
Frontend Developer Intern Task - Project Documentation

Project Name: TaskApp - Scalable Web App with Authentication & Dashboard

---

1. Project Overview

This project is a full-stack Task Management application built to demonstrate a scalable frontend-backend integration. It features secure JWT authentication, a responsive dashboard, and full CRUD capabilities with search and filtering options.

Key Features

- **Authentication:** Secure User Signup & Login using JWT (JSON Web Tokens) and Bcrypt for password hashing.
- **Dashboard:** A protected area where users can manage their specific tasks.
- **Data Management:** Users can Create, Read, Update, and Delete (CRUD) tasks.
- **Advanced UI:** Real-time Search and Status Filtering (Pending/In-Progress/Completed).
- **Profile Management:** Users can update their profile details.
- **Security:** Protected routes ensure unauthenticated users cannot access sensitive data.

---

2. Technology Stack

Frontend (Client)

- **Framework:** React.js (via Vite)
- **Styling:** Tailwind CSS (Responsive Design)
- **Routing:** React Router DOM (v6)
- **HTTP Client:** Axios (with Interceptors for automatic token handling)

Backend (Server)

- **Runtime:** Node.js
- **Framework:** Express.js
- **Database:** MongoDB (via Mongoose ODM)
- **Authentication:** JWT & BcryptJS

---

3. Setup & Installation Guide

Prerequisites

- Node.js installed on your machine.
- A MongoDB connection string (local or Atlas).

Step 1: Backend Setup

1. Navigate to the server folder: cd server
2. Install dependencies: npm install
3. Create a .env file in the root and add:

Code snippet

MONGO_URI=your_mongodb_connection_string

JWT_SECRET=your_secret_key_here

PORT=5000

4. Start the server: npm start

Step 2: Frontend Setup

1. Navigate to the client folder: cd client
2. Install dependencies: npm install
3. Start the development server: npm run dev
4. Open your browser to the URL shown (usually http://localhost:5173).

---

4. API Documentation

Base URL: http://localhost:5000/api

A. Authentication Endpoints

1. Register User

- **Endpoint:** POST /users/register
- **Description:** Creates a new user account and returns a token.
- **Body:**

JSON

```
{
  "name": "Vinit Kumar",
  "email": "vinit@example.com",
  "password": "securepassword123"
}
```

- **Response (201):**

JSON

```
{
  "_id": "651a...",
  "name": "Vinit Kumar",
  "email": "vinit@example.com",
  "token": "eyJhbG..."
}
```

2. Login User

- **Endpoint:** POST /users/login
- **Description:** Authenticates a user and returns a token.
- **Body:**

JSON

```
{
  "email": "vinit@example.com",
  "password": "securepassword123"
}
```

3. Update Profile (Protected)

- **Endpoint:** PUT /users/profile
- **Headers:** Authorization: Bearer <token>
- **Body:**

JSON

```
{
```

"name": "Vinit K. Updated"

}

B. Task Endpoints (Protected)

All Task endpoints require the Authorization: Bearer <token> header.

1. Get Tasks (With Search & Filter)

- **Endpoint:** GET /tasks
- **Query Parameters:**
    - search: (Optional) Keyword to search task titles.
    - status: (Optional) Filter by 'pending', 'in-progress', or 'completed'.
- **Example URL:** GET /tasks?search=meeting&status=pending
- **Response (200):** Array of task objects.

2. Create Task

- **Endpoint:** POST /tasks
- **Body:**

JSON

{

"title": "Complete Frontend Task",

"description": "Finish the documentation"

}

3. Update Task

- **Endpoint:** PUT /tasks/:id
- **Description:** Updates task details or status.
- **Body:**

JSON

{

"status": "completed"

}

4. Delete Task

- **Endpoint:** DELETE /tasks/:id
- **Description:** Permanently removes a task.

---

5. Scalability Strategy (Production Plan)

This section addresses how I would scale this integration for a high-traffic production environment.

Frontend Scalability

1. **State Management:** For larger datasets, I would migrate from simple React State to **Redux Toolkit** or **React Query** to handle server-state caching and background refetching efficiently.
2. **Performance:** Implement **Code Splitting** (React.lazy) to load routes only when needed, reducing the initial bundle size.
3. **CDN Integration:** Serve static assets (images, CSS) via a CDN (e.g., Cloudflare) to reduce latency for global users.

Backend Scalability

1. **Database Indexing:** Add indexes to MongoDB fields like user and status to ensure search queries remain fast as the dataset grows to millions of records.
2. **Caching Layer:** Implement **Redis** to cache frequently accessed data (like user profiles or task lists), reducing the load on the primary database.
3. **Horizontal Scaling:** Use **Node.js Clustering** or Docker/Kubernetes to run multiple instances of the backend API, managed by a Load Balancer (Nginx) to distribute traffic evenly.
4. **Microservices:** If the app grows significantly, I would decouple the "Authentication" logic and "Task Management" logic into separate microservices.