Barebones React To-Do App — Exact Scope

1) Goal

A minimal client-side To-Do app demonstrating React fundamentals

2) Feature Requirements (must-have)

1. **Create a task**
    - Input field or "Add" button (or Enter key).
    - Empty input should not create a task.
2. **List tasks**
    - Newly added tasks appear at the **top** of the list.
3. **Toggle completion**
    - Checkbox to mark a task complete/incomplete.
    - Completed tasks show a visible style change (e.g., strike-through).
4. **Delete a task**
    - Each task has a delete control (e.g., "×" button).
5. **Basic styling**
    - Clean, minimal layout; readable on desktop and mobile (no need for pixel-perfect design).

Note:

Persistence is not required, the page refresh may clear tasks (no localStorage/API).

3) Non-Functional Requirements

- **Framework:** React (Vite or CRA or Next.js app router—candidate's choice).
- **Language:** JavaScript or TypeScript (either is fine).
- **State mgmt:** React local state (useState, optional useReducer). **No Redux/3rd-party state libs.**
- **Dependencies:** Keep to a minimum; no heavy UI kits required.
- **Accessibility (a11y):**
    - Label the input; associate label and input.
    - Buttons are actual <button> elements with accessible names (e.g., aria-label="Delete task").

    - Keyboard: Enter adds a task; tab order is logical.
- **Performance:** Reasonable for small lists (no virtualization needed).
- **Browser support:** Latest Chrome/Edge/Firefox/Safari.

4) Out of Scope (not required but can do if desired)

- Authentication
- Persistence (localStorage, DB, or API)
- Filters (All/Active/Completed)
- Edit task text
- Sorting controls
- Tests (optional bonus only)

5) UI Spec (lightweight)

- **Header:** “To-Do”
- **Add Row:** Single-line text input (placeholder: “Add a task…”) + Add button.
    - Pressing **Enter** in the input is equivalent to clicking Add.
- **List Area:** Each item = [ ] checkbox + task text + delete icon/button at right.
    - Completed task style: strike-through + reduced opacity (e.g., 0.6).
- **Empty State:** If no tasks, show a subtle message: “No tasks yet.”

Example layout structure

```
<App>
 <Header />
 <TaskInput />
 <TaskList>
  <TaskItem />
 </TaskList>
</App>
```

6) Data Model

In-memory array of task objects:

```
type Task = {
  id: string;      // unique id (e.g., Date.now() or uuid)
  text: string;     // trimmed, non-empty
  completed: boolean;
  createdAt: number; // timestamp; newest first in list
}
```

7) Behaviors & Edge Cases

- **Trim input** before create; ignore if empty after trim.
- **Delete** removes the task immediately.
- **Toggle** flips completed boolean without altering ordering.
- **Ordering:** Always **newest first** (by createdAt).

8) Acceptance Criteria (testable)

1. Given an empty app, when I type “Buy milk” and press Enter, I see one task at the top with text “Buy milk”.
2. Given a task “Buy milk”, when I click its checkbox, the text shows strike-through; clicking again removes strike-through.
3. Given tasks A then B, after adding B last, B appears **above** A.
4. Given a task, when I click its Delete button, that task disappears.
5. Given an empty input or only spaces, pressing Enter/Add does **nothing**.
6. Keyboard-only: I can Tab to input, type, press Enter to add; Tab to task checkboxes and toggle them; Tab to delete and activate it via Enter/Space.
7. No network requests are required; reloading the page may reset the list.

9) Folder Structure (suggested)

```
/src
  /components
    Header.jsx/tsx
    TaskInput.jsx/tsx
    TaskList.jsx/tsx
    TaskItem.jsx/tsx
  App.jsx/tsx
```

main.jsx/tsx

index.css

10) Code Quality Expectations

- Clear, small components with single responsibility.
- No unused code/vars; basic lint cleanliness.
- Meaningful names (TaskItem, onAddTask, etc.).
- Minimal inline styles; use simple CSS classes.
- Comments only where intent isn't obvious.

11) Submission Requirements

- Git repo or zipped project with:
    - README.md including:
        - How to run locally (npm i && npm run dev or equivalent).
        - Your React + tooling choice (Vite/CRA/Next).
        - Any decisions/notes (≤5 bullets).
- App should start with a single command and open in the browser.

12) Evaluation Rubric (100 pts)

- **Correctness (35):** Meets all must-have features & acceptance criteria.
- **Code Quality (25):** Readability, component structure, state handling, naming.
- **Accessibility & UX (15):** Labels, keyboard support, clear affordances, empty state.
- **Simplicity (8):** Minimal dependencies, no over-engineering.
- **Polish (7):** Basic responsive layout, tidy styling, no console errors.
- **Documentation (10) :**Clear and understable read me, easy to follow setup instructions, etc

13) Optional Nice-to-Haves (do not grade as required)

- Lightweight unit tests for a reducer or a utility function.
- Simple count of remaining tasks.
- Micro-animation on add/delete (CSS only).