Practical 7

Project: User Profile Manager

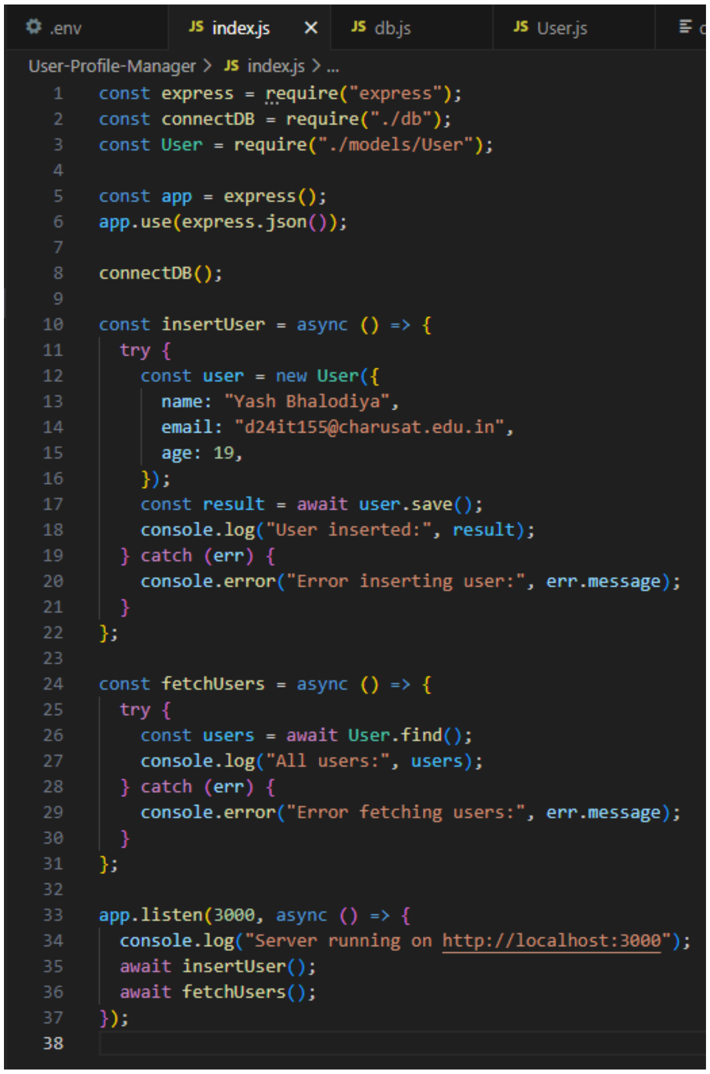
Objective: Set up MongoDB and Mongoose in a Node.js application and create a basic schema to perform simple database operations.

Tasks

- Install MongoDB locally or create a free cluster on MongoDB Atlas.
- Set up a new Node.js project and install the required dependencies (express, mongoose, dotenv).
- Create a .env file and store your MongoDB connection URI securely.
- Write a connection script in Node.js using Mongoose to connect to MongoDB.
- Define a User schema with fields like name, email, and age.
- Create a simple script to insert a user into the database and log the results in the console.
- Fetch all users from the database and display them in the console

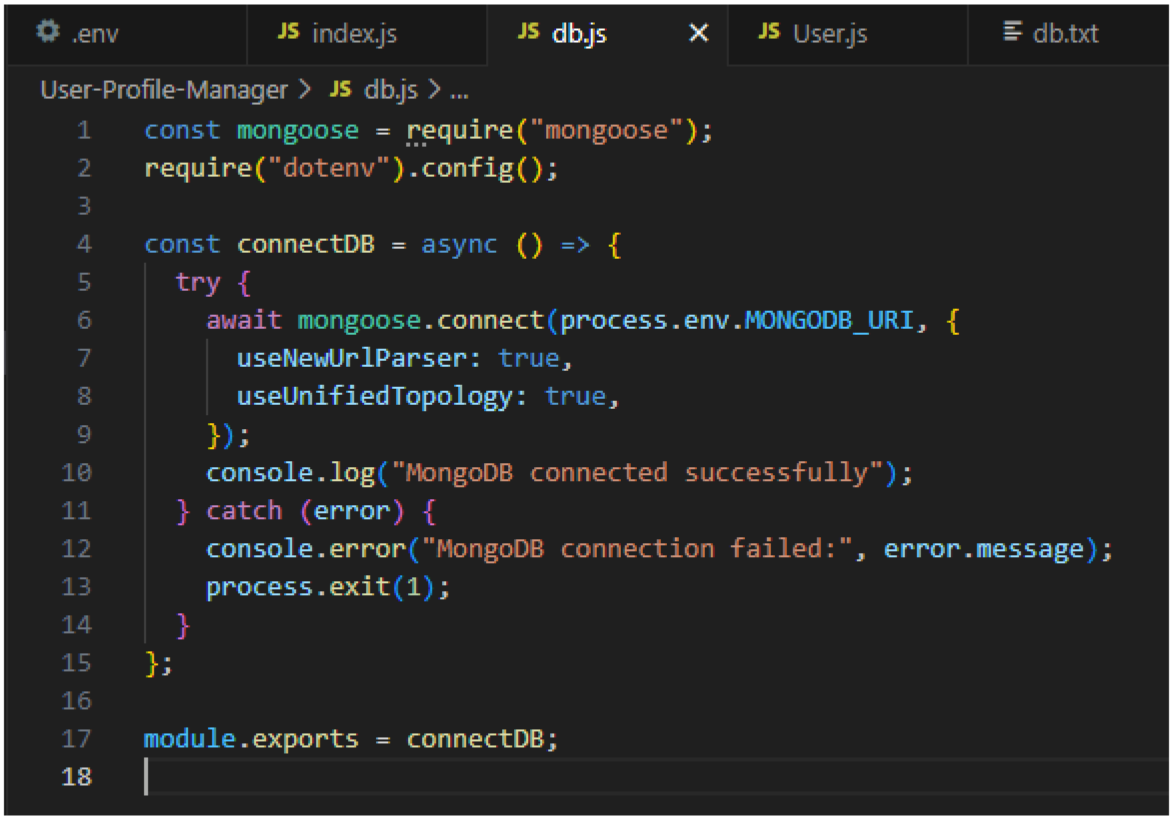
Screenshot of code:

index.js



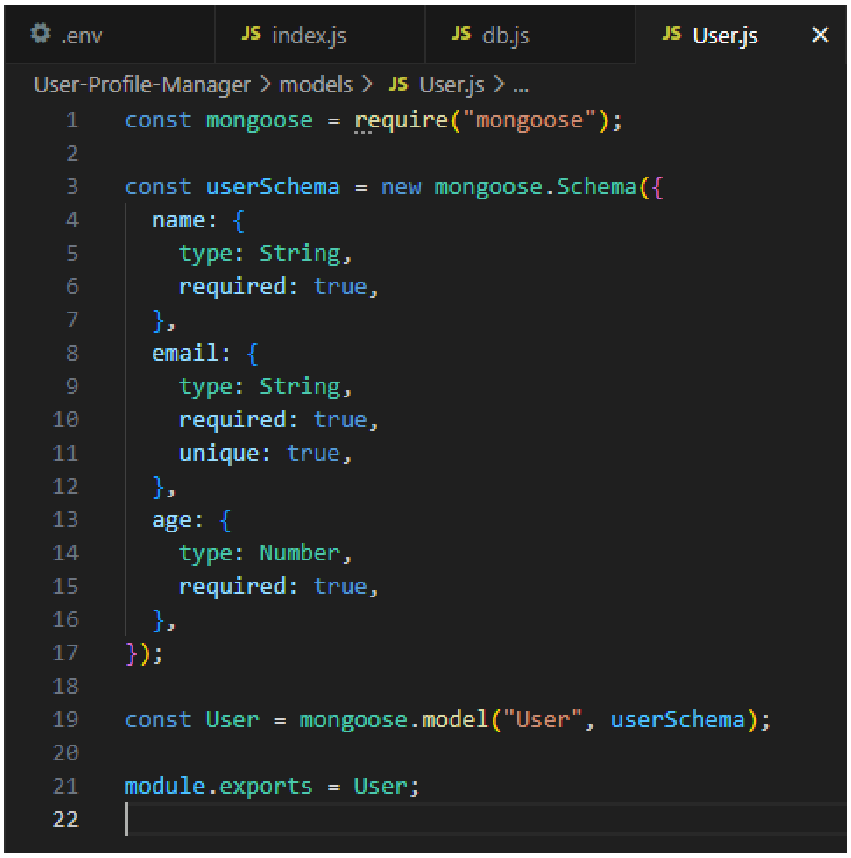
```javascript
const express = require("express");
const connectDB = require("./db");
const User = require("./models/User");

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

connectDB();

const insertUser = async () => {
  try {
    const user = new User({
      name: "Yash Bhalodiya",
      email: "d24it155@charusat.edu.in",
      age: 19,
    });
    const result = await user.save();
    console.log("User inserted:", result);
  } catch (err) {
    console.error("Error inserting user:", err.message);
  }
};

const fetchUsers = async () => {
  try {
    const users = await User.find();
    console.log("All users:", users);
  } catch (err) {
    console.error("Error fetching users:", err.message);
  }
};

app.listen(3000, async () => {
  console.log("Server running on http://localhost:3000");
  await insertUser();
  await fetchUsers();
});

```

db.js



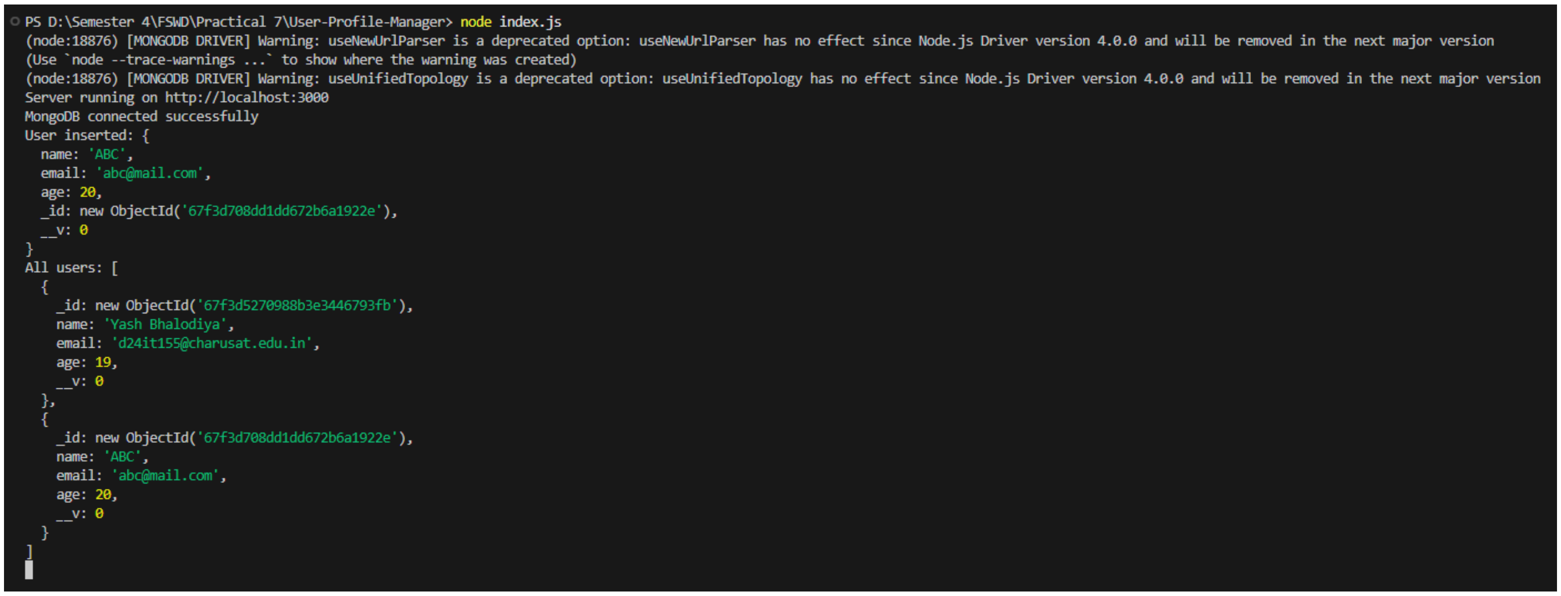
```
const mongoose = require("mongoose");
require("dotenv").config();

const connectDB = async () => {
  try {
    await mongoose.connect(process.env.MONGODB_URI, {
      useNewUrlParser: true,
      useUnifiedTopology: true,
    });
    console.log("MongoDB connected successfully");
  } catch (error) {
    console.error("MongoDB connection failed:", error.message);
    process.exit(1);
  }
};

module.exports = connectDB;
```

models/User.js

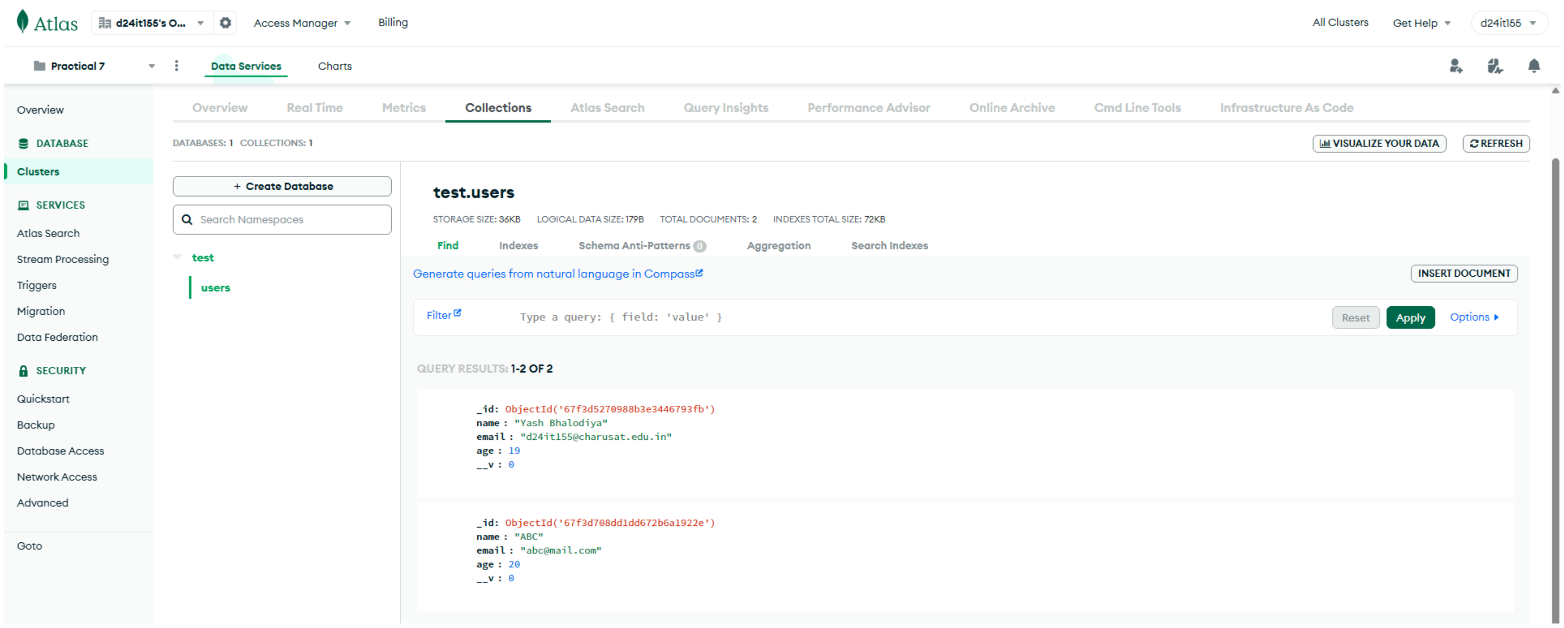


```
const mongoose = require("mongoose");

const userSchema = new mongoose.Schema({
  name: {
    type: String,
    required: true,
  },
  email: {
    type: String,
    required: true,
    unique: true,
  },
  age: {
    type: Number,
    required: true,
  },
});

const User = mongoose.model("User", userSchema);

module.exports = User;
```

Screenshot of Output:



```
PS D:\Semester 4\FSWD\Practical 7\User-Profile-Manager> node index.js
(node:18876) [MONGODB DRIVER] Warning: useNewUrlParser is a deprecated option: useNewUrlParser has no effect since Node.js Driver version 4.0.0 and will be removed in the next major version
(Use `node --trace-warnings ...` to show where the warning was created)
(node:18876) [MONGODB DRIVER] Warning: useUnifiedTopology is a deprecated option: useUnifiedTopology has no effect since Node.js Driver version 4.0.0 and will be removed in the next major version
Server running on http://localhost:3000
MongoDB connected successfully
User inserted: {
  name: 'ABC',
  email: 'abc@mail.com',
  age: 20,
  _id: new ObjectId('67f3d708dd1dd672b6a1922e'),
  __v: 0
}
All users: [
  {
    _id: new ObjectId('67f3d5270988b3e3446793fb'),
    name: 'Yash Bhalodiya',
    email: 'd24it155@charusat.edu.in',
    age: 19,
    __v: 0
  },
  {
    _id: new ObjectId('67f3d708dd1dd672b6a1922e'),
    name: 'ABC',
    email: 'abc@mail.com',
    age: 20,
    __v: 0
  }
]
```



DATABASES: 1 COLLECTIONS: 1

test.users

STORAGE SIZE: 36KB LOGICAL DATA SIZE: 179B TOTAL DOCUMENTS: 2 INDEXES TOTAL SIZE: 72KB

QUERY RESULTS: 1-2 OF 2

```
_id: ObjectId('67f3d5270988b3e3446793fb')
name : "Yash Bhalodiya"
email : "d24it155@charusat.edu.in"
age : 19
__v : 0

_id: ObjectId('67f3d708dd1dd672b6a1922e')
name : "ABC"
email : "abc@mail.com"
age : 20
__v : 0
```

Project: Task Manager API

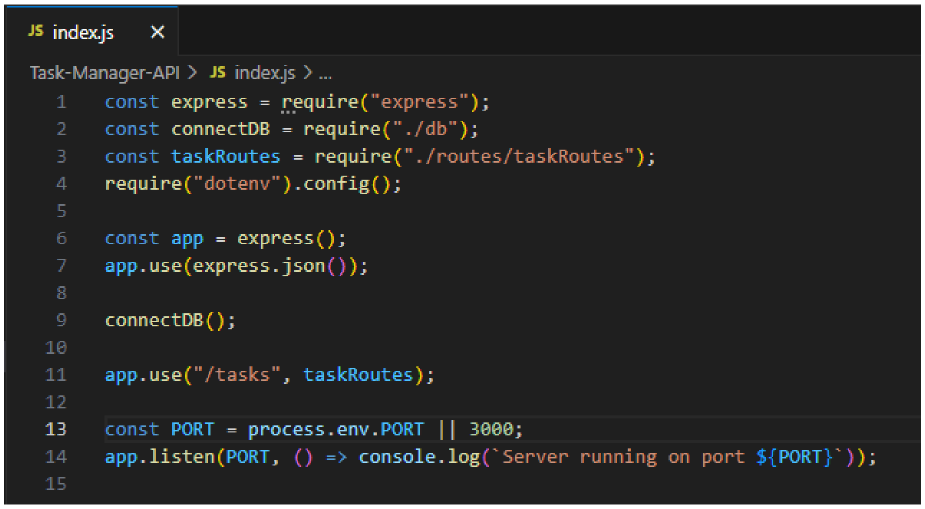
Objective: Build a RESTful API with Express and Mongoose to manage tasks in a MongoDB collection.

Tasks

- Define a Task schema with fields like title, description, status (e.g., "Pending", "Completed"), and dueDate.
- Set up the following API endpoints:
- POST /tasks: Add a new task to the database.
- GET /tasks: Retrieve all tasks.
- GET /tasks/:id: Retrieve a specific task by ID.
- PUT /tasks/:id: Update task details by ID.
- DELETE /tasks/:id: Delete a task by ID.
- Test the API endpoints using Postman or Thunder Client.
- Use filters to query tasks based on their status or dueDate.
- Handle basic errors, such as invalid task IDs or missing fields.

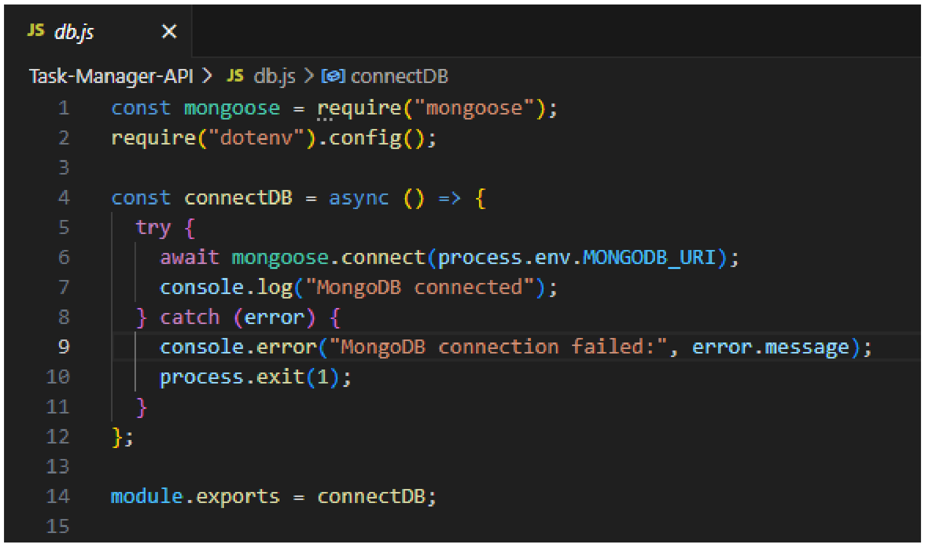
Screenshot of Code:

index.js



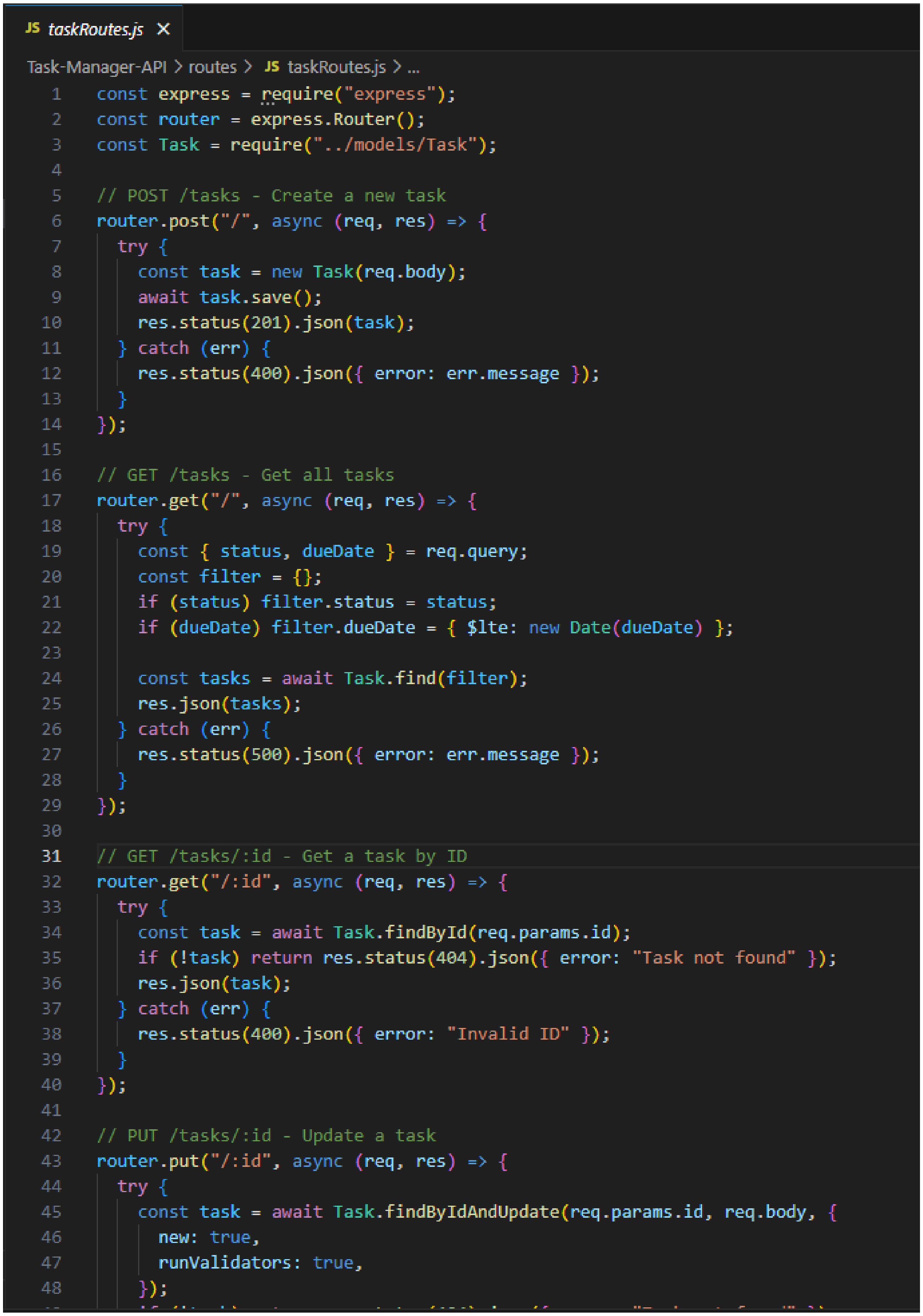
```
const express = require("express");
const connectDB = require("./db");
const taskRoutes = require("./routes/taskRoutes");
require("dotenv").config();

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

connectDB();

app.use("/tasks", taskRoutes);

const PORT = process.env.PORT || 3000;
app.listen(PORT, () => console.log(`Server running on port ${PORT}`));
```

db.js



```
const mongoose = require("mongoose");
require("dotenv").config();

const connectDB = async () => {
  try {
    await mongoose.connect(process.env.MONGODB_URI);
    console.log("MongoDB connected");
  } catch (error) {
    console.error("MongoDB connection failed:", error.message);
    process.exit(1);
  }
};

module.exports = connectDB;
```

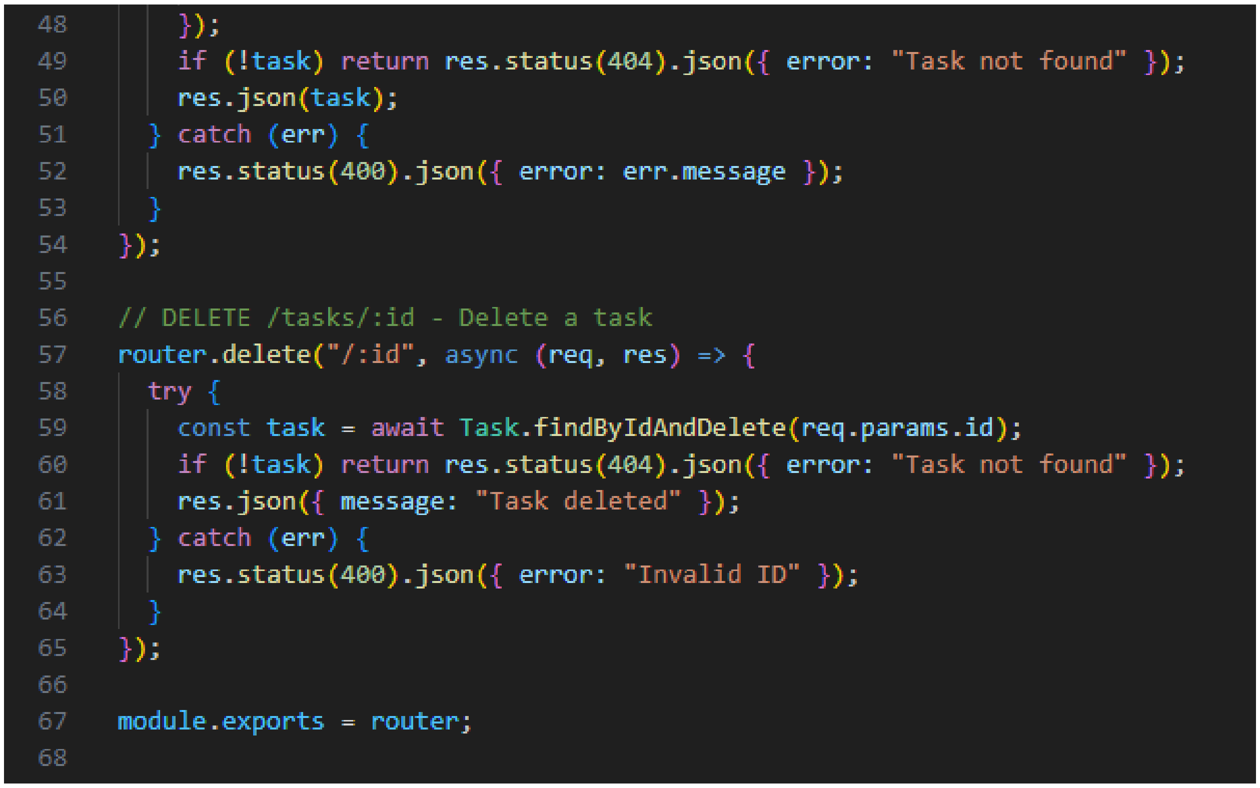
routes/taskRoutes.js



taskRoutes.js ×

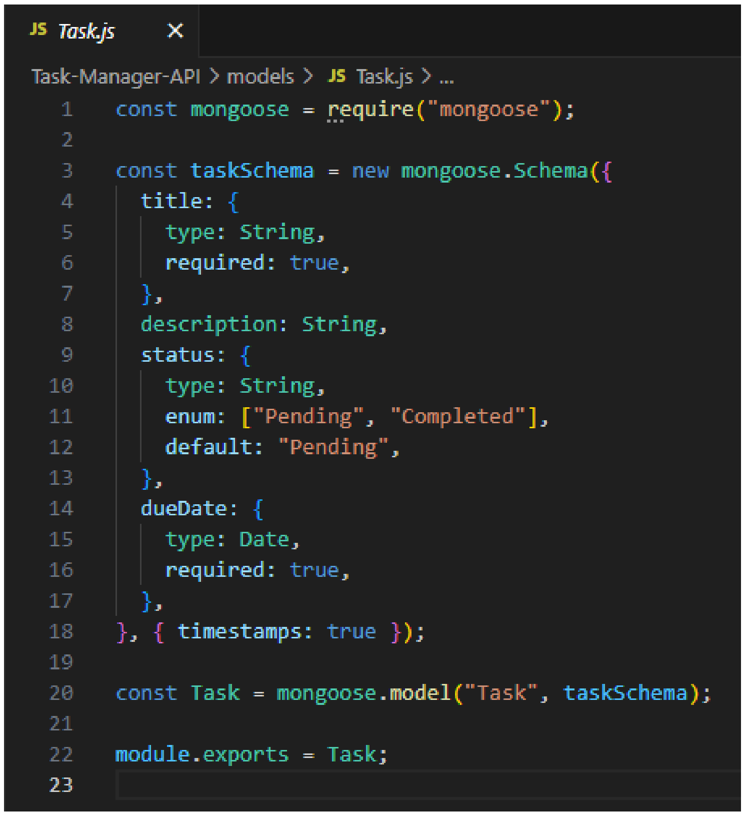
Task-Manager-API > routes > taskRoutes.js > ...

```javascript
const express = require("express");
const router = express.Router();
const Task = require("../models/Task");

// POST /tasks - Create a new task
router.post("/", async (req, res) => {
  try {
    const task = new Task(req.body);
    await task.save();
    res.status(201).json(task);
  } catch (err) {
    res.status(400).json({ error: err.message });
  }
});

// GET /tasks - Get all tasks
router.get("/", async (req, res) => {
  try {
    const { status, dueDate } = req.query;
    const filter = {};
    if (status) filter.status = status;
    if (dueDate) filter.dueDate = { $lte: new Date(dueDate) };

    const tasks = await Task.find(filter);
    res.json(tasks);
  } catch (err) {
    res.status(500).json({ error: err.message });
  }
});

// GET /tasks/:id - Get a task by ID
router.get("/:id", async (req, res) => {
  try {
    const task = await Task.findById(req.params.id);
    if (!task) return res.status(404).json({ error: "Task not found" });
    res.json(task);
  } catch (err) {
    res.status(400).json({ error: "Invalid ID" });
  }
});

// PUT /tasks/:id - Update a task
router.put("/:id", async (req, res) => {
  try {
    const task = await Task.findByIdAndUpdate(req.params.id, req.body, {
      new: true,
      runValidators: true,
    });
```



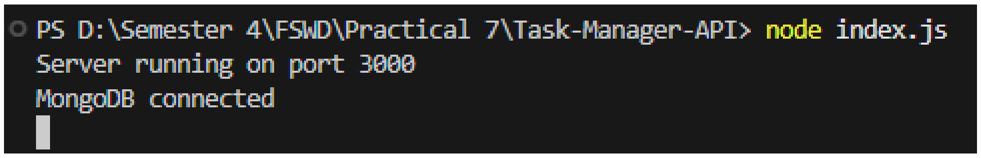
```
    });
    if (!task) return res.status(404).json({ error: "Task not found" });
    res.json(task);
  } catch (err) {
    res.status(400).json({ error: err.message });
  }
});

// DELETE /tasks/:id - Delete a task
router.delete("/:id", async (req, res) => {
  try {
    const task = await Task.findByIdAndDelete(req.params.id);
    if (!task) return res.status(404).json({ error: "Task not found" });
    res.json({ message: "Task deleted" });
  } catch (err) {
    res.status(400).json({ error: "Invalid ID" });
  }
});

module.exports = router;
```

models/Task.js

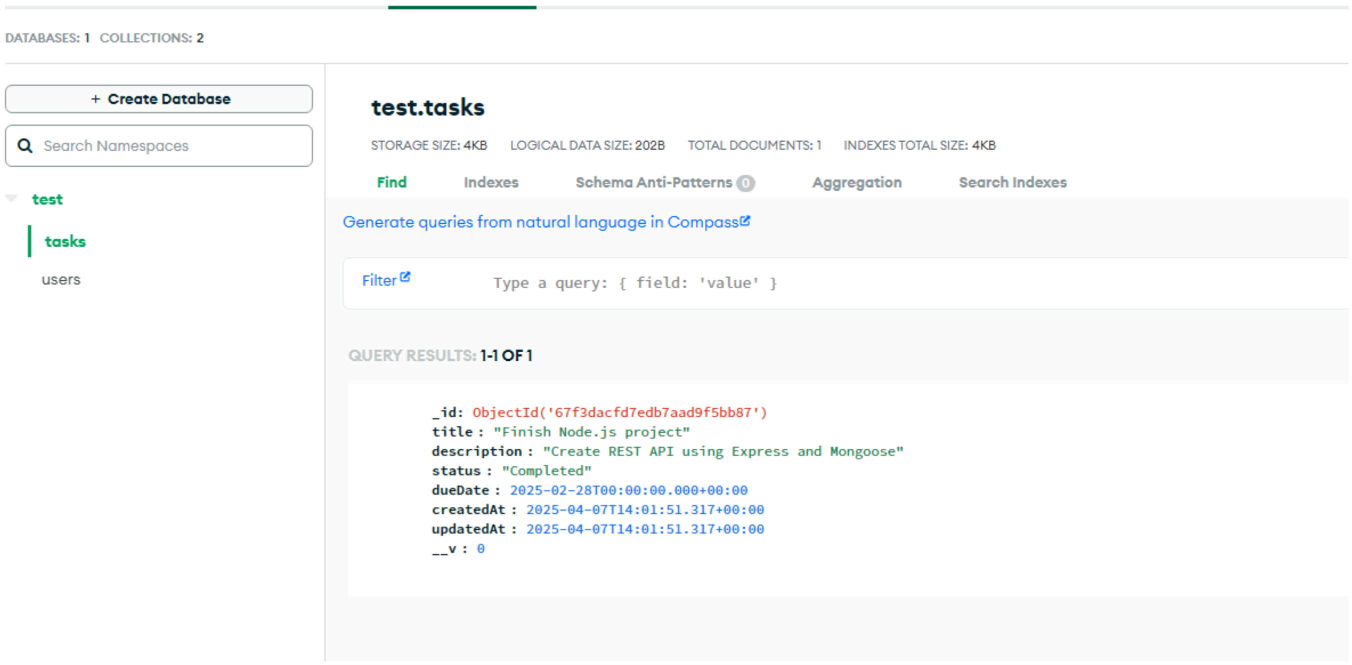


```
const mongoose = require("mongoose");

const taskSchema = new mongoose.Schema({
  title: {
    type: String,
    required: true,
  },
  description: String,
  status: {
    type: String,
    enum: ["Pending", "Completed"],
    default: "Pending",
  },
  dueDate: {
    type: Date,
    required: true,
  },
}, { timestamps: true });

const Task = mongoose.model("Task", taskSchema);

module.exports = Task;
```

Output:



```
PS D:\Semester 4\FSWD\Practical 7\Task-Manager-API> node index.js
Server running on port 3000
MongoDB connected
```

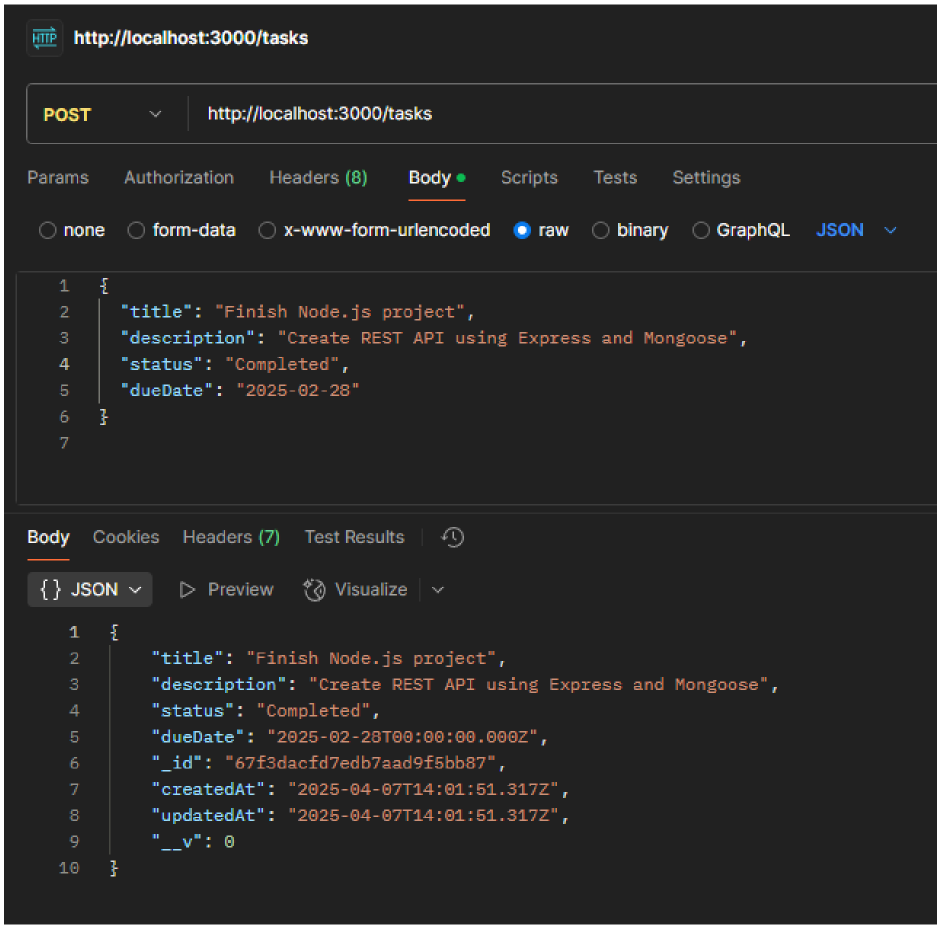


test.tasks

STORAGE SIZE: 4KB LOGICAL DATA SIZE: 202B TOTAL DOCUMENTS: 1 INDEXES TOTAL SIZE: 4KB

QUERY RESULTS: 1-1 OF 1

```
_id: ObjectId('67f3dacfd7edb7aad9f5bb87')
title : "Finish Node.js project"
description : "Create REST API using Express and Mongoose"
status : "Completed"
dueDate : 2025-02-28T00:00:00.000+00:00
createdAt : 2025-04-07T14:01:51.317+00:00
updatedAt : 2025-04-07T14:01:51.317+00:00
__v : 0
```



POST http://localhost:3000/tasks

```
{
  "title": "Finish Node.js project",
  "description": "Create REST API using Express and Mongoose",
  "status": "Completed",
  "dueDate": "2025-02-28"
}
```

Response Body: JSON

```
{
    "title": "Finish Node.js project",
    "description": "Create REST API using Express and Mongoose",
    "status": "Completed",
    "dueDate": "2025-02-28T00:00:00.000Z",
    "_id": "67f3dacfd7edb7aad9f5bb87",
    "createdAt": "2025-04-07T14:01:51.317Z",
    "updatedAt": "2025-04-07T14:01:51.317Z",
    "__v": 0
}
```

GitHub: https://github.com/BhalodiyaYash155/Practical-7

Conclusion: From This Practical I learn the following topics in node.js

- **What is MongoDB? NoSQL vs SQL databases**
- **Installing MongoDB locally and an introduction to MongoDB Atlas**
- **Setting up a Node.js project**
- **Installing and configuring Mongoose for MongoDB integration**
- **Understanding schemas and models in Mongoose**
- **Implementing Create, Read, Update, and Delete operations with Mongoose**
- **Introduction to RESTful API routes using Express**
- **Querying documents with filters and projections**