

# Service Oriented Computing

## Task Tracker

### 1. Introduction

A task management system that allows users to create, edit, delete, and complete tasks. It is designed to be simple, minimalistic, and responsive. The project integrates a React-based frontend with a NestJS backend and uses MongoDB as its database to persist data. The project follows a Service-Oriented Architecture (SOA):

**Frontend:** React (**TypeScript**).

**Backend:** NestJS for REST API development.

**Database:** MongoDB for data persistence.

### 2. Features

- **Create a Task:** Add tasks with titles and optional edit mode.
- **Read Tasks:** Fetch all tasks from the database and display them.
- **Update Tasks:** Edit task titles or mark tasks as completed.
- **Delete Tasks:** Remove unwanted tasks from the list.

### 3. API Documentation

Method	Endpoint	Desription	Body Example
GET	/tasks	Fetch All Tasks	
POST	/tasks	Create a new task	{ "title": "New Task" }
PATCH	/tasks/:id	Update a task	{ "title": "Updated Task", "completed": true }
DELETE	/tasks/:id	Delete a task by ID	

## 4. POSTMAN

### POST Request: Create a Tasks

POST

http://localhost:4000/tasks

Send

Params Authorization Headers (8) **Body** Scripts Settings

☐ none ☐ form-data ☐ x-www-form-urlencoded ☒ raw ☐ binary ☐ GraphQL **JSON**

1 {  
2   "title": "My first333 local task",  
3   "description": "This is stored in local MongoDB."  
4 }

Cookies Beautify

### GET Request: Retrieve All Tasks

Task Manager / all tasks

Save Share

GET

http://localhost:4000/tasks

Send

### GET Request: By ID

Task Manager / all tasks

Save Share

GET

http://localhost:4000/tasks

Send

### PATCH Request: Update by ID

Task Manager / update task

Save Share

PATCH

http://localhost:3000/tasks/675d757a4e9b2bd3f0bcee50

Send

Params Authorization Headers (8) **Body** Scripts Settings

☐ none ☐ form-data ☐ x-www-form-urlencoded ☒ raw ☐ binary ☐ GraphQL **JSON**

1 {  
2   "title": "Updated Local Task Title",  
3   "description": "Updated Description",  
4   "isCompleted": true  
5 }

Cookies Beautify

### DELETE Request: By ID

Task Manager / delete a task

Save Share

DELETE

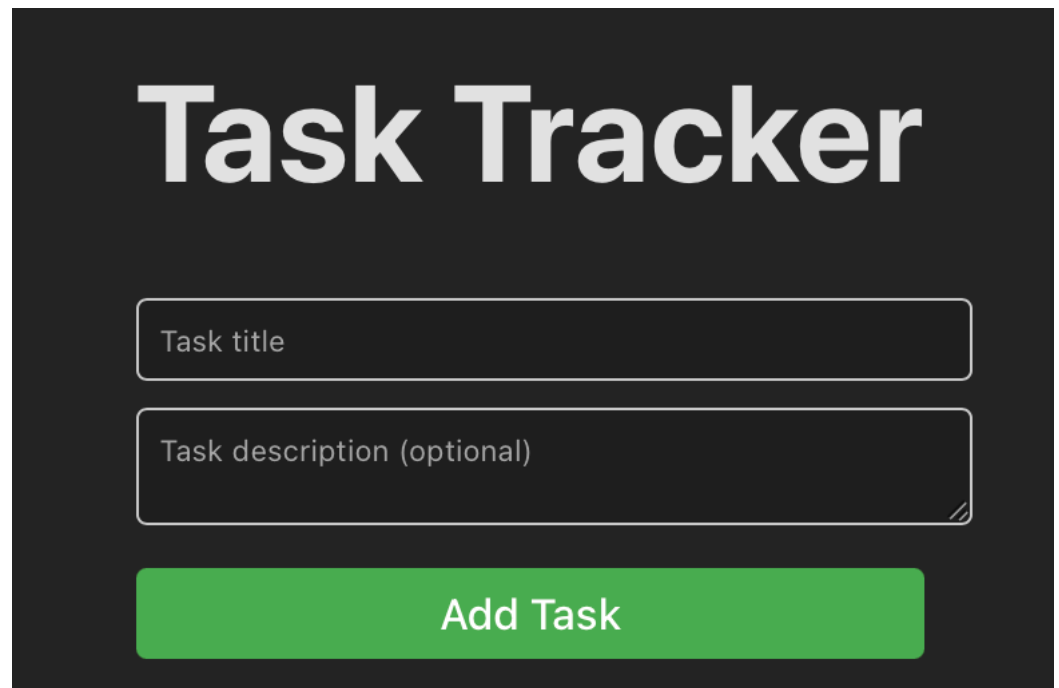
http://localhost:3000/tasks/675d757a4e9b2bd3f0bcee50

Send

## 5. GUI

### Creating a Task

Once you open the website you get to fill a form of 2 textfields, one is the title and the other is the description which is optional, once you click on **Add Task** the client sends a request to the api gateway which communicates with the backend then to the database to create a task in the local MongoDB.

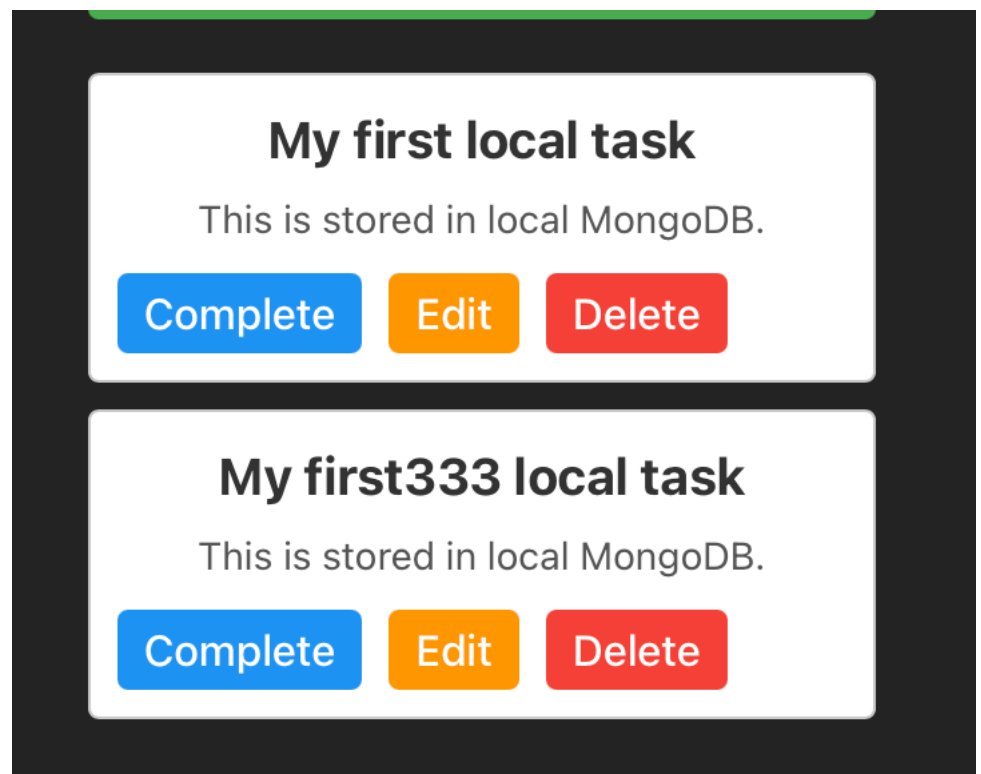
A dark-themed form titled "Task Tracker" in large white letters. Below the title are two text input fields: the first is labeled "Task title" and the second is labeled "Task description (optional)". At the bottom of the form is a large green button with the text "Add Task" in white.

### Task List

After creating the task, it is saved in the database and then retrieved again to be shown on the client side. Each task in the task list has the ability to be changed to complete and non-completed.

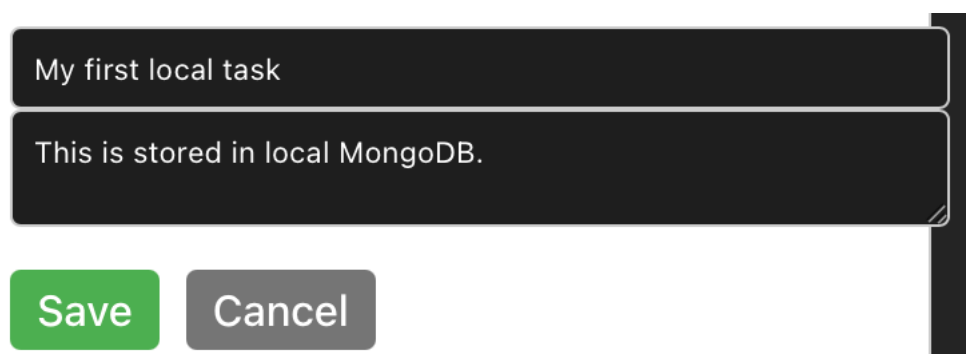
### Task Management

The user have the ability to edit, delete, or mark the task as done and even undo the changes.

A dark-themed interface showing a list of tasks. Each task is displayed in a white box with a dark border. The first task is titled "My first local task" and the second is titled "My first333 local task". Below each title is the text "This is stored in local MongoDB." and three colored buttons: "Complete" (blue), "Edit" (orange), and "Delete" (red).

### Edit Button

Once clicked, the user is shown a form where both of the title can be changed up, and there the user can either save their changes or cancel the changes

A dark-themed form for editing a task. It contains two text input fields: the first is labeled "My first local task" and the second is labeled "This is stored in local MongoDB.". At the bottom of the form are two buttons: a green "Save" button and a grey "Cancel" button.

## 6. Backend

### RESTful APIs

The backend exposes CRUD (Create, Read, Update, Delete) endpoints for managing tasks.

Endpoints are defined following RESTful principles, ensuring clear and predictable interactions.

### Task Schema

Task data is structured using a Mongoose schema, which ensures consistency in the MongoDB collection.

### The schema includes the following fields

- **title (string, required):** Title of the task.
- **completed (boolean, optional):** Indicates if the task is completed. Defaults to false.
- **isEditing (boolean, optional):** Tracks edit mode status. Defaults to false.

### DTOs (Data Transfer Objects)

DTOs are used for data validation and to enforce a consistent structure in incoming requests.

**Example DTOs include:**

- **CreateTaskDto** for creating tasks.
- **UpdateTaskDto** for updating tasks.

### Modular Architecture

The backend follows NestJS's modular structure:

- **tasks.module.ts:** Registers the Task schema and imports dependencies.
- **tasks.service.ts:** Contains the core business logic for task management.
- **tasks.controller.ts:** Defines the routes and handles HTTP requests.

## 7. Data Flow

Following a clean client-server architecture. The user interacts with the React-based UI to perform operations like adding, updating, or deleting tasks. Each action triggers event handlers in the React components, which call centralized API functions defined in `taskService.ts`. These functions use Axios to send HTTP requests to the NestJS backend. The backend receives these requests through controllers, processes them using services, and interacts with the MongoDB database to persist or retrieve data. The updated data or responses are sent back to the frontend, where the React state is updated dynamically to reflect the changes, providing a smooth and responsive user experience.

## 8- Conclusion

A robust, full-stack task management system that demonstrates the integration of modern web technologies, including React, NestJS, and MongoDB. The project follows a modular and scalable architecture, ensuring clean separation of concerns and efficient data flow between the client and server. By implementing core CRUD functionalities, validation, and error handling, TaskFlow provides a solid foundation for managing tasks effectively. Future enhancements, such as filtering, user authentication, and advanced task features, can further improve the application's capabilities. TaskFlow is an excellent showcase of full-stack development principles, delivering a functional and minimalistic user experience.