COLLEGE CODE : 2603

COLLEGE NAME : GOVERNMENT COLLEGE ENGINEERING, BARGUR.

DEPARTMENT : COMPUTER SCIENCE AND ENGINEERING.

STUDENT NM ID : 49e50cdea1910dbaea3486fd2603547f

ROLL NUMBER : 2303610710422302

DATE : 17.09.2025.

COMPLETED THE PROJECT NAMED AS PHASE 2.

PROJECT NAME : TODO APP USING REACT HOOKS

SUBMITTED BY,

NAME: MADHUMITHA P

MOBILE NUMBER: 8778010311

TODO APP USING REACT HOOKS

1. TECH STACK SELECTION

Choosing the right set of tools and technologies ensures smooth development, scalability, and maintainability.

Frontend:

- React.js (with Hooks) – to build reusable components (task list, add form, filters).
- HTML5, CSS3, JavaScript – for UI styling and interactivity.

State Management:

- React Hooks (useState, useEffect, useReducer) – for managing todos and UI state.

Backend (Optional):

- Node.js with Express.js – if persistence is required.
- Alternatively, localStorage can be used for storing todos in the browser.

Other Tools:

- Git/GitHub – version control.
- Visual Studio Code – IDE for development.
- npm/yarn – package management.

2. UI STRUCTURE & API SCHEMA DESIGN

UI Structure:

The system will contain the following main pages/components:

1. **Dashboard / Home Page**
    - Displays all tasks in a list or card format.
    - Shows key details: Task Title, Description, Status, and Deadline.
    - Options to filter/search tasks.

2. **Add New Task Page / Component**
    - A form where users can add task details (title, description, deadline).
    - A dropdown/radio for status (Pending, In Progress, Completed).

3. **Task Details / Edit Component**
    - Displays the full details of a task.
    - Allows editing status or updating details.

4. **Search & Filter Component**
    - Enables searching by keywords.
    - Filters tasks by status or due date.

API Schema Design (if backend used):

- **POST /todos** → Create a new task.
- **GET /todos** → Retrieve all tasks.
- **GET /todos/:id** → Retrieve a single task.
- **PUT /todos/:id** → Update a task's details or status.
- **DELETE /todos/:id** → Delete a task.

Each record will contain:

- **id** (unique identifier)
- **title** (string)
- **description** (string, optional)
- **deadline** (date, optional)
- **status** (enum: Pending, In Progress, Completed)

3. DATA HANDLING APPROACH

1. All tasks will be stored in a backend database (MongoDB/MySQL) or browser localStorage.
2. Frontend form inputs will be validated before saving.
3. CRUD operations will manage the lifecycle of tasks:
    - **Create**: Add new tasks.
    - **Read**: View tasks in a list or detail view.
    - **Update**: Modify task details or mark as completed.
    - **Delete**: Remove finished/unnecessary tasks.

Security Practices (if backend used):

- Input validation to prevent injection attacks.
- Error handling for missing/invalid fields.

4. COMPONENT / MODULE DIAGRAM

Frontend Components:

- **Form Component**: For adding and editing tasks.
- **List Component**: Displays all tasks in an organized manner.
- **Filter/Search Component**: For searching and filtering tasks.
- **Detail View Component**: Shows full details of a task.

Backend Modules (optional):

- **API Routes**: Define endpoints for tasks.
- **Controllers**: Contain CRUD logic.
- **Database Models**: Define task schema (fields like title, deadline, status).
- **Middleware**: For validation and error handling.

5.BASIC FLOW DIAGRAM (TEXTUAL DESCRIPTION)

1. The user opens the app and sees the Dashboard with the task list.
2. If they want to add a task, they fill in the form → React updates state with the new task (and optionally sends POST request to backend).
3. When the user views all tasks, React fetches from local state/backend → results displayed in a list.
4. If the user updates a task (e.g., mark as "Completed"), the state is updated (or PUT request sent to backend) → UI refreshes.
5. If the user deletes a task, React removes it from state (or DELETE request sent to backend) → dashboard updates.