International Islamic University Chittagong

Department of Computer Science and Engineering

LAB REPORT

Course title	: Software Engineering Sessional & Software Development 2
Course Code	: **CSE-3638 & 3640**
Report No	: **05**
Report Title	: **Project Proposal submission and Fundamentals of API Development**

<u>Submitted By</u>

Name	: **ArifulhasanAdil**
ID No	: **C223112**
Section	: **6CM**
Semester	: 6th

<u>Submitted To</u>

Mohammad Arfizurrahman
Adjunct Faculty
Department of CSE, IIUC

Submission Date : 25/05/2025

Lab Report 5: Project Proposal Submission and Fundamentals of API Development

Lab Tasks

1. Project Proposal

- Brainstorm a realistic software project idea.
- Prepare a brief proposal (1–2 pages) including:
  - Project title
  - Problem statement and motivation
  - Proposed features/modules
  - Team members (if applicable)
  - Technologies/tools you plan to use

Project TITLE:

Cloud-Integrated E-Commerce Platform Using Django and React

Problem Statement & MOTIVATION:

The exponential growth of online retail has made e-commerce platforms an essential part of the global economy. However, many current solutions suffer from issues such as complex deployment processes, difficulty in integrating secure payment systems, limited scalability, and fragmented technology stacks. These challenges can create barriers for developers and businesses looking to build or maintain modern online stores, especially when aiming to host them on cloud infrastructure.This project is driven by the goal of addressing these limitations through the development of a **cloud-ready, full-stack e-commerce platform**. By leveraging **Django** for backend operations and **React** for a dynamic and responsive frontend, the platform aims to combine performance, usability, and scalability into a single, cohesive solution.

The core motivation is to simplify the development and deployment experience, ensure secure and reliable payment processing, and offer a flexible foundation that can evolve with future business requirements. Cloud integration will further enable efficient storage, automated deployment, and better resource management, making the platform suitable for both startups and established businesses seeking to modernize their digital presence.

Proposed Features and Modules (Summary)

🔐 User Authentication

- Secure login/registration (JWT or Django Auth)
- Role-based access (Admin, Seller, Customer)
- Profile and password management

🛍️ Product Management

- Add/edit/delete products with images (stored in cloud)
- Categories, tags, variants (size, color)
- Inventory tracking

🛒 Cart & Checkout

- Add to cart, update quantity
- Address form and secure payment (Stripe/PayPal)
- Order confirmation and receipt generation

📦 Order Management

- Order tracking (Pending → Shipped → Delivered)
- View past orders
- Admin dashboard for order control

⭐ Reviews & Ratings

- User reviews and star ratings
- Display average ratings on product pages

🔍 Search & Filters

- Product search by name/category
- Filters: price range, rating, tags

☁️ Cloud Integration

- Cloud storage for media (AWS S3 / Cloudinary)
- Easy deployment on cloud platforms (Heroku, Render, AWS)

📊 Admin Panel

- Manage users, products, and orders

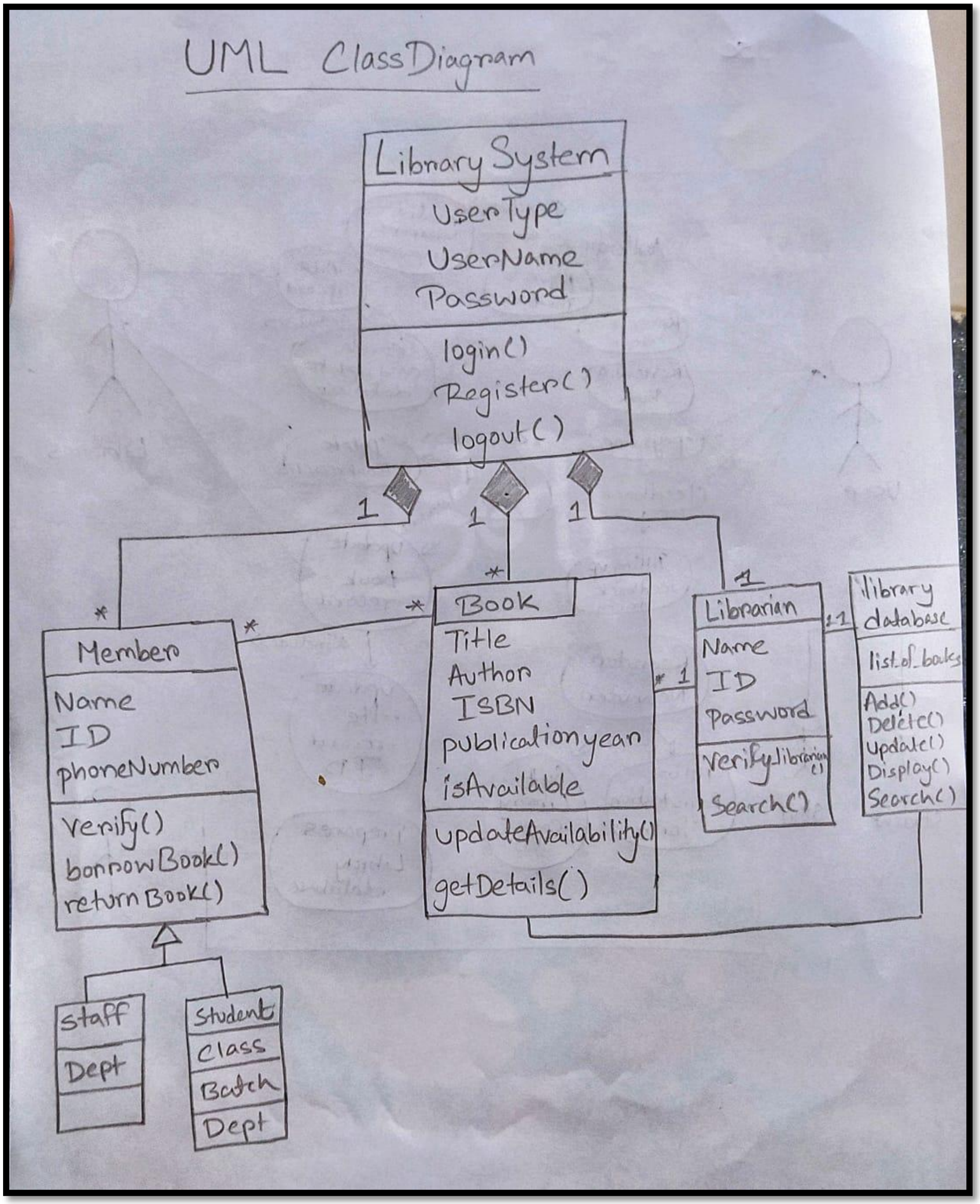
Team Members :

1. *Apan Singha— Project Leader*
2. *Chowdhury Avishek Pritom*
3. *Ariful hasan Adil*
4. *Halimur rashid*

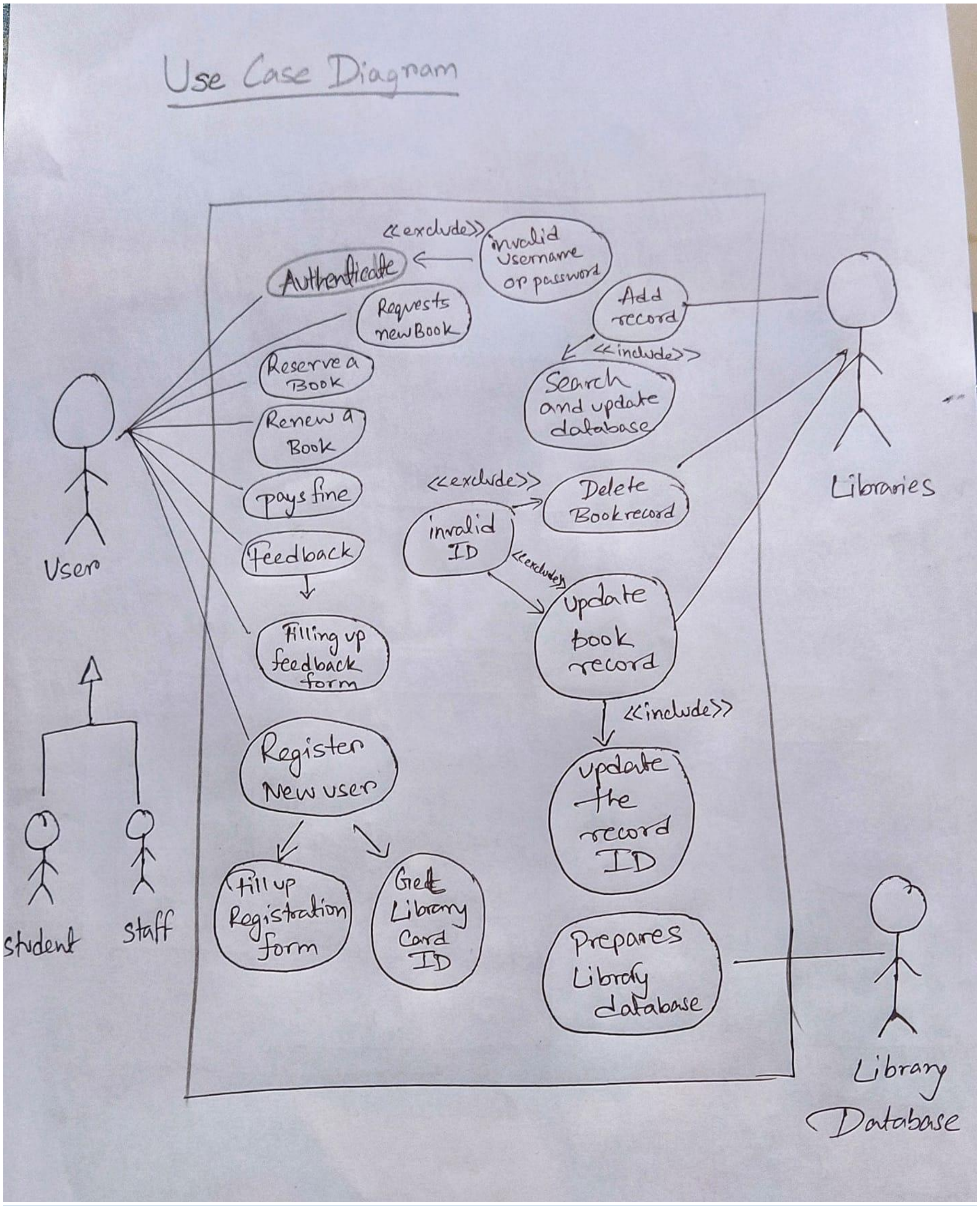
Technologies/Tools Planned to Use:

- Backend: Django, Django REST Framework,SQL
- Frontend: React.js, Bootstrap
- Payment Gateways: Stripe, PayPal SDKs
- Deployment: Docker, Docker Compose, Cloud services (AWS, GCP, or Azure)
- Version Control: Git & GitHub
- Environment Management: dotenv, environment variables in cloud environment
- Other Tools: CI/CD pipeline for automated deployment

DRAWING UML CLASS DIAGRAM:



DRAWING UML CASE DIAGRAM:



3. REST API Fundamentals

What is REST?

REST (Representational State Transfer) is an architectural style for designing networked applications. REST APIs allow different systems to communicate over HTTP using standard operations.

Key HTTP Methods in REST

1. **GET**
   The `GET` method is used to **retrieve data** from the server. It is a **read-only** operation and does not change any data. For example, fetching a list of products or the details of a specific user would use a GET request.
2. **POST**
   The `POST` method is used to **create new resources** on the server. When a client sends data to the server (such as submitting a form), the server processes it and stores the new data. For instance, adding a new product or registering a user would use POST.
3. **PUT**
   The `PUT` method is used to **update an existing resource**. It typically replaces the entire resource with new data. For example, updating all details of a product or user profile would use PUT.
4. **DELETE**
   The `DELETE` method is used to **remove a resource** from the server. For example, deleting a product, an order, or a user account would involve sending a DELETE request.

Each of these methods follows REST principles and is commonly used to perform **CRUD operations**:

- **Create** → POST
- **Read** → GET
- **Update** → PUT
- **Delete** → DELETE

Routes and Endpoints:

- An **endpoint** is a URL where a resource is accessed or manipulated.
- **Example Routes:**
    - `GET /api/products/` → Get all products
    - `GET /api/products/5/` → Get product with ID 5
    - `POST /api/products/` → Create a new product
    - `PUT /api/products/5/` → Update product with ID 5
    - `DELETE /api/products/5/` → Delete product with ID 5

JSON Response Structure:

REST APIs usually return data in **JSON** format.

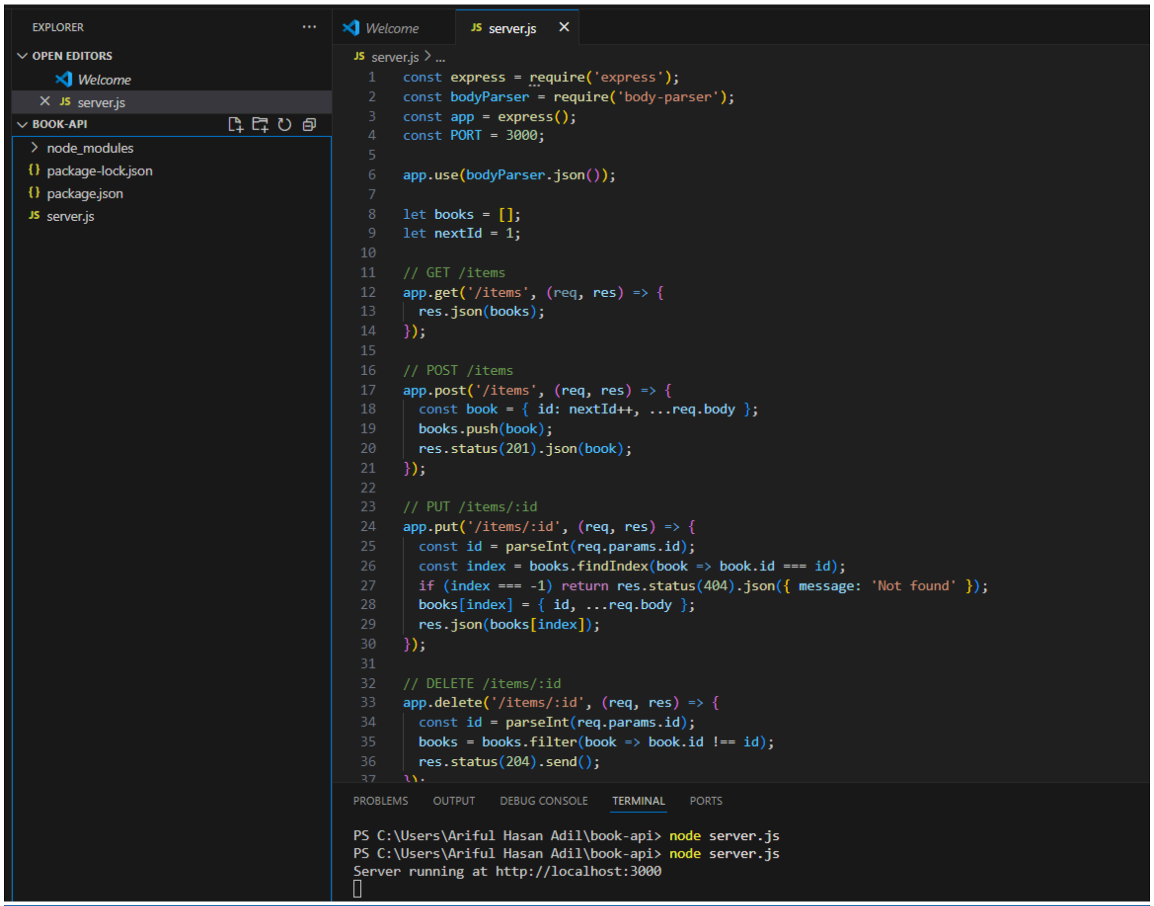
Example response:

```
{
  "id": 5,
  "name": "Wireless Mouse",
  "price": 25.99,
  "in_stock": true
}
```

Status Codes:

HTTP status codes are three-digit numbers returned by the server in response to a client's request. They indicate the outcome of the request. Understanding these codes is crucial for building robust client applications.

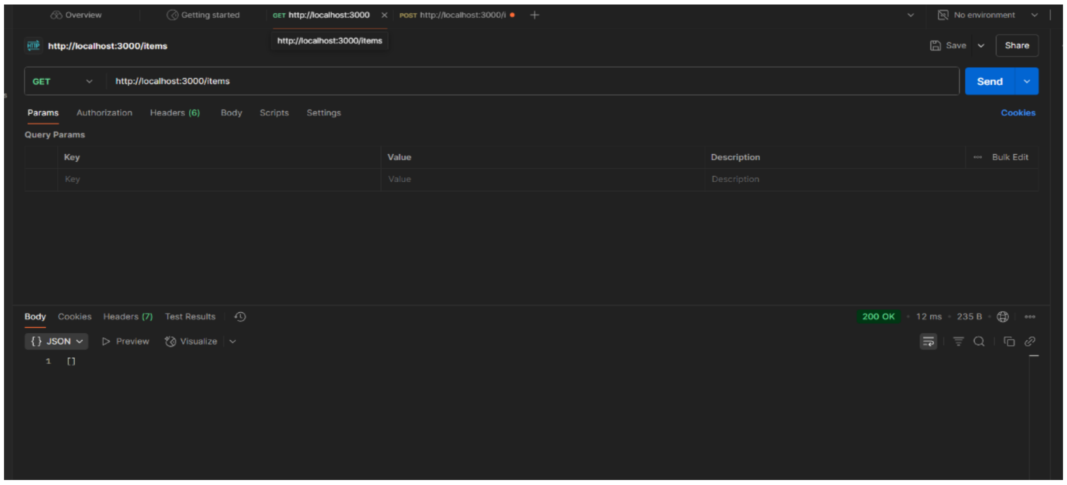
Hands-On API Implementation



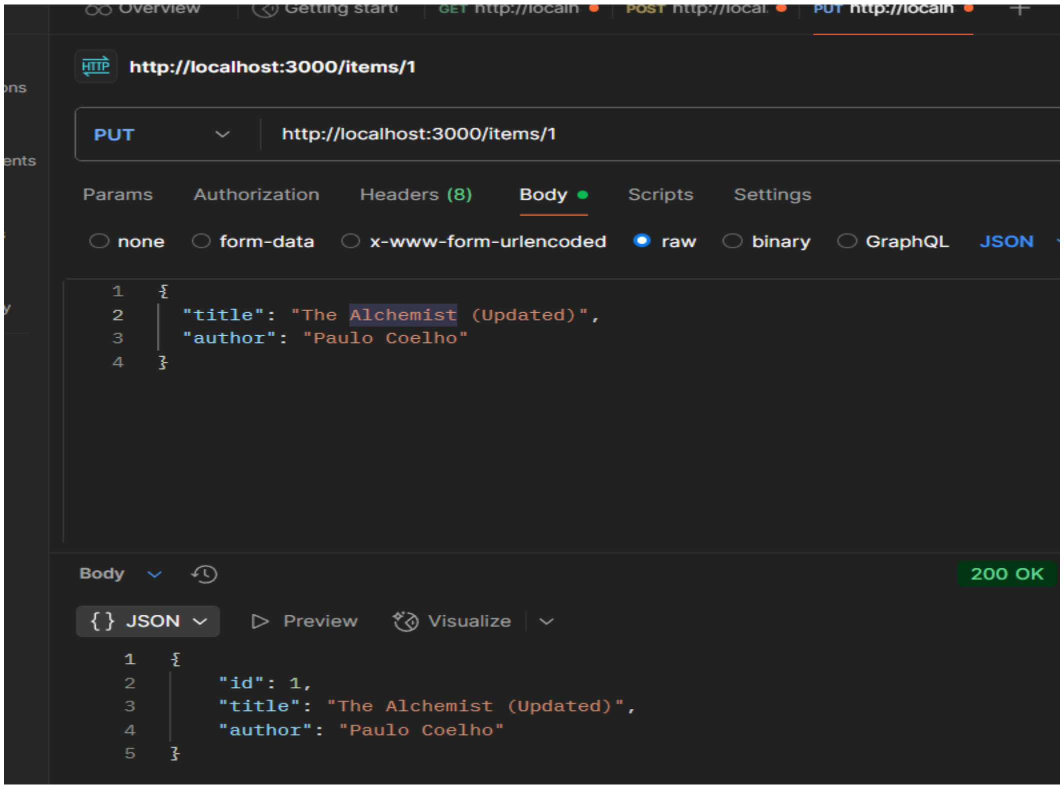
```javascript
const express = require('express');
const bodyParser = require('body-parser');
const app = express();
const PORT = 3000;

app.use(bodyParser.json());

let books = [];
let nextId = 1;

// GET /items
app.get('/items', (req, res) => {
  res.json(books);
});

// POST /items
app.post('/items', (req, res) => {
  const book = { id: nextId++, ...req.body };
  books.push(book);
  res.status(201).json(book);
});

// PUT /items/:id
app.put('/items/:id', (req, res) => {
  const id = parseInt(req.params.id);
  const index = books.findIndex(book => book.id === id);
  if (index === -1) return res.status(404).json({ message: 'Not found' });
  books[index] = { id, ...req.body };
  res.json(books[index]);
});

// DELETE /items/:id
app.delete('/items/:id', (req, res) => {
  const id = parseInt(req.params.id);
  books = books.filter(book => book.id !== id);
  res.status(204).send();
```

```
PS C:\Users\Ariful Hasan Adil\book-api> node server.js
PS C:\Users\Ariful Hasan Adil\book-api> node server.js
Server running at http://localhost:3000
```

- Implement endpoints:
  - `GET /items`
  - `POST /items`
  - `PUT /items/:id`
  - `DELETE /items/:id`



```
http://localhost:3000/items

GET  http://localhost:3000/items

Key | Value | Description

200 OK · 12 ms · 235 B

1  []
```



http://localhost:3000/items/1

PUT http://localhost:3000/items/1

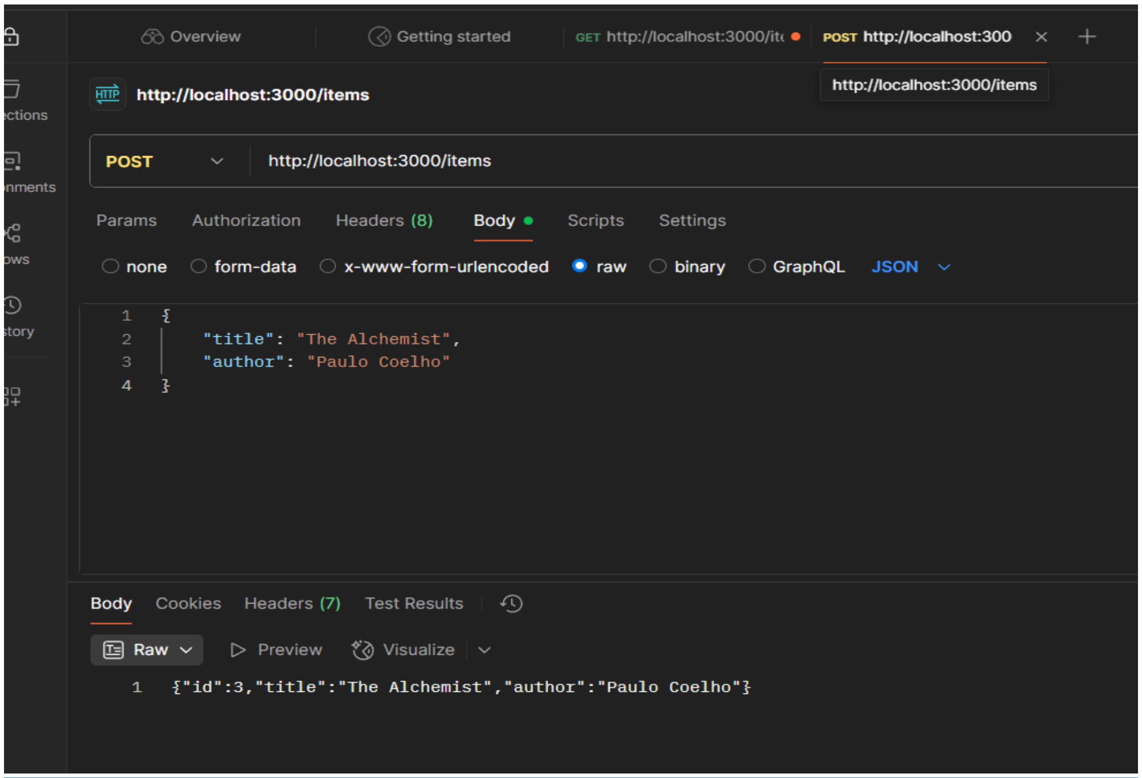
none | form-data | x-www-form-urlencoded | [✓ raw] | binary | GraphQL | JSON

```
{
  "title": "The Alchemist (Updated)",
  "author": "Paulo Coelho"
}
```

Body — 200 OK

{} JSON | Preview | Visualize

```
{
    "id": 1,
    "title": "The Alchemist (Updated)",
    "author": "Paulo Coelho"
}
```



Overview
Getting started
GET http://localhost:3000/it
POST http://localhost:300
http://localhost:3000/items
ctions
nments
ows
story

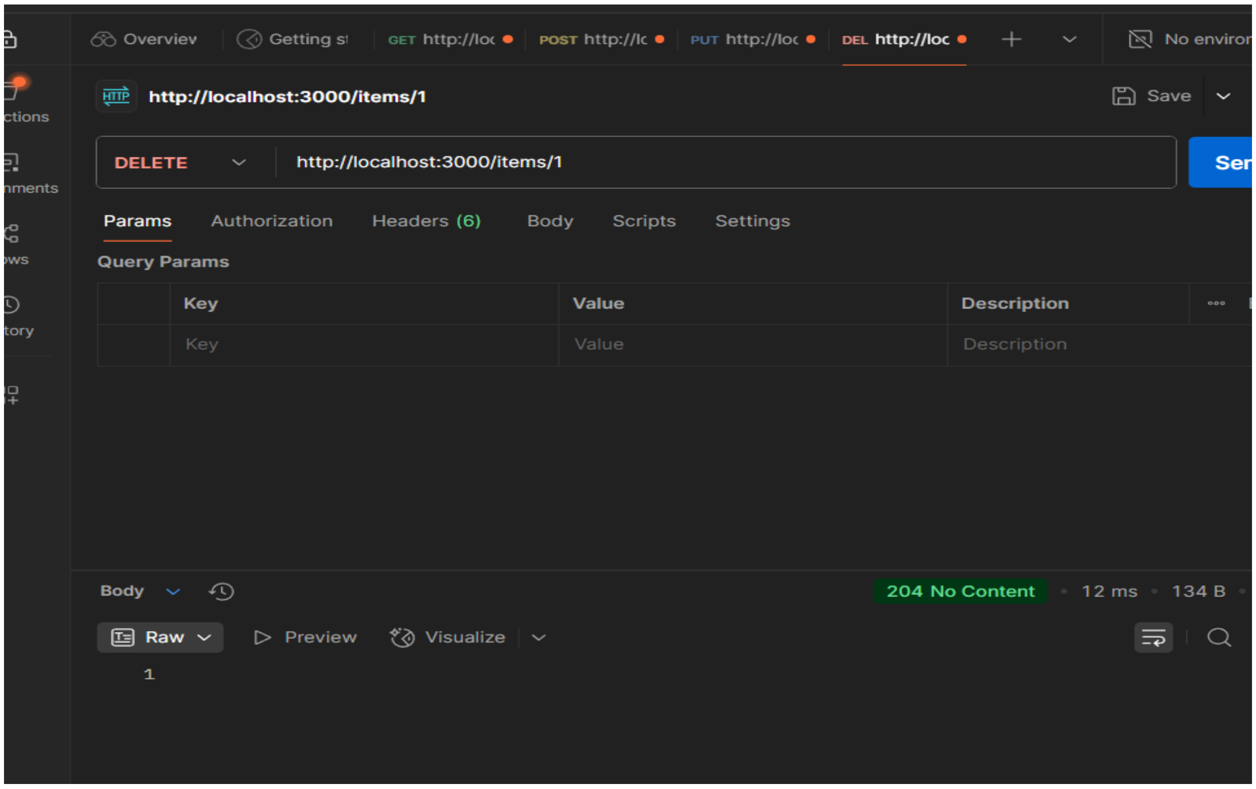
http://localhost:3000/items

POST http://localhost:3000/items

none | form-data | x-www-form-urlencoded | [✓ raw] | binary | GraphQL | JSON

```
{
  "title": "The Alchemist",
  "author": "Paulo Coelho"
}
```

```
{"id":3,"title":"The Alchemist","author":"Paulo Coelho"}
```



http://localhost:3000/items/1

DELETE http://localhost:3000/items/1

Query Params

Key	Value	Description
Key	Value	Description

Body

204 No Content • 12 ms • 134 B

```
1
```

Conclusion:

By implementing this Express.js API, I've built a solid foundational understanding of how RESTful APIs work. Key concept I practiced:

- **Routing & HTTP Methods**
- **Data Handling**
- **Status Codes & Responses**
- **Request Parsing with Middleware**
- **Testing & Debugging with Postman**