

1. Simple Task API With Middleware

1.1. Overview

The Simple Task API is a RESTful service that allows users to manage tasks. It provides endpoints to create, retrieve, update, and delete tasks, with additional features such as task status updates and task filtering. The API is built with middleware for validating requests and handling errors, ensuring smooth and consistent interactions.

1.2. Technologies used

- **Node.js:** JavaScript runtime environment used to build server-side logic.
- **Express:** A web framework used to develop the RESTful API.
- **CORS:** Middleware enabling Cross-Origin Resource Sharing, allowing the API to be accessed from different domains.
- **Core Node.js Modules:**
 - fs: Handles file system operations to store tasks in a JSON file.
 - path: Manages and resolves file paths.

1.3. API Endpoints

1.3.1. Retrieve All Tasks

- Endpoint: `/api/tasks`
- Method: `GET`
- Description: Retrieves all tasks stored in the JSON file.

1.3.2. Create a New Task

- Endpoint: `/api/tasks`
- Method: `POST`
- Description: Adds a new task with the provided name, description, and status.
- Request Body: Requires name, description, and status fields (all mandatory).

1.3.3. Update a Task by ID

- Endpoint: `/api/tasks/:id`
- Method: `PUT`
- Description: Updates an existing task using its ID. The task's name, description, and status can be updated.
- Request body: Requires `name`, `description`, and `status` fields for the update.

1.3.4. Partially Update Tasks

- Endpoint: `/api/tasks/:id`
- Method: `PATCH`
- Description: Partially updates the status of an existing task using its ID.
- Request body: Requires at least one key value.

1.3.5. Delete a note by ID

- Endpoint: `/api/notes/:id`
- Method: `DELETE`
- Description: Removes a task based on the provided ID.
- Parameter: `id` (The ID of the Task to delete)

1.4. Middleware

1.4.1. Task Validation Middleware

- **Description:** Validates that required fields (name and status) are provided in the request body. Ensures that task data is complete and correctly formatted before proceeding.

1.4.2. Error Handling Middleware

- **Description:** Handles errors consistently across the API, such as invalid input or non-existent task IDs. It returns clear and structured error messages to the client.

1.4.3. Task Existence Middleware

- **Description:** Checks whether a task exists before allowing updates or deletions. This middleware prevents operations on tasks that do not exist.

1.5. Data Storage

Tasks are stored in a local JSON file named `tasks.json`, located within the `data` directory of the project. This file is managed using the `fs` module.

1.6. Error Handling

Middleware is used to manage potential errors such as:

- **Task Not Found:** When an operation is attempted on a non-existent task.
- **Validation Errors:** When required fields are missing or improperly formatted.
- **General Errors:** Catches and handles any other errors that may occur during request processing.

1.7. Additional Features

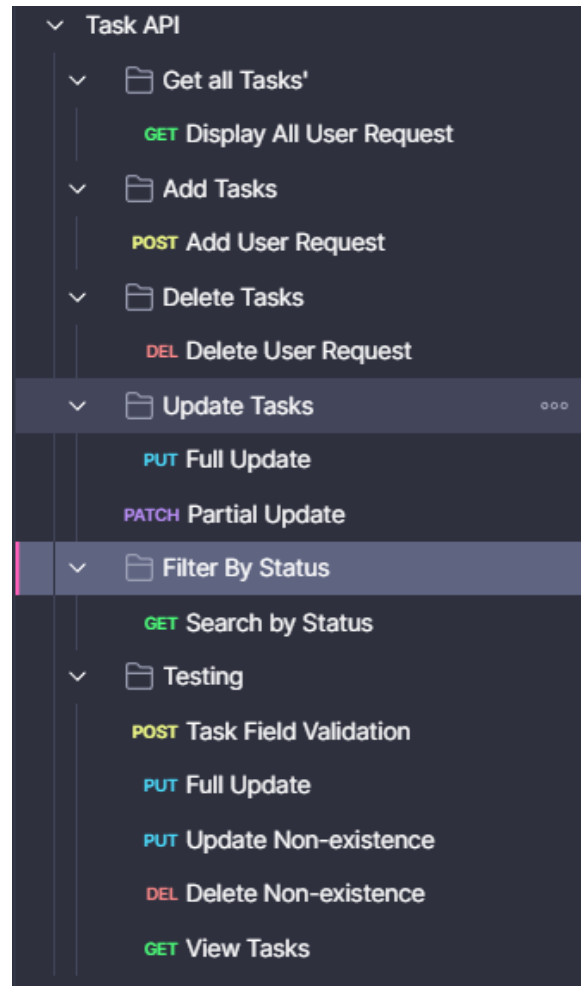
- **Task Filtering:** Implemented filters to allow tasks to be retrieved based on their status or priority, enhancing task management capabilities.

1.8. How to Run the Project

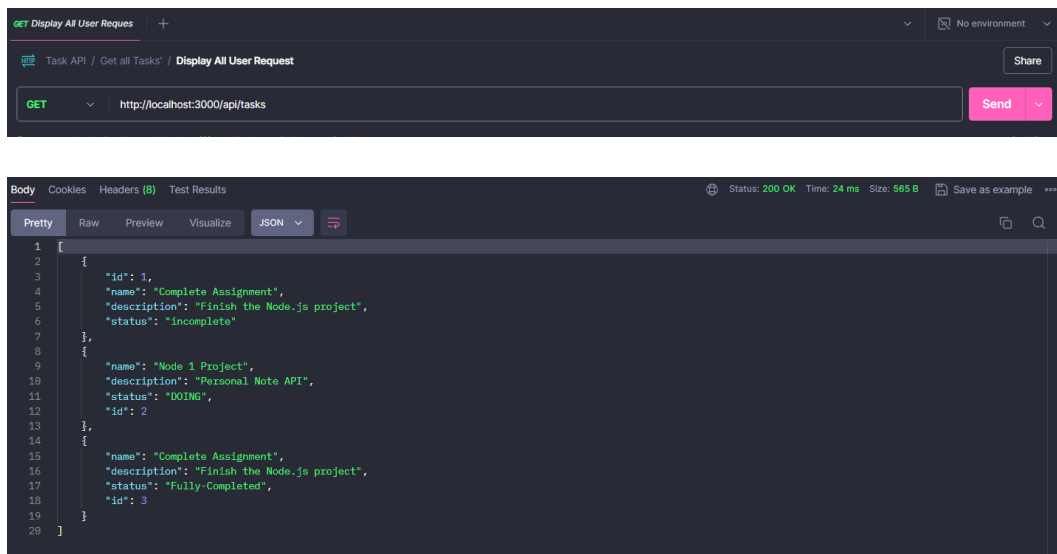
- Navigate to the main project directory and open a terminal.
- Run `npm install` to install all dependencies.
- Install necessary packages by running `npm install express cors nodemon`.
- Start the project by executing `npm -w node.js` in the terminal. The application will run on `http://localhost:3000`.

1.9. POSTMAN Configuration

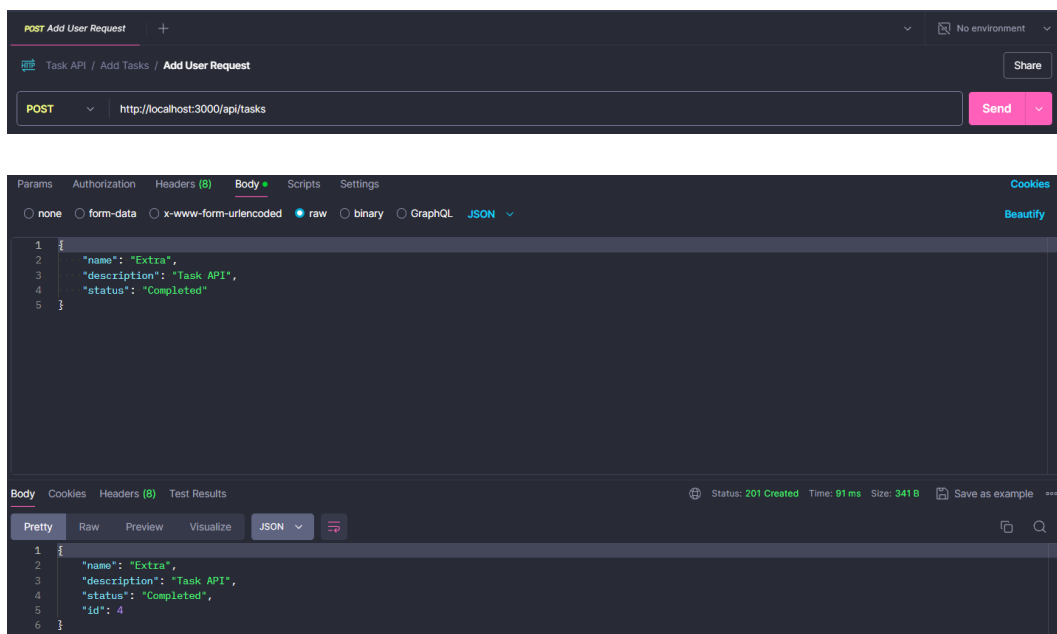
A structured folder setup is recommended for easy access and testing of the API endpoints:



a) **Get All Tasks:** Test the retrieval of all tasks.



b) **Add a Task:** Test the creation of a new task.



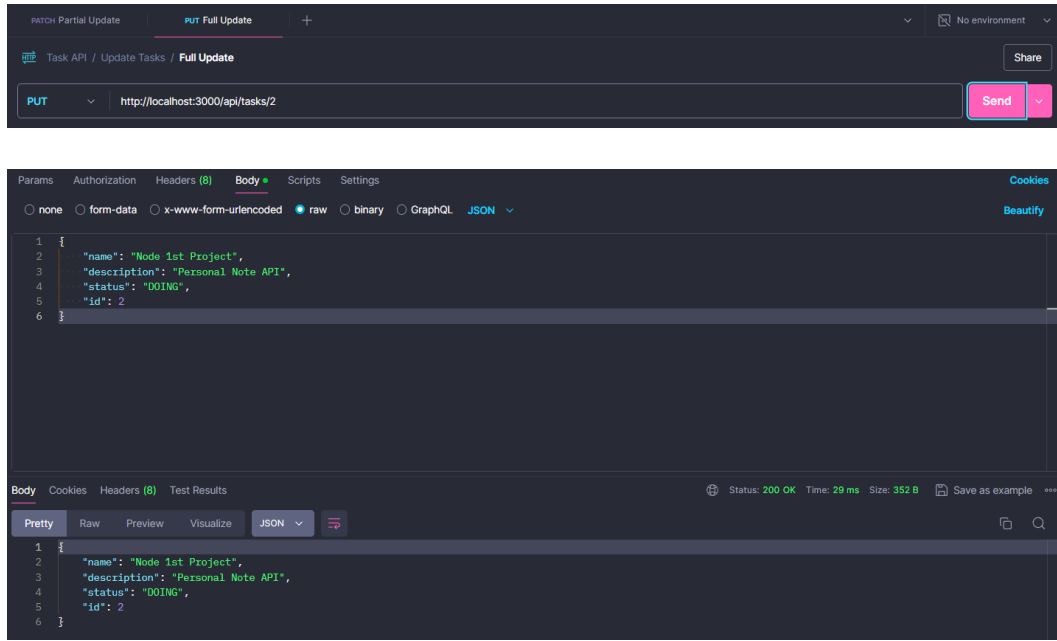
c) **Partially Update a Task:** Test the partial update of a task's status by ID.

```
{
  "name": "Complete Assignment",
  "description": "Finish the Node.js project",
  "status": "Fully-Completed",
  "id": 3
}
```

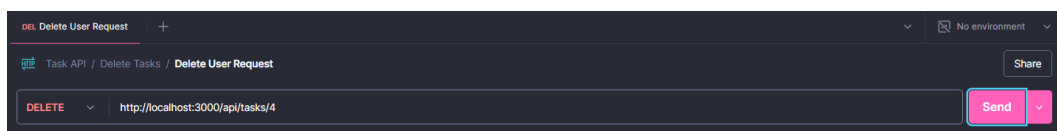
The screenshot shows a REST client interface with a PATCH request to `http://localhost:3000/api/tasks/3`. The request body is a JSON object: `{ "status": "Completed" }`. The response status is `200 OK` with a time of `31 ms` and a size of `369 B`. The response body is a JSON object: `{ "name": "Complete Assignment", "description": "Finish the Node.js project", "status": "Completed", "id": 3 }`.

d) **Full Update:** Test the Full update of a task's status by ID

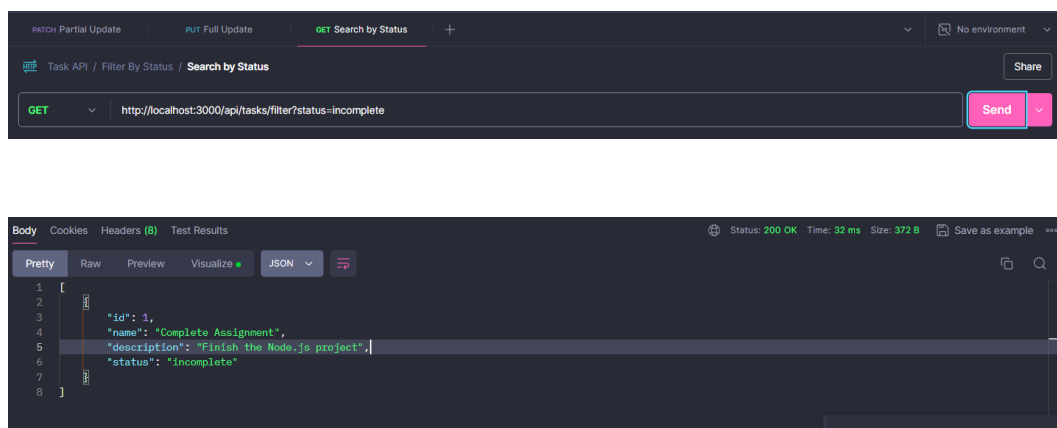
```
{
  "name": "Node 1 Project",
  "description": "Personal Note API",
  "status": "DOING",
  "id": 2
},
```



The screenshot shows a REST client interface with a PUT request to `http://localhost:3000/api/tasks/2`. The request body is a JSON object: `{ "name": "Node 1 Project", "description": "Personal Note API", "status": "DOING", "id": 2 }`. The response is a 200 OK status with a 29 ms time and 352 B size.

e) **Delete a Task:** Test the deletion of a task by ID:

The screenshot shows a REST client interface with a DELETE request to `http://localhost:3000/api/tasks/4`.

f) **Filter by Status:** Search the task Based on Status:

The screenshot shows a REST client interface with a GET request to `http://localhost:3000/api/tasks/filter?status=incomplete`. The response is a 200 OK status with a 32 ms time and 372 B size. The response body is a JSON array: `[{ "id": 1, "name": "Complete Assignment", "description": "Finish the Node.js project.", "status": "incomplete" }]`.

1.10. GitHub Profile

<https://github.com/devrajKhadka-smiley/Task-API.git>