

Modern Task Manager Application– Node.js

NAME: LANKIPALLI THARUN

E-MAIL: tharun393325@gmail.com

B-TECH-INFORMATION TECHNOLOGY

PROJECT OVERVIEW

Task manager application designed to elevate individual and team productivity. Developed using Node.js and incorporating a RESTful API, Taskify seamlessly connects to a robust database to provide users with a secure and scalable platform for efficient task management. The project focuses on integrating essential features such as authentication, file upload, sorting, pagination, and filtering to offer a comprehensive solution for organizing and optimizing tasks.

PROJECT DEVELOPMENT OVERVIEW

Define Requirements:

Clearly define the features and functionalities you want in your Task Manager application. This includes user authentication, task creation, updating, deleting, sorting, pagination, filtering, file upload, etc.

Set Up Your Development Environment:

Install Node.js and npm on your machine. Set up a version control system (e.g., Git) for tracking changes.

Database:

MongoDB is a common choice for Node.js applications.

Create a Node.js Project:

Use npm init to initialize a new Node.js project. This will generate a package.json file. Choose a Web Framework and Implement Authentication Select a web framework for building your RESTful API. Express.js is a popular choice for Node.js applications.

Set Up Routes:

Create routes for handling different API endpoints, such as creating, updating, and deleting tasks.

Connect to the Database:

Use an ODM (Object-Document Mapper) like Mongoose if you're using MongoDB to connect your application to the database.

Implement CRUD Operations:

Implement the CRUD (Create, Read, Update, Delete) operations for tasks.

File Upload:

Integrate a file upload system for attaching files to tasks. Use middleware like Multer for handling file uploads.

Sorting, Pagination, and Filtering:

Implement features for sorting tasks, pagination, and filtering based on various criteria.

Project Development Steps

1. Define Requirements

- **Features to include:**
 - **User Authentication:** Secure login and signup system using JWT or sessions.
 - **Task Management:** Create, read, update, and delete tasks.
 - **Sorting, Filtering, and Pagination:** Allow users to sort tasks by priority, due date, or other criteria; filter by status or category; and paginate results.
 - **File Upload:** Enable users to upload and attach files to tasks.
 - **Role-based Access Control (Optional):** Separate permissions for admins and regular users.

2. Set Up Development Environment

- **Install Node.js and npm:**
 - ◆ Install them from Node.js official site.
 1. > node -v
 2. > npm -v
- **Initialize the project:**
 - ◆ This creates a package.json file to manage dependencies
 1. > mkdir Taskify
 2. > cd Taskify
 3. > npm init -y

3. Database Setup OR Launch MongoDB Compass

- Open **MongoDB Compass**.
 - Use the default URI to connect to your local MongoDB instance:plaintext
mongodb://127.0.0.1:27017
- Click **Connect**
- **Create a New Database**
 1. Once connected:
 - Click the "**Create Database**" button at the top.
 2. Enter the following details:
 - **Database Name:** Taskify
 - **Collection Name:** tasks
 3. Click "**Create Database**".

4. Open the project in VS Code

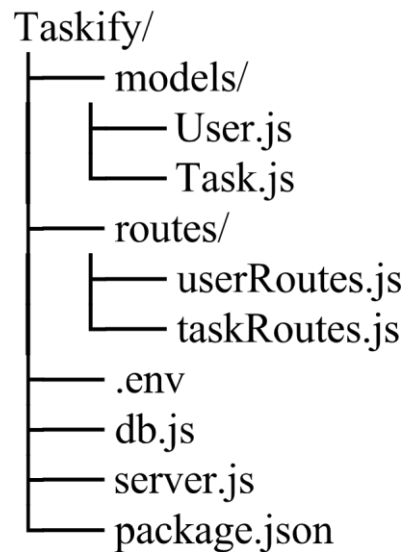
> code .

5. Install Dependencies

1. Install Express.js:
> npm install express
2. Install MongoDB and Mongoose:
> npm install mongoose
3. Install other dependencies for the project:
 1. **Multer** for file uploads:
> npm install multer
 2. **JWT (jsonwebtoken) and bcryptjs** for authentication:
> npm install jsonwebtoken bcryptjs
 3. **dotenv** for environment variables:
> npm install dotenv

6. Create Core Files

1. File structure in *VS Code*:



2. Create *server.js*:

```
const express = require('express');
const dotenv = require('dotenv');
const connectDB = require('./db');
dotenv.config();
connectDB();
```

```

const app = express();
app.use(express.json());
app.get('/', (req, res) => res.send('API is running...'));
const PORT = process.env.PORT || 5001;
app.listen(PORT, () => console.log(`Server running on port ${PORT}`));
const userRoutes = require('./routes/userRoutes');
const taskRoutes = require('./routes/taskRoutes');
app.use('/api/users', userRoutes);
app.use('/api/tasks', taskRoutes);

```

3. Set up **.env** file for environment variables:

```

PORT=5001
MONGO_URI=mongodb://127.0.0.1:27017/Taskify
JWT_SECRET=your_jwt_secret

```

4. Create **db.js**:

```

const mongoose = require('mongoose');
const connectDB = async () => {
  try {
    await mongoose.connect(process.env.MONGO_URI); // No need for
    useUrlParser or useUnifiedTopology
    console.log('MongoDB Connected...');
  } catch (error) {
    console.error('MongoDB Connection Error:', error.message);
    process.exit(1);
  }
};
module.exports = connectDB;

```

5. Create models for User and Task in **models/**:

○ **User.js**:

```

const mongoose = require('mongoose');
const bcrypt = require('bcryptjs');
const UserSchema = new mongoose.Schema({
  name: { type: String, required: true },
  email: { type: String, required: true, unique: true },
  password: { type: String, required: true },
});

```

```
UserSchema.pre('save', async function (next) {
  if (!this.isModified('password')) return next();
  this.password = await bcrypt.hash(this.password, 10);
  next();
});
module.exports = mongoose.model('User', UserSchema);
```

○ *Task.js:*

```
const mongoose = require('mongoose');
const TaskSchema = new mongoose.Schema({
  title: { type: String, required: true },
  description: { type: String },
  status: { type: String, enum: ['Pending', 'In Progress',
'Completed'], default: 'Pending' },
  dueDate: { type: Date },
  user: { type: mongoose.Schema.Types.ObjectId, ref: 'User' },
});
module.exports = mongoose.model('Task', TaskSchema);
```

6. Create routes in the *routes/* folder:

○ *userRoutes.js:*

```
const express = require('express');
const User = require('../models/User');
const jwt = require('jsonwebtoken');
const router = express.Router();

// Register user
router.post('/register', async (req, res) => {
  const { name, email, password } = req.body;
  const user = new User({ name, email, password });
  await user.save();
  res.status(201).json(user);
});

// Login user
router.post('/login', async (req, res) => {
  const { email, password } = req.body;
  const user = await User.findOne({ email });
  if (user && (await bcrypt.compare(password, user.password))) {
    const token = jwt.sign({ id: user._id },
process.env.JWT_SECRET, { expiresIn: '30d' });
```

```

    res.json({ token });
  } else {
    res.status(401).send('Invalid credentials');
  }
});
module.exports = router;

```

○ *taskRoutes.js*:

```

const express = require('express');
const Task = require('../models/Task');
const router = express.Router();

// Create a new task
router.post('/', async (req, res) => {
  try {
    const { title, status } = req.body;
    if (!title || !status) {
      return res.status(400).json({ message: 'Title and Status are required' });
    }
    const task = new Task(req.body);
    await task.save();
    res.status(201).json(task);
  } catch (error) {
    res.status(500).json({ message: error.message });
  }
});

// Get all tasks
router.get('/', async (req, res) => {
  try {
    const tasks = await Task.find();
    res.status(200).json(tasks);
  } catch (error) {
    res.status(500).json({ message: error.message });
  }
});

// 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({ message: 'Task not found' });
    }
    res.status(200).json(task);
  } catch (error) {
    res.status(500).json({ message: error.message });
  }
});

// Update a task by ID
router.put('/:id', async (req, res) => {
  try {
    const task = await Task.findByIdAndUpdate(req.params.id,
req.body, { new: true });
    if (!task) {
      return res.status(404).json({ message: 'Task not found' });
    }
    res.status(200).json(task);
  } catch (error) {
    res.status(500).json({ message: error.message });
  }
});

// Delete a task by ID
router.delete('/:id', async (req, res) => {
  try {
    const task = await Task.findByIdAndDelete(req.params.id);
    if (!task) {
      return res.status(404).json({ message: 'Task not found' });
    }
    res.status(200).json({ message: 'Task deleted successfully' });
  } catch (error) {
    res.status(500).json({ message: error.message });
  }
});
module.exports = router;

```

7. Run and Test

1. Start the server:

> **node server.js**

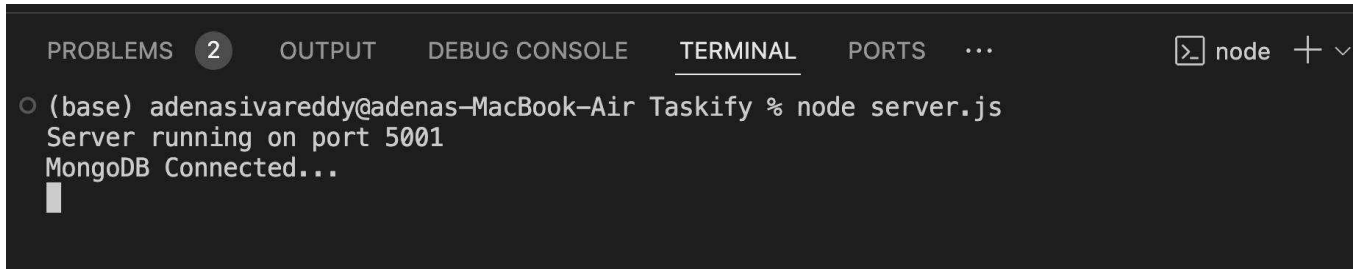
Or use nodemon for auto-restart:

> npm install -g nodemon

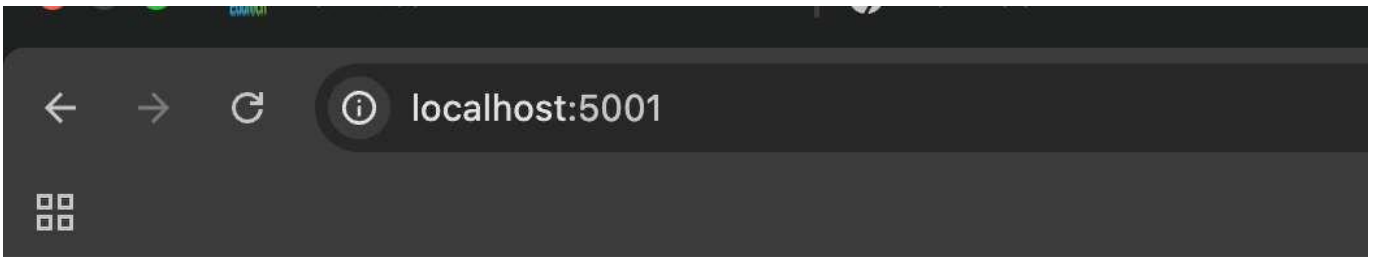
> nodemon server.js

2. Test with Postman or curl:

- Test the **GET**, **POST**, **PUT**, and **DELETE APIs**.

A screenshot of a terminal window with a dark background. The top bar shows tabs for PROBLEMS (with a count of 2), OUTPUT, DEBUG CONSOLE, TERMINAL (which is active), PORTS, and a menu icon. The terminal text shows a user prompt, the command 'node server.js', and the output 'Server running on port 5001' and 'MongoDB Connected...'.

```
(base) adenasivareddy@adenas-MacBook-Air Taskify % node server.js
Server running on port 5001
MongoDB Connected...
█
```



API is running...

8. Test the API

Now that the routes are set up, you can test them using Postman.

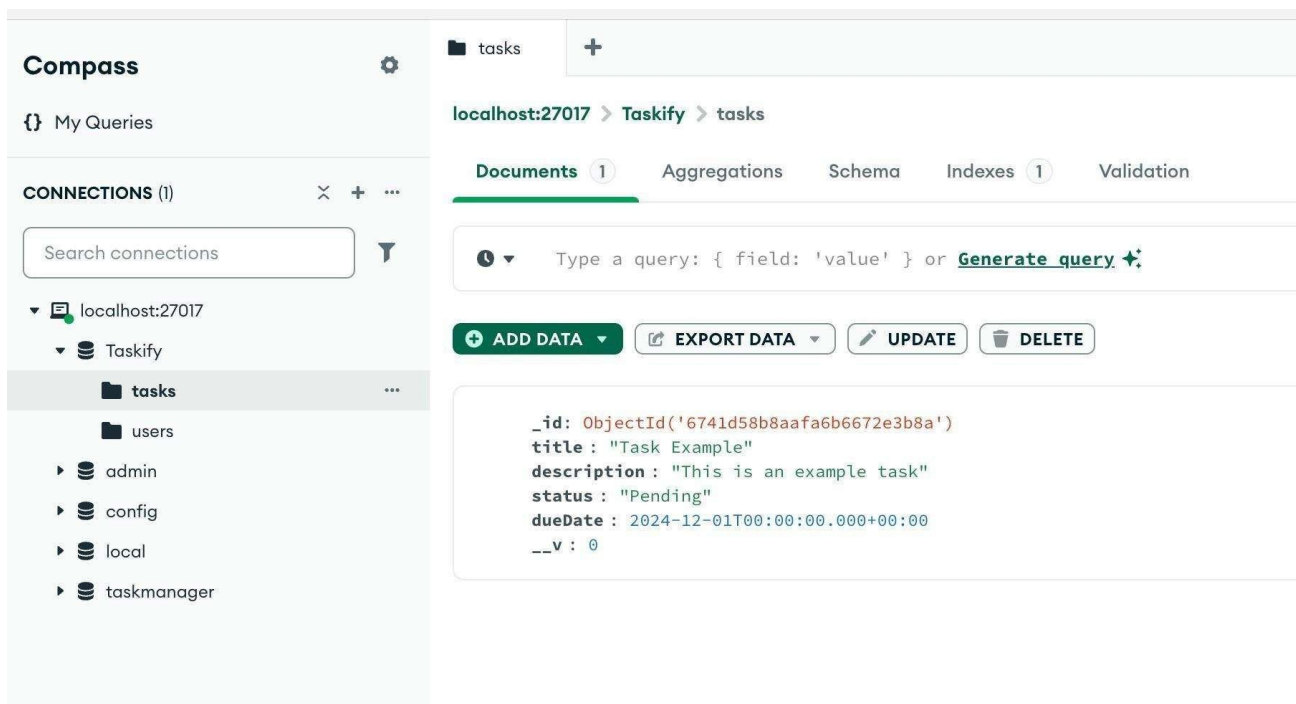
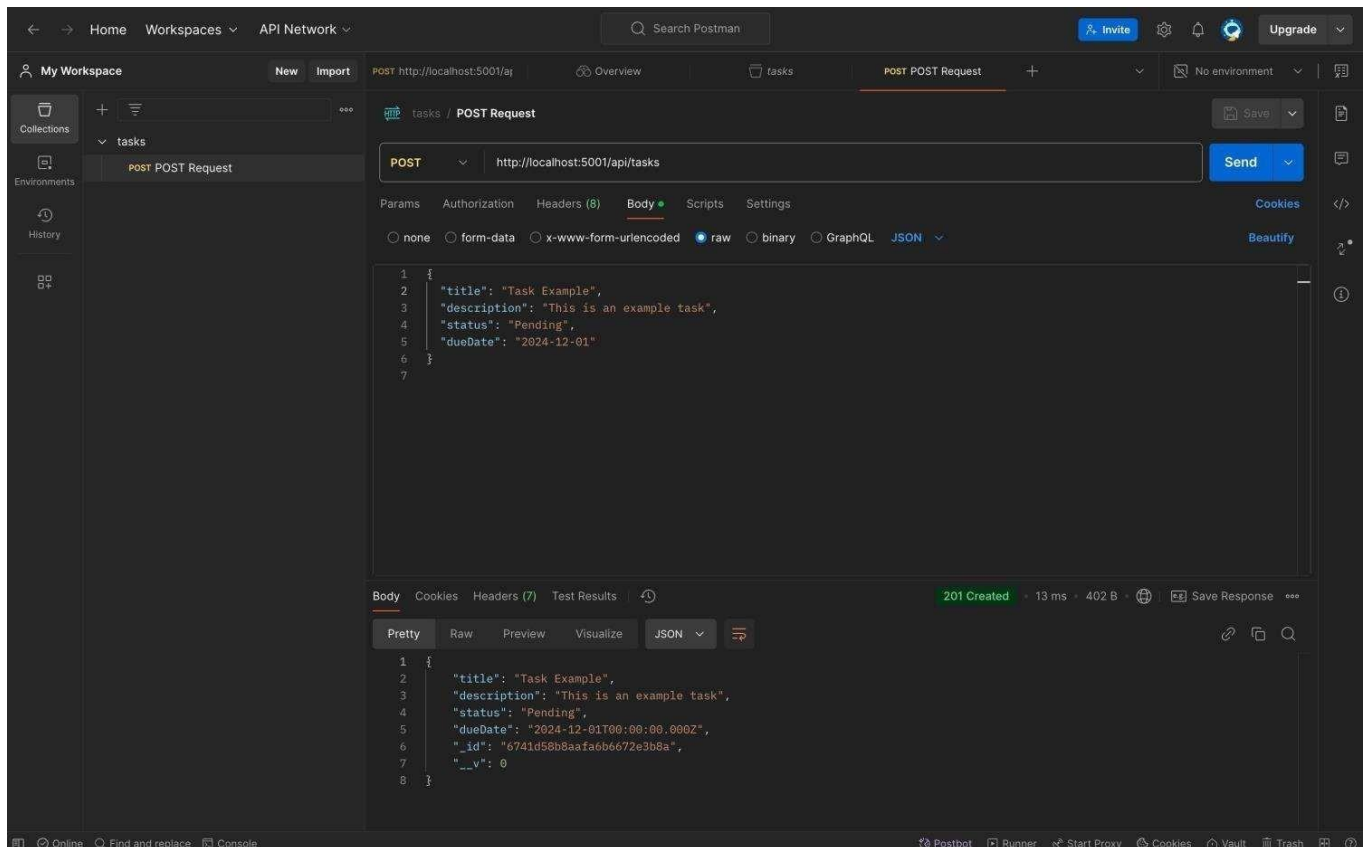
- ✓ Open **Postman**.
- ✓ Create a **new collection** and name it "tasks".
- ✓ Add **requests** under the collection with **POST** methods.

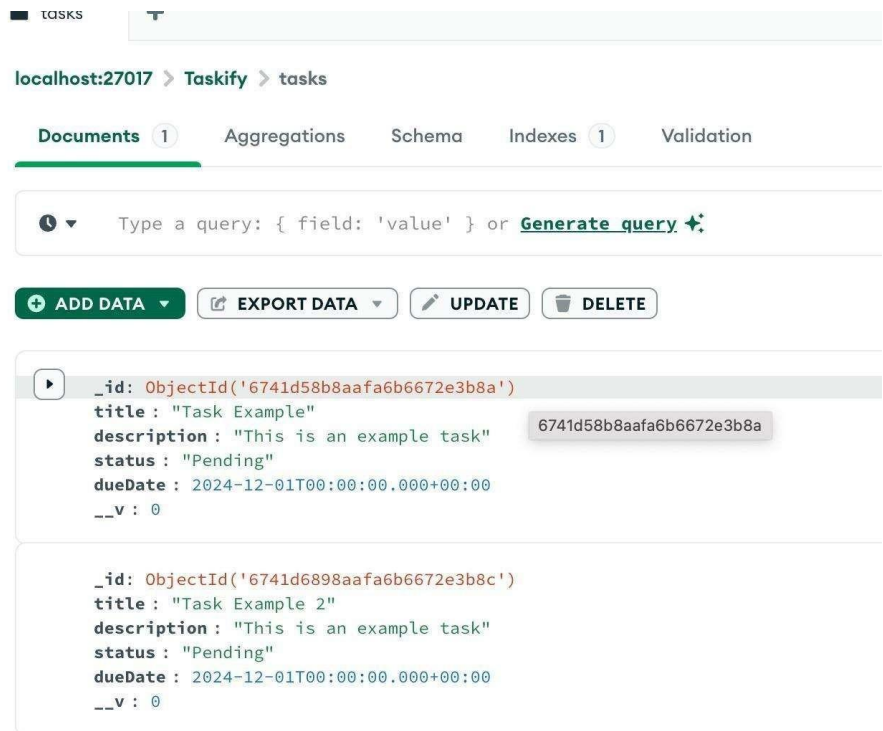
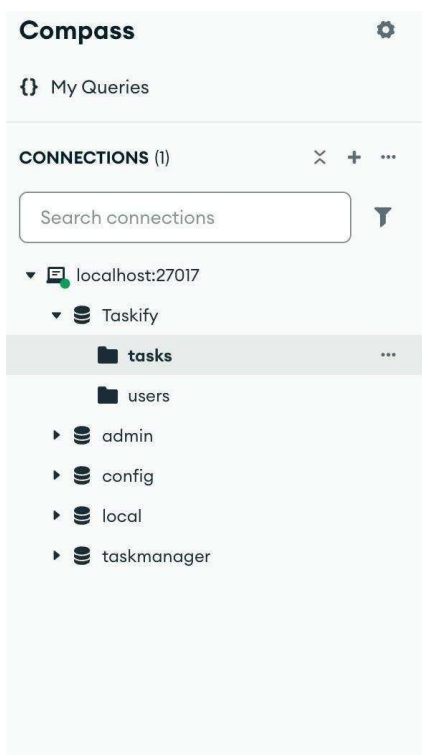
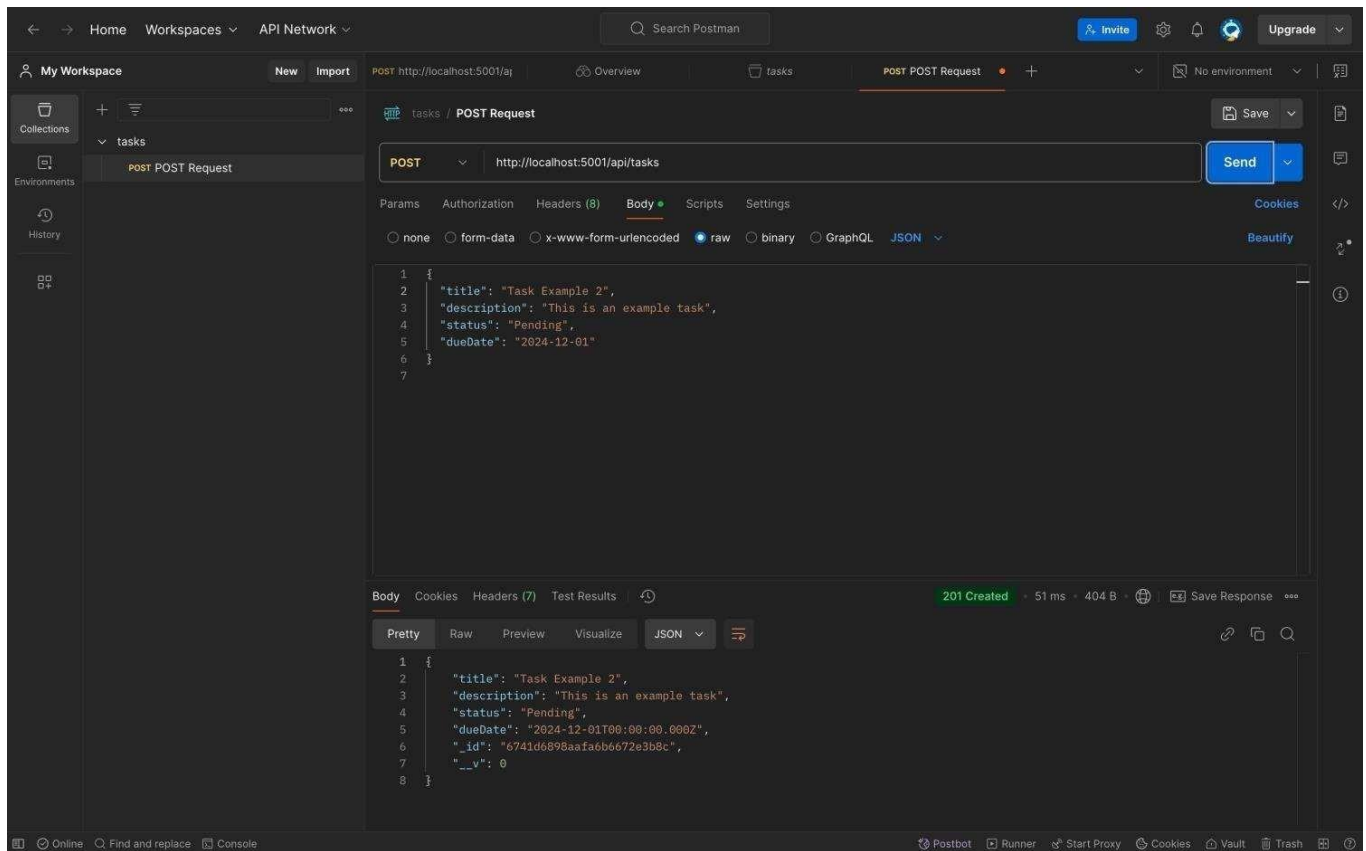
1. Create a new task:

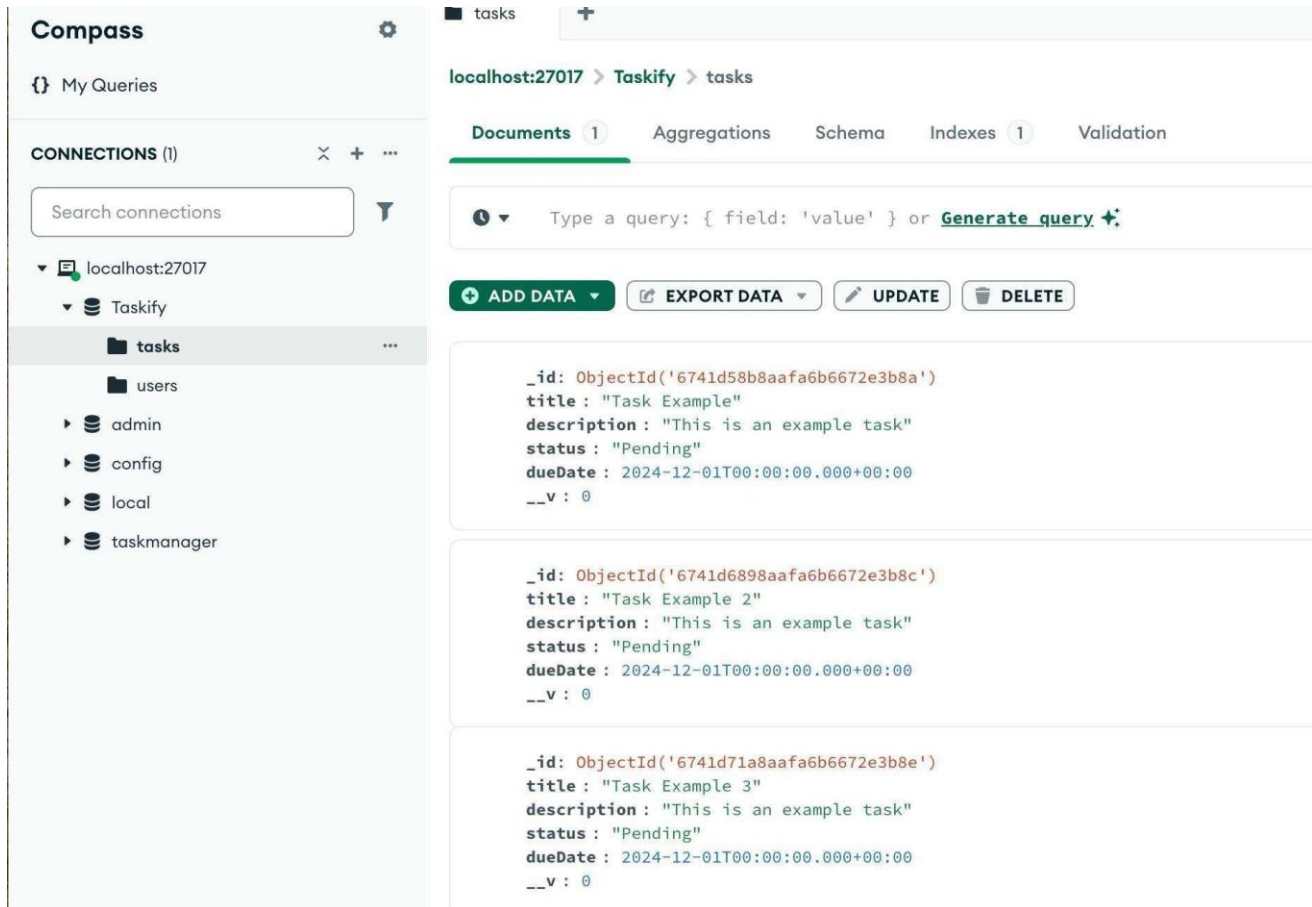
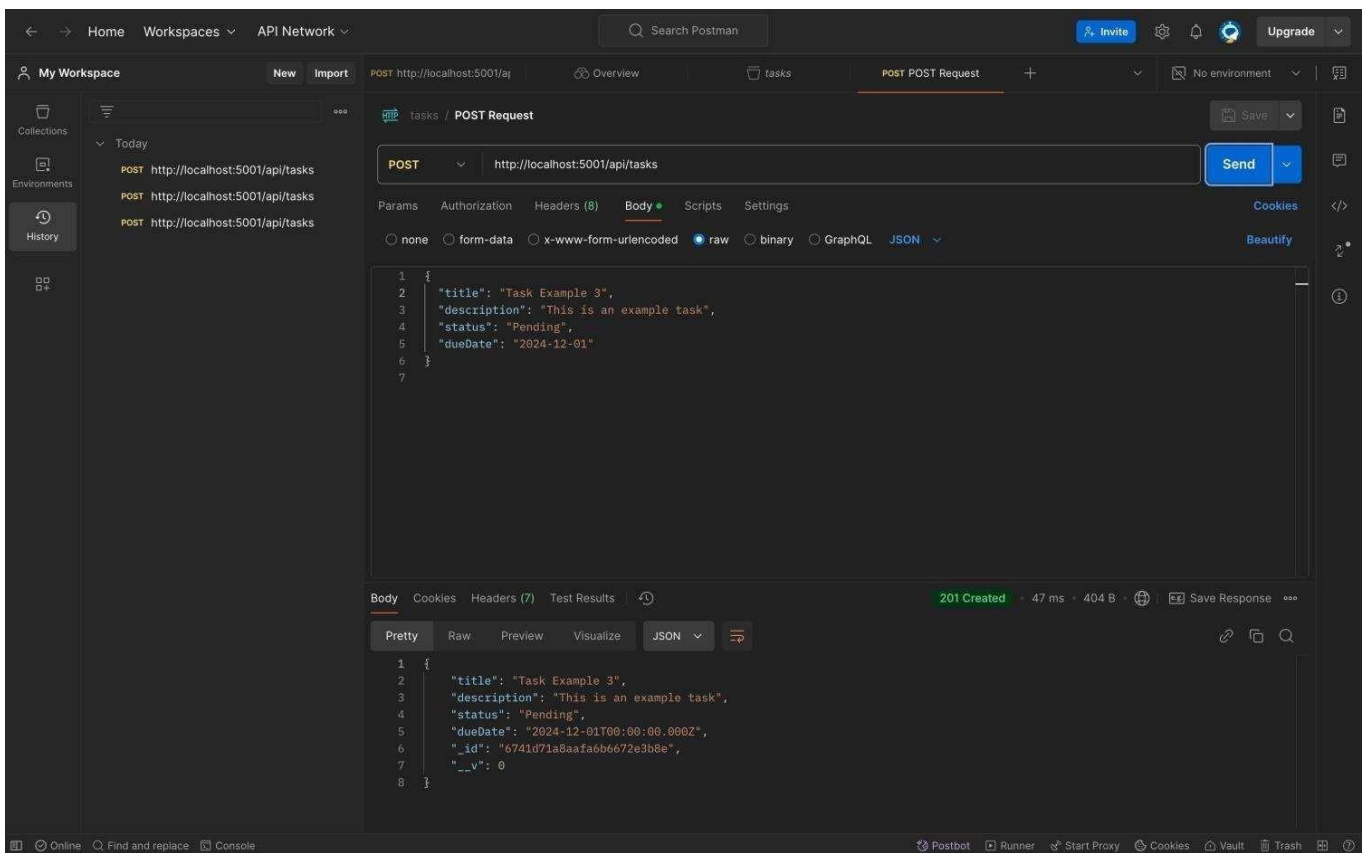
- Method: **POST**
- **URL:** `http://localhost:5001/api/tasks`
- Body: **JSON**son
- >

```
{
  "title": "Sample Task",
  "description": "This is a sample
task description",
  "status": "Pending",
  "dueDate": "2024-11-25"
}
```

- Click **Send**.
- You should receive a response with the created task's details.

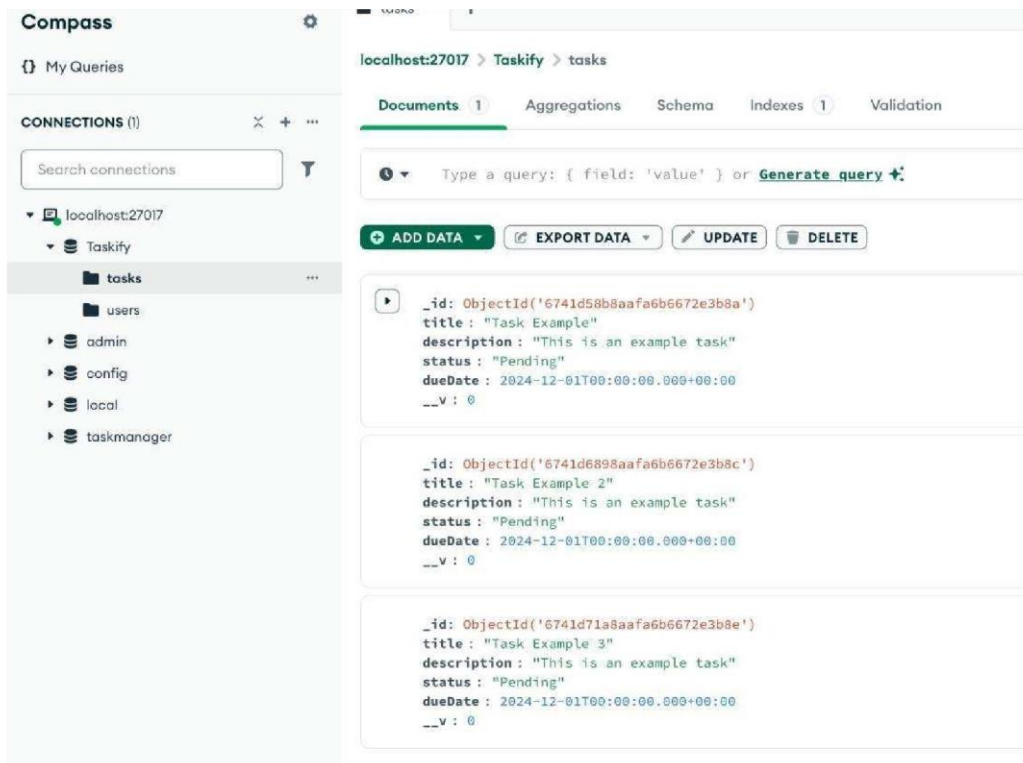
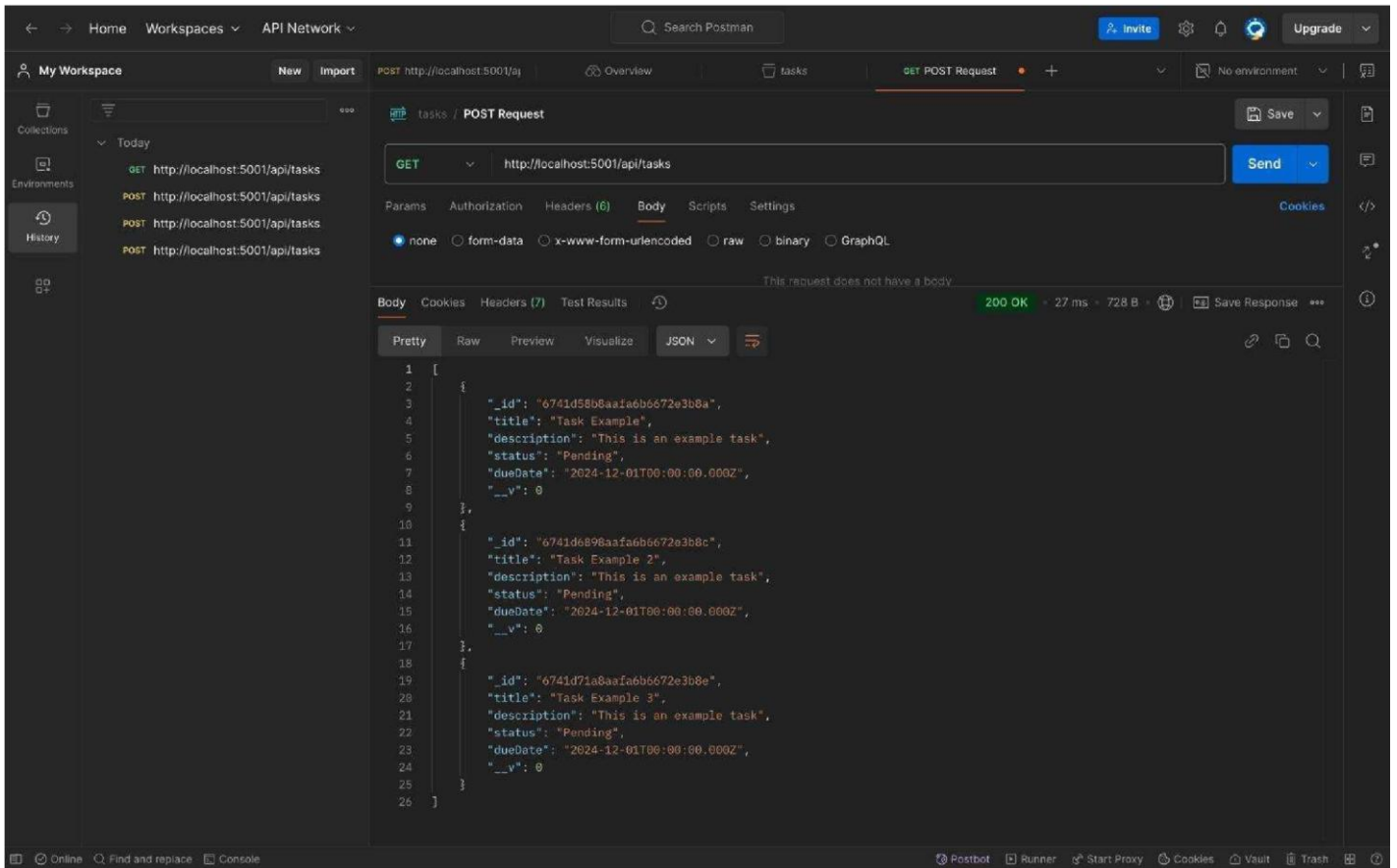






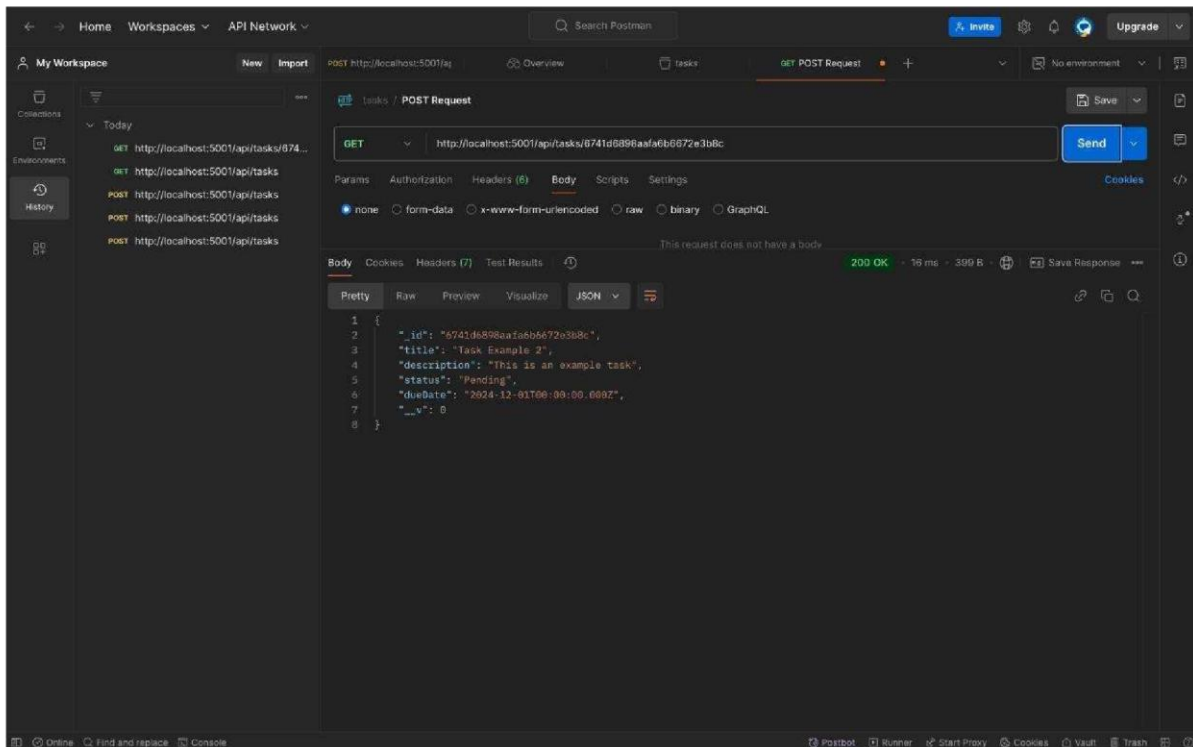
2. Get all tasks:

- Method: GET
- URL: <http://localhost:5001/api/tasks>
- Click **Send**.
- You should receive a response with a list of all tasks.



3. Get a task by ID:

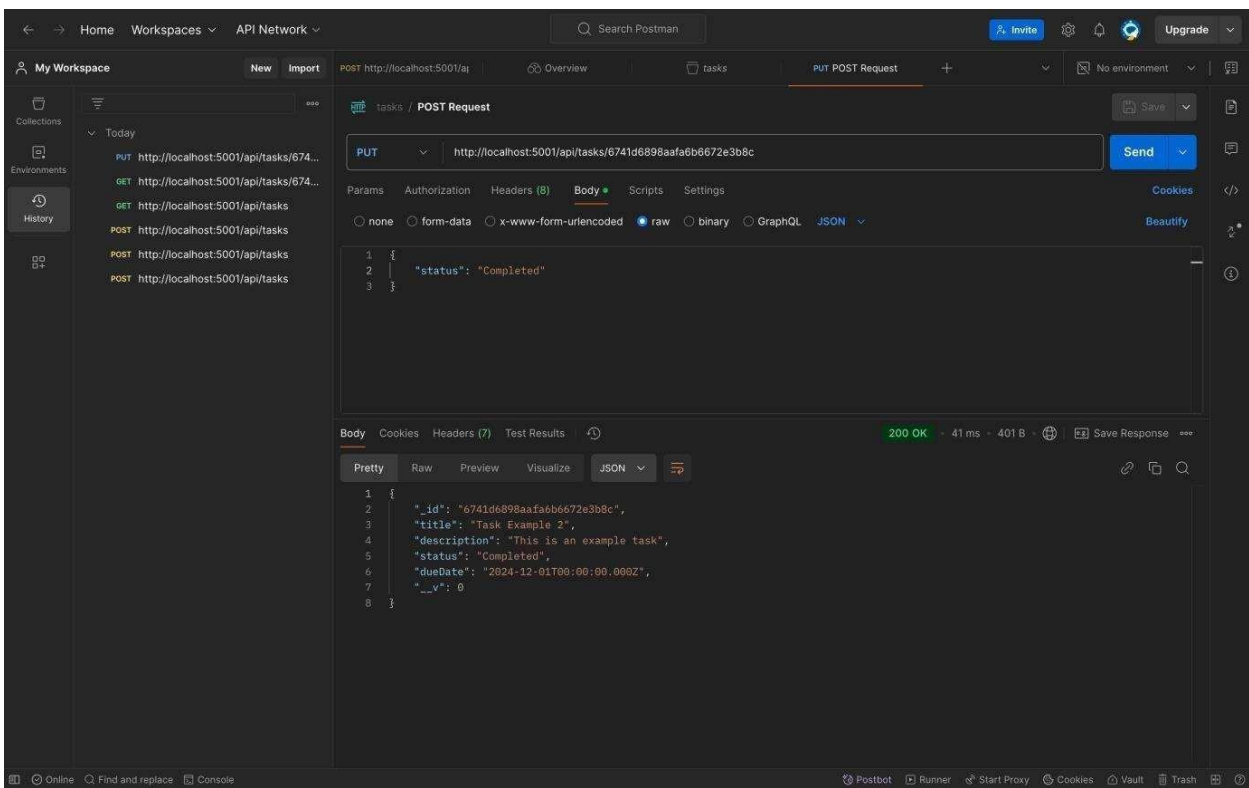
- Method: GET
- URL: `http://localhost:5001/api/tasks/<task_id>`
- **Example URL:** `http://localhost:5001/api/tasks/6741d6898aafa6b6672e3b8c`
- Click **Send**.
- You should receive the details of the specific task.



4. Update a task:

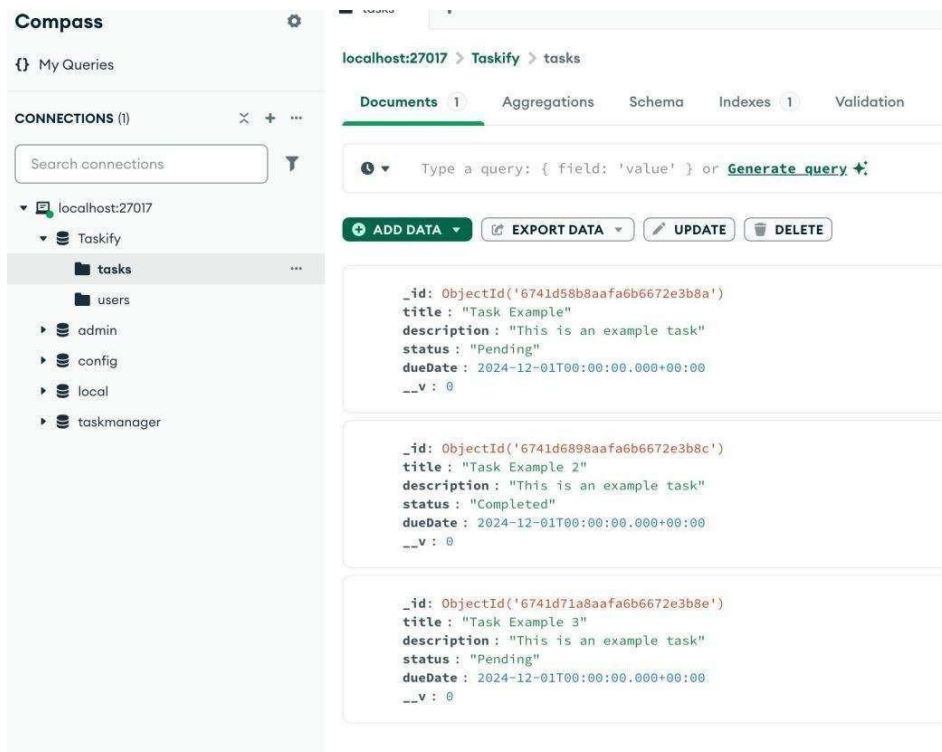
- Method: PUT
- URL: `http://localhost:5001/api/tasks/<task_id>`
- **Example URL:** `http://localhost:5001/api/tasks/6741d6898aafa6b6672e3b8c`
- Body: JSONjson
- ```
> { "status": "Completed"}
```
- Click **Send**.
- You should receive the updated task details.





```

_id: ObjectId('6741d6898aafa6b6672e3b8c')
title: "Task Example 2"
description: "This is an example task"
status: "Completed"
dueDate: 2024-12-01T00:00:00.000+00:00
__v: 0
```



## 5. Delete a task:

- Method: DELETE
- URL: `http://localhost:5001/api/tasks/<task_id>`
- **Example URL:** `http://localhost:5001/api/tasks/6741d6898aafa6b6672e3b8c`
- Click **Send**.
- You should receive a response confirming the deletion of the task.

The image shows two screenshots. The top screenshot is from Postman, showing a DELETE request to `http://localhost:5001/api/tasks/6741d6898aafa6b6672e3b8c` with a response of `200 OK` and a body containing `"message": "Task deleted successfully"`. The bottom screenshot is from MongoDB Compass, showing the 'tasks' collection in the 'Taskify' database. The collection contains two documents, both with `status: "Pending"`.

**Postman Screenshot:**

- Method: DELETE
- URL: `http://localhost:5001/api/tasks/6741d6898aafa6b6672e3b8c`
- Body: 

```
{
 "status": "Completed"
}
```
- Response: `200 OK`, `29 ms`, `274 B`. Body: 

```
{
 "message": "Task deleted successfully"
}
```

**MongoDB Compass Screenshot:**

- Database: Taskify
- Collection: tasks
- Documents:

  - ```
{
  "_id": ObjectId('6741d58b8aafa6b6672e3b8a'),
  "title": "Task Example",
  "description": "This is an example task",
  "status": "Pending",
  "dueDate": "2024-12-01T00:00:00.000+00:00",
  "__v": 0
}
```
 - ```
{
 "_id": ObjectId('6741d71a8aafa6b6672e3b8e'),
 "title": "Task Example 3",
 "description": "This is an example task",
 "status": "Pending",
 "dueDate": "2024-12-01T00:00:00.000+00:00",
 "__v": 0
}
```