CO515: Advanced Computer Networks: Selected Topics - Lab 03

E/19/446: Wijerathna I.M.K.D.I.

04/04/2024

Lab Task

The following is the file structure I used for this implementation.

```
lab-03/
├── docker-compose.yaml
├── db/
│   └── init.js
└── web/
    ├── Dockerfile
    └── index.js
```

Figure 1: File Structure

Dockerfile used for the Web Service

```
# Use the official Node.js image
FROM node:latest

# Set the working directory in the container
WORKDIR /usr/src/app

# Initialize a new Node.js project without prompts
RUN npm init -y
# Install dependencies
RUN npm install express
RUN npm install mongodb
RUN npm install

# Copy the rest of the application code to the working directory
COPY . .

# Expose the port the app runs on
EXPOSE 3000
# Command to run the application
CMD ["node", "index.js"]
```

Figure 2: "./web/Dockerfile"

- *The DB consists of sample data on different computer brands, models and their prices.*

  Eg: { "brand": "Apple", "model": "MacBook Pro", "price": 2000 },

  { "brand": "Dell", "model": "XPS 15", "price": 1800 }, ...

- *The node-express web app interacts with this db and provides an API to perform crud operations (endpoint=> localhost:3000/data)*

The following code files are attached to the Appendix.

1. *"./web/index.js"*
2. *"./db/init.js"*

The overall configurations are defined in the "docker-compose.yaml"

```yaml
version: '3.8'

services:
  mongodb:
    image: mongodb/mongodb-community-server:latest
    ports:
      - "27017:27017"
    volumes:
      - ./db/init.js:/docker-entrypoint-initdb.d/init.js
    networks:
      - myproject_network

  web:
    build:
      context: ./web
      dockerfile: Dockerfile
    ports:
      - "3000:3000"
    depends_on:
      - mongodb
    networks:
      - myproject_network


networks:
  myproject_network:
```

Figure 3: "./docker-compose"

This Docker Compose file (Figure-3) defines two services: mongodb and web.

- *mongodb service:*
  - *Uses the mongodb/mongodb-community-server:latest image.*
  - *Maps port 27017 on the host to port 27017 in the container.*
  - *Mounts the volume, ./db/init.js to /docker-entrypoint-initdb.d/init.js*
  - *Connects to the myproject_network.*
- *web service:*
  - *Builds the image using the Dockerfile located in the ./web directory.*
  - *Maps port 3000 on the host to port 3000 in the container.*
  - *Depends on the mongodb service.*
  - *Connects to the myproject_network.*
- *Additionally, it defines a network named myproject_network to allow communication between the two services. This network is by default a bridge network.*

The command used to build and run the containers using docker-compose:

```
kalindu@kalindu-Inspiron-13-5310:~/SEM-6/CO515/Lab3/lab-proj$ sudo docker-compose up
Creating network "lab-proj_myproject_network" with the default driver
Creating lab-proj_mongodb_1 ... done
Creating lab-proj_web_1     ... done
Attaching to lab-proj_mongodb_1, lab-proj_web_1
```

Figure 4: starting the containers

To Test the Web app and DB, the API testing framework "POSTMAN" was used

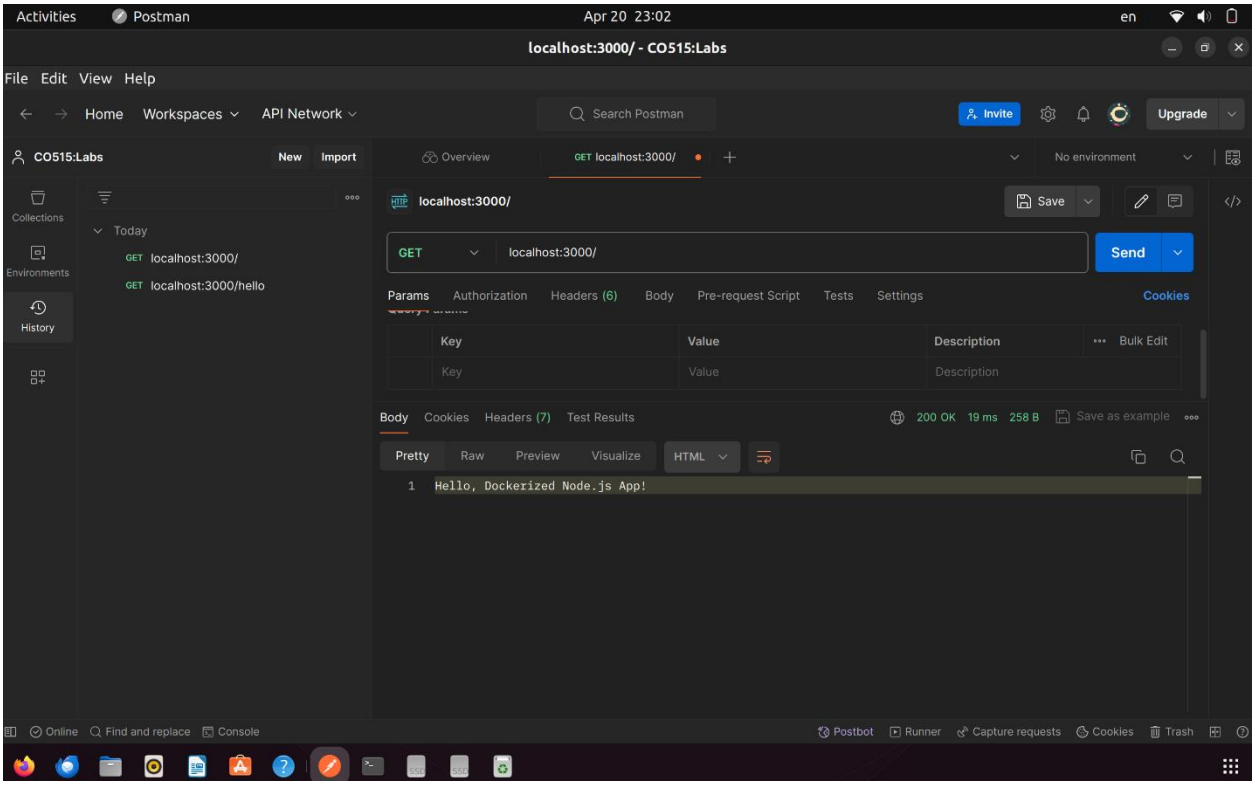


Figure 5: Overview of the POSTMAN workspace

Figure 6: Starting the Servers

Testing the connectivity to the web-app alone

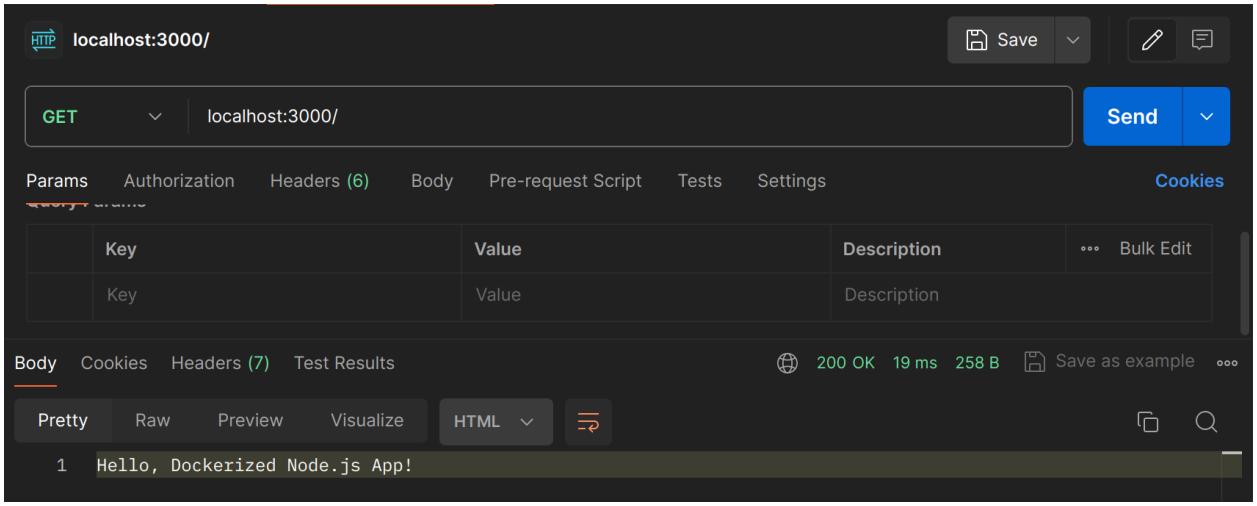
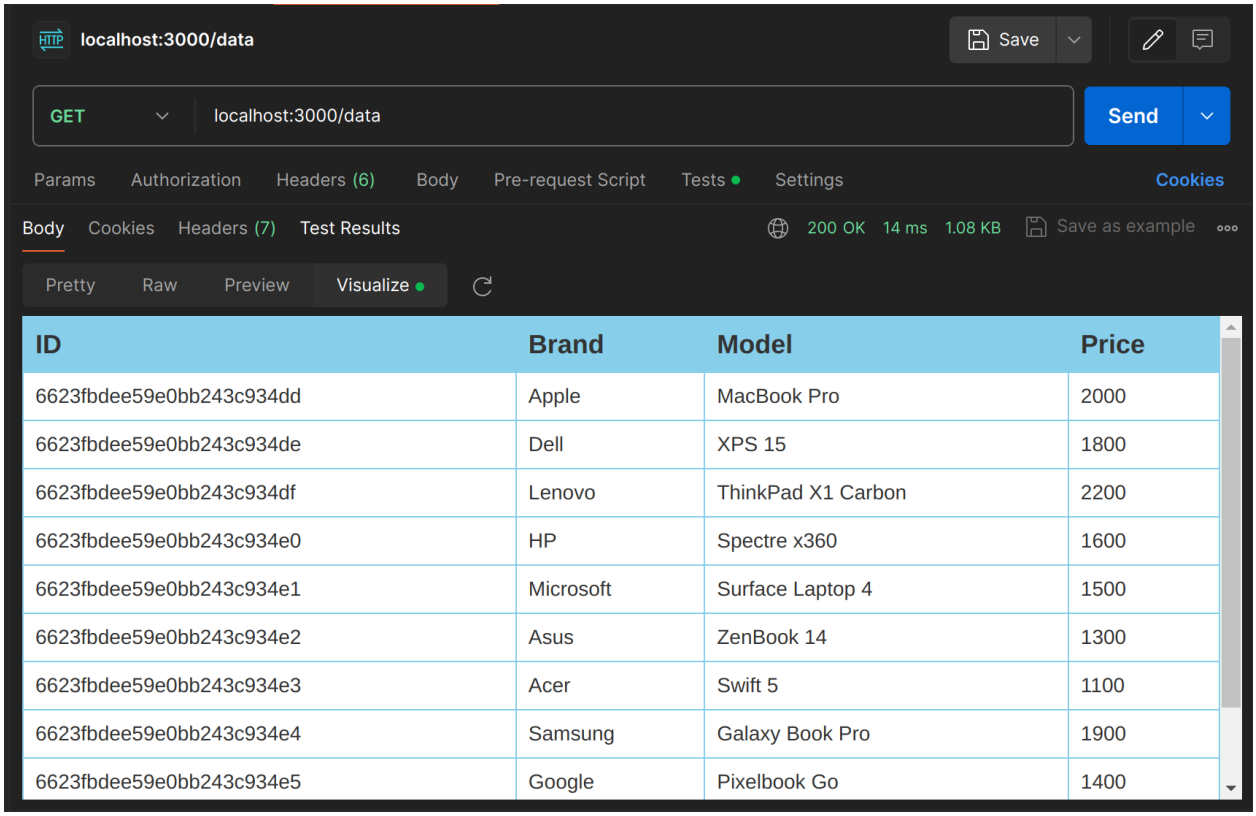


Figure 7: Testing web app connectivity

Testing the CRUD operations on the db

Read:



ID	Brand	Model	Price
6623fbdee59e0bb243c934dd	Apple	MacBook Pro	2000
6623fbdee59e0bb243c934de	Dell	XPS 15	1800
6623fbdee59e0bb243c934df	Lenovo	ThinkPad X1 Carbon	2200
6623fbdee59e0bb243c934e0	HP	Spectre x360	1600
6623fbdee59e0bb243c934e1	Microsoft	Surface Laptop 4	1500
6623fbdee59e0bb243c934e2	Asus	ZenBook 14	1300
6623fbdee59e0bb243c934e3	Acer	Swift 5	1100
6623fbdee59e0bb243c934e4	Samsung	Galaxy Book Pro	1900
6623fbdee59e0bb243c934e5	Google	Pixelbook Go	1400

Figure 8: Testing CRUD - Read

Create:

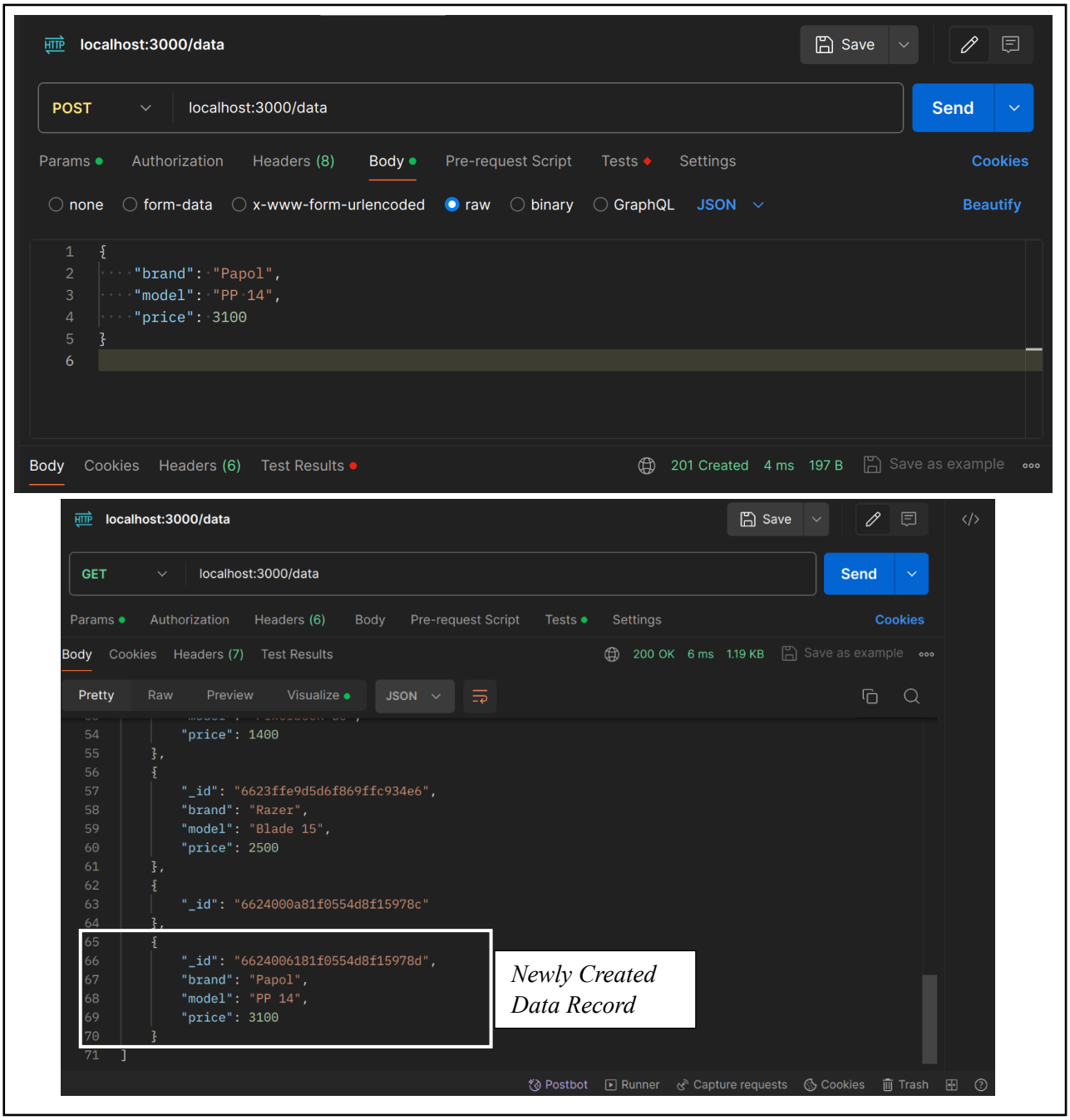


Figure 9: Testing CRUD - Create

Update:

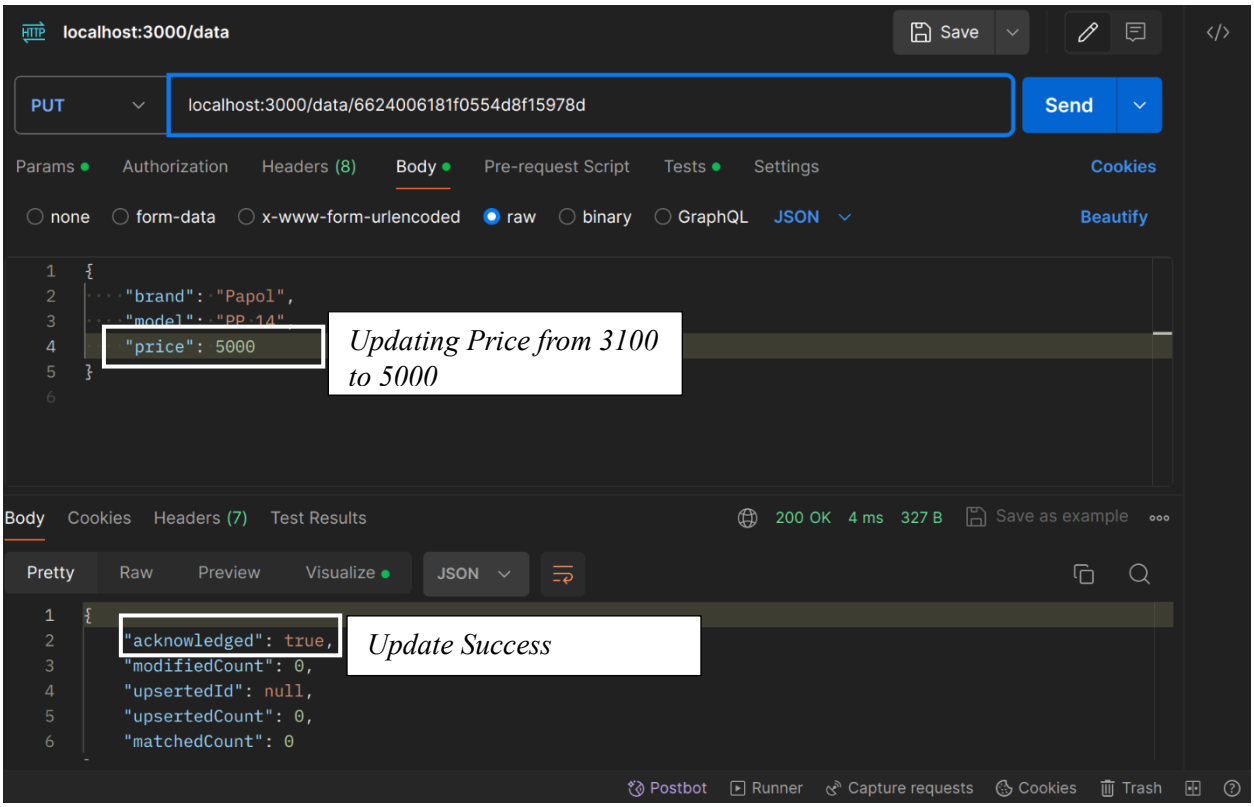


Figure 10: Testing CRUD - Update

Delete:

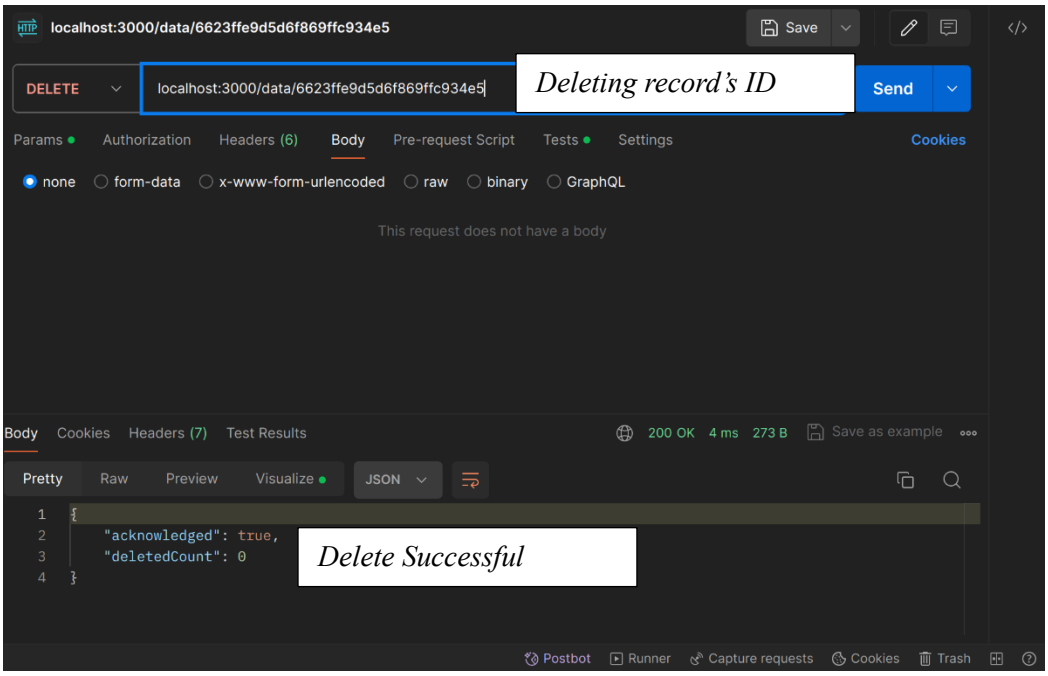


Figure 11: Testing CRUD - Delete

Cleaning Up the Containers

Ctrl+C:

Figure 12: Shutting down containers

Docker-compose down

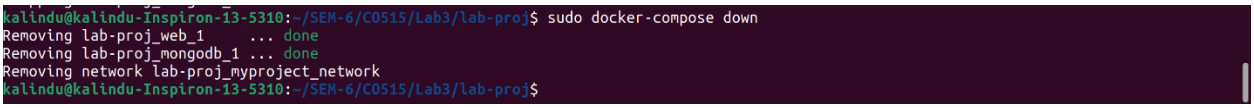


Figure 13: Ensure Cleaning up the Containers

Discussion Questions

1. **What is Docker Compose, and how does it differ from Docker?**
   - *Docker Compose is a tool for defining and running multi-container Docker applications. It allows us to use a YAML file to configure the services comprising our application and then deploy them with a single command.*
   - *While Docker is a platform for developing, shipping, and running applications in containers, Docker Compose specifically focuses on managing multi-container applications and simplifying their deployment.*

2. **What are the benefits of using Docker Compose for multi-container applications?**
   - *Simplified management: Docker Compose streamlines the management of multi-container applications by allowing us to define and configure all services in a single YAML file.*
   - *Easy orchestration: It simplifies the orchestration of multiple containers, enabling us to start, stop, and manage them collectively with a single command.*
   - *Consistent environments: Docker Compose ensures consistent environments across different development, testing, and production environments, making it easier to replicate and deploy applications.*
   - *Dependency management: It facilitates the management of dependencies between containers, ensuring that services are started in the correct order and can communicate with each other seamlessly.*

3. **Explain the purpose of each section in the docker-compose.yml file.**
   - *version: Specifies the Docker Compose file format version.*
   - *services: Defines the services that make up the application, each with its configuration.*
   - *networks: Configures networking options for the application's services.*

- *volumes: Specifies volumes that are shared among containers or with the host machine.*
- *configs and secrets (optional): Manage external configurations or secrets used by services.*
- *deploy (optional): Specifies deployment options when using Docker Swarm.*

4. **What are some real-world scenarios where Docker Compose can be useful?**
   - *Web Development Environments: Setting up consistent development environments for web applications with multiple services like databases, application servers, and caching systems.*
   - *Microservices Architectures: Orchestrating multiple microservices that work together to form a complete application, allowing for easy scaling and management.*
   - *Continuous Integration and Deployment (CI/CD): Defining and deploying complex application stacks for automated testing, staging, and production environments.*
   - *Local Testing and Debugging: Providing developers with a lightweight, reproducible environment for testing and debugging applications before deploying them to production.*
   - *Containerized Databases: Managing databases and associated services in isolated containers for development, testing, and production environments.*
   - *Multi-tier Applications: Building and deploying multi-tier applications with frontend, backend, and database components, ensuring consistency across environments.*
   - *Education and Training: Offering standardized environments for teaching Docker and containerization concepts, allowing students to experiment with multi-container applications easily.*
   - *DevOps Workflows: Streamlining the deployment and management of applications in DevOps workflows, enabling rapid iteration and delivery of software updates.*
   - *Demonstrations and Demos: Showcasing applications and prototypes in workshops, conferences, and demonstrations, providing a consistent environment for attendees to interact with.*
   - *SaaS Platforms: Building Software-as-a-Service (SaaS) platforms that require isolated environments for each tenant or customer, ensuring security and scalability.*

APPENDIX

1. *./web/index.js*

```
const express = require('express');
const { MongoClient } = require('mongodb');

const app = express();
const PORT = process.env.PORT || 3000;
const MONGODB_URI = 'mongodb://mongodb:27017/mydatabase'; // MongoDB URI

app.use(express.json()); // Parse JSON bodies
app.use(express.urlencoded({ extended: true })); // Parse URL-encoded bodies

// Connect to MongoDB
MongoClient.connect(MONGODB_URI, { useNewUrlParser: true, useUnifiedTopology: true })
  .then(client => {
    console.log('Connected to MongoDB');
    const db = client.db();

// Define routes
app.get('/', (req, res) => {
  console.log('Received request for /');
  res.send('Hello, Dockerized Node.js App!');
});

// Get data from MongoDB
app.get('/data', async (req, res) => {
  try {
    console.log('Received request for /data');

    // Ensure that the MongoDB connection is established
    if (!db) {
      console.error('MongoDB connection is not established');
      return res.status(500).send('Internal Server Error');
    }

    // Fetch data from MongoDB
    const data = await db.collection('laptops').find().toArray();

    console.log('Data fetched successfully:', data);

    // Send the fetched data as a JSON response
    res.json(data);
  } catch (error) {
```

```
    console.error('Error fetching data:', error);
    res.status(500).send('Internal Server Error');
  }
});

    // Add new data to MongoDB
    app.post('/data', async (req, res) => {
      try {
        // Assuming req.body contains the new data object
        const newData = req.body;
        const result = await db.collection('laptops').insertOne(newData);
        res.status(201).json(result.ops);
      } catch (error) {
        console.error('Error adding data:', error);
        res.status(500).send('Internal Server Error');
      }
    });

    // Update data in MongoDB
    app.put('/data/:id', async (req, res) => {
      try {
        const id = req.params.id;
        const newData = req.body;
        const result = await db.collection('laptops').updateOne({ _id: id }, { $set: newData });
        res.json(result);
      } catch (error) {
        console.error('Error updating data:', error);
        res.status(500).send('Internal Server Error');
      }
    });

    // Delete data from MongoDB
    app.delete('/data/:id', async (req, res) => {
      try {
        const id = req.params.id;
        const result = await db.collection('laptops').deleteOne({ _id: id });
        res.json(result);
      } catch (error) {
        console.error('Error deleting data:', error);
        res.status(500).send('Internal Server Error');
      }
    });
```

```
    // Start the server
    app.listen(PORT, () => {
      console.log(`Server is running on port ${PORT}`);
    });
  })
  .catch(error => {
    console.error('Error connecting to MongoDB:', error);
    process.exit(1); // Exit the process if unable to connect to MongoDB
  });
```

2. *./db/init.js*

```
// MongoDB Initialization Script

// Sample data
var sampleData = [
  { "brand": "Apple", "model": "MacBook Pro", "price": 2000 },
  { "brand": "Dell", "model": "XPS 15", "price": 1800 },
  { "brand": "Lenovo", "model": "ThinkPad X1 Carbon", "price": 2200 },
  { "brand": "HP", "model": "Spectre x360", "price": 1600 },
  { "brand": "Microsoft", "model": "Surface Laptop 4", "price": 1500 },
  { "brand": "Asus", "model": "ZenBook 14", "price": 1300 },
  { "brand": "Acer", "model": "Swift 5", "price": 1100 },
  { "brand": "Samsung", "model": "Galaxy Book Pro", "price": 1900 },
  { "brand": "Google", "model": "Pixelbook Go", "price": 1400 },
  { "brand": "Razer", "model": "Blade 15", "price": 2500 }
];

// Connect to the database
var conn = new Mongo();
var db = conn.getDB('mydatabase');

// Insert sample data into a collection
db.laptops.insertMany(sampleData);
```