Full Stack Project: Front End GitHub Information
GitHub Username: https://github.com/voidronin-glitch
Repository Name: ITWEB400 assignment 1

The Navbar component serves as the header of the web application. It displays the app's title — "Task Manager" — and could later include navigation links to different sections, such as "All Tasks," "Completed Tasks," or "Add Task." It enhances usability by keeping the app organized and easy to navigate. TaskForm Component The TaskForm component provides an input field and button for users to create new tasks.

It uses React's useState hook to manage the form's input data. When the user submits the form, the component currently logs the input to the console, but a comment in the code indicates where the API request will go once the backend is ready. This component represents how users send data to the system. TaskList Component The TaskList component displays a list of existing tasks.

For now, it uses a placeholder array of dummy data. In the future, it will fetch task data from the backend using an API call and render each task dynamically. This component demonstrates how data from the backend will be displayed to users. App Component The App component acts as the main container that integrates all other components — Navbar, TaskForm, and TaskList.

It defines the structure of the application's UI, maintaining a simple and clean layout where users can view and add tasks. Later, App will manage the flow of data between components and the backend. Summary of React Usage and Front-End Interaction React was used to create the dynamic and component-based front end for this full stack system.

React enables the UI to be modular by separating functionality into reusable components such as TaskForm, TaskList, and Navbar. Each component handles a specific part of the interface, which makes the code easier to read, test, and maintain. The React app uses useState hooks to manage and update the UI in real time as users input or modify data.

For example, when a user adds a task, the form captures it in local state and would later send it to the backend using a fetch() or axios API call. Comments in the code mark where these backend communication points will be placed once the server is implemented. React interacts with the backend by sending HTTP requests (using fetch or axios) to endpoints managed by Node.js and Express.

For example: POST requests to create a new task, GET requests to retrieve existing tasks, PUT requests to update tasks, and DELETE requests to remove tasks. React will handle the responses from these requests by updating its component states, ensuring that the front end and back end remain synchronized in real time. The UI design uses HTML for structure, CSS for layout and styling, and React (JavaScript) for interactive and dynamic functionality.

The structure was built with a focus on clarity and usability — clean layouts, clear input fields, and logical task listings. Project Experience and Challenges Working on the front end of this project was a valuable experience in learning how to structure a React application and design a clean, functional user interface. The use of React helped create reusable and modular components, which made the code easier to manage. Building the initial design using HTML and CSS provided a strong visual foundation before integrating React's dynamic features.

One challenge faced during development was managing React state effectively and ensuring that each component maintained its data independently without unnecessary re-renders. Another challenge was structuring the components so that they can later communicate easily with the backend. To prepare for this, comments were included in the code where API calls will be implemented.

Another area of focus was the visual design. Ensuring proper spacing, readability, and responsive layout required experimentation with CSS properties like flexbox and box-shadow for modern, minimal aesthetics. These small design improvements helped make the app look professional and user-friendly. Going forward, the next steps will include connecting the front end to a Node.js and Express backend, enabling real task creation, updates, and deletions.

Once the backend is integrated, this React front end will be able to dynamically display live data and provide a fully functional user experience.