Library Management System

1. Project purpose & overview

This project is a **RESTful backend API** for a simple Library Management System. It provides:

- User authentication (signup + login) using **bcrypt** and **JWT**
- Protected CRUD routes for **books** (create, read, update, delete)
- Input validation with **Joi**
- Organized code using the **MVC** pattern (models, controllers, routes, middleware)
- Connection to **MongoDB** via **Mongoose**

2. How requests flow (high-level)

1. A client (Postman, frontend app) sends an HTTP request to the server (Express).
2. Express matches the request to a **route** (e.g., `POST /api/auth/login`).
3. The route calls a **controller** function (business logic).
4. Controller may validate input with a **Joi schema** and interact with **Mongoose models** (read/write DB).
5. If the route is protected, **auth middleware** checks the JWT token and attaches `req.user`.
6. Controller returns a JSON response to the client.

This clear separation makes the app easier to test and maintain.

3. Folder structure and responsibilities

library-backend/
├─ package.json

```
├── .env
├── server.js              # App entry — sets up middleware, routes, DB
├── config/
│   └── db.js              # MongoDB connection
├── models/
│   ├── User.js            # User schema and methods
│   └── Book.js            # Book schema
├── controllers/
│   ├── authController.js  # Signup & login logic
│   └── bookController.js  # CRUD for books
├── routes/
│   ├── auth.js            # /api/auth routes
│   └── books.js           # /api/books routes (protected)
├── middleware/
│   ├── auth.js            # JWT check (protect)
│   └── errorHandler.js    # Central error handler
└── validators/
    ├── authSchemas.js     # Joi schemas for auth
    └── bookSchemas.js     # Joi schemas for books
```

Why this structure?

- `models/` -> data layer (DB logic)
- `controllers/` -> application logic and DB calls
- `routes/` -> maps HTTP endpoints to controllers
- `middleware/` -> reusable request handlers (auth, errors)
- `validators/` -> validation keeps controllers clean

4. Important files explained

`server.js`

- Loads environment variables and middleware (`express.json()`, `cors`).
- Connects to MongoDB using `connectDB(process.env.MONGO_URI)`.

- Mounts the route groups `/api/auth` and `/api/books`.
- Adds the error handler last so it catches controller errors.

Why last? Express executes middleware in order — the error handler must come after routes to catch thrown errors.

`config/db.js`

- Wraps `mongoose.connect()` in a try/catch.
- Keeps connection logic separate so `server.js` stays clean.

`models/User.js`

- Mongoose schema defines the fields: `name`, `email`, `password`.
- `pre('save')` middleware hashes the password with `bcrypt` before storing.
- `userSchema.methods.matchPassword()` compares a plain password to the hash.

Why hash passwords? Never store raw passwords — hashing prevents exposure if DB leaks.

`models/Book.js`

- Simple schema for book properties: `title`, `author`, `isbn`, `publishedDate`, `copiesAvailable`.
- All CRUD operations use this model via `Book.find()`, `Book.create()`, etc.

`validators/*.js` (Joi)

- Each file exports Joi schemas used to validate incoming JSON bodies.
- Controllers call `schema.validate(req.body)` and return a `400` with the Joi error message if validation fails.

Why Joi? Joi centralizes validation logic and produces friendly error messages.

`middleware/auth.js`

- Looks for `Authorization: Bearer <token>` header.
- Verifies token using `jwt.verify(token, process.env.JWT_SECRET)`.
- Fetches the user from DB and attaches `req.user` — downstream controllers can use `req.user`.

This middleware protects all book routes by requiring a valid token.

`controllers/authController.js`

- `signup`: validate input, check duplicate email, create user (password hashed automatically), generate JWT, return token + user info.
- `login`: validate input, find user by email, compare password, generate JWT, return token + user info.

JWT payload is small: `{ id: user._id }` so we can retrieve full user from DB when needed.

`controllers/bookController.js`

- `createBook`: validate request, `Book.create(req.body)` and return the created book.
- `getBooks`: `Book.find()` returns all books.
- `getBook`: `Book.findById(id)`.
- `updateBook`: `Book.findByIdAndUpdate(id, req.body, { new: true })`.
- `deleteBook`: `Book.findByIdAndDelete(id)`.

All routes are protected by `protect` middleware to ensure only authenticated users can modify books.

5. Authentication flow (JWT) — step by step

1. **Signup** (`POST /api/auth/signup`):
    - Client sends `{ name, email, password }`.
    - Server creates user and responds with a token.
2. **Login** (`POST /api/auth/login`):
    - Client sends `{ email, password }`.
    - Server verifies credentials and returns a token.
3. **Access protected** routes:
    - Client includes header `Authorization: Bearer <token>`.
    - Server verifies token, looks up user, and allows request to continue.

Tokens are stateless — server only needs the secret to validate.

6. How to run locally (quick start)

Install dependencies:
npm install
Create `.env` with values (PORT, MONGO_URI, JWT_SECRET, JWT_EXPIRES_IN).
Start MongoDB Compass. Set `MONGO_URI` accordingly.
Run the app:
npm run dev
Use Postman to test the endpoints described later in this doc.

7. Example HTTP requests (Postman)

Signup

POST /api/auth/signup
Content-Type: application/json

```
{
  "name": "Aline",
  "email": "aline@gmail.com",
  "password": "mypassword"
```

}

Login

POST /api/auth/login
Content-Type: application/json

```
{
  "email": "aline@gmail.com",
  "password": "mypassword"
}
```

Response includes `token`.

Create Book (authorized)

POST /api/books
Headers: Authorization: Bearer <token>
Content-Type: application/json

```
{
  "title": "The Great Gatsby",
  "author": "F. Scott Fitzgerald",
  "isbn": "9780743273565",
  "publishedDate": "1925-04-10",
  "copiesAvailable": 5,
  "description": "A novel about the American dream."
}
```

Get Books

GET /api/books
Headers: Authorization: Bearer <token>