COLLEGE CODE : 9509

COLLEGE NAME:Holycross Engineering College

DEPARTMENT :CSE

STUDENT NM-ID: DAE074EB4A8886B8B224E01DC0C33A55

ROLL NUMBER:950923104044

DATE :06-10-2025

Completed the project named as:TO DO LIST APPLICATION.

SUBMITTED BY:K.SANJAY.

MOBILE NO : 9342667280

1. Final Demo Walkthrough

Application Overview:
Our To-Do List Application is a full-stack web solution designed for efficient task management. It provides a clean, intuitive interface for users to create, read, update, and delete their tasks. The application features a responsive design, ensuring a seamless experience across both desktop and mobile devices.

Key Features Demonstrated:

- **Adding a New Task:** Enter a task description in the input field and click "Add" or press Enter. The task is instantly displayed in the pending list.
- **Viewing Tasks:** All tasks are displayed in a clear list. Pending tasks are shown with an empty checkbox, while completed tasks are shown with a checked checkbox and a strikethrough.
- **Marking Tasks as Complete/Incomplete:** Click the checkbox next to a task to toggle its completion status. The task will visually move between the active and completed states.
- **Editing Tasks:** Click the "Edit" button next to a task. The task text will change into an editable input field, allowing for quick modifications. Save the changes to update the task.
- **Deleting Tasks:** Click the "Delete" button to permanently remove a task from the list. A confirmation dialog may be implemented for safety.
- **Persistence:** Refresh the browser page. All your tasks (both pending and completed) will remain, as they are stored in a database.

---

2. Project Report

1. Introduction

This document outlines the development process of the To-Do List Application. The project aimed to build a robust, user-friendly task management tool using modern web technologies to demonstrate proficiency in full-stack development.

2. Objectives

- To design and implement a functional, responsive To-Do List web application.
- To provide full CRUD (Create, Read, Update, Delete) functionality for tasks.
- To ensure data persistence using a backend database.
- To create a clean and intuitive user interface.

3. Technologies Used

- **Frontend:** HTML5, CSS3, JavaScript (React.js / Vanilla JS)
- **Backend:** Node.js with Express.js
- **Database:** MongoDB / PostgreSQL
- **Other Tools:** Git for version control, Postman for API testing, Vercel/Netlify for frontend deployment, Render/Heroku for backend deployment.

4. System Architecture

The application follows a client-server architecture:

1. The **Client** (Frontend) runs in the user's browser, handling the user interface and user interactions.
2. The **Server** (Backend), built with Node.js and Express, listens for HTTP requests from the client.
3. The **Database** stores all task data persistently. The server interacts with the database to perform all CRUD operations.

5. Implementation Details

- The frontend was built with a component-based architecture for reusability and maintainability.
- RESTful API endpoints were designed on the backend to handle all task-related operations.
- Asynchronous JavaScript (async/await) was used for all database operations and API calls.
- Responsive CSS design techniques (Flexbox/Grid) were employed to ensure compatibility with various screen sizes.

6. Conclusion

The project was successfully completed, meeting all initial objectives. The application is fully functional, performs all core task management operations, and is deployed on a live server. This project served as an excellent exercise in full-stack development, API design, and deployment workflows.

3. Screenshots / API Documentation

Screenshots

(Imagine screenshots inserted here in your Word document)

- **Figure 1: Main Application View** - Shows the main interface with an input field, a list of pending tasks, and a list of completed tasks.
- **Figure 2: Adding a New Task** - Demonstrates the process of typing a new task and seeing it appear in the list.
- **Figure 3: Task Completion** - Shows a task with a strikethrough after its checkbox has been checked.
- **Figure 4: Editing a Task** - Displays the inline edit functionality where a task is in an editable input field.

API Documentation

Base URL: `https://your-backend-api.herokuapp.com/api`

Endpoint	Method	Description	Request Body	Success Response
`/tasks`	`GET`	Get all tasks.	-	`200 OK` + Array of tasks
`/tasks`	`POST`	Create a new task.	`{"title": "New Task"}`	`201 Created` + New task
`/tasks/:id`	`PUT`/`PATCH`	Update a specific task.	`{"title": "Updated Task", "completed": true}`	`200 OK` + Updated task
`/tasks/:id`	`DELETE`	Delete a specific task.	-	`200 OK` or `204 No Content`

---

4. Challenges & Solutions

Challenge	Solution
Connecting Frontend to Backend: Initial CORS errors prevented the frontend from communicating with the backend API.	Implemented the CORS middleware in the Express.js server to explicitly allow requests from the frontend's origin.
Data Persistence: Tasks were disappearing on page refresh when using only frontend state.	Integrated a backend database (MongoDB) to store tasks permanently. The frontend now fetches tasks from the API on load.
State Management: Managing the state of tasks (adding, updating, deleting) across different components became complex.	Utilized React's Context API / useState hooks to create a centralized state management system, ensuring the UI is always in sync with the data.
Deployment Configuration: Environment variables and database connection strings failed in the production deployment environment.	Used environment configuration files (`.env`) for local development and set the environment variables directly in the deployment platform's (e.g., Heroku, Render) dashboard.

---

5. GitHub README & Setup Guide

(This section can be copied directly into your GitHub README.md file)

To-Do List Application

A full-stack, responsive To-Do List application built with the MERN stack (MongoDB, Express.js, React, Node.js).

Features

- **Add Tasks:** Quickly add new tasks to your list.
- **Mark Complete/Incomplete:** Toggle task completion status.
- **Edit Tasks:** Update your existing tasks inline.
- **Delete Tasks:** Remove tasks you no longer need.
- **Persistent Storage:** Your tasks are saved and will persist after closing the browser.

Live Demo

Check out the live application: [Frontend Deployed Link (e.g., Vercel)]

Check out the live API: [Backend Deployed Link (e.g., Render)]

Tech Stack

- **Frontend:** React, CSS3
- **Backend:** Node.js, Express.js
- **Database:** MongoDB

Local Setup & Installation

Follow these steps to run the project locally.

Prerequisites

- Node.js and npm installed on your machine.
- A MongoDB database (local or cloud cluster like MongoDB Atlas).

Installation

1. **Clone the repository:**

   bash

   ```bash
   git clone https://github.com/your-username/your-todo-app-repo.git
   cd your-todo-app-repo
   ```

2. **Backend Setup:**

   ```bash
   cd backend
   npm install
   ```

   - Create a `.env` file in the `backend` directory.
   - Add your MongoDB connection string:

   ```text
   MONGODB_URI=your_mongodb_connection_string_here
   PORT=5000
   ```

   - Start the backend server:

   ```bash
   npm run dev
   ```

   The backend API will run on `http://localhost:5000`.

3. **Frontend Setup:**

   ```bash
   cd ../frontend
   npm install
   ```

   - Create a `.env` file in the `frontend` directory.
   - Add the backend API URL:

   ```text
   REACT_APP_API_BASE_URL=http://localhost:5000/api
   ```

   - Start the frontend development server:

   ```bash
   npm start
   ```

   The frontend application will open in your browser at `http://localhost:3000`.

API Endpoints

Refer to the API Documentation section above for details.

6. Final Submission

- **GitHub Repository Link:** https://github.com/sanjaykrvilai01-crypto/todo_list.git
- **Deployed Application Link:** https://sanjaykrvilai01-crypto.github.io/todo_list/