WEBDEVELOPMENT

MERN Task

Fast-NUCES
Faisal jabbar

Contents

Task Management Application with User Authentication — MERN Stack

1. Project Overview

This project is a **full-stack task management application** built using the MERN stack:

- **MongoDB** for the database (NoSQL document storage),
- **Express.js** as the backend web framework,
- **React** for building the frontend user interface,
- **Node.js** as the backend runtime environment.

The application enables users to **register and log in** securely, then create, edit, delete, and manage their tasks with priority, due dates, and completion status. It features a **secure user authentication system** with password hashing and JWT tokens, and a clean, responsive UI suitable for desktop and mobile.

2. Core Functionalities Explained & Implementation Details

2.1 User Authentication System

What it means:
Users must be able to create accounts (register), securely log in and log out. Passwords are stored securely (hashed), and only authenticated users can access certain features or pages.

How we implemented it:

- **User Registration:**
  - Frontend registration form collects email and password.
  - Backend route /api/auth/register receives data.
  - Passwords hashed using bcrypt before saving in MongoDB.
  - Input validated for format and completeness on frontend and backend.
- **User Login/Logout:**
  - Login form sends email/password to /api/auth/login.
  - Backend verifies password hash with bcrypt.
  - On success, backend generates a JWT (JSON Web Token).
  - Token stored in frontend (e.g., localStorage) and sent with API requests in Authorization headers.
  - Logout clears token from frontend storage.
- **Protected Routes:**
  - Backend middleware verifies JWT on routes that require authentication (e.g., /api/tasks).
  - Unauthorized requests are blocked and receive 401 errors.

  - Frontend redirects to login page if no valid token.

2.2 Task Management Features

What it means:
Users can create tasks with details, view tasks in an organized way, modify tasks, delete them, mark complete/incomplete, and filter tasks by their status or priority.

How we implemented it:

- **Creating Tasks:**
  - User fills form with title, description, due date, priority.
  - Frontend sends POST request to /api/tasks with task data.
  - Backend stores tasks in MongoDB linked to the user.
- **Viewing Tasks:**
  - GET request to /api/tasks fetches all tasks for the logged-in user.
  - Frontend displays tasks in a clean list or card view.
  - Filters available to show completed, pending, or by priority.
- **Editing & Deleting Tasks:**
  - PUT /api/tasks/:id updates a task with new data.
  - DELETE /api/tasks/:id removes the task from DB.
  - Frontend UI updates accordingly after each operation.
- **Marking Complete/Incomplete:**
  - PATCH /api/tasks/:id/toggle flips the completed status.
  - UI shows completed tasks differently (e.g., strikethrough or checked).

2.3 User Interface Requirements

What it means:
The app must be responsive, visually clean and easy to use, with proper feedback and validation.

How we implemented it:

- **Responsive Design:**
  - Used CSS Flexbox and media queries to support desktop and mobile layouts.
  - Forms and task lists adapt to screen size.
- **Clean UI:**
  - Simple color scheme, intuitive navigation.

  - Clear call-to-action buttons for task creation and actions.
- **Loading States:**
  - Show spinners or “loading...” text during API calls to indicate progress.
- **Error Handling & User Feedback:**
  - Display error messages on failed operations (e.g., invalid login).
  - Success messages on actions like task creation or updates.
- **Form Validation:**
  - Frontend validation for empty fields, valid email format.
  - Backend validation for security and data integrity.

3. Backend Details

Backend Tech & Folder Structure

- Built using **Node.js** and **Express.js**.
- Organized routes, controllers, and middleware.
- Used **bcrypt** for password hashing.
- Used **jsonwebtoken (JWT)** for token-based authentication.
- Connected to **MongoDB** via Mongoose ORM.
- Middleware implemented to protect routes (JWT verification).

Key Backend Implementations

- **User Model:**
  Schema with fields for email (unique), password (hashed), timestamps.
- **Task Model:**
  Fields: title, description, dueDate, priority (enum), completed (boolean), userId (reference to User).
- **Authentication Routes:**

Method	**Endpoint**	**Description**
POST	/api/auth/register	Register new user
POST	/api/auth/login	Login user
GET	/api/auth/profile	Fetch logged-in user info (protected)

- **Task Routes:**

Method	Endpoint	Description
GET	/api/tasks	List all tasks for user
POST	/api/tasks	Create a new task
PUT	/api/tasks/:id	Update a specific task
DELETE	/api/tasks/:id	Delete a task
PATCH	/api/tasks/:id/toggle	Toggle completion status

- **Middleware:**
  JWT verification to protect task routes.
- **Database Connection & Error Handling:**
  MongoDB connected with error listeners to ensure smooth operation.

4. Database Details

- **Database:** MongoDB (NoSQL document store)
- Used **Mongoose** for schema and model definition.

Schemas

- **User Schema:**
  - email (unique, required, validated)
  - password (hashed, required)
  - Timestamps for creation and update
- **Task Schema:**
  - title (string, required)
  - description (string, optional)
  - dueDate (Date, optional)
  - priority (enum: Low, Medium, High)
  - completed (boolean, default false)
  - user (ObjectId reference to User)
  - Timestamps

Database Performance

- Indexed email in User for fast lookups.
- Indexed user field in Task schema for efficient querying by user.

5. Testing API with Thunder Client

Used **Thunder Client** (VSCode extension) to manually test API endpoints during development.

- Tested **Registration** by sending POST to /api/auth/register with email/password JSON payload.
- Tested **Login** with POST to /api/auth/login, verified returned JWT token.
- Used JWT token in **Authorization** header for protected routes.
- Tested **Fetching Profile** with GET /api/auth/profile.
- For tasks:
    - Created tasks with POST /api/tasks.
    - Retrieved list with GET /api/tasks.
    - Updated tasks with PUT /api/tasks/:id.
    - Deleted tasks with DELETE /api/tasks/:id.
    - Toggled completion status with PATCH /api/tasks/:id/toggle.
- Verified proper status codes, response messages, and database updates after each request.

6. Bonus Features & Implementation

- **Drag-and-drop task reordering:**
  (Planned) Would use React DnD or similar library to allow users to reorder tasks visually. Backend API would store task order as an extra field.
- **Task Categories or Tags:**
  Could add category or tags field in Task schema and implement filtering by tags on frontend.
- **Task Search Functionality:**
  Search bar implemented on frontend filters tasks by title or description dynamically.
- **Data Export (CSV):**
  Export tasks as downloadable CSV via backend route generating the file on demand.

7. Setup Instructions

Prerequisites

- Node.js installed (v14+ recommended)
- npm or yarn
- MongoDB (local installation or MongoDB Atlas cloud instance)
- Git (optional)

Clone Project

```
git clone https://github.com/your-username/task-manager.git
cd task-manager
```

Backend Setup

```
cd server
npm install
```

Create a .env file in the server folder with the following:

```
PORT=5000
MONGO_URI=mongodb://localhost:27017/taskmanager
JWT_SECRET=abec813133042da2bd3b5c7acd94f1bc08dfccd5389d7de0372728f011d12d3634084d0159b1938 6c5790466d9c4fc40f8ac6e2c5ce7d9d8bf3b267226a1ce6a
```

Start backend server:

```
npm start
```

Backend runs on: http://localhost:5000

Frontend Setup

Open a new terminal:

```
cd client
npm install
npm start
```

Frontend runs on: http://localhost:3000

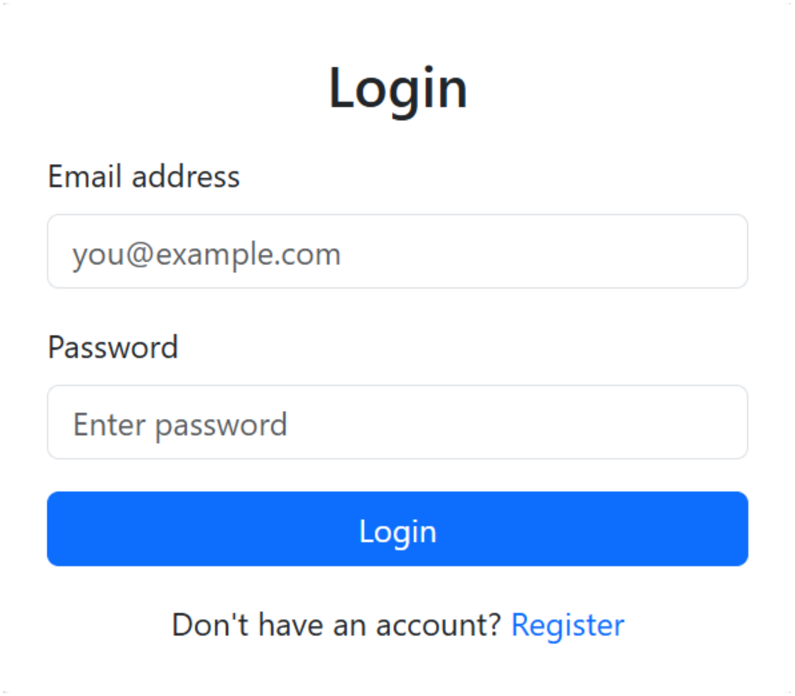
Access the Application

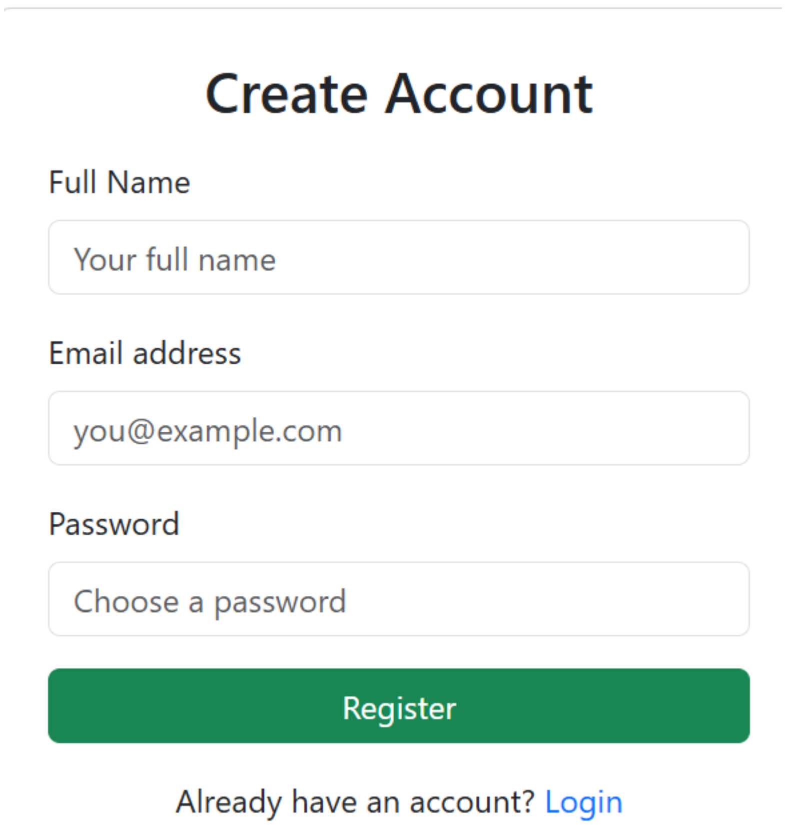
Open browser at http://localhost:3000.

Summary

This project provides a solid foundation for user-authenticated task management with secure backend services, a responsive React frontend, and integration with MongoDB. It follows best practices in security, API design, and UX.

UI





Create Account

Full Name

Your full name

Email address

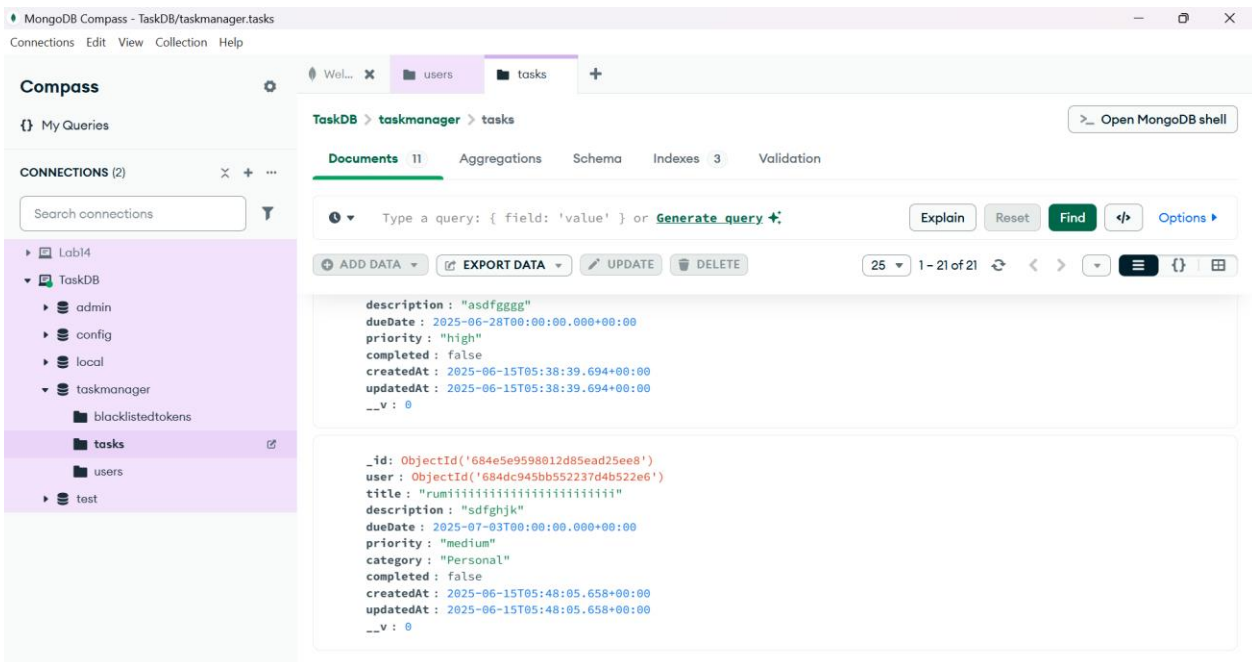
you@example.com

Password

Choose a password

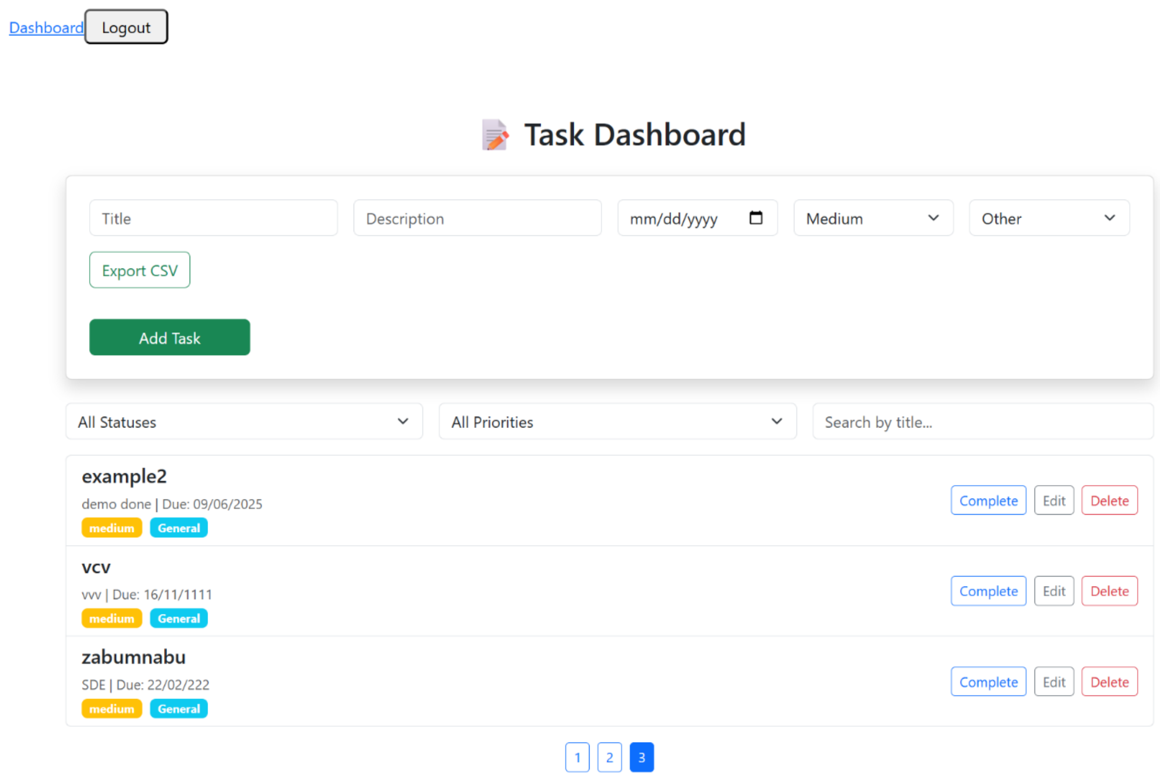
Register

Already have an account? Login



```
description : "asdfgggg"
dueDate : 2025-06-28T00:00:00.000+00:00
priority : "high"
completed : false
createdAt : 2025-06-15T05:38:39.694+00:00
updatedAt : 2025-06-15T05:38:39.694+00:00
__v : 0

_id: ObjectId('684e5e9598012d85ead25ee8')
user : ObjectId('684dc945bb552237d4b522e6')
title : "rumiiiiiiiiiiiiiiiiiiiiiiii"
description : "sdfghjk"
dueDate : 2025-07-03T00:00:00.000+00:00
priority : "medium"
category : "Personal"
completed : false
createdAt : 2025-06-15T05:48:05.658+00:00
updatedAt : 2025-06-15T05:48:05.658+00:00
__v : 0
```

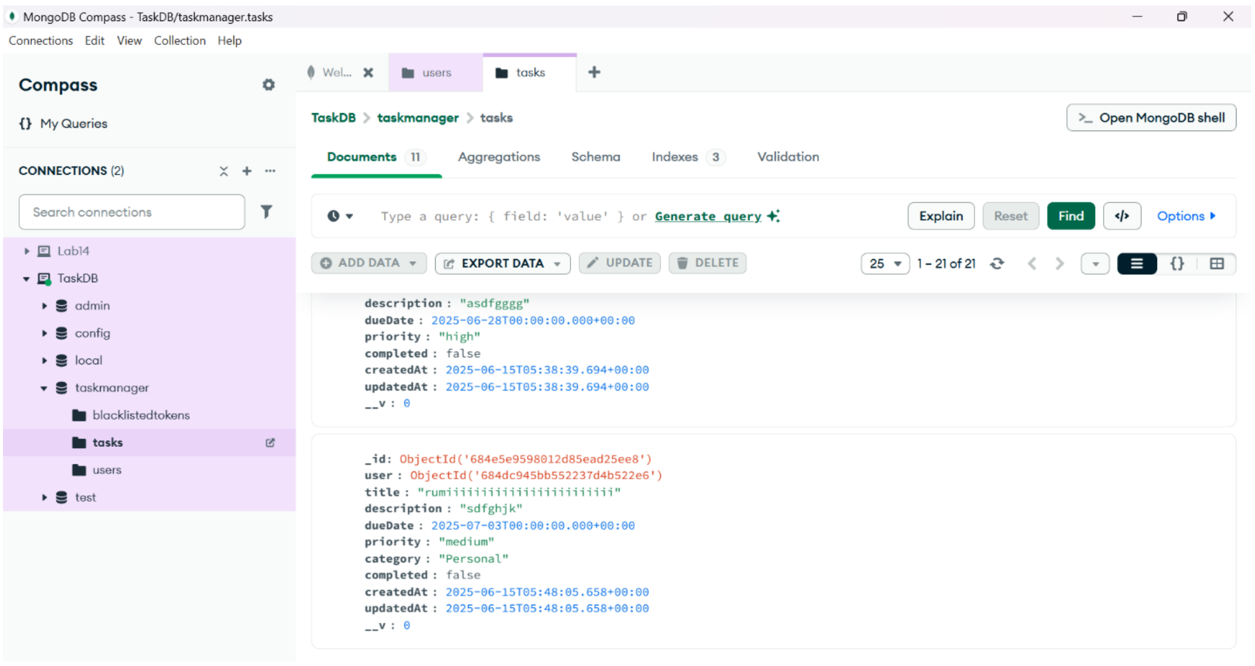


Task Dashboard

example2
demo done | Due: 09/06/2025
medium General

vcv
vvv | Due: 16/11/1111
medium General

zabumnabu
SDE | Due: 22/02/222
medium General



```
description : "asdfgggg"
dueDate : 2025-06-28T00:00:00.000+00:00
priority : "high"
completed : false
createdAt : 2025-06-15T05:38:39.694+00:00
updatedAt : 2025-06-15T05:38:39.694+00:00
__v : 0
```

```
_id: ObjectId('684e5e9598012d85ead25ee8')
user : ObjectId('684dc945bb552237d4b522e6')
title : "rumiiiiiiiiiiiiiiiiiiiiiii"
description : "sdfghjk"
dueDate : 2025-07-03T00:00:00.000+00:00
priority : "medium"
category : "Personal"
completed : false
createdAt : 2025-06-15T05:48:05.658+00:00
updatedAt : 2025-06-15T05:48:05.658+00:00
__v : 0
```