Full-Stack Todo List Application

Step-by-Step Project Documentation

1. Introduction

This documentation details the process I followed to develop the Full-Stack Todo List Application. The application allows users to create, view, update, and delete tasks. It features a React-based frontend, a Node.js/Express backend, and MongoDB for data persistence. I built the project entirely on my own, applying modern web development practices and containerization with Docker. This documentation serves as an overview of the design, implementation, and deployment steps I took during the project.

2. Project Overview

- **Objective:**
  Develop a responsive, full-stack web application for managing tasks.
- **Technologies Used:**
  - **Frontend:** React, Material-UI, Vite, React Toastify, Lucide Icons
  - **Backend:** Node.js, Express, Mongoose
  - **Database:** MongoDB
  - **Containerization:** Docker
  - **Development Tools:** Visual Studio Code, Git/GitHub
- **Outcome:**
  A fully functional Todo List application with a demo.

3. Step-by-Step Development Process

Step 1: Setting Up the Development Environment

- **Install Required Software:**

    - **Node.js:** Download and install Node.js (v18.9.1 was used for this project).
    - **MongoDB:** Install MongoDB locally for development or configure access to a remote MongoDB instance.
    - **Docker:** Install Docker Desktop for containerization.
    - **Visual Studio Code:** Set up VS Code with necessary extensions such as Docker and Remote - Containers.
- **Version Control:**
  Initialize a Git repository and push the initial project structure to GitHub.

Step 2: Establishing the Project Structure

- **Directory Layout:**
    - Create a `frontend/` directory for the React application.
    - Create a backend/ directory for the Node.js/Express server.
- **Documentation:**
  Add separate README files in both directories outlining specific setup instructions.

Step 3: Backend Development

- **Initialize the Backend:**
    - Run `npm init` in the backend/ directory.
    - Install necessary dependencies (Express, Mongoose, etc.) with `npm install express mongoose`.
- **Database Connection:**

- Create a connection module (e.g., `connectDb.js`) to establish a connection with MongoDB using Mongoose.
- Example snippet:

```
javascript
Copy
import mongoose from "mongoose";

const connectDb = () => {
  return mongoose
    .connect(`mongodb://mongo-shared-dev:fikTpih4U2!@20.218.241.192:27017/?directConnection=true&appName=mongosh+1.8.2&authMechanism=DEFAULT`)
    .then(() => console.log("connected"))
    .catch((err) => console.log("catch error", err));
};

export default connectDb;
```

- **API Endpoints:**
  - Design and implement RESTful endpoints for CRUD operations on todo items.
  - Organize routes and controllers for modularity and maintainability.

Step 4: Frontend Development

- **Initialize the Frontend:**
  - Use Vite to create a new React project in the `frontend/` directory.
  - Run `npm install` to install project dependencies.
- **User Interface Design:**
  - Build the UI components using React and Material-UI.
  - Integrate Lucide Icons for modern visual elements.
  - Use React Toastify to handle user notifications for actions like task creation and deletion.
- **Connecting to the Backend:**
  - Set up API service functions to call the backend endpoints and manage task data.
  - Ensure the UI updates in real time as tasks are created, updated, or deleted.

Step 5: Integrating Frontend and Backend

- **API Integration:**
    - Test endpoints using tools like Postman during development.
    - Ensure proper communication between the React frontend and Express backend.
- **Error Handling and Data Validation:**
    - Implement error handling both on the client and server sides to manage unexpected issues gracefully.

Step 6: Containerization with Docker

- **Dockerizing the Backend:**
    - Write a Dockerfile in the project root or within the backend/ directory:

```
dockerfile
Copy
FROM node:18.9.1
WORKDIR /app
COPY package.json .
RUN npm install
COPY . .
EXPOSE 5000
CMD ["npm", "start"]
```

- **Docker Compose (Optional):**
    - If using Docker Compose for multi-container setups (e.g., running both frontend and backend in containers), create a `docker-compose.yml` file to orchestrate services.
- **Testing Containers:**
    - Build the Docker image using `docker build -t fullstack-todo .`
    - Run the container locally with `docker run -p 5000:5000 fullstack-todo`

Step 7: Testing and Debugging

- **Local Testing:**
    - Test the application locally by running the backend (`npm start`) and the frontend (`npm run dev`).

    - Verify the functionality of task creation, updates, and deletion.
- **Debugging:**
    - Use VS Code's debugging tools to step through code and resolve issues.
    - Check logs and error messages for troubleshooting.

Step 8: Deployment and Submission

- **Deployment:**
    - Deploy the application to a hosting platform (e.g., Render) to make it publicly accessible.
    - Ensure that the live demo URL is functional and reflects the latest changes.
- **Final Submission:**
    - Finalize all documentation, update READMEs, and commit the final version of the project to GitHub.
    - Submit the assignment as required.

4. Conclusion

In this assignment, I developed a comprehensive Full-Stack Todo List Application by integrating modern technologies and best practices in web development. I handled all aspects of the project—from setting up the development environment, establishing the project structure, and coding the backend and frontend, to containerizing the application with Docker and deploying it for public access. This documentation outlines every step of the process, demonstrating my ability to manage and deliver a complete, production-ready application. I take full responsibility for the project’s development and submission, which reflects my skills and dedication to full-stack development.