Ganpat University
॥ विद्यया समाजोत्कर्षः ॥
Institute of Computer Technology

Name: Tushar Panchal

En.No: 21162101014

Sub: MICROSEVICES

Branch: CBA

Batch:51

----------PRACTICAL 10----------

❖ Question (TASK) :

Build a basic demo application for illustrating Microservices with Node.js.

Implement an application on Microservices using Node js where each of the services will have individual servers running on different ports. These services will communicate with each other through REST APIs. Example application three services: Book, Customer, and Order services. The following tasks should be performed:

1. Creating Database Connection.

2. Creating Book Service.

3. Creating Customer Service.

4. Creating Order Service.

🔗 **Github Link :**
https://github.com/Tushar007079/MICROSERVICES_PRACTICALS/tree/49fd765ee0c54184cff204bcfc464f649ba5d9c1/10

❖ **STEPS TO PERFORM THIS TASK :**

📌 Step 1 :

» Install Dependencies.

» Run this following command to initialize the package.json file :
`npm init -y`

» Run this following command to install the mongoose , express body-parser :
`npm install express mongoose body-parser`

📌 Step 2 :

» Define Database connection.

» n the db.js file, define the Mongoose schemas and the database connection.

» Here's how to create a MongoDB database and collections:
→Create a Database
In the MongoDB shell, create a new database. the database is named myappdb. You can create it with the use command:
`use myappdb`

» Create Collections

➔ Next, create collections for each of the services: "books," "customers," and "orders." Here's how you can create collections for each service:
`db.createCollection('books')`
`db.createCollection('customers')`
`db.createCollection('orders')`

✓ **db.js :-**

```
const mongoose = require('mongoose');

mongoose.connect('mongodb://localhost/microservices-demo', {
    useNewUrlParser: true,
    useUnifiedTopology: true,
});
```

```
const BookSchema = new mongoose.Schema({
    title: String,
    author: String,
    isbn: String,
});

const CustomerSchema = new mongoose.Schema({
    name: String,
    email: String,
});

const OrderSchema = new mongoose.Schema({
    customerId: mongoose.Schema.Types.ObjectId,
    bookId: mongoose.Schema.Types.ObjectId,
});

const Book = mongoose.model('Book', BookSchema);
const Customer = mongoose.model('Customer', CustomerSchema);
const Order = mongoose.model('Order', OrderSchema);

module.exports = {
    Book,
    Customer,
    Order,
};
```

In MongoDB Compass shell :

```
>_MONGOSH
> use myappdb
< switched to db myappdb
> db.createCollection('books')
  db.createCollection('customers')
  db.createCollection('orders')
< { ok: 1 }
```

Step 3 :

Define Service Files.

✓ **bookService.js :-**

```
const express = require('express');
const app = express();
const bodyParser = require('body-parser');
const db = require('./db');

app.use(bodyParser.json());

app.get('/books', async (req, res) => {
    const books = await db.Book.find();
    res.json(books);
});

app.post('/books', async (req, res) => {
    const newBook = new db.Book(req.body);
    await newBook.save();
    res.status(201).json(newBook);
});

// Update a book by isbn(International Standard Book Number)
app.put('/books/:isbn', async (req, res) => {
    const { isbn } = req.params;
    const updatedBook = await db.Book.findOneAndUpdate({ isbn }, req.body, {
new: true });
    res.json(updatedBook);
  });


// Delete a book by isbn(International Standard Book Number)
app.delete('/books/:isbn', async (req, res) => {
    const { isbn } = req.params;
    const deletedBook = await db.Book.findOneAndRemove({ isbn });
    res.json(deletedBook);
  });


const PORT = 3000;
app.listen(PORT, () => {
    console.log(`Book service is running on port ${PORT}`);
});
```

✓ **customerService.js :-**

```
const express = require('express');
const app = express();
```

```
const bodyParser = require('body-parser');
const db = require('./db');

app.use(bodyParser.json());

app.get('/customers', async (req, res) => {
    const customers = await db.Customer.find();
    res.json(customers);
});

app.post('/customers', async (req, res) => {
    const newCustomer = new db.Customer(req.body);
    await newCustomer.save();
    res.status(201).json(newCustomer);
});

// Update a customer by name
app.put('/customers/:name', async (req, res) => {
    const { name } = req.params;
    const updatedCustomer = await db.Customer.findOneAndUpdate({ name },
req.body, { new: true });
    res.json(updatedCustomer);
  });

// Delete a customer by name
app.delete('/customers/:name', async (req, res) => {
    const { name } = req.params;
    const deletedCustomer = await db.Customer.findOneAndRemove({ name });
    res.json(deletedCustomer);
  });

const PORT = 3001;
app.listen(PORT, () => {
    console.log(`Customer service is running on port ${PORT}`);
});
```

✓ **orderServices.js :-**

```
const express = require('express');
const app = express();
const bodyParser = require('body-parser');
const db = require('./db');

app.use(bodyParser.json());

app.get('/orders', async (req, res) => {
    const orders = await db.Order.find();
    res.json(orders);
});
```

```
app.post('/orders', async (req, res) => {
    const newOrder = new db.Order(req.body);
    await newOrder.save();
    res.status(201).json(newOrder);
});

// Update an order by customerId
app.put('/orders/:customerId', async (req, res) => {
    const { customerId } = req.params;
    const updatedOrder = await db.Order.findOneAndUpdate({ customerId },
req.body, { new: true });
    res.json(updatedOrder);
  });

// Delete an order by customerId
app.delete('/orders/:customerId', async (req, res) => {
    const { customerId } = req.params;
    const deletedOrder = await db.Order.findOneAndRemove({ customerId });
    res.json(deletedOrder);
  });

const PORT = 3002;
app.listen(PORT, () => {
    console.log(`Order service is running on port ${PORT}`);
});
```

👉 Step 4 :

- Run the Services.
- Run this following command to run each services :

```
node bookService.js
```

```
node customerService.js
```

```
node orderService.js
```

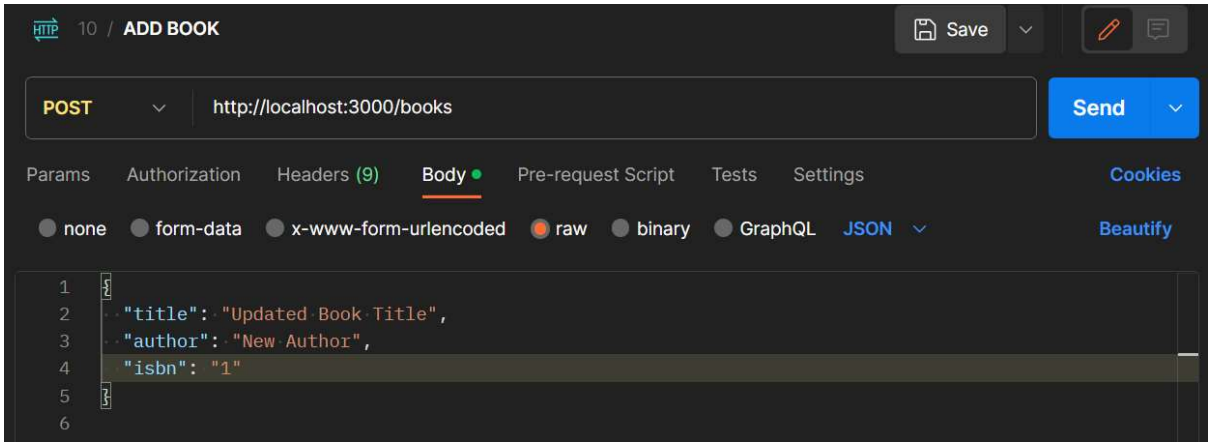
Step 5 :

Test the services with POSTMAN.

Book Service :

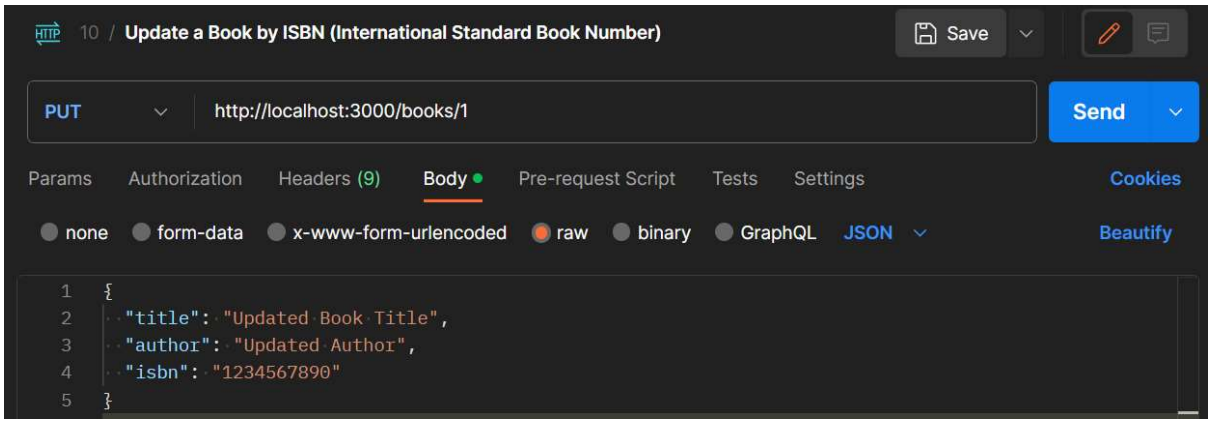
1. Add a Book :

- **Method: *POST***
- **Endpoint: *http://localhost:3000/books***
- **Body: *Raw JSON***



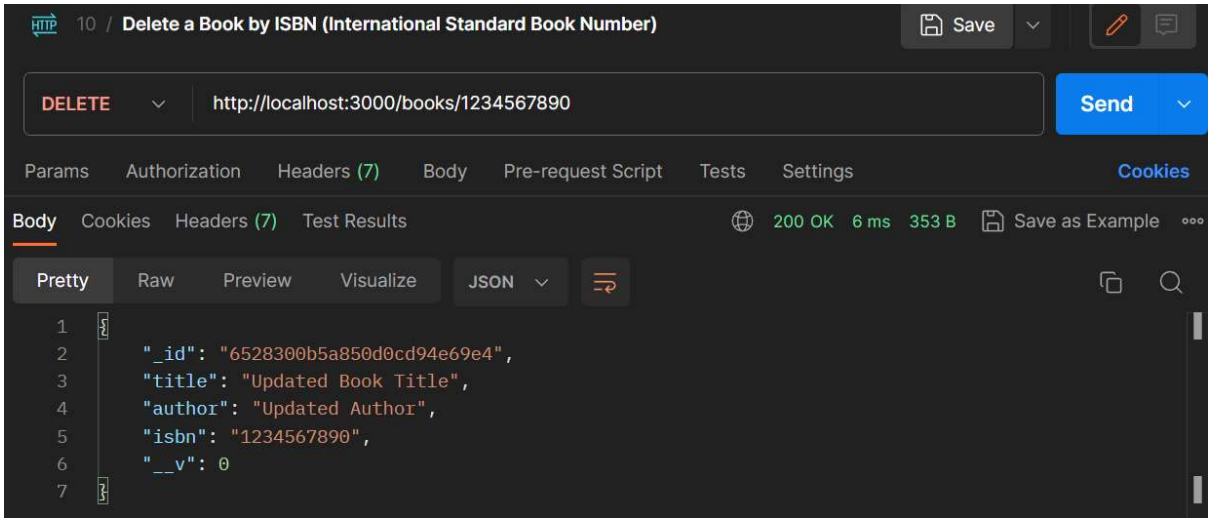
2. Update a book :

- **Method: *PUT***
- **Endpoint: *http://localhost:3000/books/{isbn}***
- **Body: *Raw JSON***



3. Delete a book :

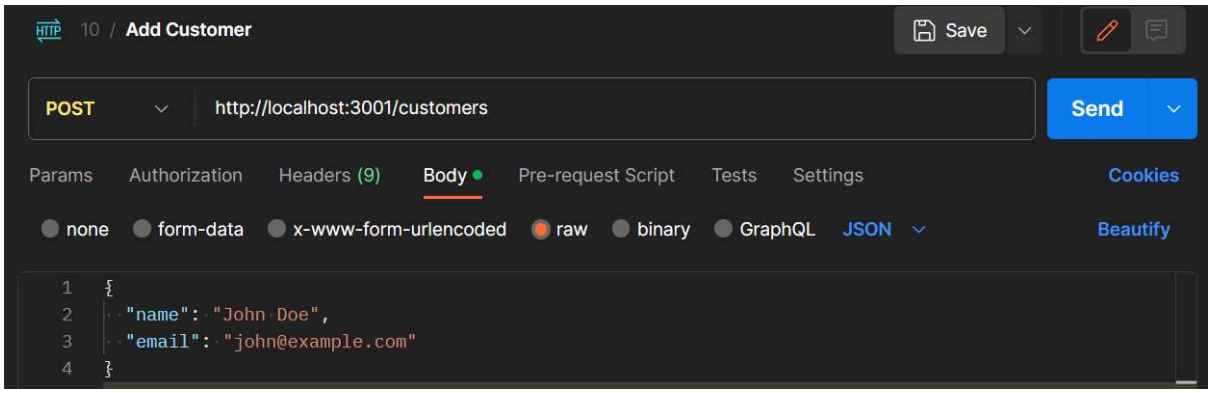
- **Method: *DELETE***
- **Endpoint: *http://localhost:3000/books/{isbn}***



→ ***Customer Service :***

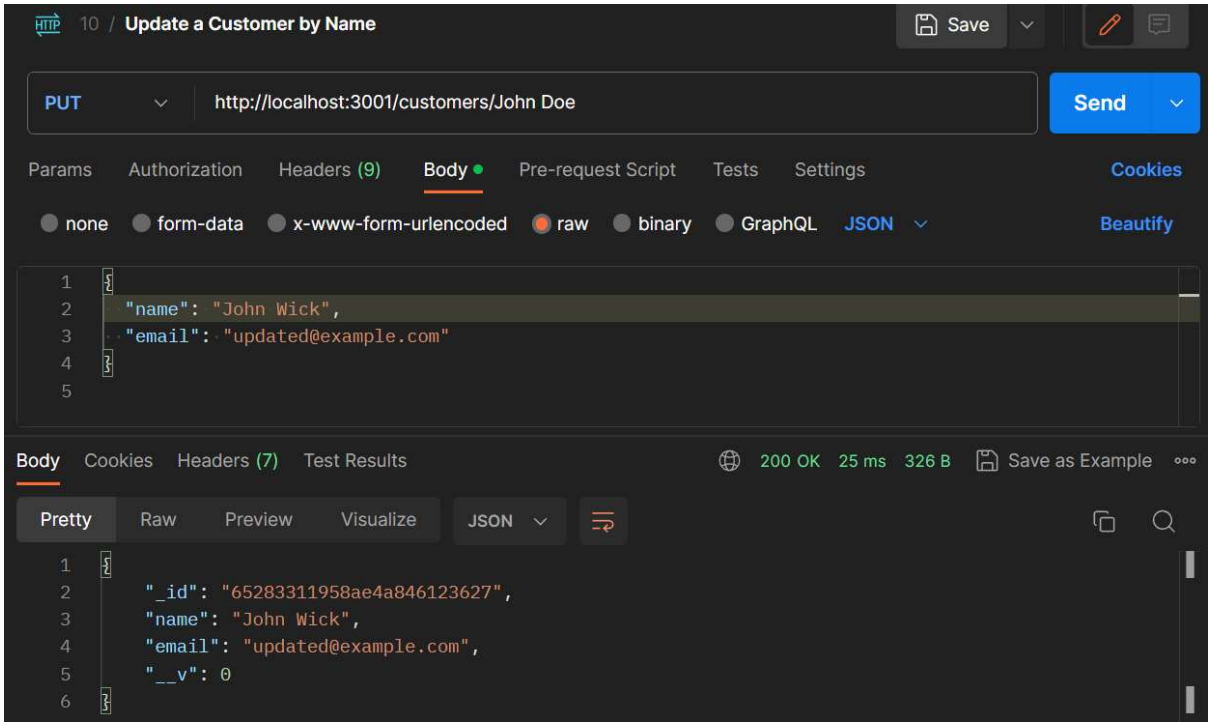
1. Add a Customer :

- **Method: *POST***
- **Endpoint: *http://localhost:3001/customers***
- **Body: *Raw JSON***



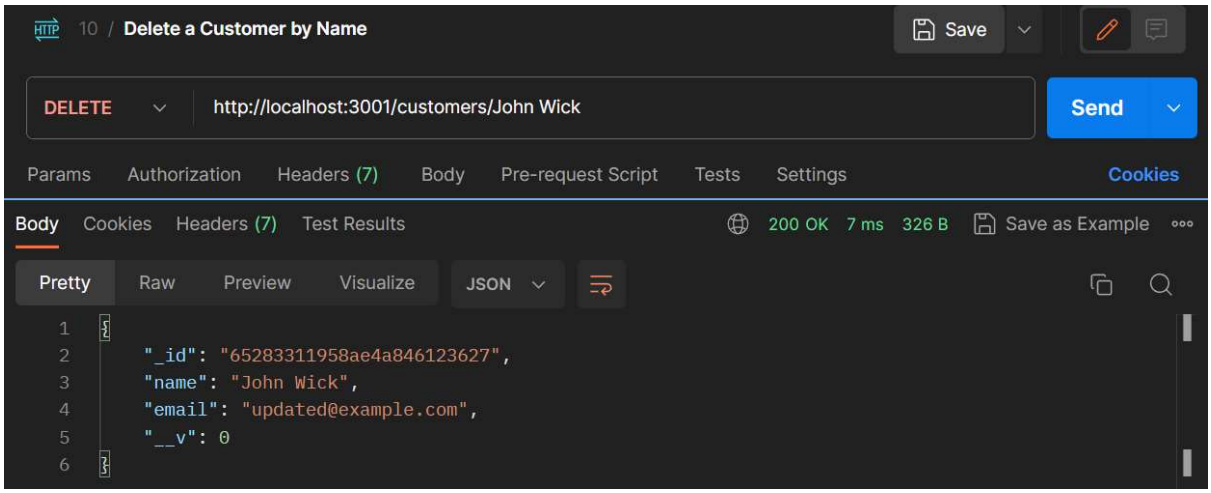
2. Update a Customer :

- **Method: *POST***
- **Endpoint: *http://localhost:3001/customers/{name}***
- **Body: *Raw JSON***



3. Delete a Customer :

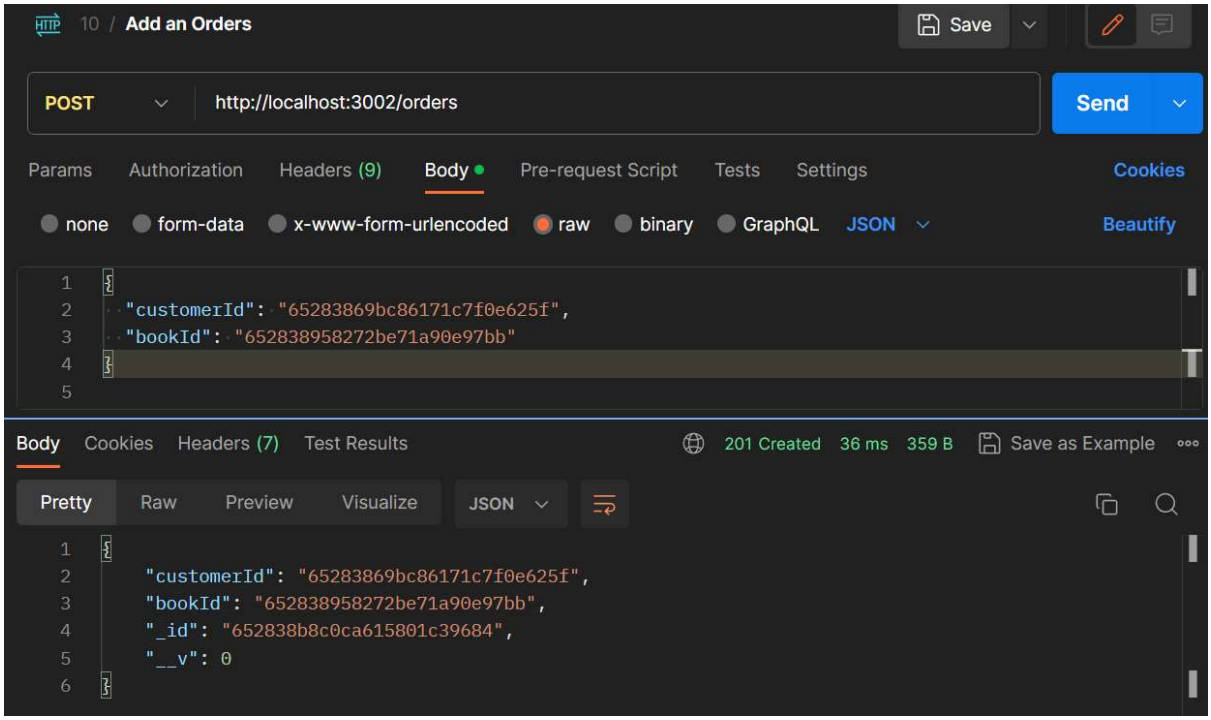
- **Method:** ***DELETE***
- **Endpoint:** ***http://localhost:3001/customers/{name}***
- **Body:** ***Raw JSON***



→ *Order Service :*

1. Add an Orders :

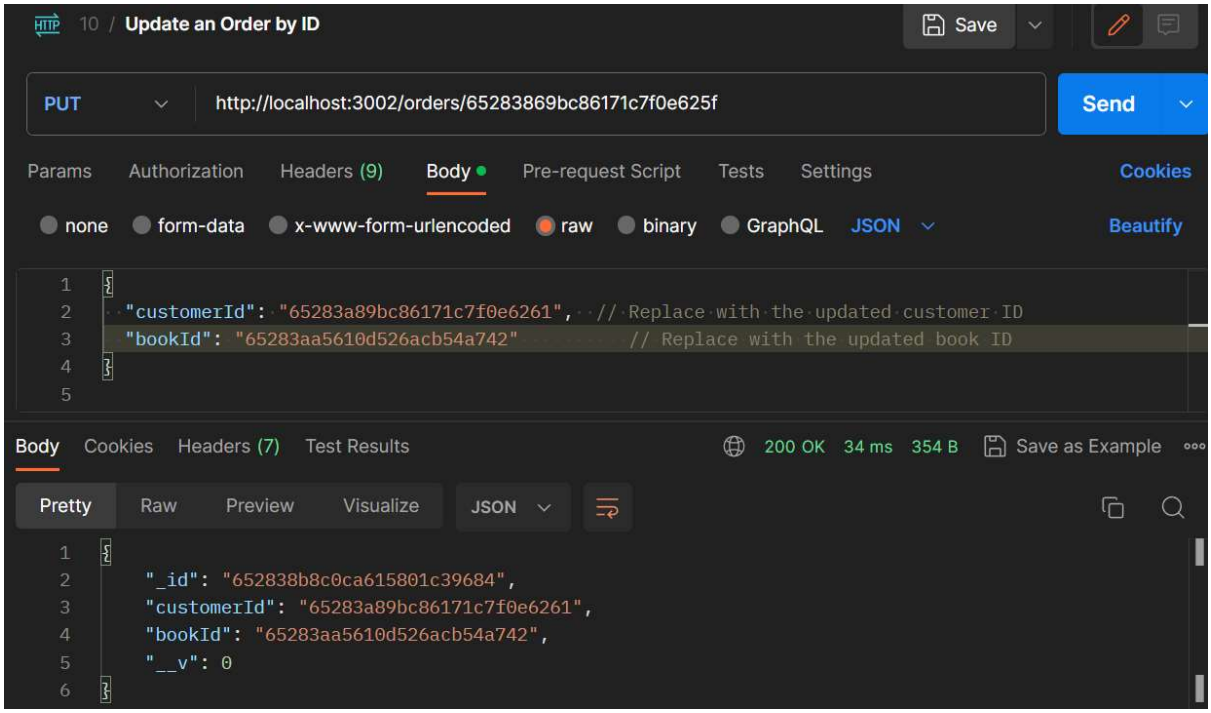
- **Method:** ***POST***
- **Endpoint:** ***http://localhost:3002/orders***
- **Body:** ***Raw JSON***



In this We need customerID from customers and bookId from Books Services then we put in this order service to make it order.

2. Update an Order :

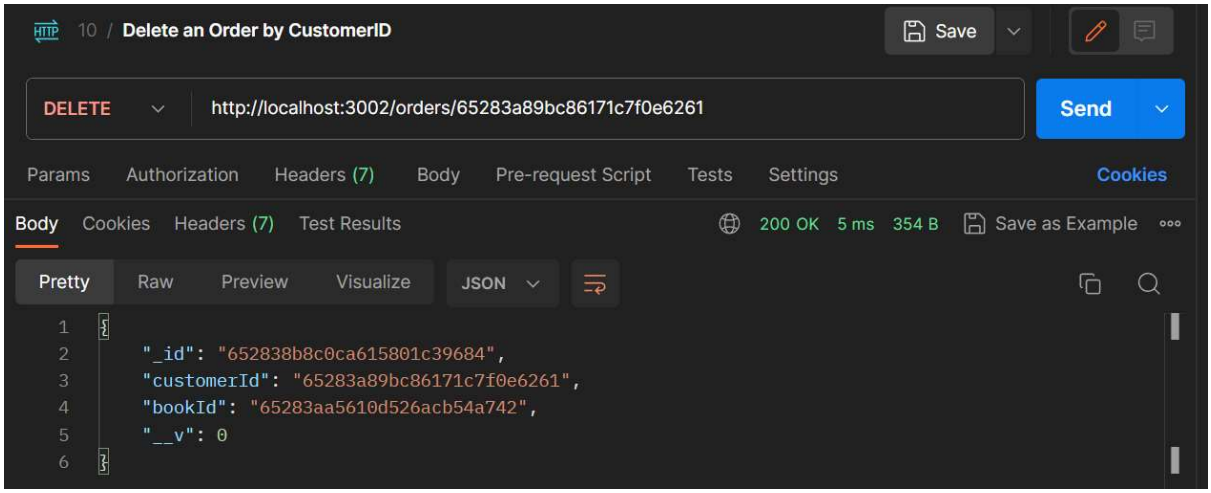
- **Method: *POST***
- **Endpoint: *http://localhost:3002/orders/{id}***
- **Body: *Raw JSON***



In this We need another customerID from customers and bookId from Books Services then we put in this order service to make it Update.

3. Delete a Customer :

- **Method:** ***DELETE***
- **Endpoint:** ***http://localhost:3002/orders/{id}***
- **Body:** ***Raw JSON***



In this We need customerID from orders to delete order.