Name: Pratiksha Angad Gore

Email: pratikshagore08@gmail.com

Contact No: 9834939003

Task Manager API Documentation

- ## Introduction:

The Task Manager API provides endpoints to manage a collection of tasks, including creating, retrieving, updating, and deleting tasks. This API is built using Node.js and Express.js and stores tasks in memory.

- ## Table of Contents

1. Prerequisites

Node.js and npm installed

Git installed

2. Installation

Clone the repository:

git clone https://github.com/Pratiksha1302/osumare-backend.git

cd osumare-backend

Setup the backend:

cd backend

npm install

3.Running the Server

npm start

The server will run at http://localhost:8080.

4. API Endpoints

5. GET /tasks

Retrieve a list of all tasks.

URL: http://localhost:8080 /tasks

Method: GET

Response:

```
[
  {
    "id": 1,
    "title": "Task Title",
    "description": "Task Description"
  }
]
```

6. GET /tasks/

Retrieve a specific task by ID.

URL: /tasks/:id

Method: GET

Response:

```
{
   "id": 1,
   "title": "Task Title",
   "description": "Task Description"
}
```

7. POST /tasks

Create a new task.

URL: /tasks

Method: POST

Request Body:

```
{
   "title": "Task Title",
   "description": "Task Description"
}
```

Response:

```
{
"id": 1,
 "title": "Task Title",
  "description": "Task Description"
}
```

8. PUT /tasks/

Update an existing task by ID.

URL: /tasks/:id

Method: PUT

Request Body:

```
{
  "title": "Updated Title",
  "description": "Updated Description"
}
```

Response:

```
{
  "id": 1,
  "title": "Updated Title",
  "description": "Updated Description"
}
```

9. DELETE /tasks/

Delete a task by ID.

URL: /tasks/:id

Method: DELETE

Response:

```
[
  {
    "id": 1,
    "title": "Deleted Title",
    "description": "Deleted Description"
  }
]
```

10. Testing the API

You can test the API using tools like Postman or curl.

1. **Create a Task**:

   curl -X POST http://localhost:8080/tasks -H "Content-Type: application/json" -d '{"title": "Sample Task", "description": "This is a sample task."}'

2. **Get All Tasks**:

   curl http://localhost:8080/tasks

3. **Get a Task by ID**:

   curl http://localhost:8080/tasks/1

4. **Update a Task**:

   curl -X PUT http://localhost:8080/tasks/1 -H "Content-Type: application/json" -d '{"title": "Updated Task", "description": "This is an updated task."}'

5. **Delete a Task**:

   curl -X DELETE http://localhost:8080/tasks/1

11. Error Handling

The API uses appropriate status codes to indicate the result of operations:

- 200 OK: The request was successful.
- 201 Created: The resource was successfully created.
- 400 Bad Request: The request was invalid or cannot be served.
- 404 Not Found: The requested resource could not be found.
- 500 Internal Server Error: An error occurred on the server.

Additionally, an error handling middleware is used to catch unexpected errors and respond with a 500 Internal Server Error.

```
app.use((err, req, res, next) => {
  console.error(err.stack);
```

```
  res.status(500).send('Something broke!');
});
```

12. Approach

1. **Environment Setup**
    - Installed necessary dependencies: express, ejs, method-override, and uuid.

2. **Routes and Handlers**
    - Defined routes for each CRUD operation.
    - Implemented handlers to process incoming requests and generate appropriate responses.

3. **Views**
    - Created EJS templates for displaying tasks, creating new tasks, editing tasks, and showing error messages.
4. **Validation and Error Handling**
    - Added validation to ensure that tasks have both a title and description.
    - Implemented error handling to provide user-friendly error messages.
5. **Testing**
    - Tested the API using curl commands and a web browser to ensure all endpoints function correctly.
    - Verified that appropriate status codes and error messages are returned.

13. Challenges and Solutions

1. **Validation**
    - **Challenge**: Ensuring that all required fields are present for POST and PUT requests.

- **Solution**: Implemented validation logic in the POST and PUT request handlers.

2. **In-Memory Storage**
    - **Challenge**: Maintaining unique IDs for tasks.
    - **Solution**: Used the uuid package to generate unique IDs for each task.

14. Conclusion

The Task Manager API was successfully developed and tested. It supports all required CRUD operations and includes validation and error handling. The use of EJS for rendering views enhances the user experience by providing a clear and organized way to interact with the tasks.