**Answers to Development Questions**

**1. How did you handle inline editing in the table?**

Inline editing was implemented by opening a modal dialog when the edit button was clicked for a specific row. The modal allowed users to update the title and status of the task. The edited data was managed in a temporary state (editingTask) and updated in the main task data array (data) upon saving. The saveEditing function ensured that the changes were applied and re-filtered the displayed tasks based on the current filter.

**2. Explain how you fetched and processed the data from the dummy API.**

Data was fetched from the dummy API using the fetch function inside a useEffect hook. The API (https://jsonplaceholder.typicode.com/todos) provided an array of tasks. Upon receiving the data, it was stored in two states: data (containing all fetched tasks) and filteredData (containing tasks to be displayed based on the current filter). This separation allowed us to efficiently manage filtering without modifying the original dataset.

**3. What approach did you use to filter tasks based on status?**

A statusFilter state variable was used to track the selected filter option ("To Do", "Done", or "All"). When the filter value changed, the handleStatusFilterChange function was triggered to update the filteredData state by applying the appropriate filter condition:

* "To Do": Display tasks where completed is false.
* "Done": Display tasks where completed is true.
* "All": Display all tasks. This ensured that the table dynamically updated to reflect the selected status filter.

**4. How did you manage the state of tasks when adding or editing them?**

State management for tasks involved the following:

* **Adding Tasks**: When a new task was added, it was temporarily stored in the newTask state. Upon saving, the new task was prepended to the data array using the spread operator, ensuring it appeared at the top of the table. The filteredData state was updated based on the current filter to reflect the addition.
* **Editing Tasks**: The task being edited was stored in the editingTask state. Changes were applied to this state during editing, and upon saving, the data array was updated by mapping through the tasks and replacing the edited task. In both cases, state updates triggered re-renders, and appropriate filtering logic ensured consistency in the displayed tasks.

**5. What challenges did you face during development, and how did you overcome them?**

* **Challenge 1: Ensuring data consistency after updates**
  + *Problem*: Maintaining synchronization between data and filteredData was tricky, especially when tasks were added, edited, or deleted.
  + *Solution*: Always updated both data and filteredData in functions handling these operations, applying the current filter logic to filteredData.
* **Challenge 2: Displaying newly added tasks at the top**
  + Problem: By default, new tasks were appended to the array, leading to them being displayed at the bottom of the table.
  + Solution: Used the spread operator to prepend new tasks to the data array ([newTask, ...data]) when adding a task.
* **Challenge 3: Providing user feedback for operations**
  + Problem: The lack of visual feedback made it difficult for users to know if their actions (e.g., adding, editing, or deleting tasks) were successful.
  + Solution: Integrated the react-toastify library to display toast notifications for success messages, enhancing the user experience.
* **Challenge 4: Handling task completion input in the "Add Task" modal**
  + Problem: Managing true/false values for the completed field as user input was challenging, as users might enter invalid values.
  + Solution: Replaced the text input with a dropdown (Select component), allowing users to explicitly choose between "To Do" and "Done" options.

These solutions ensured that the application was robust, user-friendly, and responsive to user actions.