Name: Devireddy Harika

Roll Number: 20NN1A05D3

Vignan's Nirula Institute of Technology and Science for Women

FULL STACK WITH MERN

ASSIGNMENT-III

Aim: Created a RESTful API using Express.js and integrated it with MongoDB to perform CRUD operations on a database.

- Here we used two main files named “app.js” and “package.json”.
- We installed Thunder Client to create RESTful API and perform CRUD operations.
- The output of the operations performed are shown in the terminal rather than in the local host.

Code in **app.js**

```
const express = require('express');
const mongoose = require('mongoose');
const bodyParser = require('body-parser');

const app = express();
const PORT = process.env.PORT || 3000;

// Middleware
app.use(bodyParser.json());

// Connect to MongoDB
mongoose.connect('mongodb://localhost:27017/test/developer', {
  useNewUrlParser: true,
  useUnifiedTopology: true,
});
const db = mongoose.connection;
db.on('error', console.error.bind(console, 'MongoDB connection error:'));

// Define schema and model for MongoDB
const Schema = mongoose.Schema;
const myResourceSchema = new Schema({
  name: String,
  description: String,
});
```

```
const MyResource = mongoose.model('MyResource', myResourceSchema);

// Routes
// Create
app.post('/api/resources', async (req, res) => {
  try {
    const { name, description } = req.body;
    const newResource = new MyResource({ name, description });
    await newResource.save();
    res.status(201).json(newResource);
  } catch (err) {
    res.status(500).json({ message: err.message });
  }
});

// Read
app.get('/api/resources', async (req, res) => {
  try {
    const resources = await MyResource.find();
    res.json(resources);
  } catch (err) {
    res.status(500).json({ message: err.message });
  }
});

// Update
app.put('/api/resources/:id', async (req, res) => {
  try {
    const { id } = req.params;
    const { name, description } = req.body;
    const updatedResource = await MyResource.findByIdAndUpdate(
      id,
      { name, description },
      { new: true }
    );
    res.json(updatedResource);
  } catch (err) {
    res.status(500).json({ message: err.message });
  }
});

// Delete
app.delete('/api/resources/:id', async (req, res) => {
  try {
    const { id } = req.params;
    await MyResource.findByIdAndDelete(id);
    res.json({ message: 'Resource deleted' });
  } catch (err) {
```

```
    res.status(500).json({ message: err.message });
  }
});

// Start server
app.listen(PORT, () => {
  console.log(`Server is running on http://localhost:${PORT}`);
});
```

The **app.js** file contains the implementation of a RESTful API using Express.js and integration with MongoDB to perform CRUD operations on a database. Here's a breakdown of its contents:

- **Dependencies**: The file requires the necessary Node.js modules such as **express**, **mongoose**, and **body-parser**.
- **Express App Setup**: It sets up an Express application using **express()** and assigns it to the variable **app**.
- **Middleware**: It uses **body-parser** middleware to parse incoming JSON requests.
- **MongoDB Connection**: It establishes a connection to a MongoDB database named "test" with the collection "developer" using Mongoose.
- **Schema and Model Definition**: It defines a schema for MongoDB using Mongoose's **Schema** constructor and creates a model named **MyResource**.
- **Routes**: It defines routes for performing CRUD operations:

    POST /api/resources: Creates a new resource.

    GET /api/resources: Retrieves all resources.

    PUT /api/resources/:id: Updates an existing resource by its ID.

    DELETE /api/resources/:id: Deletes an existing resource by its ID.
- **Error Handling**: It handles errors that may occur during CRUD operations by responding with appropriate status codes and error messages in JSON format.
- **Server Start**: It starts the Express server, listening on the specified port (default is 3000), and logs the server's URL upon successful startup.

Code in **package.json**

```
{
  "name": "express-mongodb-api",
  "version": "1.0.0",
  "description": "RESTful API using Express.js and MongoDB",
  "main": "app.js",
  "scripts": {
```

```
    "start": "node app.js"
  },
  "dependencies": {
    "body-parser": "^1.19.0",
    "express": "^4.17.1",
    "mongoose": "^5.11.15"
  }
}
```

The **package.json** file contains metadata about the Node.js project and its dependencies. Here's a breakdown of its contents:

- **Name**: The name of the project.
- **Version**: The version of the project.
- **Description**: A brief description of the project.
- **Main**: The entry point file for the Node.js application.
- **Scripts**: Defines npm scripts, such as the "start" script to run the application (**node app.js**).
- **Dependencies**: Lists the dependencies required for the project, including **express**, **mongoose**, and **body-parser**, along with their respective versions.

Readme.md

```
This provides a RESTful API for CRUD operations on a MongoDB database. It
allows you to manage resources with basic functionalities such as create,
read, update, and delete.


Navigate to the project directory:
cd express-mongodb-api


Install dependencies:
npm install
Make sure you have MongoDB installed and running locally on the default port
(27017).

Running the Application
To start the Express.js server, following command is runned,
npm start
The server will start running on http://localhost:3000 by default.

Routes and Functionality
```

```
POST /api/resources
Description: Creates a new resource.
Request Body: JSON object with properties name and description.
Example:
{
  "name": "New Resource",
  "description": "This is a new resource"
}
Response: JSON object of the created resource.

GET /api/resources
Description: Retrieves all resources.
Response: JSON array of all resources.

PUT /api/resources/:id
Description: Updates an existing resource by its ID.
Request Body: JSON object with properties name and description.
Example:
{
  "name": "Updated Resource",
  "description": "This resource has been updated"
}
Response: JSON object of the updated resource.

DELETE /api/resources/:id
Description: Deletes an existing resource by its ID.
Response: JSON object with a success message.

Error Handling
In case of any errors during API operations, the server responds with
appropriate status codes and error messages in JSON format.
```

The **Readme.md** file provides instructions and information about the project. Here's a summary of its contents:

- **Description**: Describes the purpose of the project, which is to provide a RESTful API for CRUD operations on a MongoDB database.
- **Installation**: Provides instructions on how to install dependencies using **npm install**.
- **Running the Application**: Explains how to start the Express.js server using the **npm start** command and specifies the default server URL (**http://localhost:3000**).
- **Routes and Functionality**: Describes the available routes and their corresponding functionalities for creating, reading, updating, and deleting resources.
- **Error Handling**: Mentions how the server handles errors during API operations by responding with appropriate status codes and error messages in JSON format.