Project Title: Task Manager REST API

Developer

Bhumika Dadlani
B. Tech – Artificial Intelligence and Data Science

Objective

The primary goal of this project is to build a full-stack Task Management System that allows users to **create, read, update, and delete tasks**, along with tracking progress via statistics. The backend exposes a RESTful API built with **Node.js and Express.js**, and the frontend provides a dynamic and styled interface using **React.js**, consuming the API to offer a seamless user experience.

This Task Manager aims to help users:

- Keep track of multiple tasks
- Monitor task progress using status tracking
- Visually manage task lifecycles with simple interactions

Tech Stack

- **Backend:** Node.js, Express.js
- **Testing Tool:** Postman
- **Frontend (Optional Visualization):** React.js (from App.js and App.css)
- **Styling:** CSS (for UI visualization)
- **Middleware:** Express JSON parser, CORS

Core Functionalities:

1. Add New Task
2. View All Tasks
3. Update Task Status
4. Edit Task Content

5. Delete Task
6. Task Statistics Panel (Total, Pending, In Progress, Completed)

Task Properties:

- `id` (unique identifier)
- `title` (required)
- `description` (optional)
- `status` (`pending`, `in-progress`, `completed`)
- `createdAt`, `updatedAt`

API Features and Endpoints

API Endpoints

Method	Endpoint	Description
GET	/api/tasks	Get all tasks
GET	/api/tasks/:id	Get task by ID
POST	/api/tasks	Create new task
PUT	/api/tasks/:id	Update title, description, and status
PATCH	/api/tasks/:id/status	Update status only
DELETE	/api/tasks/:id	Delete a task
GET	/api/stats	Get task statistics summary

1.Add New Task

- **Method:** POST
- **Endpoint:** `/api/tasks`
- **Request Body:**

```
json
{
  "title": "Example Task",
  "description": "This is a new task",
```

```
  "status": "pending"  // Optional, defaults to "pending"
}
```

- **Response:** JSON object with created task and success message.

2. Get All Tasks

- **Method:** GET
- **Endpoint:** `/api/tasks`
- **Query Parameters (Optional):** `?status=pending` to filter
- **Response:** List of all or filtered tasks.

3. Get Specific Task

- **Method:** GET
- **Endpoint:** `/api/tasks/:id`
- **Response:** JSON object of the task by ID.

4. Update Task

- **Method:** PUT
- **Endpoint:** `/api/tasks/:id`
- **Request Body:**

```
json
{
  "title": "Updated Title",
  "description": "Updated description",
  "status": "in-progress"
}
```

- **Response:** Updated task with success message.

5. Update Task Status Only

- **Method:** PATCH
- **Endpoint:** `/api/tasks/:id/status`
- **Request Body:**

```
json
{
  "status": "completed"
}
```

- **Response:** Updated task with status changed.

6. Delete Task

- **Method:** DELETE
- **Endpoint:** `/api/tasks/:id`
- **Response:** Deleted task with confirmation.

7. Task Statistics

- **Method:** GET
- **Endpoint:** `/api/stats`
- **Response:**

```
json
{
  "total": 10,
  "pending": 4,
  "in-progress": 3,
  "completed": 3
}
```

Input Validation

- Title is mandatory.
- Status must be one of: `pending`, `in-progress`, `completed`.

Middleware Used

- `express.json()` and `express.urlencoded()` to parse JSON and form data.
- `cors()` to enable frontend-backend communication.
- Logging middleware for timestamped request logs.
- Error-handling and 404 middleware for API robustness.

Testing Using Postman

- All endpoints were tested thoroughly using Postman.
- Status codes and error messages were verified.
- Test cases included:
    - Creating tasks with and without description
    - Invalid status inputs
    - Fetching non-existent task ID

- Deletion and re-fetch of deleted task

Test Cases Covered

Test Case Description	Endpoint Tested	Expected Result
Add valid task	POST `/api/tasks`	201 Created with task data
Add task without title	POST `/api/tasks`	400 Bad Request
Update status to invalid value	PATCH `/api/tasks/:id/status`	400 Bad Request
Get tasks with status filter	GET `/api/tasks?status=pending`	200 OK with filtered list
Delete non-existing task	DELETE `/api/tasks/:id`	404 Not Found
Get task statistics	GET `/api/stats`	200 OK with JSON summary

Frontend Design (React.js)

The frontend (`App.js` and `App.css`) is a **single-page React application** that interacts with the backend API. It enables task creation, viewing, editing, deletion, and status updates, while dynamically rendering statistics.

Components & State

- `tasks`, `title`, `description`, `error`, `stats` are managed via `useState`.
- `fetchData` uses `useEffect` and `useCallback` to handle API calls on load or after state updates.
- Buttons in the UI call `handleAddTask`, `handleDeleteTask`, and `handleToggleStatus` respectively.

UI Functionalities

- Form to add new tasks with validations
- Buttons for status toggling:
  - **Start** → changes status from `pending` to `in-progress`
  - **Complete** → from `in-progress` to `completed`
  - **Reopen** → from `completed` to `pending`
- Real-time statistics using `/api/stats` endpoint
- Responsive, styled list of tasks with class-based visual cues (color-coded by status)

Styling (App.css)

- Clean, minimalistic styling for improved UX
- Task statuses are color-coded:
  - `pending` → Yellow border
  - `in-progress` → Blue border
  - `completed` → Green border + strikethrough
- Form and buttons are styled for accessibility and ease of use

Example CSS Logic:

```
css
.task-item.pending { border-left: 5px solid #ffc107; }
.task-item.in-progress { border-left: 5px solid #007bff; }
.task-item.completed {
  border-left: 5px solid #28a745;
  text-decoration: line-through;
}
```

Future Scope

1. **Database Integration**: Persist tasks using MongoDB or PostgreSQL.
2. **User Authentication**: JWT-based login system.
3. **Deadline & Priority Features**: Add due dates, reminders, and priorities.
4. **Notifications**: Email or push notifications on deadlines.
5. **Search and Filter UI**: Add client-side filtering by keyword, date, etc.
6. **Deployment**: Host backend on platforms like Render, and frontend via Netlify or Vercel.

Sample Statistics Output

```
json
{
  "total": 5,
  "pending": 2,
  "in-progress": 1,
  "completed": 2
}
```

Workflow Diagram

```
text
User → React UI → REST API (Express) → Task Operations in Memory
              ↑                              ↓
       Real-time stats            Error handling
```

Project Folder Structure

```
bash
TaskManager/
├── server.js          # Node.js backend
├── App.js             # React frontend logic
├── App.css            # Styling
├── package.json       # Node project metadata
```

Learning Outcomes

- Implementing REST APIs using Node.js and Express.js
- Managing full-stack state using React hooks
- Handling async API requests and error boundaries
- Designing user-friendly frontend interfaces
- Structuring scalable project architecture

Conclusion

The Task Manager project showcases an end-to-end full-stack application involving RESTful APIs and React-based UI integration. It is highly extendable, simple to deploy, and serves as a solid foundation for advanced task management systems.

To View the Project

CodeSandbox

Github