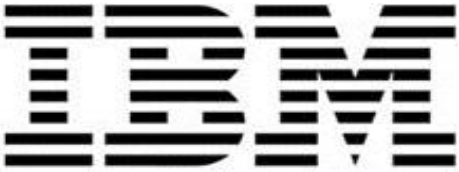

COLLEGE CODE : 8203

COLLEGE NAME : A.V.C College of Engineering

DEPARTMENT : B.tech – Information Technology

STUDENTNM-ID : C9C912D4BD1EB53B5A04BF0E7A44DD4E

ROLL NO : 23IT84

DATE : 22-09-2025

Completed the project named as Phase 3

TECHNOLOGY PROJECT NAME : To-Do App with React Hooks

SUBMITTED BY,

Name : Ragulkumar A

MOBILE NO : 9360457452

Project Setup

Initialize React app with create-react-app or Vite.

Setup Node.js + Express backend project structure.

Connect MongoDB (local or Atlas).

Install dependencies: React Hooks, Axios/Fetch, Express, Mongoose, CORS, Nodemon.

Setup .env for environment variables (DB URI, PORT).

Core Features Implementation

The core functionality of the application includes full CRUD (Create, Read, Update, Delete) operations for tasks:

- Create: Users can add new tasks through an input form.
- Read: All existing tasks are fetched from the database and displayed on page load.
- Update: Users can toggle a task's completion status or edit the task's text content.
- Delete: Users can permanently remove tasks from the list.
- **Add Task:** User can add new task with title.
- **View Tasks:** Fetch tasks from backend and display in UI

- **Edit Task:** Update existing task details.
- **Delete Task:** Remove task permanently from DB.
- **Mark Complete/Incomplete:** Toggle status with checkbox.
- **Real-time UI Update:** Reflect changes immediately after action.

Data Storage

- Client-Side (Local State): The React front-end uses useState and useEffect hooks to manage the application's state. This provides real-time feedback to the user as tasks are added, updated, or deleted, creating a smooth user experience.
- Server-Side (Database): MongoDB serves as the persistent database. A Task model was defined with the following schema:
    - text (Type: String, Required: true) ◦ completed (Type: Boolean, Default: false)
    - createdAt (Type: Date, Default: Date.now)

Testing Core Features

- API Testing: The backend REST API endpoints were tested using tools like Postman to ensure they correctly handled GET, POST, PUT, and DELETE requests and performed the intended database operations.

- Front-End Testing: Manual testing was conducted on the user interface to verify that all interactive elements functioned as expected, state was managed correctly, and API calls were triggered appropriately.

Unit Testing:

- Test React components (TaskInput, TaskList).
- Verify API routes with Postman.

Integration Testing:

- End-to-end task flow (Add → Update → Delete → Fetch).

Error Handling:

- Empty input validation.
- Invalid ID or missing task handling in backend.

Version Control

Github : https://github.com/ragulkumar839/NM-IBM-AVCCE-Ragulkumar.git