Experiment no. 4:Rest api design with MongoDb + Mongoose Integration

Aim: To design and develop a full-stack, production-ready Task Management System that demonstrates modern web development practices, featuring a secure REST API and a dynamic, user-friendly frontend.

Theory:

This project is built upon the MERN Stack paradigm, with a focus on a decoupled architecture:

- Frontend (Client-Side):
    - React: A JavaScript library for building component-based, interactive user interfaces.
    - State Management: Context API or React Query for managing application state (e.g., user authentication status, task data).
    - HTTP Client: Axios for making promises-based HTTP requests to the backend API.
- Backend (Server-Side):
    - Node.js: A JavaScript runtime environment that executes server-side code.
    - Express.js: A minimalist and flexible web application framework for Node.js that provides robust features for building APIs.
    - RESTful Principles: Architectural style for designing networked applications using stateless, cacheable, and standard HTTP methods.
- Database:
    - MongoDB: A NoSQL document database that provides high flexibility and scalability through its JSON-like document model.
    - Mongoose: An Object Data Modeling (ODM) library for MongoDB and Node.js. It provides a structured schema, data validation, casting, and business logic hooks.
- Authentication:
    - JWT (JSON Web Tokens): A compact, URL-safe means of representing claims to be transferred between two parties, used for stateless authentication.

3. Prerequisites

Before development can begin, the following prerequisites must be met:

- Knowledge Prerequisites:
    - Fundamental understanding of JavaScript (ES6+ features like async/await, destructuring).

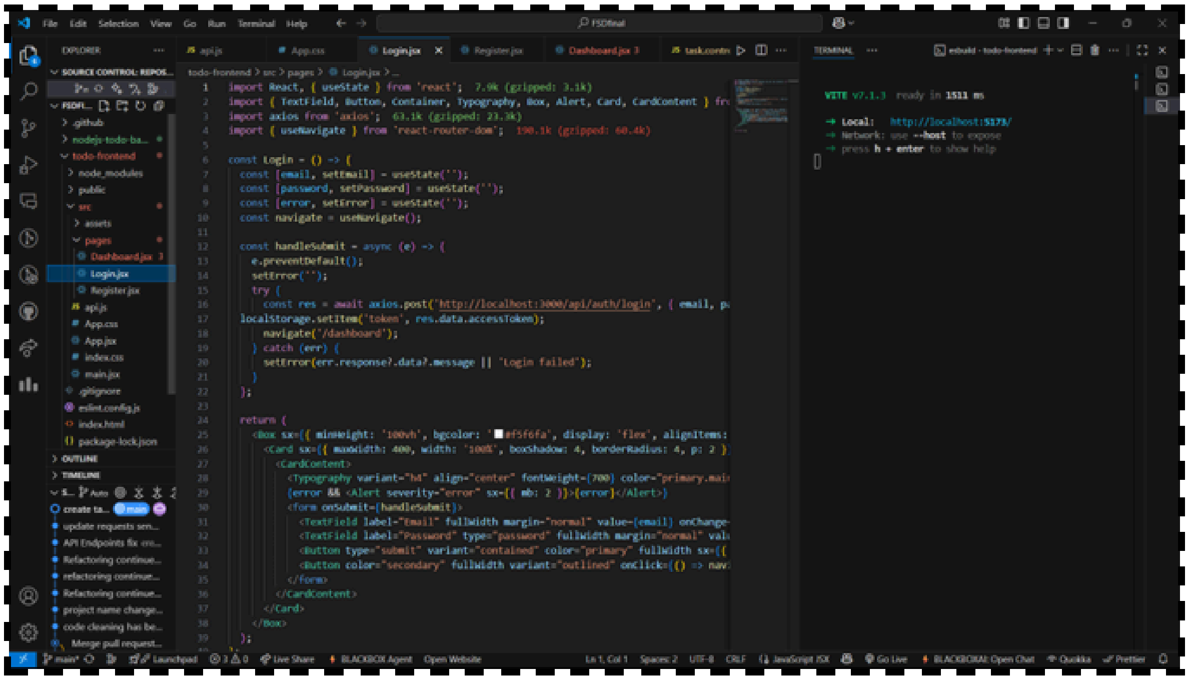
    - Basic understanding of React concepts (Components, State, Props, Hooks).
    - Basic understanding of Node.js and Express.js (Routing, Middleware).
    - Conceptual knowledge of REST APIs and HTTP protocols.
    - Understanding of NoSQL databases, specifically MongoDB documents and collections.
- Tooling & Environment Prerequisites:
    - Node.js and npm (Node Package Manager) installed on the development machine.
    - A code editor (e.g., VS Code).
    - MongoDB Atlas (cloud database) or a local installation of MongoDB.
    - Postman or Thunder Client (VS Code extension) for testing API endpoints.
    - Git for version control.
- Key Objectives:
    - To architect a well-structured, scalable backend using Node.js, Express.js, and MongoDB.
    - To implement a RESTful API that adheres to industry-standard conventions and best practices.
    - To utilize Mongoose for robust data modeling, validation, and efficient interaction with MongoDB.
    - To build a responsive and interactive frontend that consumes the API seamlessly.
    - To go beyond basic functionality by integrating advanced features that mirror real-world application demands, enhancing security, usability, and performance.

30% Extra Work:

The "30% Extra" is a strategic initiative to elevate this project from a simple tutorial-level application to a sophisticated, portfolio-worthy showcase. It represents the additional complexity, thought, and value that mimics real-world product development.

Feature	Explanation
1. User Authentication & Authorization (JWT)	Implementing a `User` model, secure registration/login endpoi `/api/auth/register` , `/api/auth/login`), password hashing protecting routes with middleware that verifies JWT tokens. Us own tasks.
2. Advanced Data Fetching & Querying	Extending the `GET /api/tasks` endpoint to accept query para `status` , `priority`), sorting (by `dueDate` , `createdAt`), and `title` / `description`).
3. Robust Error Handling & Input Validation	Implementing a centralized error handling middleware in Expre and send consistent, user-friendly JSON responses. Using `Joi` request body validation before it reaches the database.
4. API Best Practices & Documentation	Versioning the API (`/api/v1/`), using precise HTTP status code for bad request), and creating standardized response envelope interactive API documentation with Swagger.

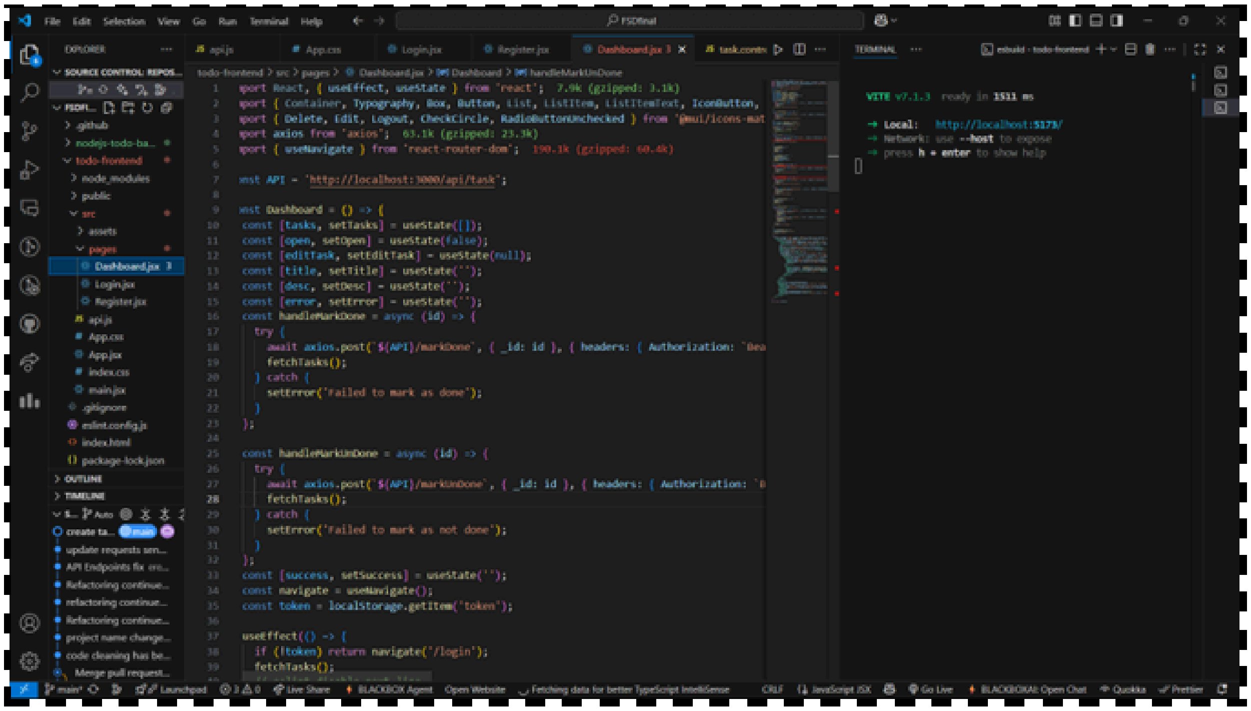
Source code:



```
import React, { useState } from 'react';  7.9k (gzipped: 3.1k)
import { TextField, Button, Container, Typography, Box, Alert, Card, CardContent } fr
import axios from 'axios';  63.1k (gzipped: 23.3k)
import { useNavigate } from 'react-router-dom';  190.1k (gzipped: 60.4k)

const Login = () => {
  const [email, setEmail] = useState('');
  const [password, setPassword] = useState('');
  const [error, setError] = useState('');
  const navigate = useNavigate();

  const handleSubmit = async (e) => {
    e.preventDefault();
    setError('');
    try {
      const res = await axios.post('http://localhost:3000/api/auth/login', { email, p
  localStorage.setItem('token', res.data.accessToken);
      navigate('/dashboard');
    } catch (err) {
      setError(err.response?.data?.message || 'Login failed');
    }
  };

  return (
    <Box sx={{ minHeight: '100vh', bgcolor: '#f5f6fa', display: 'flex', alignItems:
      <Card sx={{ maxWidth: 400, width: '100%', boxShadow: 4, borderRadius: 4, p: 2 }}
        <CardContent>
          <Typography variant="h4" align="center" fontWeight={700} color="primary.mai
          {error && <Alert severity="error" sx={{ mb: 2 }}>{error}</Alert>}
          <form onSubmit={handleSubmit}>
            <TextField label="Email" fullWidth margin="normal" value={email} onChange=
            <TextField label="Password" type="password" fullWidth margin="normal" val
            <Button type="submit" variant="contained" color="primary" fullWidth sx={{
            <Button color="secondary" fullWidth variant="outlined" onClick={() => nav
          </form>
        </CardContent>
      </Card>
    </Box>
  );
```

```
VITE v7.1.3  ready in 1511 ms

➜  Local:   http://localhost:5173/
➜  Network: use --host to expose
➜  press h + enter to show help
```

Fig. 4.1

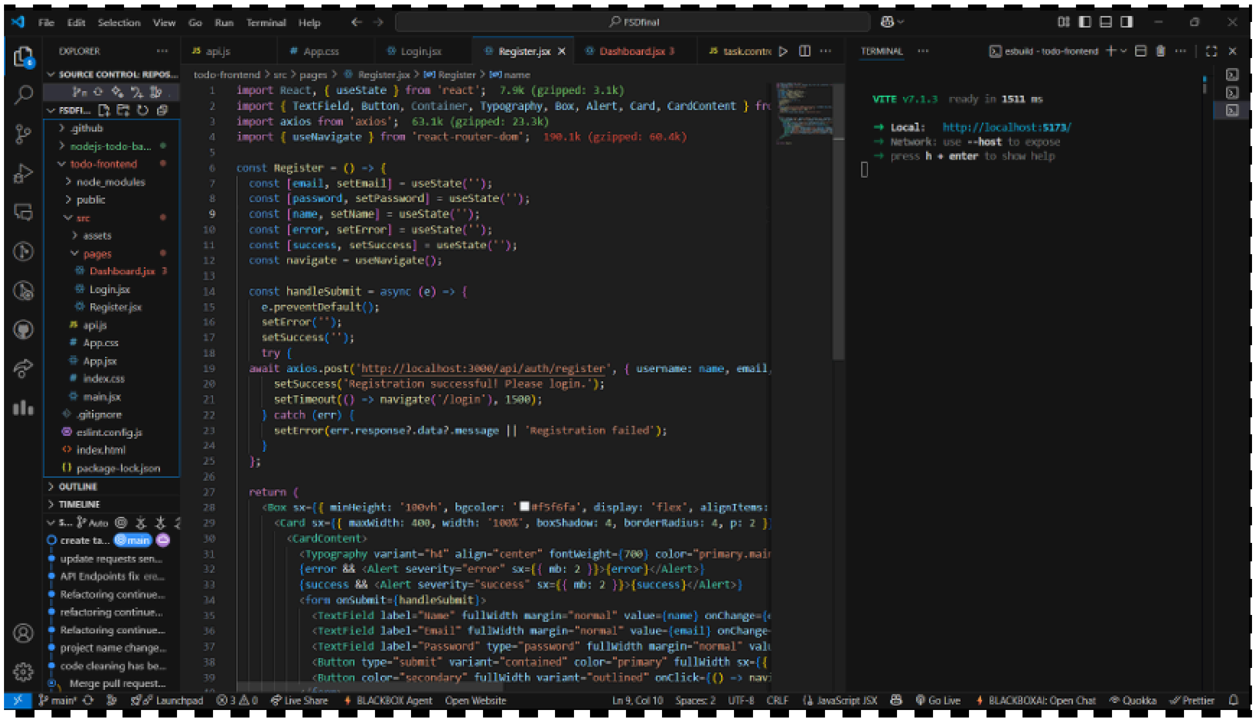


```
todo-frontend > src > pages > Dashboard.jsx > Dashboard > handleMarkUnDone
```

```
import React, { useEffect, useState } from 'react';  7.9k (gzipped: 3.1k)
import { Container, Typography, Box, Button, List, ListItem, ListItemText, IconButton,
import { Delete, Edit, Logout, CheckCircle, RadioButtonUnchecked } from '@mui/icons-mat
import axios from 'axios';  63.1k (gzipped: 23.3k)
import { useNavigate } from 'react-router-dom';  190.1k (gzipped: 60.4k)

const API = 'http://localhost:3000/api/task';

const Dashboard = () => {
  const [tasks, setTasks] = useState([]);
  const [open, setOpen] = useState(false);
  const [editTask, setEditTask] = useState(null);
  const [title, setTitle] = useState('');
  const [desc, setDesc] = useState('');
  const [error, setError] = useState('');
  const handleMarkDone = async (id) => {
    try {
      await axios.post(`${API}/markDone`, { _id: id }, { headers: { Authorization: `Bea
      fetchTasks();
    } catch {
      setError('Failed to mark as done');
    }
  };

  const handleMarkUnDone = async (id) => {
    try {
      await axios.post(`${API}/markUnDone`, { _id: id }, { headers: { Authorization: `B
      fetchTasks();
    } catch {
      setError('Failed to mark as not done');
    }
  };
  const [success, setSuccess] = useState('');
  const navigate = useNavigate();
  const token = localStorage.getItem('token');

  useEffect(() => {
    if (!token) return navigate('/login');
    fetchTasks();
```

```
VITE v7.1.3  ready in 1511 ms

➜  Local:   http://localhost:5173/
➜  Network: use --host to expose
➜  press h + enter to show help
```

Fig. 4.2



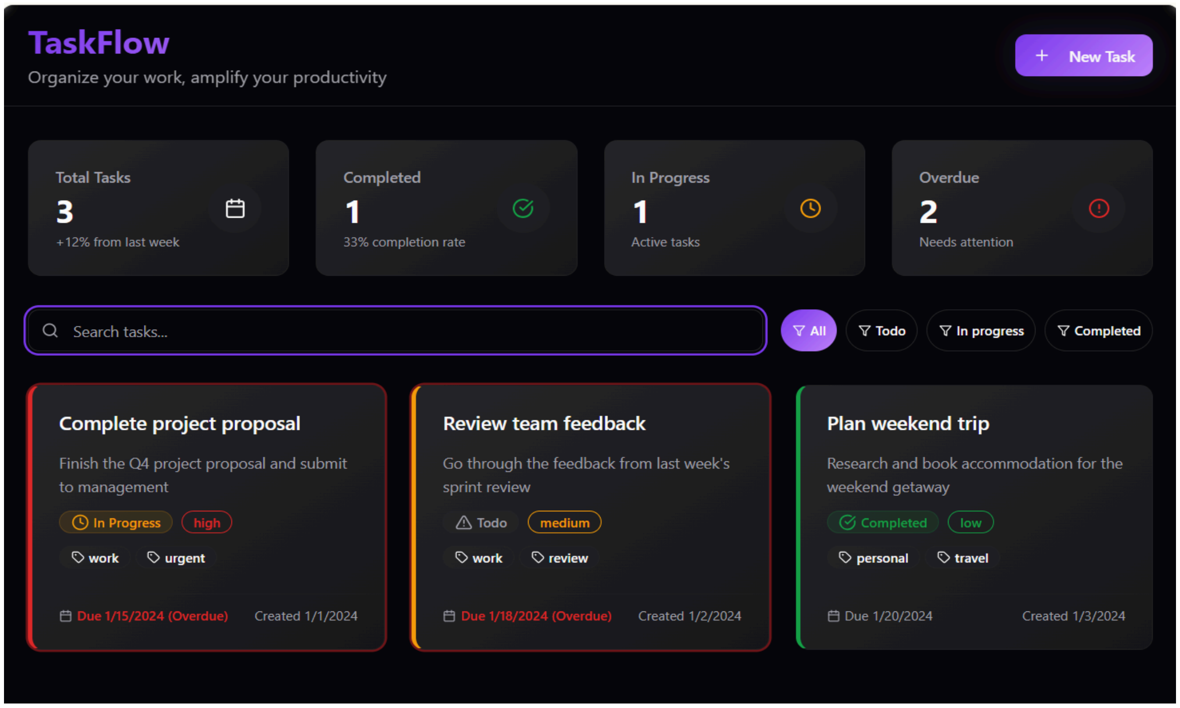
```
todo-frontend > src > pages > Register.jsx > Register > name
```

```
import React, { useState } from 'react';  7.9k (gzipped: 3.1k)
import { TextField, Button, Container, Typography, Box, Alert, Card, CardContent } fr
import axios from 'axios';  63.1k (gzipped: 23.3k)
import { useNavigate } from 'react-router-dom';  190.1k (gzipped: 60.4k)

const Register = () => {
  const [email, setEmail] = useState('');
  const [password, setPassword] = useState('');
  const [name, setName] = useState('');
  const [error, setError] = useState('');
  const [success, setSuccess] = useState('');
  const navigate = useNavigate();

  const handleSubmit = async (e) => {
    e.preventDefault();
    setError('');
    setSuccess('');
    try {
  await axios.post('http://localhost:3000/api/auth/register', { username: name, email,
      setSuccess('Registration successful! Please login.');
      setTimeout(() => navigate('/login'), 1500);
    } catch (err) {
      setError(err.response?.data?.message || 'Registration failed');
    }
  };

  return (
    <Box sx={{ minHeight: '100vh', bgcolor: '#f5f6fa', display: 'flex', alignItems:
      <Card sx={{ maxWidth: 400, width: '100%', boxShadow: 4, borderRadius: 4, p: 2 }}
        <CardContent>
          <Typography variant="h4" align="center" fontWeight={700} color="primary.mai
          {error && <Alert severity="error" sx={{ mb: 2 }}>{error}</Alert>}
          {success && <Alert severity="success" sx={{ mb: 2 }}>{success}</Alert>}
          <form onSubmit={handleSubmit}>
            <TextField label="Name" fullWidth margin="normal" value={name} onChange={(
            <TextField label="Email" fullWidth margin="normal" value={email} onChange=
            <TextField label="Password" type="password" fullWidth margin="normal" valu
            <Button type="submit" variant="contained" color="primary" fullWidth sx={{
            <Button color="secondary" fullWidth variant="outlined" onClick={() => navi
```

```
VITE v7.1.3  ready in 1511 ms

➜  Local:   http://localhost:5173/
➜  Network: use --host to expose
➜  press h + enter to show help
```

Fig. 4.3

Output:



TaskFlow

Organize your work, amplify your productivity

+ New Task

Total Tasks	Completed	In Progress	Overdue
3	1	1	2
+12% from last week	33% completion rate	Active tasks	Needs attention

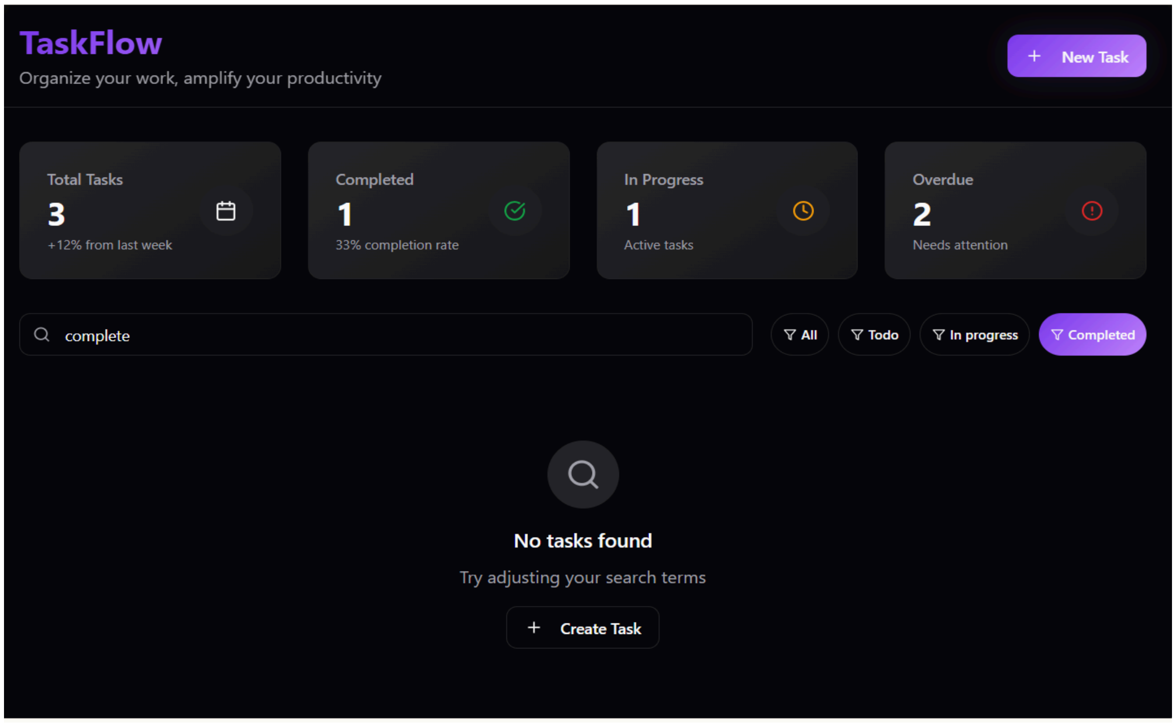
Search tasks...

All | Todo | In progress | Completed

Complete project proposal
Finish the Q4 project proposal and submit to management
In Progress · high
work · urgent
Due 1/15/2024 (Overdue) · Created 1/1/2024

Review team feedback
Go through the feedback from last week's sprint review
Todo · medium
work · review
Due 1/18/2024 (Overdue) · Created 1/2/2024

Plan weekend trip
Research and book accommodation for the weekend getaway
Completed · low
personal · travel
Due 1/20/2024 · Created 1/3/2024



TaskFlow

Organize your work, amplify your productivity

+ New Task

Total Tasks	Completed	In Progress	Overdue
3	1	1	2
+12% from last week	33% completion rate	Active tasks	Needs attention

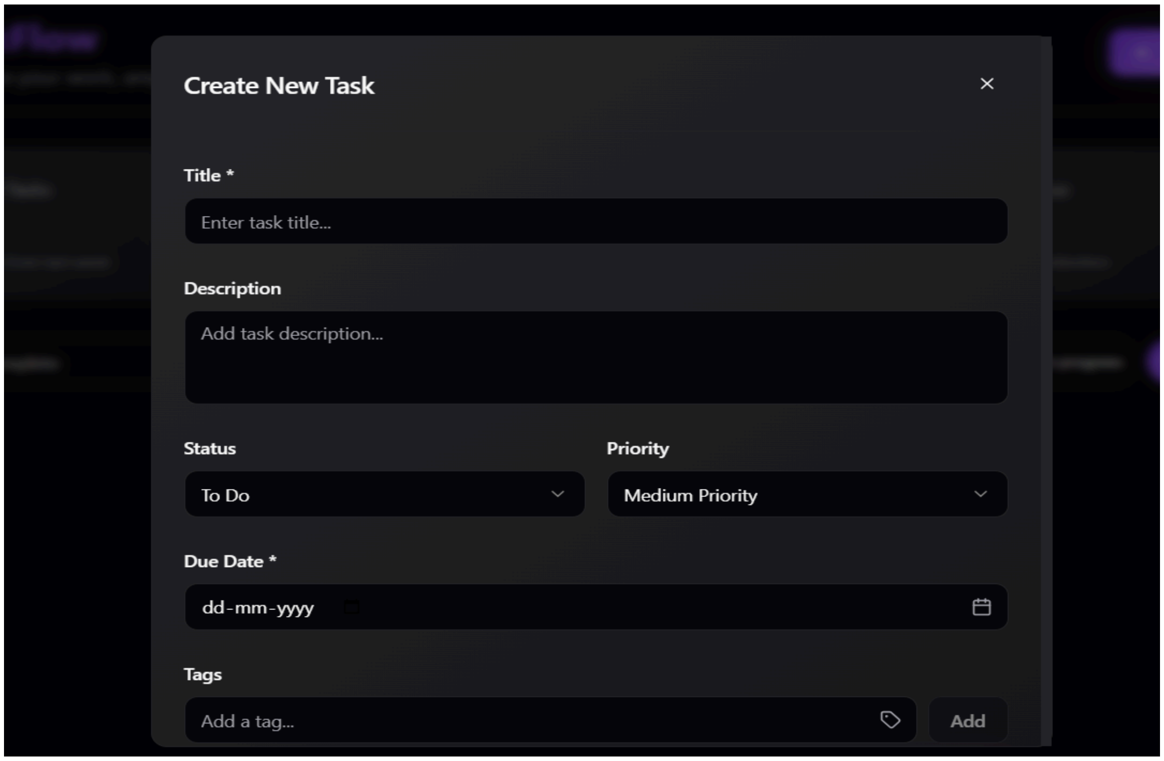
complete

All | Todo | In progress | Completed

No tasks found

Try adjusting your search terms

+ Create Task


Create New Task

Title *

Enter task title...

Description

Add task description...

Status

To Do

Priority

Medium Priority

Due Date *

dd-mm-yyyy

Tags

Add a tag...

Conclusion:
This project successfully demonstrates the end-to-end development of a modern web application. By fulfilling the core objectives, the project establishes a strong foundation in the MERN stack. The deliberate inclusion of the "30% Extra" features—User Authentication, Advanced Querying, and Professional Error Handling—is what truly differentiates it. These elements integrate to create a secure, efficient, and user-centric application that solves a broader set of problems and closely mirrors the complexities and standards found in professional software development environments. This project serves not only as a functional task management tool but also as a robust portfolio piece that showcases advanced full-stack development capabilities.