DTS NodeJS Capstone Assignment

Name: Atharva Athanikar 72287317E

Batch: DTS Batch-3

Server.cjs:

const express = require('express');

const mongoose = require('mongoose');

const bodyParser = require('body-parser');

const Book = require('./models/Book.cjs');

const User = require('./models/User.cjs');

const userRouter = require('./routes/users.cjs');

const authRoutes = require('./routes/auth.cjs');

const bookRoutes = require('./routes/books.cjs');

const borrowRoutes = require('./routes/borrow.cjs');

const app = express();

app.use(bodyParser.json());

const mongoURI = 'mongodb://127.0.0.1:27017/library-management';

mongoose.connect(mongoURI, { useNewUrlParser: true, useUnifiedTopology: true })

    .then(() => {

        console.log('MongoDB connected');

        seedDatabase();

    })

    .catch((err) => console.log('Error connecting to MongoDB:', err));

const PORT = process.env.PORT || 7000;

app.get('/', (\_req, res) => {

    res.send('Library Management System Backend');

});

app.use('/api/auth', authRoutes);

app.use('/api', bookRoutes);

app.use('/api', borrowRoutes);

app.use('/api', userRouter);

async function seedDatabase() {

    try {

        const userExists = await User.countDocuments();

        if (userExists === 0) {

            await new User({ name: 'Admin User', username: 'admin', password: 'admin123', email: 'admin@library.com', mobile: 1234567890, admin: true }).save();

            await new User({ name: 'Atharva Athanikar', username: 'Atharva', password: 'atharva2003', email: 'aa@gmail.com', mobile: 9123456780, admin: false }).save();

        }

        const bookExists = await Book.countDocuments();

        if (bookExists === 0) {

            await new Book({ name: 'Wings of fire', author: 'APJ Abdul kalam', genre: 'Motivation', type: 'Book', available: true }).save();

            await new Book({ name: '1984', author: 'George Orwell', genre: 'Dystopian', type: 'Novel', available: true }).save();

            await new Book({ name: 'To Kill a Mockingbird', author: 'Harper Lee', genre: 'Fiction', type: 'Novel', available: true }).save();

            await new Book({ name: 'The Catcher in the Rye', author: 'J.D. Salinger', genre: 'Fiction', type: 'Novel', available: true }).save();

            await new Book({ name: 'The Great Gatsby', author: 'F. Scott Fitzgerald', genre: 'Fiction', type: 'Novel', available: true }).save();

        }

    } catch (err) {

        console.log('Error seeding database:', err);

    }

}

app.listen(PORT, () => {

    console.log(`Server is running on port ${PORT}`);

});

Models:

Book.cjs

const mongoose = require('mongoose');

const bookSchema = new mongoose.Schema({

  name: { type: String, required: true },

  author: { type: String, required: true },

  genre: { type: String, required: true },

  type: { type: String, required: true },

  available: { type: Boolean, default: true }

}, { timestamps: true });

module.exports = mongoose.model('Book', bookSchema);

Borrow.cjs

const mongoose = require('mongoose');

const borrowSchema = new mongoose.Schema({

  username: { type: String, required: true },

  bookId: { type: mongoose.Schema.Types.ObjectId, ref: 'Book', required: true },

  dueDate: { type: Date, required: true }

}, { timestamps: true });

module.exports = mongoose.model('Borrow', borrowSchema);

Return.cjs

const mongoose = require('mongoose');

const returnSchema = new mongoose.Schema(

  {

    username: { type: String, required: true },

    bookid: { type: mongoose.Schema.Types.ObjectId, ref: 'Book', required: true },

    duedate: { type: Date, ref: 'Borrow', required: true },

    fine: { type: Number, default: 0 }

  },

  { timestamps: true }

);

module.exports = mongoose.model('Return', returnSchema);

User.cjs

const mongoose = require('mongoose');

const userSchema = new mongoose.Schema({

  name: { type: String, required: true },

  username: { type: String, required: true, unique: true },

  password: { type: String, required: true },

  email: { type: String, required: true, unique: true },

  mobile: { type: Number, required: true, unique: true },

  admin: { type: Boolean, default: false }

});

module.exports = mongoose.model('User', userSchema);

Routes:

auth.cjs

const express = require('express');

const bcrypt = require('bcryptjs');

const jwt = require('jsonwebtoken');

const User = require('../models/User.cjs');

const router = express.Router();

router.post('/register', async (req, res) => {

  const { name, username, password, email, mobile, admin } = req.body;

  try {

    const userExists = await User.findOne({ username });

    if (userExists) return res.status(400).json({ msg: 'Username already exists' });

    const salt = await bcrypt.genSalt(10);

    const hashedPassword = await bcrypt.hash(password, salt);

    const newUser = new User({ name, username, password: hashedPassword, email, mobile, admin });

    await newUser.save();

    res.status(201).json({ msg: 'User registered successfully' });

  } catch (err) {

    res.status(500).json({ msg: 'Server Error' });

  }

});

router.post('/login', async (req, res) => {

  const { username, password } = req.body;

  try {

    const user = await User.findOne({ username });

    if (!user) return res.status(400).json({ msg: 'Invalid credentials' });

    const isMatch = await bcrypt.compare(password, user.password);

    if (!isMatch) return res.status(400).json({ msg: 'Invalid credentials' });

    const token = jwt.sign({ userId: user.\_id }, 'secret', { expiresIn: '1h' });

    res.json({ token });

  } catch (err) {

    res.status(500).json({ msg: 'Server Error' });

  }

});

module.exports = router;

books.cjs

const express = require('express');

const Book = require('../models/Book.cjs');

const router = express.Router();

router.get('/books', async (req, res) => {

  try {

    const books = await Book.find({ available: true });

    res.json(books);

  } catch (err) {

    res.status(500).json({ msg: 'Server Error', err });

  }

});

router.post('/books', async (req, res) => {

  const { name, author, genre, type } = req.body;

  try {

    const newBook = new Book({ name, author, genre, type });

    await newBook.save();

    res.status(201).json({ msg: 'Book created successfully' });

  } catch (err) {

    res.status(500).json({ msg: 'Server Error', err });

  }

});

router.put('/books/:id', async (req, res) => {

  try {

    const { id } = req.params;

    const { name, author, genre, type, available, admin } = req.body;

    if (!admin) return res.status(403).json({ msg: 'Access denied. Admin privileges required.' });

    const book = await Book.findByIdAndUpdate(id, { name, author, genre, type, available }, { new: true, runValidators: true });

    if (!book) return res.status(404).json({ msg: 'Book not found' });

    res.json({ msg: 'Book updated successfully', book });

  } catch (error) {

    res.status(500).json({ msg: 'Server error' });

  }

});

module.exports = router;

borrow.cjs

const express = require('express');

const Borrow = require('../models/Borrow.cjs');

const Book = require('../models/Book.cjs');

const User = require('../models/User.cjs');

const router = express.Router();

router.post('/borrow', async (req, res) => {

  try {

    const { username, bookId, dueDate } = req.body;

    const user = await User.findOne({ username });

    if (!user) return res.status(400).json({ msg: 'User not found' });

    const book = await Book.findById(bookId);

    if (!book) return res.status(400).json({ msg: 'Book not found' });

    if (!book.available) return res.status(400).json({ msg: 'Book not available' });

    const borrow = new Borrow({ username, bookId, dueDate });

    await borrow.save();

    book.available = false;

    await book.save();

    res.json({ msg: 'Book borrowed successfully', borrow });

  } catch (error) {

    res.status(500).json({ msg: 'Server error' });

  }

});

router.post('/return', async (req, res) => {

  try {

    const { username, bookId } = req.body;

    const borrowRecord = await Borrow.findOne({ username, bookId });

    if (!borrowRecord) return res.status(404).json({ msg: 'Borrow record not found' });

    const book = await Book.findById(bookId);

    if (book) {

      book.available = true;

      await book.save();

    }

    await Borrow.deleteOne({ \_id: borrowRecord.\_id });

    res.json({ msg: 'Book returned successfully' });

  } catch (error) {

    res.status(500).json({ msg: 'Server error' });

  }

});

module.exports = router;

users.cjs

const express = require('express');

const User = require('../models/User.cjs');

const router = express.Router();

router.get('/users', async (req, res) => {

  try {

    const users = await User.find();

    res.json(users);

  } catch (error) {

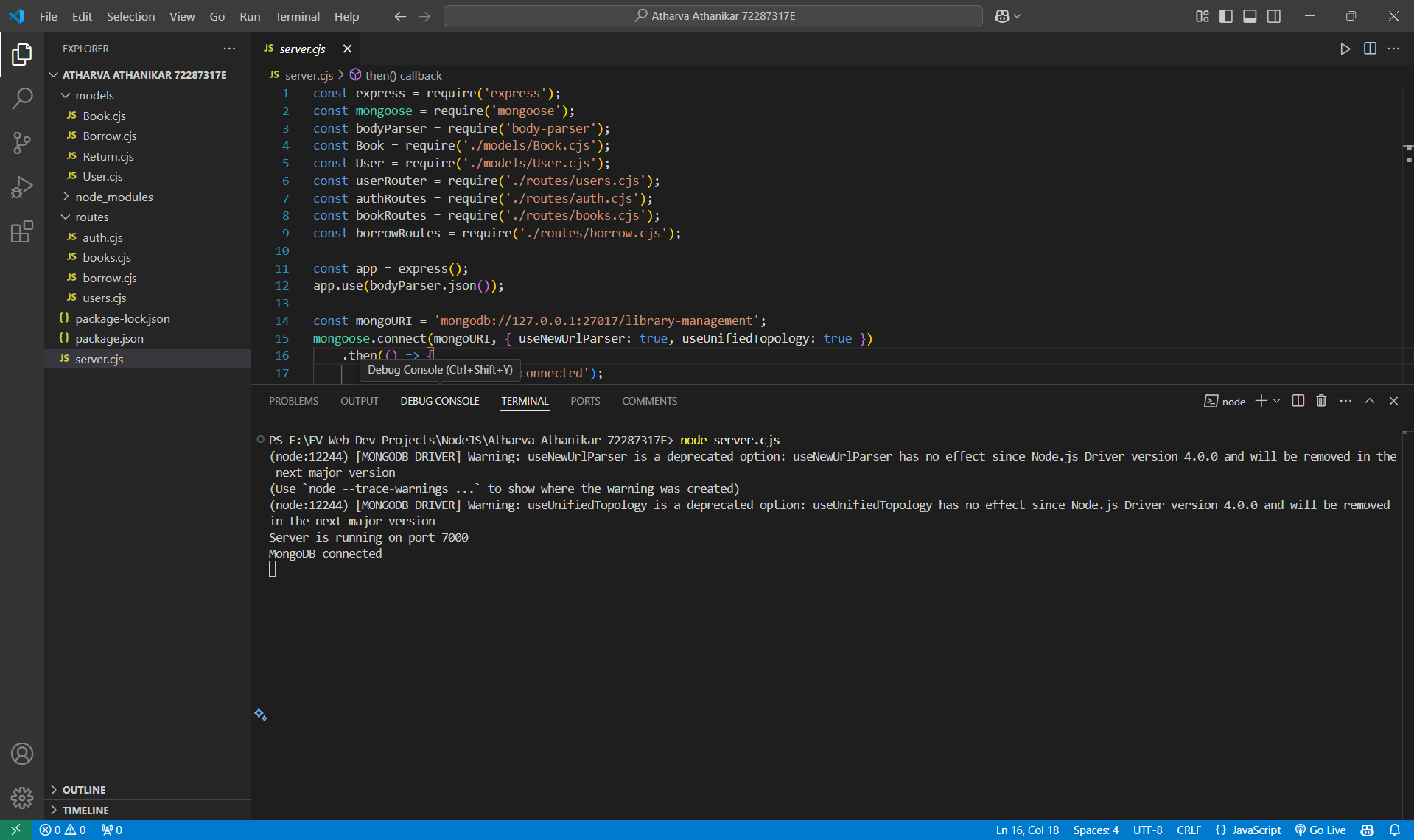
    res.status(500).json({ msg: 'Server error' });

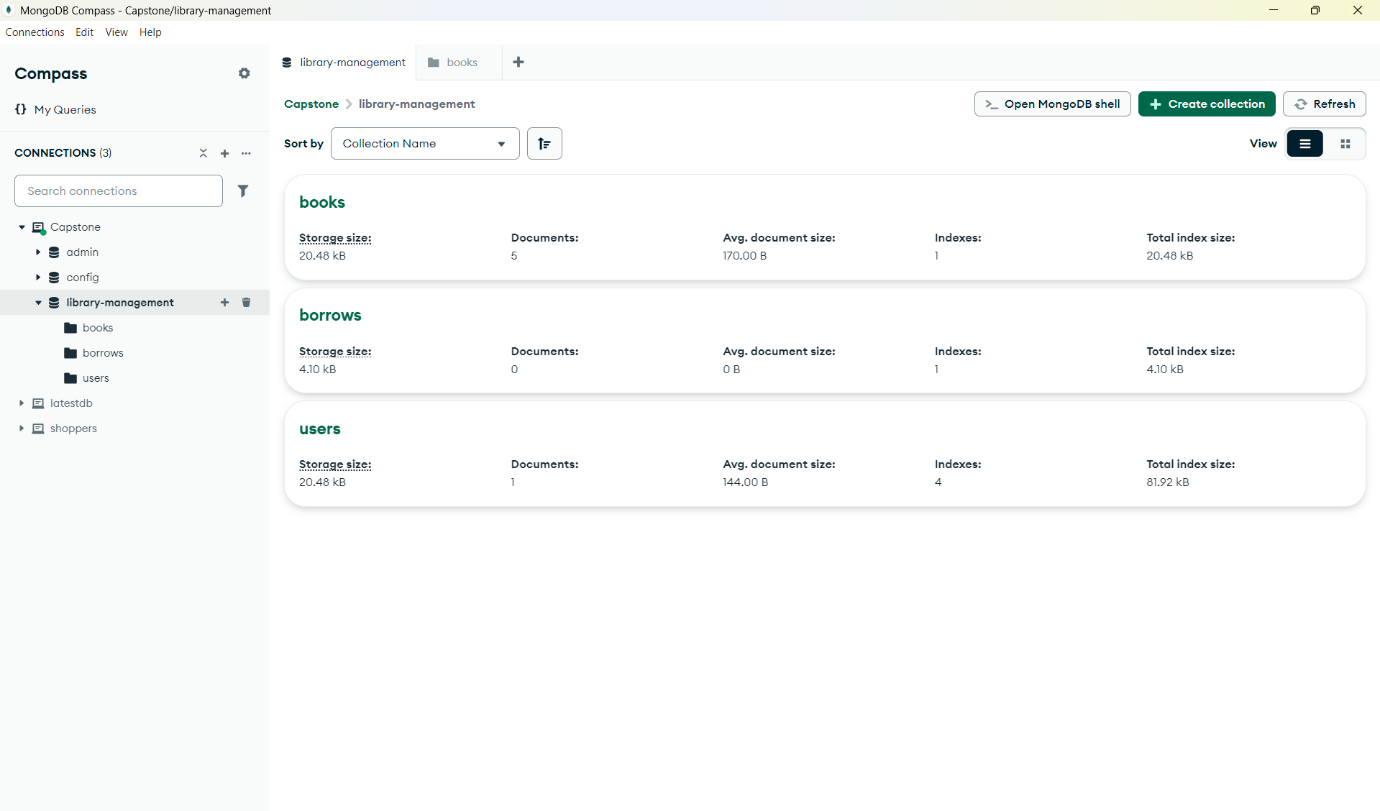
  }

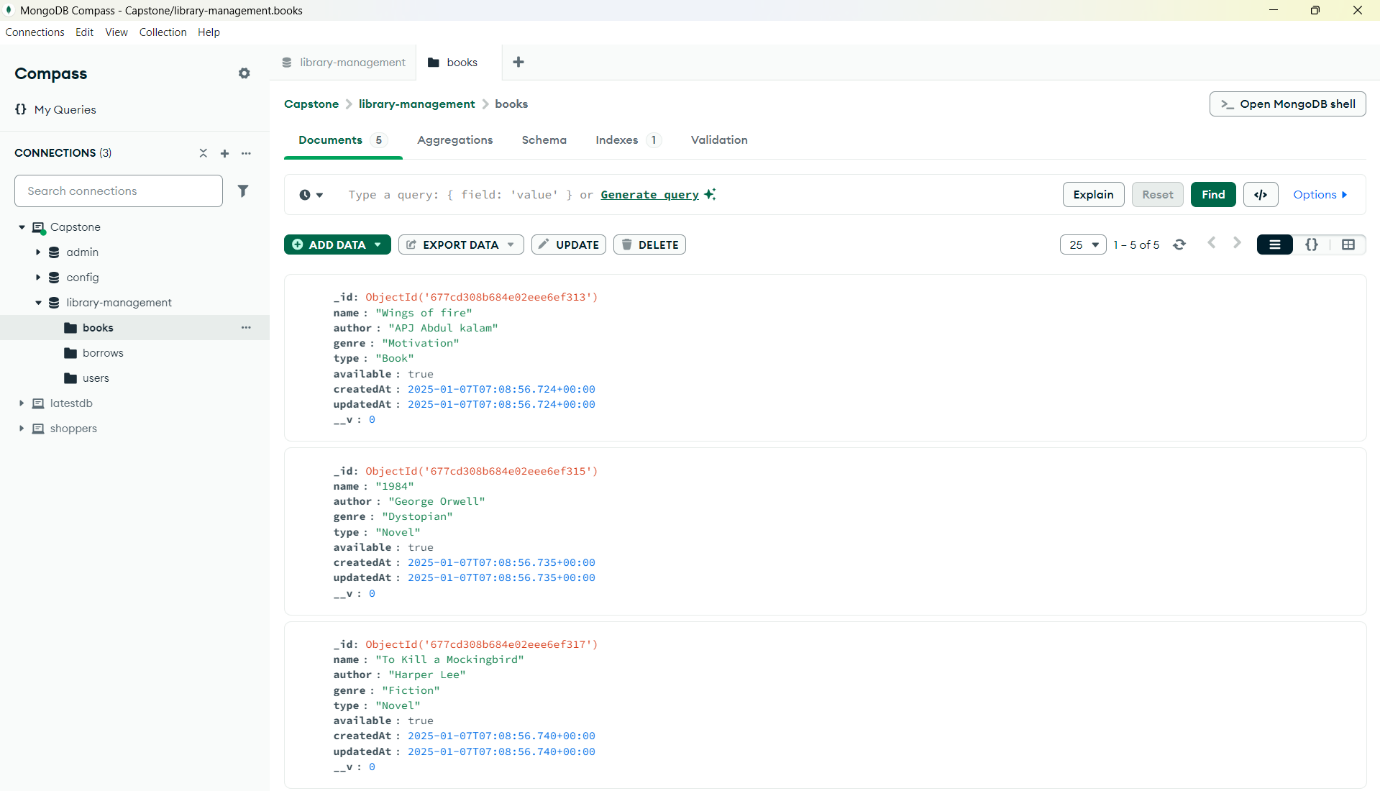
});

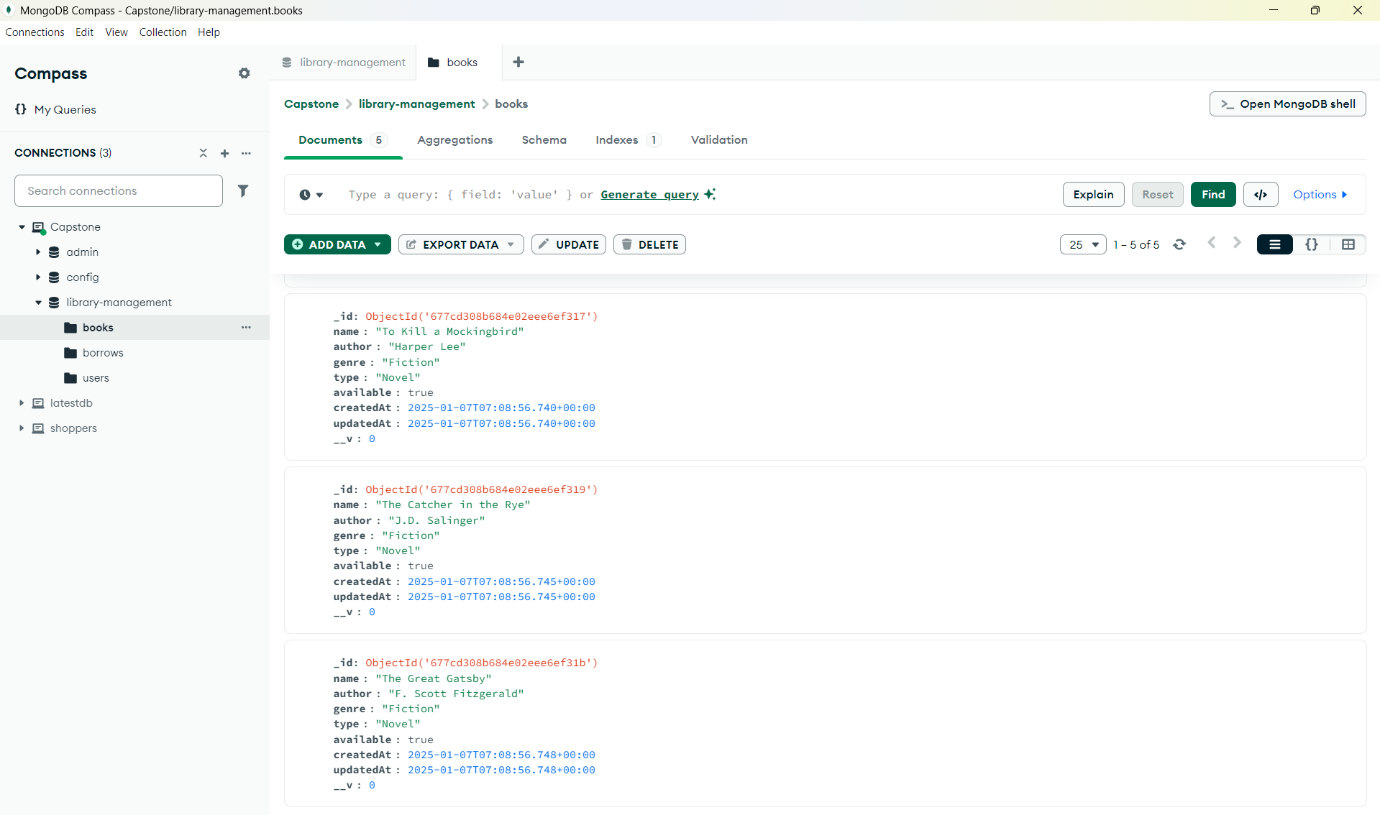
module.exports = router;

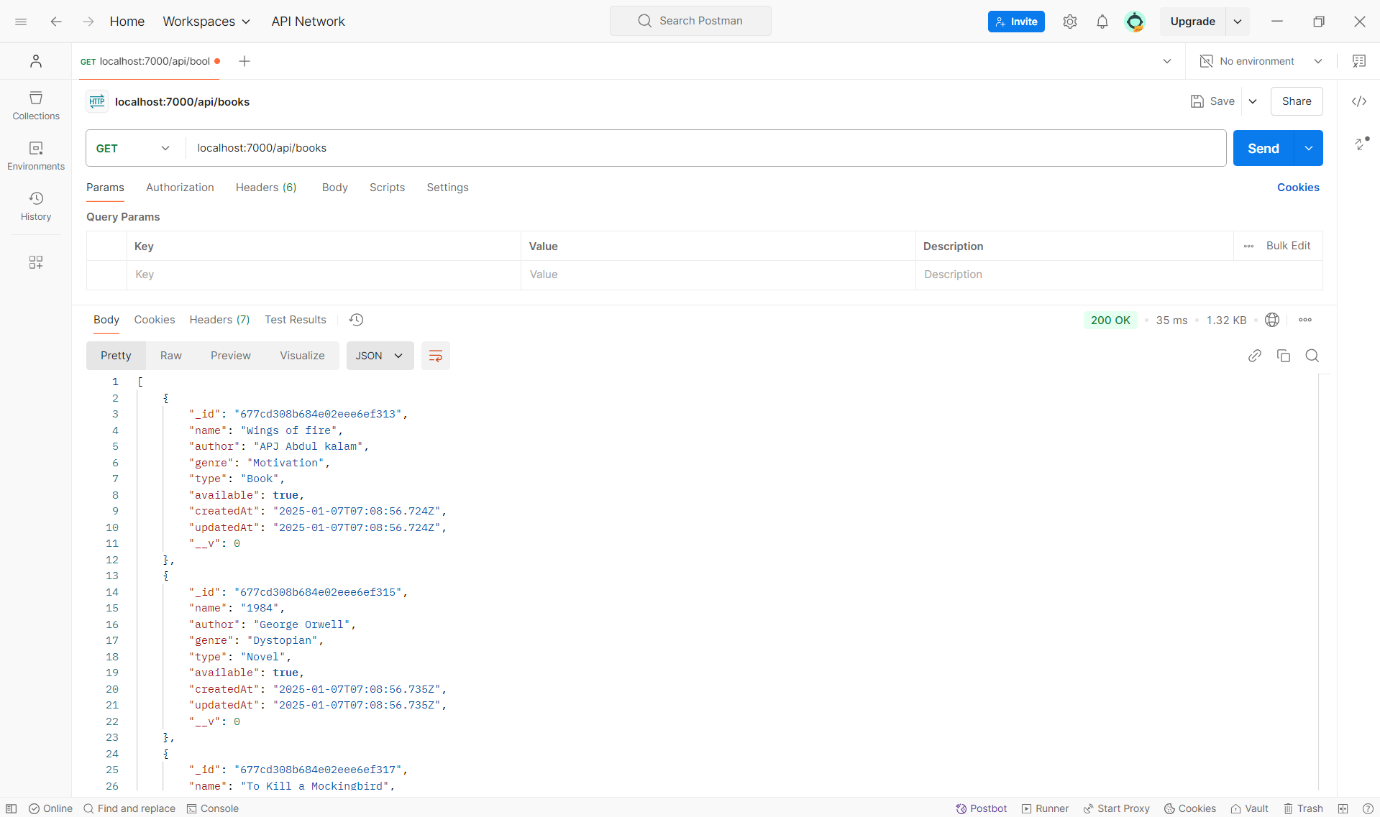
OUTPUT:

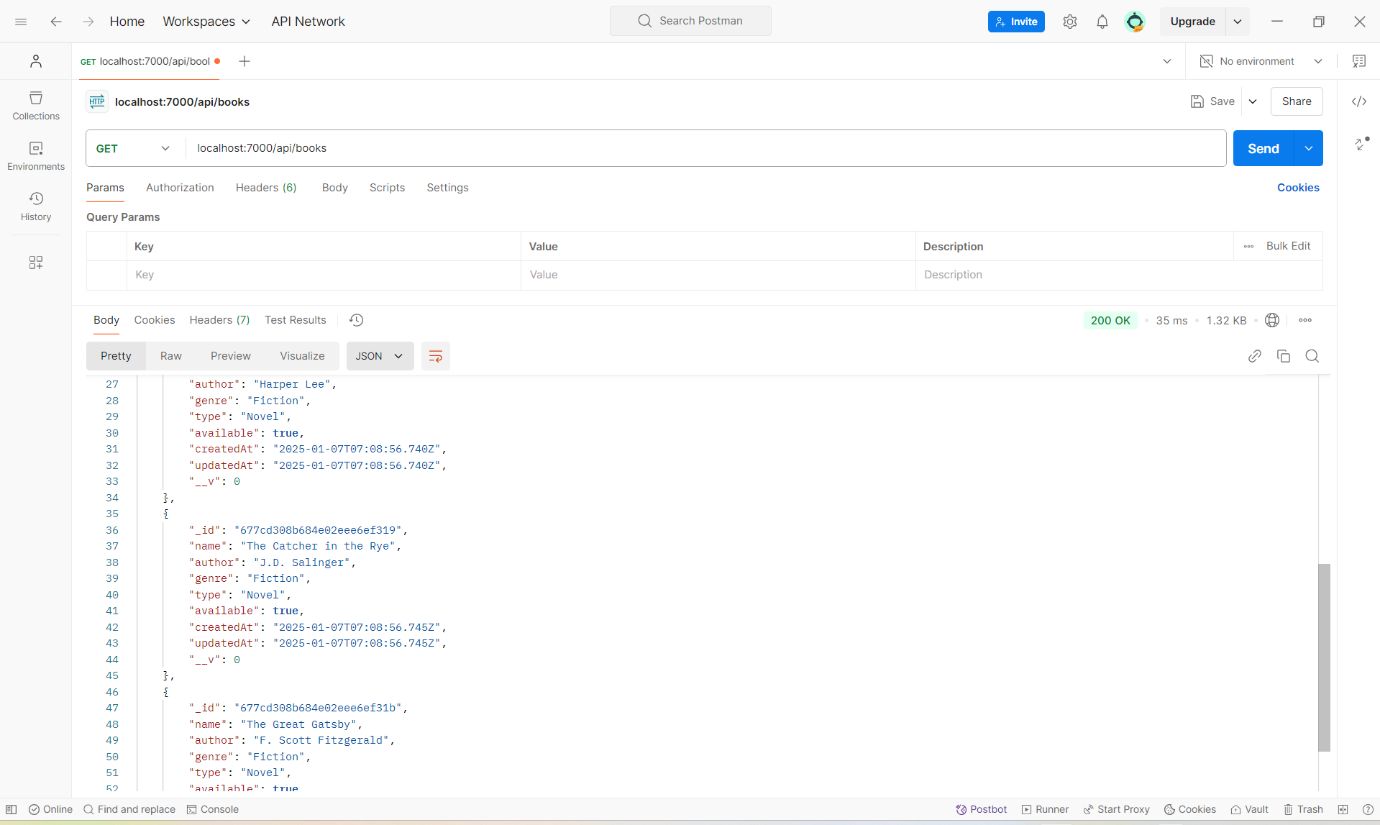


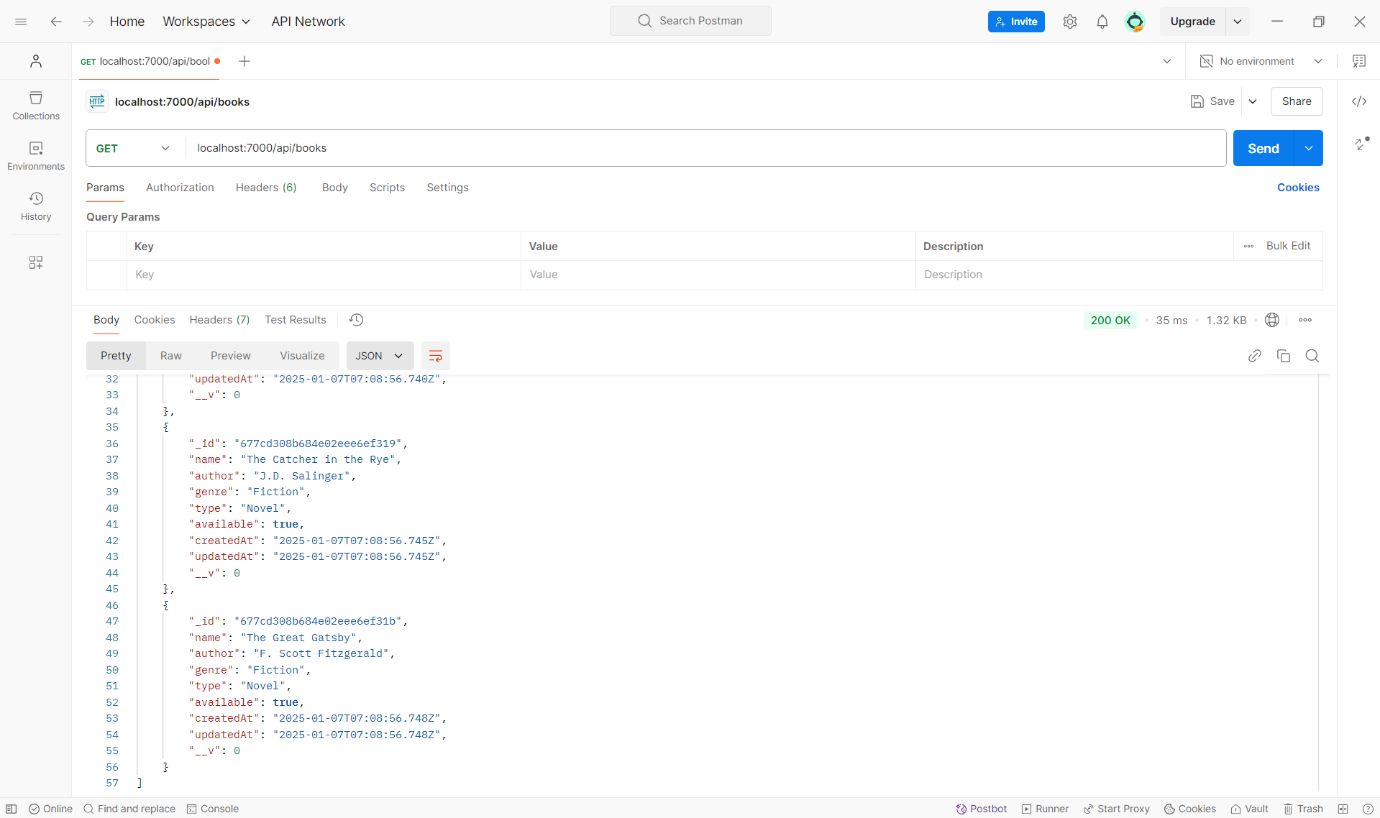


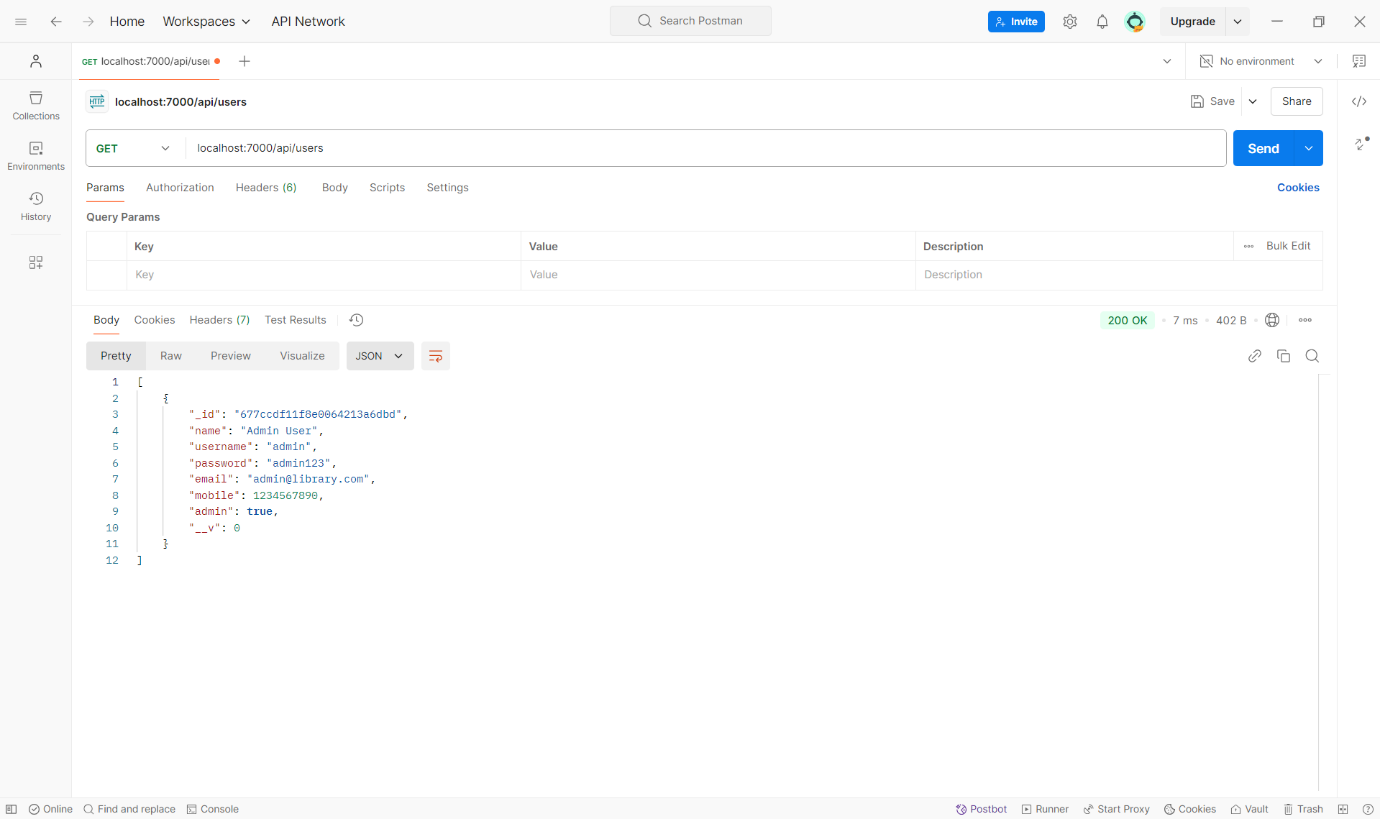


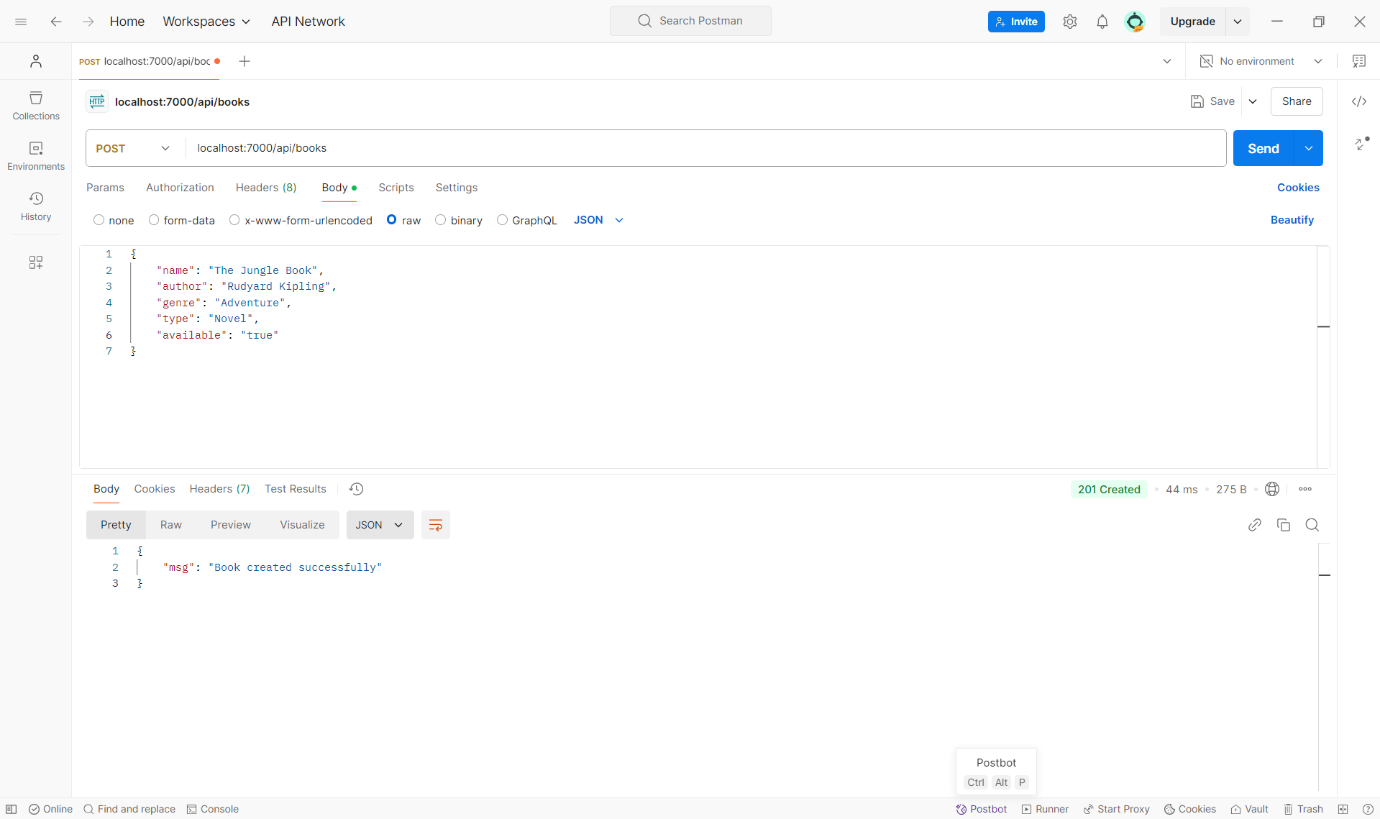


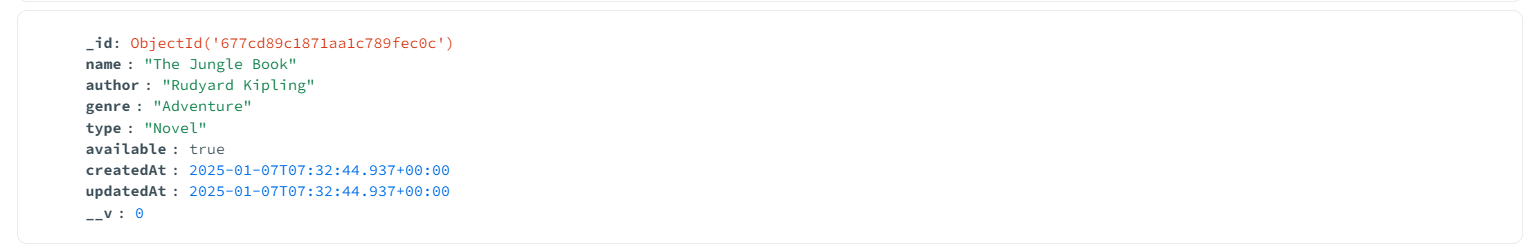


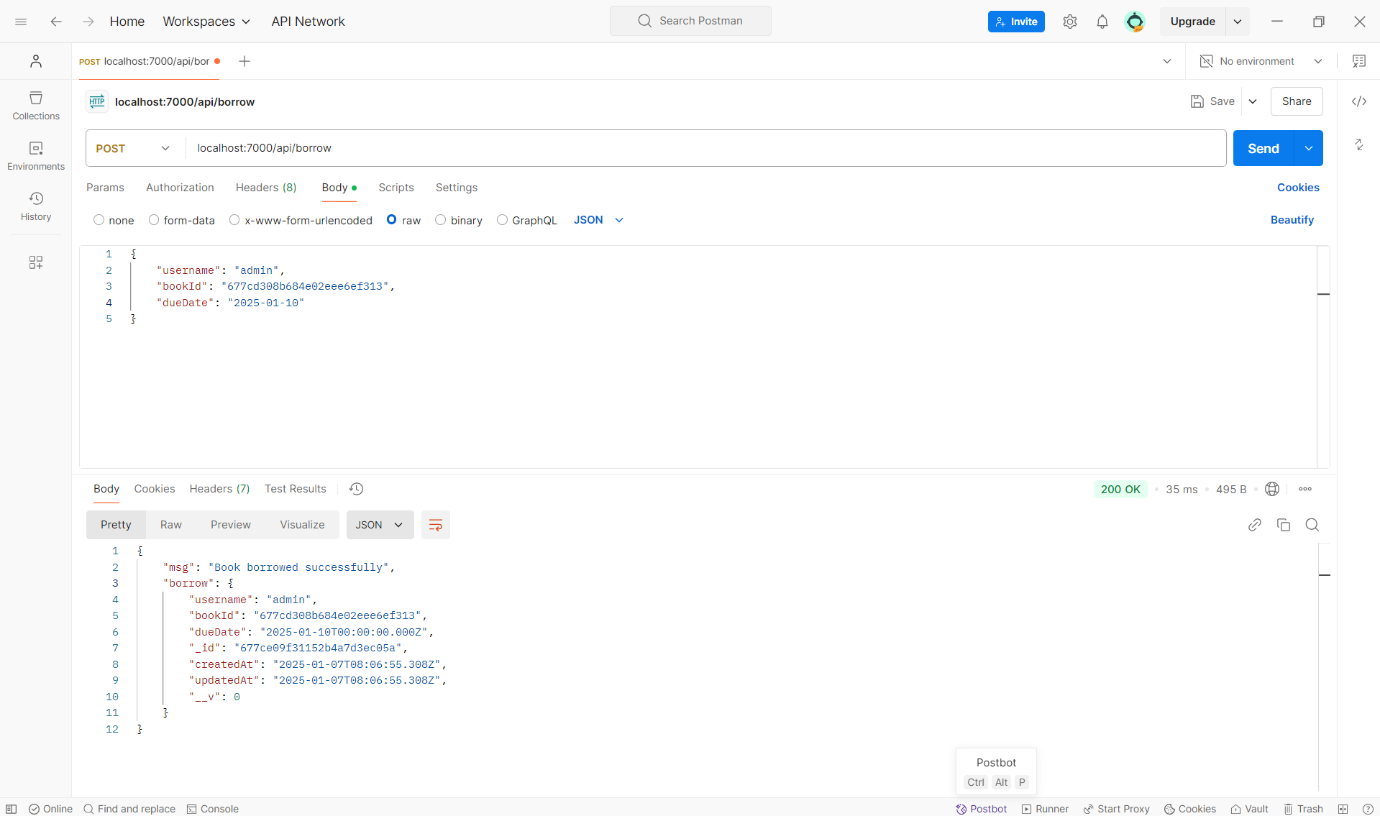


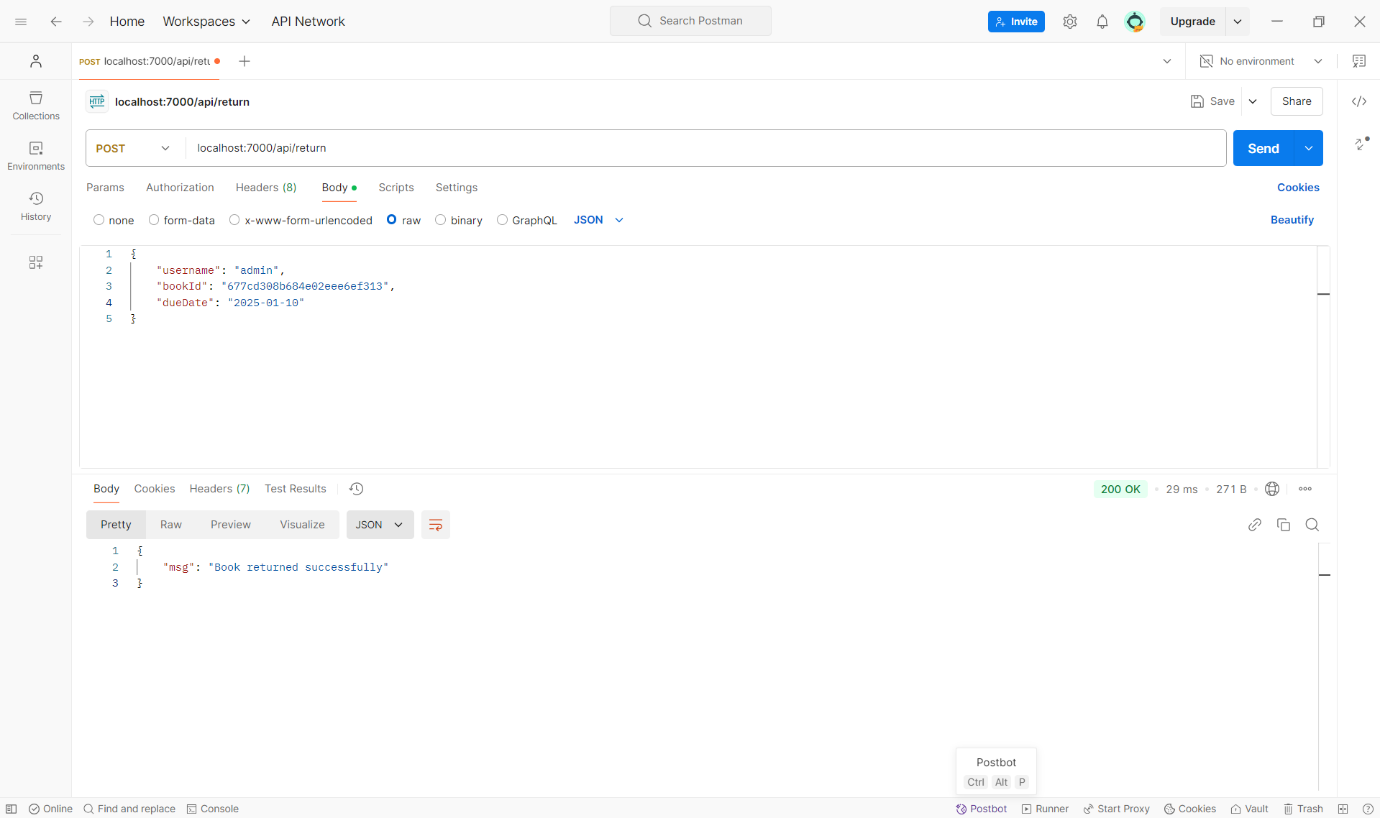


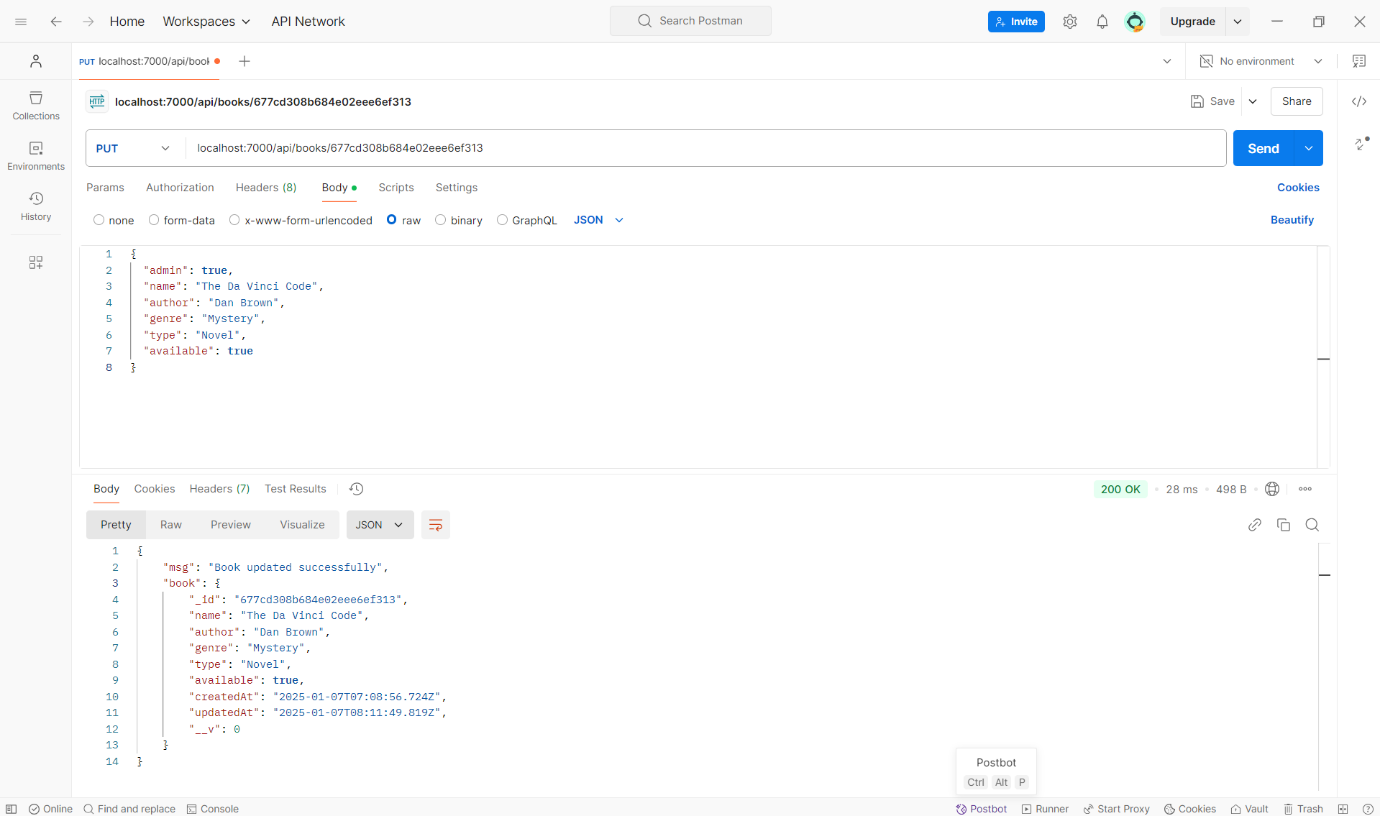


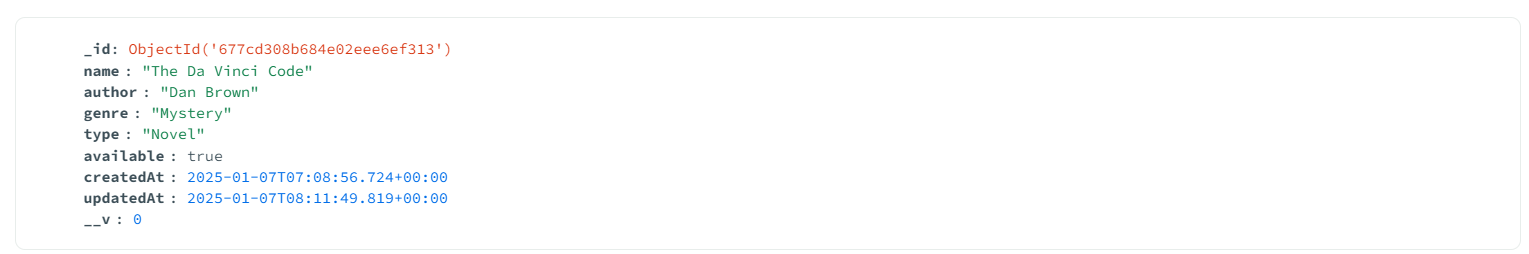


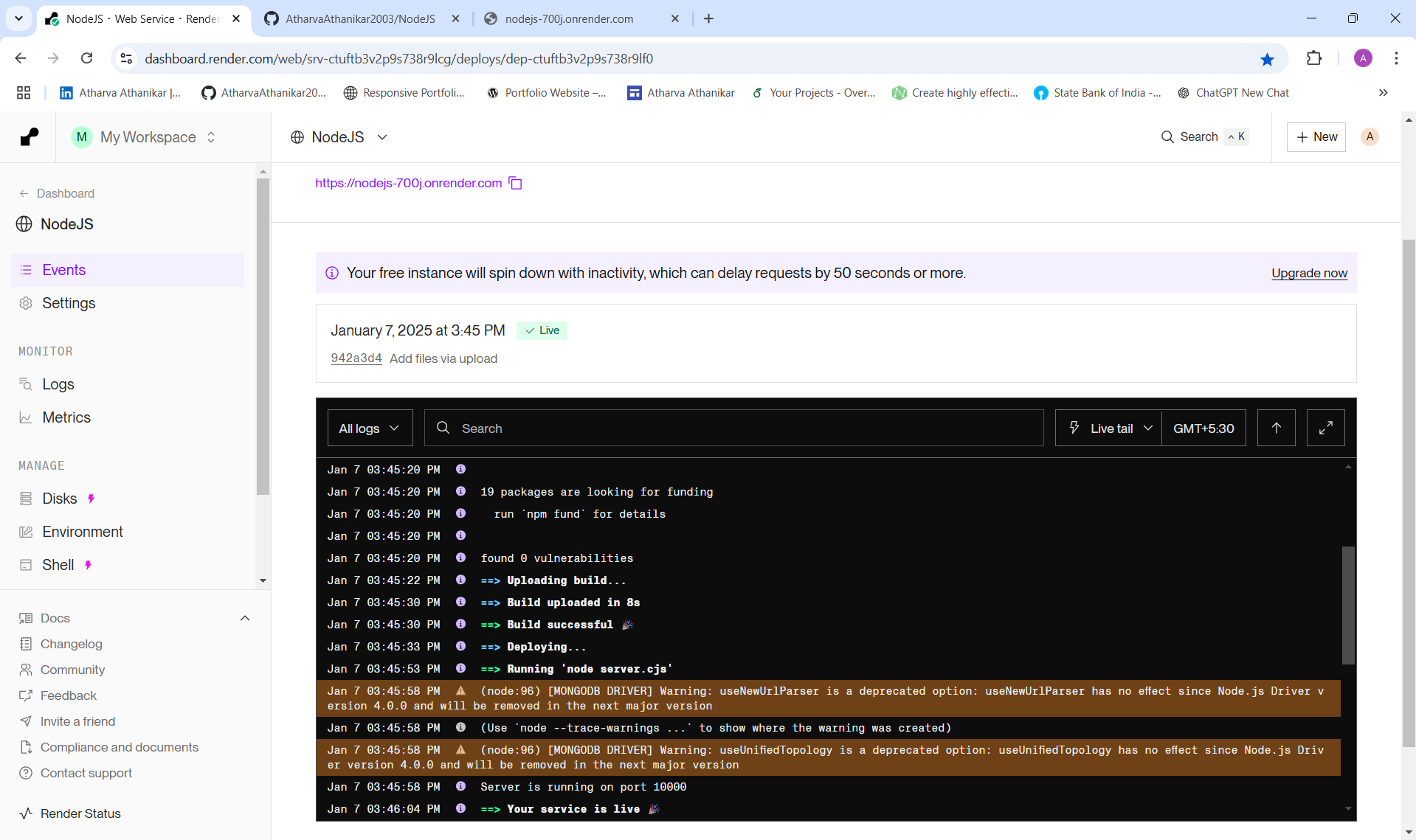


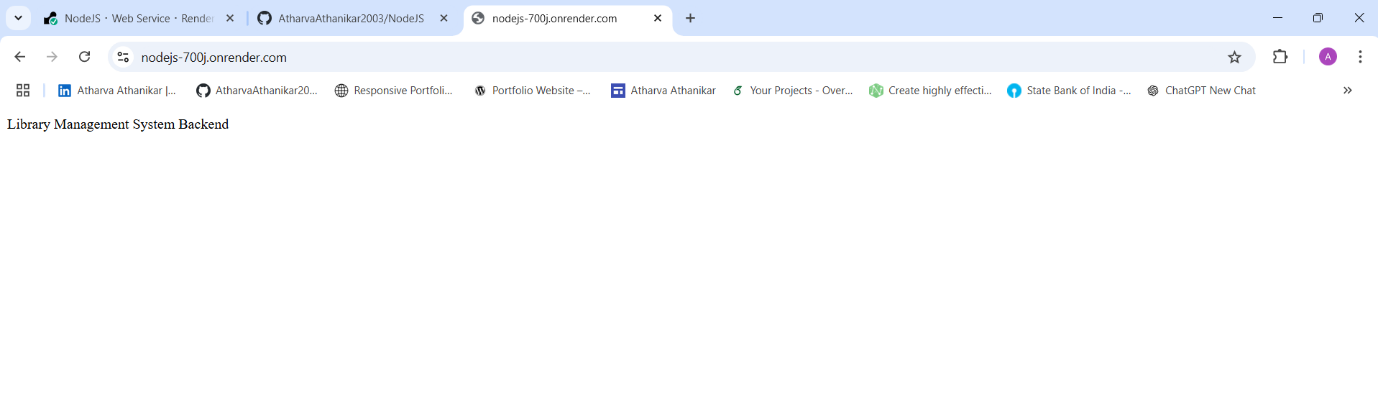












Github :

<https://github.com/AtharvaAthanikar2003/NodeJS>

Render:

<https://nodejs-700j.onrender.com>