Developing a Todo Application using the MERN Stack

The MERN stack comprises MongoDB, Express.js, React, and Node.js. This guide will walk you through creating a Todo application using these technologies.

Setup Instructions

1. Backend Setup with Node.js, Express, and MongoDB

Step 1: Initialize the Backend Project

Create a new directory for your project and initialize a Node.js project:

```
mkdir todo-app-backend
cd todo-app-backend
npm init -y
```

Step 2: Install Required Dependencies

Install Express, Mongoose (for MongoDB), and other dependencies:

```
npm install express mongoose body-parser cors
```

Step 3: Set Up MongoDB

- If you don't have MongoDB installed, follow the instructions on the MongoDB website.
- Alternatively, use a cloud database like MongoDB Atlas.

Step 4: Create the Express Server

Create an index.js file and set up the server:

```
const express = require('express');
const mongoose = require('mongoose');
const bodyParser = require('body-parser');
const cors = require('cors');

const app = express();
app.use(bodyParser.json());
app.use(cors());

mongoose.connect('mongodb+srv://sandarbh:getting_node@cluster0.mongodb.net/<dbname>?retryWrites=true&w=majority, { useNewUrlParser: true, useUnifiedTopology: true });

const TodoSchema = new mongoose.Schema({
  text: String,
  completed: Boolean
});

const Todo = mongoose.model('Todo', TodoSchema);

app.get('/todos', async (req, res) => {
  const todos = await Todo.find();
```

```
  res.json(todos);
});

app.post('/todos', async (req, res) => {
  const newTodo = new Todo({
    text: req.body.text,
    completed: false
  });
  const savedTodo = await newTodo.save();
  res.status(201).json(savedTodo);
});

app.put('/todos/:id', async (req, res) => {
  const updatedTodo = await Todo.findByIdAndUpdate(
    req.params.id,
    { completed: req.body.completed },
    { new: true }
  );
  res.json(updatedTodo);
});

app.delete('/todos/:id', async (req, res) => {
  await Todo.findByIdAndDelete(req.params.id);
  res.status(204).end();
});

const PORT = process.env.PORT || 5000;
app.listen(PORT, () => {
  console.log(`Server running on port ${PORT}`);
});
```

Step 5: Run the Backend Server

```
node index.js
```

2. Frontend Setup with React

Step 1: Initialize the Frontend Project

Create a new React application using `create-react-app`:

```
npx create-react-app todo-app-frontend
cd todo-app-frontend
```

Step 2: Install Axios for HTTP Requests

```
npm install axios
```

Step 3: Create React Components

Modify `src/App.js` to include a simple todo list interface that interacts with the backend.

App Component (`src/App.js`):

```
import React, { useState, useEffect } from 'react';
import axios from 'axios';
```

```
import './App.css';

const App = () => {
  const [todos, setTodos] = useState([]);
  const [text, setText] = useState('');

  useEffect(() => {
    axios.get('http://localhost:5000/todos')
      .then(response => setTodos(response.data))
      .catch(error => console.error('Error:', error));
  }, []);

  const addTodo = (e) => {
    e.preventDefault();
    axios.post('http://localhost:5000/todos', { text })
      .then(response => setTodos([...todos, response.data]))
      .catch(error => console.error('Error:', error));
    setText('');
  };

  const toggleComplete = (id) => {
    const todo = todos.find(t => t._id === id);
    axios.put(`http://localhost:5000/todos/${id}`, { completed: !todo.completed })
      .then(response => setTodos(todos.map(todo =>
        todo._id === id ? response.data : todo
      )))
      .catch(error => console.error('Error:', error));
  };

  const deleteTodo = (id) => {
    axios.delete(`http://localhost:5000/todos/${id}`)
      .then(() => setTodos(todos.filter(todo => todo._id !== id)))
      .catch(error => console.error('Error:', error));
  };

  return (
    <div className="App">
      <h1>Todo List</h1>
      <form onSubmit={addTodo}>
        <input
          type="text"
          value={text}
          onChange={(e) => setText(e.target.value)}
          placeholder="Add a new todo"
        />
        <button type="submit">Add</button>
      </form>
      <div className="todo-list">
        {todos.map(todo => (
          <div key={todo._id} className="todo">
            <span
              style={{ textDecoration: todo.completed ? 'line-through' : '' }}
              onClick={() => toggleComplete(todo._id)}
            >
              {todo.text}
            </span>
            <button onClick={() => deleteTodo(todo._id)}>x</button>
          </div>
        ))}
      </div>
```

```
    </div>
  );
};

export default App;
```

App CSS (src/App.css):

```
.App {
  text-align: center;
}

.todo-list {
  margin: 0 auto;
  width: 300px;
}

.todo {
  display: flex;
  justify-content: space-between;
  background: #f4f4f4;
  margin: 5px 0;
  padding: 10px;
  border-radius: 5px;
}

.todo span {
  cursor: pointer;
}

input {
  padding: 10px;
  margin: 10px 0;
  width: calc(100% - 24px);
  border: 1px solid #ddd;
  border-radius: 5px;
}

button {
  padding: 10px;
  border: none;
  background: #007bff;
  color: #fff;
  cursor: pointer;
  border-radius: 5px;
}

button:hover {
  background: #0056b3;
}
```

Step 4: Run the Frontend Application

```
npm start
```