# Project Description: Task Manager API

**Objective:**

Build a Task Manager API to practice JavaScript and Express.js basics. You'll be working with a provided project structure..

# Project Requirements:

## Endpoints to Implement:

* GET /api/tasks: Fetch and return all tasks.
* GET /api/tasks/:id: Fetch and return a specific task by ID.
* POST /api/tasks: Create a new task. Set id, created\_at, updated\_at and all the required fields.
* PUT /api/tasks/:id: Update an existing task by ID. Modify fields and update updated\_at.
* DELETE /api/tasks/:id: Delete a task by ID.

**NOTE:** Make sure newly created and updated task follows the old tasks schema.

## Folder Structue:

Week3\_project/

 ├── src/

 │   ├── controllers/   ***# Handling HTTP requests***

 │   │   ├── taskController.js

 │   ├── errors/            ***# Custom error handling***

 │   │   └── errorHandler.js

 │   ├── middlewares/       ***# Middleware functions***

 │   │   └── validationMiddleware.js

 │   ├── routes/            ***# Route definitions***

 │   │   ├── taskRoutes.js ***# All tasks routes***

 │   │   ├── index.js ***# all routes combined in this file***

 │   ├── utils/             ***# Utility functions and helpers***

 │   │   ├── catchAsyncError.js

 │   ├── data/              ***# Dummy data files***

 │   │   └── tasks.json

 │   ├── server.js          ***# Entry point for the application***

 |

 ├── .gitignore

 ├── package.json

 ├── README.md

## Additional Notes:

* The tasks.json file in the data folder will be used to store task data. Implement file handling to read from and write to this file.
* Ensure to validate and handle errors properly for every route.
* Refer to the hints and TODO comments in the server.js and index.js files for guidance.

## Instructions:

1. **Set Up Your Environment:**
   * Set up your .env file for environment variables in code.
2. **Implement Routes and Controllers:**
   * Complete the TODO tasks in the server.js and routes/index.js files.
   * Implement the required CRUD operations and ensure they are functioning correctly.
3. **Validation Middleware:**

* Implement middleware to validate incoming requests:
  + Ensure title and description are non-empty strings.
  + Check that status is one of "pending", "in-progress", "completed".
  + Validate priority as "low", "medium", or "high".
* **Examples:**
  + For POST /api/tasks, validate title and description:
    - Return an error if either is a number or empty.
  + For PUT /api/tasks/:id, ensure status is a valid value:
    - Return an error for values other than "pending", "in-progress", or "completed".

1. **Helper Functions in utils Folder:**

* Use helper functions to keep controllers clean:
  + **Sanitization Example:** Create a function to trim whitespace from strings and format data properly.
* **How to Use:**
  + Import helper functions into controllers.
  + Use them to sanitize or prepare data before processing.

1. **Testing:**
   * Test your API endpoints using tools like Postman and verify that the API behaves as expected and handles errors properly.

By implementing these steps, you ensure clean, well-organized code and consistent data handling.