Task Management App Development Document

1. Introduction

The Task Management App is designed to help users manage their tasks efficiently. The application consists of a frontend built with React, a backend developed using Node.js/Express, and a database using MongoDB.

2. Technology Stack

Frontend

- **Framework:** React.js
- **State Management:** Redux
- **Styling:** CSS Modules / Bootstrap
- **Routing:** React Router
- **Package Management:** npm
- **API Communication:** Axios / Fetch API
- **Build Tool:** Webpack

Backend

- **Framework:** Node.js
- **Authentication:** JWT (JSON Web Token)
- **Database Interaction:** MySQL DB
- **API Security:** CORS

Database

- **Database Type:** MySQL DB (SQL)
- **Hosting:** Local MySQLDB Server
- **Schema Design:** Collections for Users and Tasks

3. Project Structure

```
Task Management App/
│── frontend/          # Frontend React app
│── backend/           # Backend Node.js server
```

```
├── database/          # Database configurations and migrations
├── README.md          # Project documentation
├── .gitignore         # Git ignored files
```

Frontend Structure

```
frontend/
├── public/            # Static assets
├── src/               # Source code
│   ├── components/    # Reusable UI components
│   ├── pages/         # Page components (e.g., Dashboard, Task View)
│   ├── store/         # State management (if using Redux)
│   ├── hooks/         # Custom hooks (if applicable)
│   ├── services/      # API communication services
│   ├── App.js         # Main application file
├── package.json       # Dependency management
├── README.md          # Project documentation
├── .gitignore         # Git ignored files
```

Backend Structure

```
backend/
├── controllers/       # Handles business logic
├── models/            # Database models
├── routes/            # API endpoints
├── middleware/        # Authentication, error handling
├── config/            # Environment variables
├── server.js          # Main server entry point
├── package.json       # Dependency management
├── .env               # Environment variables
```

Database Schema

- **Users Table**
    - `id`: Unique identifier
    - `name`: String
    - `email`: String (unique)
    - `password`: Hashed password
    - `createdAt`: Timestamp
- **Tasks Table**

- `id`: Unique identifier
- `userId`: Reference to Users
- `title`: String
- `description`: String
- `status`: Enum (Pending, In Progress, Completed)
- `dueDate`: Timestamp
- `createdAt`: Timestamp

4. Features

4.1 User Authentication

- Users can sign up, log in, and log out securely.
- Authentication handled using JWT.

4.2 Task Management

- Users can create, edit, and delete tasks.
- Task status tracking (e.g., Pending, In Progress, Completed).

4.3 Dashboard

- Displays task summary, upcoming tasks, and productivity insights.

4.4 Notifications

- Users receive alerts for upcoming deadlines and important updates.

5 API Endpoints

Base URL

http://localhost:5000/api

Authentication

1. User Registration

Endpoint:

POST /auth/register

Description: Registers a new user and returns a JWT token.

Request Body:

```
{
  "name": "John Doe",
  "email": "johndoe@example.com",
  "password": "securepassword"
}
```

Response:

```
{
  "message": "User registered successfully",
  "token": "JWT_TOKEN"
}
```

2. User Login

Endpoint:

POST /auth/login

Description: Authenticates a user and returns a JWT token.

Request Body:

```
{
  "email": "johndoe@example.com",
  "password": "securepassword"
}
```

Response:

```
{
  "message": "Login successful",
  "token": "JWT_TOKEN"
}
```

Tasks

3. Create Task

Endpoint:

POST /tasks

Description: Creates a new task for the authenticated user.

Headers:

Authorization: Bearer JWT_TOKEN

Request Body:

```
{
  "title": "Complete Project Report",
  "description": "Finish writing the report by Monday",
  "category": "Work",
  "status": "pending",
  "priority": "High",
  "due_date": "2024-03-01T10:00:00Z"
}
```

Response:

```
{
  "message": "Task created successfully",
  "task": { "id": 1, "title": "Complete Project Report", "status": "pending" }
}
```

4. Get All Tasks

Endpoint:

GET /tasks

Description: Fetches all tasks for the authenticated user.

Headers:

Authorization: Bearer JWT_TOKEN

Response:


```
[
  {
    "id": 1,
    "title": "Complete Project Report",
    "category": "Work",
    "status": "pending",
    "priority": "High",
    "due_date": "2024-03-01T10:00:00Z"
  },
  {
    "id": 2,
    "title": "Grocery Shopping",
    "category": "Personal",
    "status": "completed",
    "priority": "Low",
    "due_date": "2024-02-25T15:00:00Z"
  }
]
```


5. Get Task by ID

Endpoint:

GET /tasks/{taskId}

Description: Fetches a specific task by its ID.

Headers:

Authorization: Bearer JWT_TOKEN

Response:


```
{
```

```
  "id": 1,
  "title": "Complete Project Report",
  "category": "Work",
  "status": "pending",
  "priority": "High",
  "due_date": "2024-03-01T10:00:00Z"
}
```


6. Update Task

Endpoint:

PUT /tasks/{taskId}

Description: Updates an existing task.

Headers:

Authorization: Bearer JWT_TOKEN

Request Body:


```
{
  "title": "Updated Task Title",
  "description": "Updated description",
  "category": "Work",
  "status": "completed",
  "priority": "Medium",
  "due_date": "2024-03-02T12:00:00Z"
}
```


Response:


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


7. Delete Task

Endpoint:

DELETE /tasks/{taskId}

Description: Deletes a task by its ID.

Headers:

Authorization: Bearer JWT_TOKEN

Response:

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

Task Filtering & Search

8. Filter Tasks by Category

Endpoint:

GET /tasks?category=Work

Description: Fetches tasks belonging to a specific category.

Response:

```
[
  {
    "id": 1,
    "title": "Complete Project Report",
    "category": "Work",
    "status": "pending"
  }
]
```

9. Filter Tasks by Status

Endpoint:

GET /tasks?status=completed

Description: Fetches tasks based on completion status.

Response:

```
[
  {
    "id": 2,
    "title": "Grocery Shopping",
    "category": "Personal",
    "status": "completed"
  }
]
```

10. Search Tasks by Title

Endpoint:

GET /tasks?search=Project

Description: Searches tasks by title.

Response:

```
[
  {
    "id": 1,
    "title": "Complete Project Report",
    "category": "Work",
    "status": "pending"
  }
]
```

Optional Enhancements (Bonus)

11. Add Priority Level

- Users can assign a priority level (`High`, `Medium`, `Low`) when creating or updating a task.
- Tasks can be filtered by priority using:

GET /tasks?priority=High

12. Due Date Reminder

- Implement automatic notifications for tasks nearing their due date.
- Example of a due date reminder notification:

Task "Complete Project Report" is due on 2024-03-01T10:00:00Z

13. Dashboard Summary

- A dashboard endpoint that provides an overview of task statistics:

Endpoint:

GET /dashboard

Response:


```
{
  "total_tasks": 10,
  "pending_tasks": 6,
  "completed_tasks": 4,
  "high_priority_tasks": 3
}
```