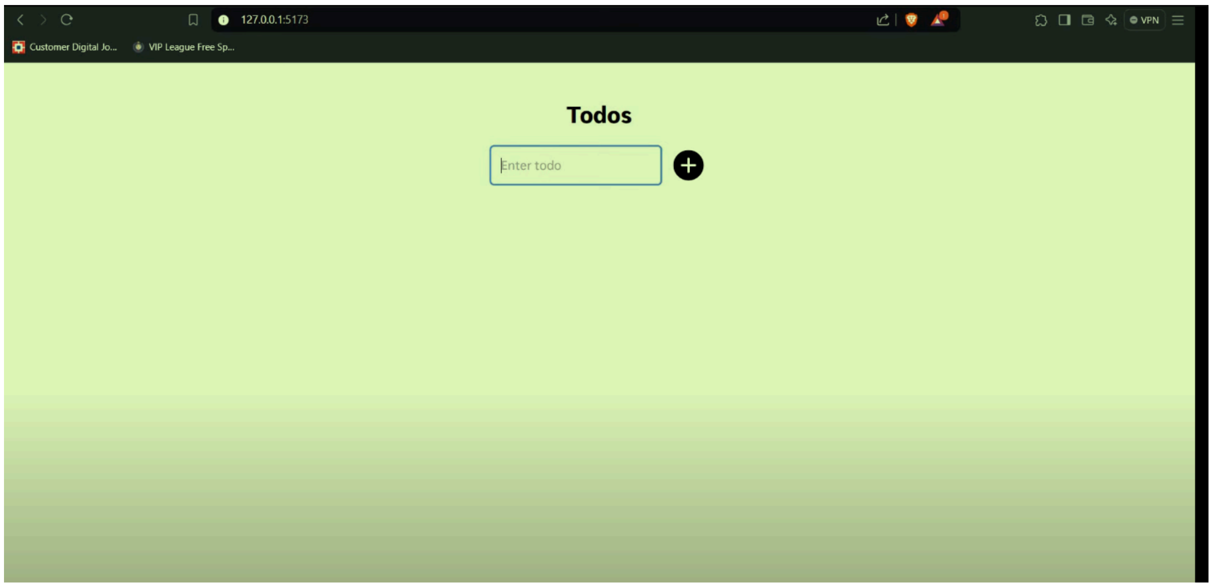
Building a Todo List Application with MERN Stack and Tailwind CSS

Table of Contents

Project Overview



This project is a full-stack Todo List application that allows users to create, read, update, and delete tasks. The application features a clean user interface built with React.js and Tailwind CSS, with a robust backend powered by Node.js, Express, and MongoDB.

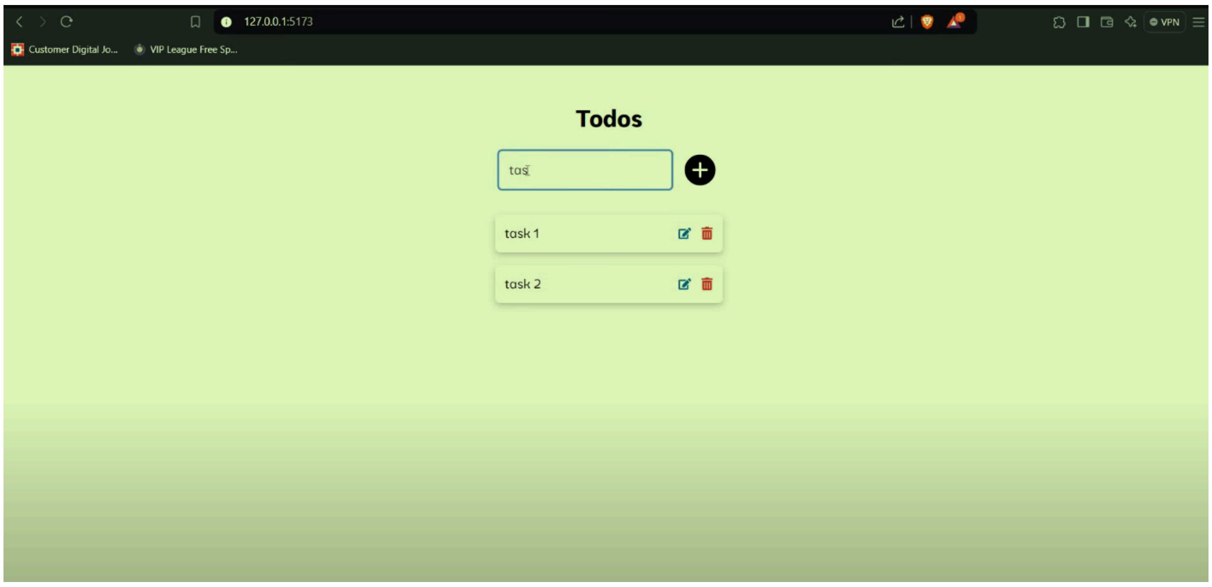
Key Features

- CRUD operations for todo items
- Real-time updates
- Responsive design
- Data persistence

- Task categorization

Prerequisites

- Node.js (v14 or higher)
- MongoDB
- npm or yarn
- Git
- Code editor (VS Code recommended)



Tech Stack

Frontend

- React.js
- Tailwind CSS
- React Icons
- Axios for API calls
- React Router for navigation

Backend

- Node.js
- Express.js
- MongoDB
- Mongoose
- JSON Web Tokens (JWT)
- Cors
- Dotenv

Project Structure

Copy

```
frontend/
├── public/
├── src/
│    │    ├── components/
│    │    ├── context/
│    │    ├── pages/
│    │    ├── services/
│    │    └── App.js
│    ├── tailwind.config.js
│    └── package.json
backend/
│    ├── controllers/
│    ├── models/
│    ├── routes/
└── README.md
```

Frontend Implementation

1. Setup React Project

bash

Copy

```bash
npx create-react-app client
cd client
npm install -D tailwindcss postcss autoprefixer
npx tailwindcss init -p
```

2. Configure AppCSS

```css
@import
url('https://fonts.googleapis.com/css2?family=Moderustic:wght@300..800&display=swap');


* {

    box-sizing: border-box;

    margin: 0;

    padding: 0;

    list-style: none;
}


.moderustic-12345 {
    font-family: "Moderustic", sans-serif;
    font-optical-sizing: auto;
    font-weight: 400;
    font-style: normal;
}
```

Desgin App.js

```
import { useEffect, useState } from 'react'
import './App.css'
import { IoIosAddCircle } from "react-icons/io";
import axios from 'axios';
import Todo from './components/Todo';


function App() {


  const [userInput, setUserInput] = useState('')
  let [todos, setTodos] = useState([])
  const [id, setId] = useState('')
  const [isUpdate, setIsUpdate] = useState(false)

  const url = 'http://localhost:3000'


  useEffect(() => {

    fetchTodos(`${url}/todo`)

  }, [])


  const fetchTodos = (url) => {

    axios.get(url).then((response) => {

      setTodos(response.data);

      setLoading(false)

    }).catch(error => console.log(error));

  }


  const handleSubmit = (e) => {

    e.preventDefault()


    const todo = {

      value: userInput

    }
```

```
if (!isUpdate) {


  axios
    .post(`${url}/todo/add`, todo)
    .then((res) => {
      const { _id, value } = res.data
      setUserInput('')
      setTodos((prev) => {
        return [...prev, { _id, value }]
      })
    }).catch(error => console.log(error.message));
}
else {
  axios
    .patch(`${url}/todo/${id}`, todo)
    .then((res) => {
      const { _id, value } = res.data
      setTodos((prev) => {
        return prev.map((todo) =>
            todo._id === _id ? { _id, value } : todo
          )
        })
        setUserInput('')
        setIsUpdate(false)
      }).catch(error => console.log(error.message));
  }

}

const getTodoById = (id) => {
  axios.get(`${url}/todo/${id}`).then((response) => {
    setUserInput(response.data.value)
  }).catch(error => console.log(error));
}

const editTodo = (id) => {
  setIsUpdate(true)
  getTodoById(id)
  setId(id)
  }


  const deleteTodo = (id) => {

    axios.delete(`${url}/todo/${id}`).then((response)
=> {

      todos = todos.filter(todo => todo._id !== id)

      setTodos(todos)

    }).catch(error => console.log(error));

  }


  return (

    <>

      <div className='moderustic-12345 w-full h-screen
flex flex-col'>

        <h1 className="text-3xl font-bold text-center
mt-12">

          Todos

        </h1>

        <div className='flex justify-center my-6'>

          <form onSubmit={handleSubmit} className='flex
justify-evenly gap-3'>

            <input
```

```
              type="text"

              value={userInput}

              onChange={(e) =>
setUserInput(e.target.value)}

              placeholder='Enter todo'

              className='shadow border rounded w-full
py-2 px-3 text-gray-700 focus:outline-none focus:ring-2
focus:ring-blue-500'

            />

            <button type="submit"><IoIosAddCircle
className='text-5xl' /></button>

          </form>

        </div>

        <ul className='todos flex justify-center
flex-col items-center'>

          {

            todos?.map(todo => {

              return <Todo key={todo._id}
_id={todo._id} value={todo.value} editTodo={() =>
editTodo(todo._id)} deleteTodo={() =>
deleteTodo(todo._id)} />

            })

          }

        </ul>
```

```
      </div>

    </>

  )

}



export default App

```

3. Create Todo Component

```
import { FaEdit } from "react-icons/fa";

import { FaTrashAlt } from "react-icons/fa";


const Todo = ({ _id, value, editTodo, deleteTodo }) =>
{

    return (<li className='rounded-md flex
justify-between w-72 p-3 shadow-item my-2'>

        <p className=''>{value}</p>

        <div className="update-delete flex items-center
gap-3">

            <FaEdit className=' text-sky-700
hover:cursor-pointer' onClick={editTodo} />

            <FaTrashAlt className='text-red-600
hover:cursor-pointer' onClick={deleteTodo} />

        </div>

    </li>)
}


export default Todo
```

Backend Implementation

1. Setup todo.js

```
const Todos = require('../models/todo')


const GetTodos = async (req, res) => {
    try {
        const todos = await Todos.find()
        if (todos)
            res.status(200).json(todos)
        else
            res.status(404).json({ message: 'No Todos
found!' })
    }
    catch (error) {
        res.status(500).json({ message: "Some error
occurred!" })
    }
}


const AddTodo = async (req, res) => {
    const { value } = req.body
```

```javascript
    if (!value) {
        res.status(400).json({ message: "All fields are required..." })
        return
    }
    try {
        const result = await Todos.create({
            value
        })
        // res.status(201).json({ message: "Todo Inserted!" })
        res.status(201).json(result)
    }
    catch (error) {
        // console.log(error.message)
        res.status(500).json({ message: "Some error occurred!" })
    }
}

const UpdateTodo = async (req, res) => {
    const { id } = req.params
    const { value } = req.body

    if (!id) {
        res.status(400).json({ message: "Invalid Id" })
        return
    }

    if (!value) {
        res.status(400).json({ message: "Field is
required..." })
        return
    }

    try {
        const result = await
Todos.findByIdAndUpdate(id, { value },{new:true})

        if (result)
            res.status(201).json(result)
        else
            res.status(404).json({ message: "Invalid
Id" })

    } catch (error) {
        // console.log(error.message)
        res.status(500).json({ message: "Invalid Id" })
    }
}

const DeleteTodo = async (req, res) => {

    const { id } = req.params
    if (!id) {
        res.status(400).json({ message: "Invalid Id" })
        return
    }

    try {
        const result = await
Todos.findByIdAndDelete(id)
        // console.log('result', result)

        if (result)
            res.status(200).json({ message: "Todo
deleted!" })
        else
            res.status(404).json({ message: "Invalid
Id" })
    }
    catch (err) {
        // console.log('err', err.message)
        res.status(500).json({ message: "Invalid Id" })
    }
}

const GetTodoById = async (req, res) => {

    const { id } = req.params
    if (!id) {
        res.status(400).json({ message: "Invalid Id" })
        return
    }

    try {
        const result = await Todos.findById(id)
        // console.log('result', result)

        if (result)
            res.status(200).json(result)
        else
            res.status(404).json({ message: "Invalid
Id" })
    }
    catch (err) {
        // console.log('err', err.message)
        res.status(500).json({ message: "Invalid Id" })
    }
}

module.exports = {
    GetTodos,
    AddTodo,
    UpdateTodo,
    DeleteTodo,
    GetTodoById
}
```

2. Create Todo Model

```
const mongoose=require('mongoose')
const TodoSchema=mongoose.Schema({
    value:{
        type:String,
        required:true
    }
})

const TodoModel=mongoose.model('todo',TodoSchema)
module.exports=TodoModel
```

3. Implement API Routes

```
const express = require('express')
const { GetTodos, AddTodo, DeleteTodo, UpdateTodo,
GetTodoById } = require('../controllers/todo')
const router = express.Router()

router.get('/', GetTodos)
router.post('/add', AddTodo)
router.route('/:id').get(GetTodoById).patch(UpdateTodo)
.delete(DeleteTodo)

module.exports = router
```

Database Setup

1. Create a MongoDB Atlas account
2. Create a new cluster
3. Get your connection string
4. Create a `.env` file in the server directory:

Copy

```
MONGODB_URI=your_mongodb_connection_string
JWT_SECRET=your_jwt_secret
```

Authentication

1. User Model

```
const mongoose=require('mongoose')

const TodoSchema=mongoose.Schema({

    value:{

        type:String,

        required:true

    }

})


const TodoModel=mongoose.model('todo',TodoSchema)

module.exports=TodoModel
```

Deployment

Frontend Deployment (Netlify)

1. Build your React application:

bash
Copy

```
cd client
npm run build
```

2. Deploy to Netlify:
- Connect your GitHub repository
- Set build command: `npm run build`
- Set publish directory: `build`

Backend Deployment (Heroku)

1. Create a Procfile:

Copy
```
web: node server.js
```

2. Deploy to Heroku:

bash
Copy
```
heroku create
git push heroku main
```

Additional Resources

- [React Documentation]
- [Tailwind CSS Documentation]
- [MongoDB Documentation]
- [Express.js Documentation]
- [Node.js Documentation]

Contributing

1. Fork the repository
2. Create your feature branch (`git checkout -b feature/AmazingFeature`)
3. Commit your changes (`git commit -m 'Add some AmazingFeature'`)
4. Push to the branch (`git push origin feature/AmazingFeature`)
5. Open a Pull Request