**THOUGHT NOTE**: To-Do List

I delete node-modules file from both (frontend & backend) folder

* FRONTEND PART:- (App.js)

1. **State Management:**

* The component uses the **useState** hook to manage the state of the **task** (input for new tasks) and **newTask** (array of tasks).

2. **HTTP Requests:**

* Axios is used to make asynchronous HTTP requests. The **fetchTodos** function retrieves existing tasks from the server, and the **addTodo** function adds a new task to the server and updates the state with the new data.

3. **Effect Hook:**

* The **useEffect** hook is employed to fetch the initial set of tasks when the component mounts (**[]** as the dependency array ensures it runs only once).

4. **Delete Functionality:**

* The **deleteTodo** function sends a DELETE request to the server to remove a task with a specific ID. It then updates the state to reflect the deletion.

5. **UI Structure:**

* The UI is divided into two sections: the left container for adding new tasks and the right container for displaying existing tasks.
* Bootstrap styling is used for layout and some basic styling.

6. **Input Handling:**

* The input for adding new tasks is controlled by the state (**task**), and the **addTodo** function is triggered by a button click.

7. **Rendering Tasks:**

* Existing tasks are rendered in the right container, and each task has a corresponding delete button (**X**) to remove the task.

8. **Styling:**

* Some basic styling is applied to the components using Bootstrap classes and custom CSS.
* BACKEND PART:- (index.js)

1. **Dependencies:**

* **express**: Web framework for Node.js.
* **cors**: Middleware for handling Cross-Origin Resource Sharing.
* **mysql**: MySQL driver for Node.js.

2. **Express App Setup:**

* An Express app is initialized.
* Middleware is added to parse JSON and enable CORS.

3. **Database Connection:**

* A MySQL database connection is established using the **mysql** module.
* Connection details include the host, user, password, and database name.

4. **POST Endpoint (/addTodo):**

* Accepts JSON data with a task property in the request body.
* Inserts a new task into the "todo" table in the database.
* Responds with the inserted data or an error message.

5. **GET Endpoint (/getTodos):**

* Retrieves all tasks from the "todo" table in the database.
* Responds with the retrieved data or an error message.

6. **DELETE Endpoint (/deleteTodo/:id):**

* Accepts a task ID as a parameter in the URL.
* Deletes the corresponding task from the "todo" table in the database.
* Responds with the deleted data or an error message.

7. **Port Configuration:**

* The server is configured to listen on port 5000, and a log statement is provided when the server starts.

**Creating database: 1.Database Name-** Massage

**2.Table Name-** todo

**3.Column Name (2)-** ID, Task