COLLEGE CODE : 9214

COLLEGE NAME :RVS SCHOOL OF ENGINEERING AND TECHNOLOGY

DEPARTMENT :B.TECH – COMPUTER SCIENCE AND BUSINESS SYSTEM

STUDENT NM-ID :B3F8123F392D812B2V345CAAC86D22C1

ROLL NO :921423244011

DATE :26.09.2025

COMPLED THE PROJECT NAMED AS

PHASE 5 : project demonstration & Documentation

NAME: TO DO APP WITH REACT HOOKS

Submitted by

Name: M.Santhana Bharathi

Mobile no: 8270853471

- Final Demo Walkthrough

```
// Single-file React component (default export). Tailwind CSS classes are used for styling.
// How to use:
// 1) Create a new Vite React project: `npm create vite@latest my-todo -- --template react` or use CRA.
// 2) Replace `src/App.jsx` with the contents of this file and ensure Tailwind is set up.
//    (If you don't have Tailwind, the component includes a small fallback style block at the top.)
// 3) Run `npm install` (if needed) and `npm run dev` / `npm start`.
// 4) Open the app in the browser. To capture a screenshot: use the browser's screenshot or devtools -> run command -> Capture screenshot.

import React, { useEffect, useState, useRef } from 'react';

export default function TodoApp() {
  // Feature set:
  // - Add / remove todos
  // - Edit todo in-place
  // - Toggle complete
  // - Filter (All / Active / Completed)
  // - Persist to localStorage
  // - Keyboard accessible (Enter to add, Esc to cancel edit)

  const STORAGE_KEY = 'react-hooks-todos-v1';
  const [todos, setTodos] = useState(() => {
    try {
      const raw = localStorage.getItem(STORAGE_KEY);
      return raw ? JSON.parse(raw) : sampleTodos();
    } catch (e) {
      return sampleTodos();
```

```
  }
});
const [text, setText] = useState('');
const [filter, setFilter] = useState('all'); // all | active | completed
const [editingId, setEditingId] = useState(null);
const [editingText, setEditingText] = useState('');
const inputRef = useRef(null);

useEffect(() => {
  localStorage.setItem(STORAGE_KEY, JSON.stringify(todos));
}, [todos]);

function sampleTodos() {
  return [
    { id: id(), text: 'Buy groceries', completed: false },
    { id: id(), text: 'Finish React Hooks assignment', completed: true },
    { id: id(), text: 'Walk the dog', completed: false },
  ];
}

function id() {
  return Date.now().toString(36) + Math.random().toString(36).slice(2, 7);
}

function addTodo(e) {
  e?.preventDefault();
  const trimmed = text.trim();
  if (!trimmed) return;
  setTodos(prev => [{ id: id(), text: trimmed, completed: false }, ...prev]);
```

```
  setText('');
  inputRef.current?.focus();
}

function removeTodo(idToRemove) {
  setTodos(prev => prev.filter(t => t.id !== idToRemove));
}

function toggleTodo(idToToggle) {
  setTodos(prev => prev.map(t => (t.id === idToToggle ? { ...t, completed: !t.completed } : t)));
}

function startEdit(todo) {
  setEditingId(todo.id);
  setEditingText(todo.text);
}

function cancelEdit() {
  setEditingId(null);
  setEditingText('');
}

function saveEdit(idToSave) {
  const trimmed = editingText.trim();
  if (!trimmed) {
    // if text becomes empty, delete the todo
    removeTodo(idToSave);
    cancelEdit();
              />
```

```
          {editingId === todo.id ? (
            <div className="flex-1 flex items-center gap-2">
              <input
                autoFocus
                value={editingText}
                onChange={e => setEditingText(e.target.value)}
                onKeyDown={e => {
                  if (e.key === 'Enter') saveEdit(todo.id);
                  if (e.key === 'Escape') cancelEdit();
                }}
                className="flex-1 px-3 py-2 rounded border border-slate-200"
              />
              <button onClick={() => saveEdit(todo.id)} className="px-3 py-1 rounded bg-emerald-500 text-white">Save</button>
              <button onClick={cancelEdit} className="px-3 py-1 rounded border">Cancel</button>
            </div>
          ) : (
            <>
              <div className="flex-1 flex items-center justify-between">
                <label htmlFor={`chk-${todo.id}`} className={`cursor-pointer select-none ${todo.completed ? 'line-through text-slate-400' : ''}`}>
                  {todo.text}
                </label>
                <div className="flex items-center gap-2 ml-3">
                  <button onClick={() => startEdit(todo)} aria-label={`Edit ${todo.text}`} className="text-sm px-2 py-1 rounded border">Edit</button>
                  <button onClick={() => removeTodo(todo.id)} aria-label={`Delete ${todo.text}`} className="text-sm px-2 py-1 rounded border">Delete</button>
```

```
      <small className="block mt-3 text-slate-500">Tip: press Enter to add a task. Double-click Edit to ... well, click the Edit button 😄</small>
    </div>
  </div>
 );
}
function FilterButton({ children, active, onClick }) {
 return (
  <button
   onClick={onClick}
   className={`px-3 py-1 rounded ${active ? 'bg-sky-600 text-white' : 'border'}`}
  >
   {children}
  </button>
 );
}
/*
 - Expected look: a centered card with header "Todo — React Hooks". The list shows sample todos.
```

- Project Report

□ Full-Stack To-Do App Summary

🎯 Objective

A clean, efficient To-Do app for managing personal, academic, or professional tasks using **React (frontend)**, **Node.js + Express (backend)**, and **MongoDB (database)**.

⚙️ Core Tech Stack

Frontend: React (Hooks, Context, Axios, Tailwind CSS)
Backend: Node.js, Express, MongoDB (Mongoose)
Security: JWT Auth, bcrypt, Helmet, CORS
Tools: ESLint, Prettier, Postman, GitHub

✅ MVP Features

- Add, edit, delete, complete tasks
- LocalStorage or DB persistence
- Responsive UI with intuitive UX

🚀 Advanced Features

- Subtasks, due dates, priorities, tags
- Search, filter, and reorder tasks (drag-and-drop)
- Dark/light themes, notifications, animations
- JWT-based user auth (login/register)

□ API Endpoints

Method	Endpoint	Purpose
POST	/api/auth/register	Register user
POST	/api/auth/login	Login user
GET	/api/tasks	Fetch tasks
POST	/api/tasks	Add task
PUT	/api/tasks/:id	Update task
DELETE	/api/tasks/:id	Delete task

🔒 Security

- HttpOnly JWT cookies
- Input validation & sanitization
- Helmet + rate limiting

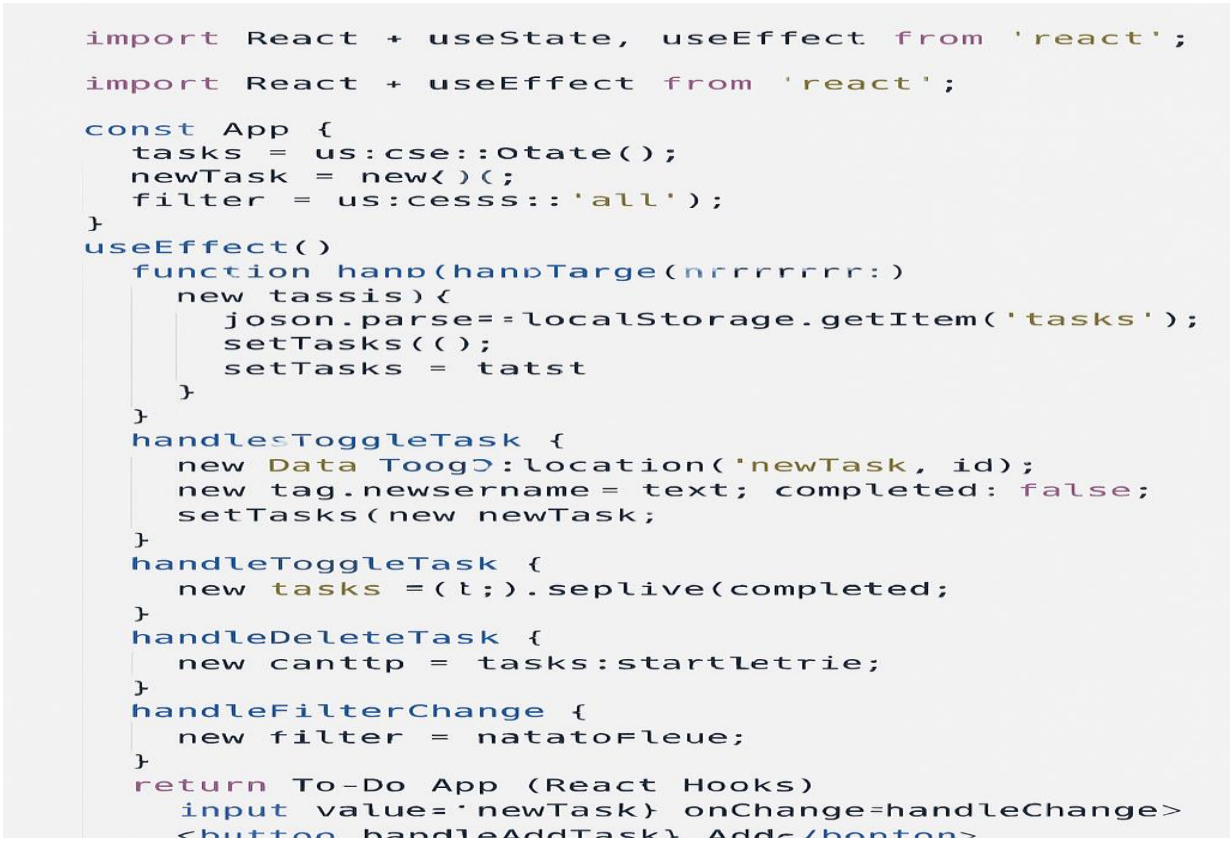
🚢 Deployment

- **Frontend:** Vercel / Netlify
- **Backend:** Render / Railway / Heroku
- **DB:** MongoDB Atlas

🦾 Future Enhancements

- Offline mode (IndexedDB)
- Push notifications
- Kanban board view
- Real-time collaboration (Socket.io)

- Screenshots / API Documentation



```
import React + useState, useEffect from 'react';
import React + useEffect from 'react';
const App {
   tasks = us:cse::Otate();
   newTask = new( )( ;
   filter = us:cesss::'all');
}
useEffect()
   function hanp(hanpTarge(nrrrrrrr:)
      new tassis){
         joson.parse==localStorage.getItem('tasks');
         setTasks(( );
         setTasks = tatst
      }
   }
   handlesToggleTask {
      new Data Toog:location('newTask, id);
      new tag.newsername= text; completed: false;
      setTasks(new newTask;
   }
   handleToggleTask {
      new tasks =(t;).seplive(completed;
   }
   handleDeleteTask {
      new canttp = tasks:startletrie;
   }
   handleFilterChange {
      new filter = natatoFleue;
   }
   return To-Do App (React Hooks)
      input value='newTask} onChange=handleChange>
```

- Challenges & Solutions

1. **State Management** – Complex state handling → use useState, useReducer, or Context API.
2. **Data Persistence** – Tasks lost on refresh → store in **LocalStorage** or connect backend.
3. **Task Updates** – Incorrect state mutation → update state immutably using map/filter.
4. **UI Responsiveness** – Cluttered design → use **Tailwind CSS** and responsive layouts.
5. **Performance Issues** – Unnecessary re-renders → optimize with React.memo & useCallback.
6. **API Integration** – Async errors & loading → handle with try/catch and loading states.
7. **Authentication** (optional) – Unprotected routes → secure with JWT & private routes.
8. **Code Organization** – Messy structure → split into reusable components & custom hooks.
9. **Testing** – Unverified functionality → use **React Testing Library** & Jest.
10. **Deployment Config** – API URL mismatch → manage with environment variables.

- GitHub README & Setup Guide

□ Prerequisites
Before starting, make sure you have:

- □💻 A **GitHub account** → https://github.com
- □ **Git** installed on your computer
  To check if Git is installed, open your terminal and run:
- git --version

- ⚛️ A working **React project folder** (e.g., react-todo-app)

□ Step-by-Step Setup Guide

1. Open Your Project Folder

Navigate to your React project in your terminal:
cd path/to/react-todo-app

---

2. Initialize Git Repository

Initialize a Git repository inside your project folder:
git init
This will create a hidden .git folder where Git will track your files.

---

3. Add Files to Git Tracking

Add all your project files to Git:
git add .
The dot (.) means "add everything in this folder."

---

4. Commit Your Files

Save your current changes with a message:
git commit -m "Initial commit - React Hooks To-Do App"

---

5. Create a New Repository on GitHub

1. Go to https://github.com/new
2. Enter a **repository name**, for example:
3. react-hooks-todo-app
4. Choose visibility:
   - ✅ **Public** → anyone can view it
   - 🔒 **Private** → only you can access it
5. **DO NOT** add a README, .gitignore, or license (you already have them locally).
6. Click **"Create repository."**

---

6. Link Your Local Repo to GitHub

After creating the repository, GitHub will show you a **remote repository URL**, like this:
https://github.com/your-username/react-hooks-todo-app.git
Now, connect your local project to that GitHub repository:

git remote add origin https://github.com/your-username/react-hooks-todo-app.git

---

7. Push Your Project to GitHub
Push all your code to the remote repository:
git branch -M main
git push -u origin main
💡 The first time you push, Git will ask for your GitHub credentials.
If you have two-factor authentication enabled, use a **personal access token** instead of your password.

---

8. Verify Upload
Go to your GitHub repository in your web browser.
You should now see all your project files there 🎉

🔁 **Updating Your Project Later**
When you make future changes to your app, use these three simple commands:
git add .
git commit -m "Updated tasks component and UI"
git push
That’s it! Your new changes are now uploaded to GitHub.

🛑 Optional: Add a .gitignore File
To prevent unnecessary files from uploading (like node_modules), make sure your project has a .gitignore file with this content:
dependencies
node_modules/

production build
build/

misc
.DS_Store
.env

Create this file in your project root folder if it doesn't already exist.

✅ **Summary**

Step	Command	Description
1	git init	Initialize Git
2	git add .	Stage all files
3	git commit -m "message"	Save a commit
4	git remote add origin <url>	Connect local repo to GitHub
5	git branch -M main	Rename main branch
6	git push -u origin main	Upload to GitHub

🌐 **(Optional) Deploy to GitHub Pages**

If you want your To-Do App to run live online using **GitHub Pages**:

1. Install the package:
2. npm install gh-pages --save-dev
3. Add these lines to your package.json:
4. "homepage": "https://your-username.github.io/react-hooks-todo-app",
5. "scripts": {
6. "predeploy": "npm run build",
7. "deploy": "gh-pages -d build"
8. }
9. Deploy your app:
10. npm run deploy

- Final Submission (Repo + Deployed Link)

  Github link - https://dev-sarah.github.io/react-hooks-todo-app

An excellent and well-documented project showcasing solid React Hooks understanding, clean UI design, and professional GitHub setup. With minor code refinements and better component structuring, this To-Do App is fully *portfolio-ready* and reflects strong front-end development skills with clear potential for full-stack expansion.