Developing & Testing REST API

Prof. P. M. Jadav
Associate Professor
Computer Engineering Department
Faculty of Technology
Dharmsinh Desai University, Nadiad

./book_api.js

```
const express = require("express");
const mongoose = require("mongoose");
const bookRoutes = require("./routes/bookRoutes"); // Import routes
const app = express();
app.use(express.json()); // middleware to parse JSON
// Connect to MongoDB and Create a database 'LibDB' if it doesn't exist
mongoose.connect('mongodb://127.0.0.1:27017/LibDB')
  .then(() => console.log('MongoDB connected successfully!'))
  .catch(err => console.error('MongoDB connection error:', err));
// --- Routes ---
app.use("/books", bookRoutes);
app.listen(3000, () => console.log(`Server running`));
```

./models/Book.js (Without Validation)

```
const mongoose = require("mongoose");
const bookSchema = new mongoose.Schema({
  title: String,
  author: String,
  genres: [String],
  year: Number,
  price: Number,
  stock: Number,
  ratings: [Number]
});
module.exports = mongoose.model("Book", bookSchema);
```

./models/Book2.js (With Validation)

```
const bookSchema = new mongoose.Schema({
  title: {
    type: String,
    required: [true, "Title is required"],
    minlength: [3, "Title must be at least 3 characters long"]
  },
  author: {
    type: String,
    required: [true, "Author is required"]
  },
  genres: {
    type: [String],
    validate: {
      validator: (arr) => arr.length > 0,
      message: "At least one genre is required"
    }  },
```

./models/Book2.js (With Validation)

```
year: {
  type: Number,
  min: [1500, "Year must be later than 1500"],
  max: [new Date().getFullYear(), "Year cannot be in the future"]
},
price: {
  type: Number,
  required: [true, "Price is required"],
  min: [0, "Price cannot be negative"]
},
stock: {
  type: Number,
  min: [0, "Stock cannot be negative"],
  default: 0  },
```

./models/Book2.js (With Validation)

```
  ratings: {
    type: [Number],
    validate: {
      validator: (arr) => arr.every(num => num >= 1 && num <= 5),
      message: "Ratings must be between 1 and 5"
    }
  }
});
```

./routes/bookRoutes.js (POST Request)

```
const express = require("express");
const Book = require("../models/Book");  // Import model
const router = express.Router();
router.post("/", async (req, res) => {// Create a book
  try {
    const book = new Book(req.body);
    await book.save();
    res.status(201).json(book); // 201 - Created
  } catch (err) {
    res.status(400).json({ error: err.message }); // 400 - Bad Request
  }
});
```

./routes/bookRoutes.js (GET Request)

```
// Get all books
router.get("/", async (req, res) => {
  const books = await Book.find();
  res.json(books);
});
```

./routes/bookRoutes.js (GET Request)

```
router.get("/:id", async (req, res) => {// Get a book by ID
  try {
    const book = await Book.findById(req.params.id);
    if (!book)
        return res.status(404).json({ error: "Book not found"
 });
    res.json(book);
  }
  catch (err) {
    res.status(400).json({ error: "Invalid ID" });
  }
});
```

./routes/bookRoutes.js (PUT Request)

```
// Update a book
router.put("/:id", async (req, res) => {
  try {
const book = await Book.findByIdAndUpdate(req.params.id, req.body, { new: true });
    if (!book)
        return res.status(404).json({ error: "Book not found" });
    res.json(book);
  }
  catch (err) {
    res.status(400).json({ error: "Invalid ID" }); // 400 - Bad Request
  }
});
```

./routes/bookRoutes.js (DELETE Request)

```
router.delete("/:id", async (req, res) => {// Delete a book
  try {
    const book = await Book.findByIdAndDelete(req.params.id);
    if (!book)
        return res.status(404).json({ error: "Book not found" });
    res.json({ message: "Book deleted successfully" });
  }
  catch (err) {
    res.status(400).json({ error: "Invalid ID" });
  }
});
module.exports = router;
```

Testing POST Endpoint

```
### Create a new book
POST http://localhost:3000/books
Content-Type: application/json

{
  "title": "Node.js in Action",
  "author": "Mike Cantelon",
  "genres": ["Programming", "Node.js"],
  "year": 2017,
  "price": 39.99,
  "stock": 15,
  "ratings": [5, 4, 5, 5, 4]
}
```

Testing GET Endpoint

```
### Get all Books
GET http://localhost:3000/books

### Get a book by Id
GET http://localhost:3000/books/68755d745c36694526e100e2
```

Testing PUT Endpoint

```
### Update a book
PUT http://localhost:3000/books/68755d745c36694526e100e3
Content-Type: application/json

{
  "price": 39.99,
  "stock": 10
}
```

Testing DELETE Endpoint

```
### Delete a Book

DELETE http://localhost:3000/books/68a5f28d7b9845c406d9d62a
```