

**Name: Patel Shivam S.
Enroll no: 21162101019
Sem: 5 Batch: 51 Branch: CBA**

Sub: Microservices

Practical 10

AIM: 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/Shivam3783/microservice_practicals/tree/main/prac10

Practical 8.1: Creating Database Connection.

The screenshot shows the VS Code interface with the following details:

- Explorer View:** Shows a project structure with files like Book.js, books.js, customers.js, db.js, Order.js, orders.js, package-lock.json, and package.json.
- Code Editor:** Displays the content of db.js, which attempts to connect to a MongoDB database using Mongoose.

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

mongoose.connect('mongodb://localhost:27017/books', {
  useNewUrlParser: true,
  useUnifiedTopology: true,
  //useFindAndModify: false,
  //useCreateIndex: true
}).then(() => {
  console.log('Connection successfull!');
}).catch((e) => {
  console.log('Connection failed!');
  console.log(e.message);
});
```

- Terminal:** Shows the output of running the script, indicating a successful connection.

```
Node.js v20.6.1
~/.nvm/versions/node/v20.6.1/bin/node ~/Downloads/SEM 5/Micro Services/prac10/main.js
cd db
node db
Connection successfull!
```

The screenshot shows the MongoDB Compass interface connected to a database named 'prac10'. The 'books' collection is selected, displaying the following information:

- Collections:** A list of collections including books, config, db2, local, and student.
- books Collection Details:**
 - Storage size: 4.10 kB
 - Documents: 0
 - Avg. document size: 0 B
 - Indexes: 1
 - Total index size: 4.10 kB

Practical 8.2: Creating Book Service.

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

const bookSchema = mongoose.Schema({
  title: {
    type: String,
    require: true
  },
  author: {
    type: String,
    require: true
  },
  numberPages: {
    type: Number,
    require: false
  },
  publisher: {
    type: String,
    require: false
  }
});

const Book = mongoose.model("book", bookSchema);

module.exports = Book;
```

Terminal output:

```
- cd books
- node Book
```

```
//require("dotenv").config();
const express = require('express');
const Book = require('../Book');

const app = express();
const port = 3000;
app.use(express.json());
app.post('/book', (req, res) => {
  const newBook = new Book({
    title:req.body.title,
    author:req.body.author,
    numberPages:req.body.numberPages,
    publisher:req.body.publisher,
  });
  newBook.save().then(() => {
    res.send('New Book created successfully!')
  }).catch((err) => {
    res.status(500).send('Internal Server Error!');
  })
});

found 0 vulnerabilities
```

Terminal output:

```
- node books
Up and Running on port 3000 - This is Book service
Connection successfull!
^C
```

→ Post data

The screenshot shows the Postman interface. A POST request is being made to `http://localhost:3000/book/`. The request body is a JSON object:

```
1  {
2     "title": "title",
3     "author": "author",
4     "numberPages": 23,
5     "publisher": "publisher"
6 }
```

The response status is 200 OK, and the message is "New Book created successfully".

The screenshot shows the MongoDB Compass interface connected to `localhost:27017`. A document in the `books.books` collection is displayed:

```
_id: ObjectId('651becd545fd3e01092723df')
title: "title"
author: "author"
numberPages: 23
publisher: "publisher"
```

→ Get data

The screenshot shows the Postman interface with a dark theme. On the left, the sidebar displays 'My Workspace' with a collection named 'exam_1' expanded, showing three items: 'New Collection', 'PRAC 10', and 'PRAC 7'. The main workspace shows a 'New Request' tab for 'PRAC 10'. The request method is 'GET' and the URL is 'http://localhost:3000/books'. The 'Params' tab is selected. In the 'Body' section, the response is displayed in 'Pretty' format:

```
1 {
2   "_id": "651becd545fd3e01092723df",
3   "title": "title",
4   "author": "author",
5   "numberPages": 23,
6   "publisher": "publisher",
7   "__v": 0
8 }
```

→ Get data by ID

The screenshot shows the Postman interface with a dark theme. On the left, the sidebar displays 'My Workspace' with a collection named 'exam_1' expanded, showing three items: 'New Collection', 'PRAC 10', and 'PRAC 7'. The main workspace shows a 'New Request' tab for 'PRAC 10'. The request method is 'GET' and the URL is 'http://localhost:3000/book/651becd545fd3e01092723df'. The 'Params' tab is selected. In the 'Body' section, the response is displayed in 'Pretty' format:

```
1 {
2   "_id": "651becd545fd3e01092723df",
3   "title": "title",
4   "author": "author",
5   "numberPages": 23,
6   "publisher": "publisher",
7   "__v": 0
8 }
```

→ Delete data by ID

The screenshot shows the Postman interface with a dark theme. On the left, the sidebar displays 'My Workspace' with a collection named 'exam_1' expanded, showing three requests: 'GET New Request', 'POST New Request', and 'DELETE New Request'. The main area shows a 'New Request' window for a 'DELETE' operation at the URL `http://localhost:3000/book/651becd545fd3e01092723df`. The 'Params' tab is selected, showing a single parameter 'Key' with 'Value' as 'Key'. The 'Body' tab shows the response body: `1 "Book deleted Successfully!"`. The status bar at the bottom indicates 'Status: 200 OK'.

This screenshot shows the same Postman interface. The 'exam_1' collection is still expanded, but the 'DELETE New Request' item is now highlighted. In the main area, a 'New Request' window is open for a 'GET' operation at the URL `http://localhost:3000/books`. The 'Params' tab is selected, showing a single parameter 'Key' with 'Value' as 'Key'. The 'Body' tab shows the response body: `1 Books not found`. The status bar at the bottom indicates 'Status: 404 Not Found'.

Practical 8.3: Creating Customer Service.

The screenshot shows the VS Code interface with the file `Customer.js` open in the editor. The code defines a `CustomerSchema` using Mongoose, specifying fields for name, age, and address. It then creates a `Customer` model based on this schema and exports it. The terminal below shows the command line history, including the creation of the `customers` directory and running the `Customer` script.

```
const mongoose = require('mongoose');
const CustomerSchema = mongoose.Schema({
  name: {
    type: String,
    require: true
  },
  age: {
    type: Number,
    require: true
  },
  address: {
    type: String,
    require: true
  }
});
const Customer = mongoose.model("customer", CustomerSchema);
module.exports = Customer;
```

```
cd books
node Book
cd ..
cd customers
node Customer
```

The screenshot shows the VS Code interface with the file `customers.js` open in the editor. The code implements two routes: a POST endpoint to create a new customer and a GET endpoint to find all customers. It uses the `Customer` model defined earlier. The terminal shows the command line history, including the creation of the `customers` directory and running the `customers` script. The output indicates the service is up and running on port 5000.

```
require('../db/db');
const Customer = require('./Customer');

const app = express();
const port = 5000;
app.use(express.json());

app.post('/customer', (req, res) => {
  const newCustomer = new Customer({...req.body});
  newCustomer.save().then(() => {
    res.send('New Customer created successfully!');
  }).catch((err) => {
    res.status(500).send('Internal Server Error!');
  })
}

app.get('/customers', (req, res) => {
  Customer.find().then((customers) => {
    if (customers) {
      res.send(customers);
    } else {
      res.send('No customers found');
    }
  })
})

app.listen(port, () => {
  console.log(`Up and Running on port ${port}- This is Customer service Connection successfull!`);
})
```

```
cd customers
node customers
```

The screenshot shows the MongoDB Compass interface at `localhost:27017`. On the left, the database structure is visible with the `books` database selected, and the `customers` collection is highlighted. The main area is titled `books.customers` and shows the following details:

- Documents: 0
- Indexes: 1

Below this, a search bar contains the query `{ field: 'value' }`. There are buttons for `Explain`, `Reset`, `Find`, and `Options`. A message states "This collection has no data" and provides an "Import Data" button.

→ Post data

The screenshot shows the Postman application interface. A POST request is being prepared to the URL `http://localhost:5000/customer/`. The request body is defined as follows:

```
1 {  
2   "name": "newname",  
3   "age": "88",  
4   "address": "not"  
5 }
```

The response status is 200 OK with a message: "New Customer created successfully".

localhost:27017

books.customers

0 DOCUMENTS 1 INDEXES

My Queries

Databases

Search

admin

books

books

customers

config

db2

local

student

ADD DATA EXPORT DATA

```
_id: ObjectId('651bf3129d9840e010443e4a')
name: "newname"
age: 88
address: "not"
__v: 0
```

1-1 of 1

→ Get data

Home Workspaces API Network Explore

My Workspace

New Import

exam_1

New Collection

PRAC 10

POST New Request

GET New Request

DEL New Request

POST New Request

GET New Request

PRAC 4

PRAC 5

PRAC 7

PRAC 10 / New Request

GET http://localhost:5000/customers

Params Authorization Headers (6) Body Pre-request Script Tests Settings Cookies

Key	Value	Description	Bulk Edit
Key	Value	Description	

Send

Status: 200 OK Time: 15 ms Size: 321 B Save as Example

Body Cookies Headers (7) Test Results

Pretty Raw Preview Visualize JSON

```
1 [ { 2   "_id": "651bf2b69d9840e010443e45", 3   "name": "newname", 4   "age": 88, 5   "address": "not", 6   "__v": 0 7 } ]
```

Online Find and replace Console Postbot Runner Start Proxy Cookies Trash

→ Get data by ID

The screenshot shows the Postman interface with a collection named "exam_1". A specific request titled "GET New Request" is selected. The request method is set to "GET" and the URL is "http://localhost:5000/customer/651bf2b69d9840e010443e45". The "Params" tab is active, showing a table with one row and two columns: "Key" and "Value". The "Body" tab is selected, displaying a JSON response with the following content:

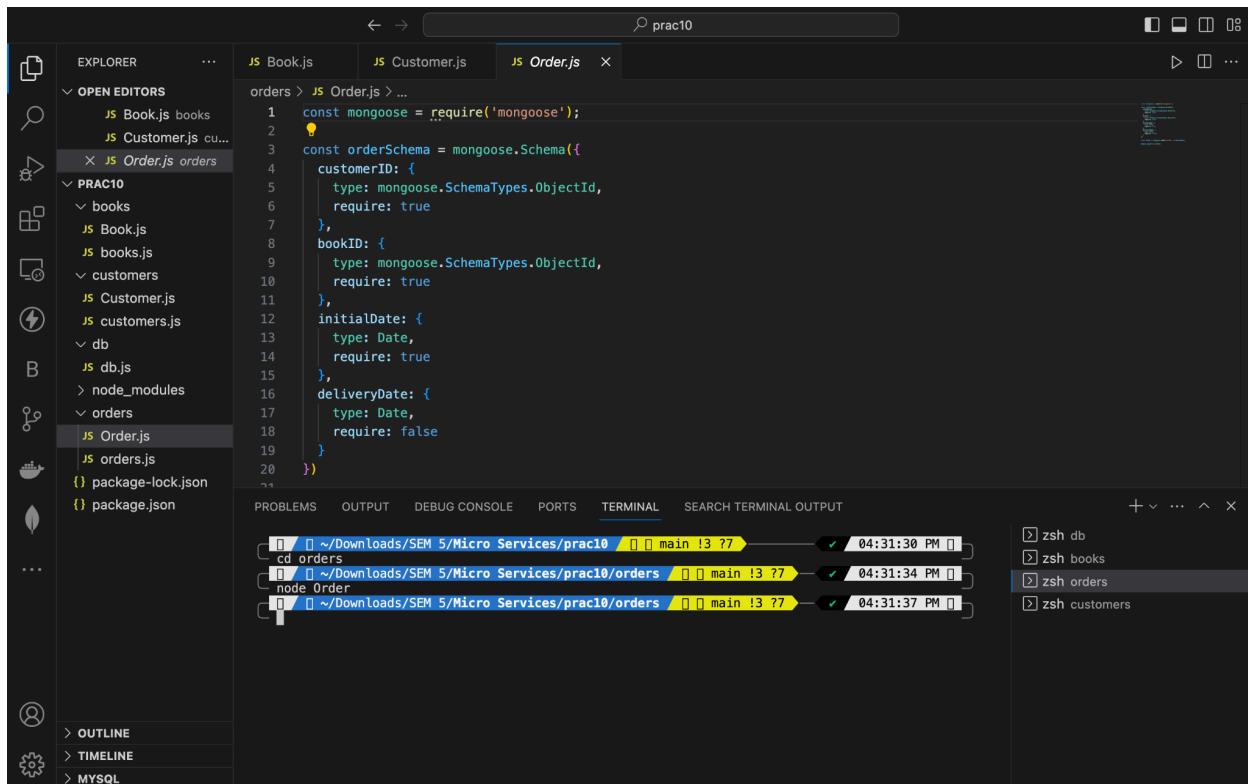
```
1  "_id": "651bf2b69d9840e010443e45",
2  "name": "newname",
3  "age": 88,
4  "address": "not",
5  "__v": 0
```

→ Delete data

The screenshot shows the Postman interface with the same "exam_1" collection. A request titled "DELETE New Request" is selected. The request method is set to "DELETE" and the URL is "http://localhost:5000/customer/651bf2b69d9840e010443e45". The "Params" tab is active, showing a table with one row and two columns: "Key" and "Value". The "Body" tab is selected, displaying a JSON response with the following content:

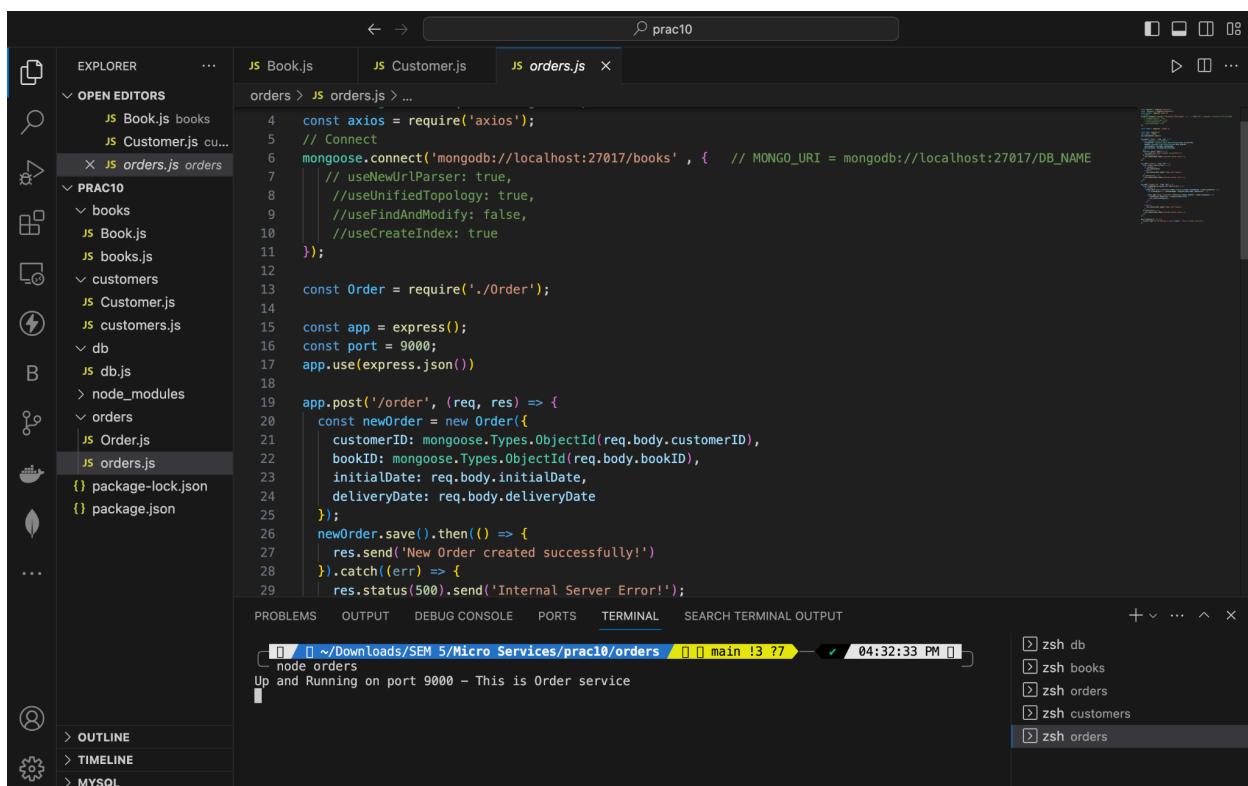
```
1  "customer deleted Successfully!"
```

Practical 8.4: Creating Order Service.



The screenshot shows the VS Code interface with the following details:

- Explorer View:** Shows a project structure under "PRAC10" with files: Book.js, books.js, Customer.js, customers.js, db.js, Order.js, orders.js, package-lock.json, and package.json.
- Editor View:** The "Order.js" tab is active, displaying the schema definition for an "Order".
- Terminal View:** The terminal shows three command-line sessions:
 - "cd orders"
 - "node Order"
 - "node Order"Each session has a timestamp: 04:31:30 PM, 04:31:34 PM, and 04:31:37 PM respectively.
- Output View:** Shows log entries for each command execution.
- Status Bar:** The status bar indicates the current workspace is "prac10".



The screenshot shows the VS Code interface with the following details:

- Explorer View:** Shows a project structure under "PRAC10" with files: Book.js, books.js, Customer.js, customers.js, db.js, Order.js, orders.js, package-lock.json, and package.json.
- Editor View:** The "orders.js" tab is active, displaying the code for an Express.js application that handles POST requests to "/order".
- Terminal View:** The terminal shows the command "node orders" being run, with the output: "Up and Running on port 9000 - This is Order service".
- Output View:** Shows log entries for the application startup.
- Status Bar:** The status bar indicates the current workspace is "prac10".

localhost:27017

Documents books.orders

My Queries Databases

Search

books

orders

config db2 local student

books.orders

0 DOCUMENTS 1 INDEXES

Documents Aggregations Schema Indexes Validation

Filter Type a query: { field: 'value' }

ADD DATA EXPORT DATA

0 - 0 of 0

This collection has no data

It only takes a few seconds to import data from a JSON or CSV file.

Import Data

> MONGOSH

The screenshot shows the Mongosh interface at localhost:27017. On the left, a sidebar lists databases: admin, books, config, db2, local, and student. The books database is selected, and its collections are listed: books, customers, and orders. The orders collection is currently selected. The main panel displays the books.orders collection. At the top right, it shows 0 DOCUMENTS and 1 INDEXES. Below that, a navigation bar includes Documents, Aggregations, Schema, Indexes, and Validation. A search bar is present with the placeholder "Type a query: { field: 'value' }". Below the search bar are buttons for ADD DATA and EXPORT DATA. The document list area shows 0 - 0 of 0. A message states "This collection has no data" and "It only takes a few seconds to import data from a JSON or CSV file." A green "Import Data" button is available. The bottom of the screen shows the command line prompt > MONGOSH.

→ Post data

The screenshot shows the Postman interface. On the left, the 'My Workspace' sidebar lists collections like 'exam_1', 'PRAC 10', 'PRAC 4', 'PRAC 5', and 'PRAC 7'. The main area shows a 'New Request' dialog for a 'POST' method to 'http://localhost:9000/order'. The 'Body' tab is selected, showing a JSON payload:

```
1 {
2   "customerID": "651bf5bc9d9840e010443e52",
3   "bookID": "651beec745fd3e01092723e7",
4   "initialDate": "2023-04-22",
5   "deliveryDate": "2023-04-27"
6 }
```

The 'Test Results' panel at the bottom displays the response: 'Status: 200 OK Time: 10 ms Size: 259 B' and the message 'New Order created successfully!'

The screenshot shows the MongoDB Compass interface connected to 'localhost:27017'. The left sidebar shows databases like 'admin', 'books', 'config', 'db2', 'local', and 'student', with 'orders' selected. The main area shows the 'books.orders' collection with 0 documents and 1 index. The 'Documents' tab is selected, displaying two documents:

```
_id: ObjectId('651bf96df16a21d58d864505')
customerID: ObjectId('651bf5bc9d9840e010443e52')
bookID: ObjectId('651beecf4fd3e01092723e5')
initialDate: 2023-04-12T00:00:00.000+00:00
deliveryDate: 2023-04-17T00:00:00.000+00:00
__v: 0
```

```
_id: ObjectId('651bf9bcf16a21d58d864507')
customerID: ObjectId('651bf5bc9d9840e010443e52')
bookID: ObjectId('651beec745fd3e01092723e7')
initialDate: 2023-04-22T00:00:00.000+00:00
deliveryDate: 2023-04-27T00:00:00.000+00:00
__v: 0
```

→ Get data

The screenshot shows the Postman interface with a collection named "PRAC 10". A new request is being created for the URL `http://localhost:9000/orders`. The "Params" tab is selected. In the "Body" tab, the response is displayed as a JSON array:

```
1 [  
2 {  
3     "_id": "651bf96df16a21d58d864505",  
4     "customerID": "651bf9bc9d9840e010443e50",  
5     "bookID": "651beeb145fd3e01092723e5",  
6     "initialDate": "2023-04-12T00:00:00.000Z",  
7     "deliveryDate": "2023-04-17T00:00:00.000Z",  
8     "__v": 0  
9 },  
10 {  
11     "_id": "651bf9bcf16a21d58d864507",  
12     "customerID": "651bf9bc9d9840e010443e52",  
13     "bookID": "651bec145fd3e01092723e7",  
14     "initialDate": "2023-04-22T00:00:00.000Z",  
15     "deliveryDate": "2023-04-27T00:00:00.000Z",  
16     "__v": 0  
17 }  
18 ]
```

→ Get data like CustomerName and BookTitle by id of order

The screenshot shows the Postman interface with the same collection "PRAC 10". A new request is being created for the URL `http://localhost:9000/order/651bf9bcf16a21d58d864507`. The "Params" tab is selected. In the "Body" tab, the response is displayed as a JSON object:

```
1 {  
2     "CustomerName": "newname12",  
3     "BookTitle": "title2"  
4 }
```

Postman screenshot showing a GET request to `http://localhost:5000/customers`. The response status is 200 OK with a time of 6 ms and a size of 511 B. The response body is:

```
7 |     "__v": 0
8 |
9 |   {
10|     "_id": "651bf5bc9d9840e010443e52",
11|     "name": "newname12",
12|     "age": 8812,
13|     "address": "no1t2",
14|     "__v": 0
15|
16|
17|   {
18|     "_id": "651bf5c59d9840e010443e54",
19|     "name": "newname123",
20|     "age": 88123,
21|     "address": "no1t23",
22|     "__v": 0
```

Postman screenshot showing a GET request to `http://localhost:3000/books`. The response status is 200 OK with a time of 10 ms and a size of 608 B. The response body is:

```
4 |   {
5|     "title": "title1",
6|     "author": "author1",
7|     "numberPages": 231,
8|     "publisher": "publisher1",
9|     "__v": 0
10|    },
11|    {
12|      "_id": "651beec745fd3e01092723e7",
13|      "title": "title2",
14|      "author": "author2",
15|      "numberPages": 2312,
16|      "publisher": "publisher12",
17|      "__v": 0
18|    },
19|    {
20|      "_id": "651beed045fd3e01092723e9",
```

