err) {
98          console.error(err);
99          res.status(500).json({ error: 'Internal Server Error' });
100     }
101 };
102
103
```

- **Pagination Logic**: The `pagination` function extracts the `page` and `limit` from the request query, retrieves authors from MongoDB, and calculates the total pages.
    - `limit`: The number of authors per page.
    - `skip`: Skips the authors from previous pages based on the current page number.

3.2 Book Controller (`book.controller.js`)

The controller for book-related CRUD operations.

JS book.controller.js ×

src > modules > book > JS book.controller.js > ...

```
// Importing the Book and Author models for database operations
import Book from "../../../db/models/book.model.js";
import Author from "../../../db/models/author.model.js";

// POST request to create a new book
Codeium: Refactor | Explain | ×
export const createBook = async (req, res) => {
    const { title, content, authorId, publishedDate } = req.body;
    try {
        const author = await Author.findById(authorId);
        if (!author) {
            return res.status(404).json({ msg: "Author not found" });
        }

        const newBook = await Book.create({ title, content, author: authorId, publishedDate });
        author.books.push(newBook._id);
        await author.save();

        res.status(201).json(newBook);
    } catch (error) {
        console.error(error);
        res.status(500).json({ error: 'Internal Server Error' });
    }
};
```

- **Creating Books**: The `createBook` function ensures that the provided `authorId` exists, creates the book, and adds its reference to the author's `books` array.

---

4. Routes

4.1 Author Routes (`author.routes.js`)

Defines the routes for author-related operations.

JS author.routes.js ×

src > modules > auhor > JS author.routes.js > ...

```
1   import { Router } from "express";
2   const router = Router();
3   import * as BC from "./author.controller.js"; // Importing controller functions
4
5   // Pagination: Fetch authors with pagination support
6   router.get("/pagination", BC.pagination);
7
8   // Get all authors
9   router.get("/getAuthors", BC.getAuthors);
10
11  // Create a new author
12  router.post("/createAuthor", BC.createAuthor);
13
14  // Get author by ID (includes books)
15  router.get("/:id", BC.getAuthorByID);
16
17  // Delete an author by ID
18  router.delete("/:id", BC.deleteAuthor);
19
20  // Update an author's details by ID
21  router.patch("/:id", BC.updateAuthor);
22
23  export default router;
24
```

- **Pagination Route**: `GET /Author/pagination` calls the `pagination` function to retrieve authors with pagination.
- **CRUD Routes**: Routes for creating, retrieving, updating, and deleting authors.

4.2 Book Routes (`book.routes.js`)

Defines the routes for book-related operations.

JS book.routes.js ×

src > modules > book > JS book.routes.js > ...

```
1   import { Router } from "express";
2   const router = Router();
3   import * as BC from "./book.controller.js"; // Importing controller functions
4
5   // Pagination: Fetch books with pagination support
6   router.get("/pagination", BC.pagination);
7
8   // Get all books
9   router.get("/getBook", BC.getBook);
10
11  // Create a new book
12  router.post("/createBook", BC.createBook);
13
14  // Get a book by ID
15  router.get("/:id", BC.getBookByID);
16
17  // Delete a book by ID
18  router.delete("/:id", BC.DeleteBook);
19
20  // Update a book's details by ID
21  router.patch("/:id", BC.updateBook);
22
23  export default router;
24
```

- **Pagination Route**: `GET /Book/pagination` retrieves books with pagination.
- **CRUD Routes**: Routes for creating, retrieving, updating, and deleting books.

---

5. Main Application (`index.js`)

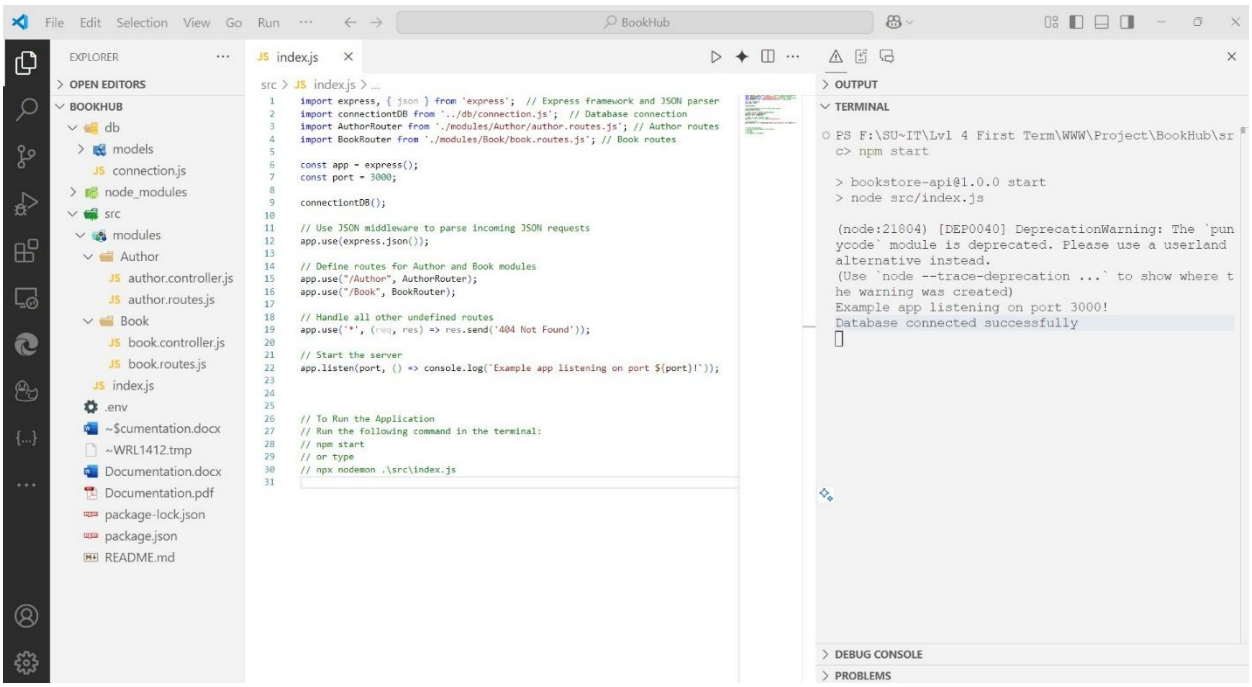
The entry point for the application that configures the Express server and routes.

JS index.js

src > JS index.js > ...

```
1   import express, { json } from 'express';  // Express framework and JSON parser
2   import connectiontDB from '../db/connection.js';  // Database connection
3   import AuthorRouter from './modules/Author/author.routes.js'; // Author routes
4   import BookRouter from './modules/Book/book.routes.js'; // Book routes
5
6   const app = express();
7   const port = 3000;
8
9   connectiontDB();
10
11  // Use JSON middleware to parse incoming JSON requests
12  app.use(express.json());
13
14  // Define routes for Author and Book modules
15  app.use("/Author", AuthorRouter);
16  app.use("/Book", BookRouter);
17
18  // Handle all other undefined routes
19  app.use('*', (req, res) => res.send('404 Not Found'));
20
21  // Start the server
22  app.listen(port, () => console.log(`Example app listening on port ${port}!`));
23
24
25
26  // To Run the Application
27  // Run the following command in the terminal:
28  // npm start
29  // or type
30  // npx nodemon .\src\index.js
31
```

- **Database Connection**: Calls `connectiontDB()` to connect to the MongoDB database.
- **Middleware**: Uses `express.json()` to parse incoming JSON requests.
- **Routes**: Mounts author and book routes for handling requests.



Conclusion

The **BookHub** project provides a complete solution for managing authors and books, with features such as CRUD operations and pagination.

Supervisors:

Dr. Hossam Eldin Aladly – Eng. Akram Medhat

Team Members:

- **Mahmoud Atef Abdelaziz**
- **Nourhan Yasser Fathy**
- **Nada Ahmed Ibrahim**