**ToDos Context Potential Improvements**

**1. Add unique IDs to todos:**

const addTodo = (text) => {

setTodos((prevTodos) => [

...prevTodos,

{

id: Date.now() + Math.random(), *// or use crypto.randomUUID()*

text,

completed: false

},

]);

setNewTodo("");

};

You don't need an import for crypto.randomUUID().

crypto.randomUUID() is part of the Web Crypto API that's built into modern browsers - it's a global browser API, so you can use it directly without any imports (see below for use).

**2. Update toggleTodo to use ID instead of index:**

const toggleTodo = (id) => {

setTodos((prevTodos) =>

prevTodos.map(todo =>

todo.id === id

? { ...todo, completed: !todo.completed }

: todo

)

);

};

**3. Add a deleteTodo function:**

const deleteTodo = (id) => {

setTodos((prevTodos) => prevTodos.filter(todo => todo.id !== id));

};

**4. Add error handling for localStorage:**

const [todos, setTodos] = useState(() => {

try {

const savedTodos = localStorage.getItem("todos");

return savedTodos ? JSON.parse(savedTodos) : [];

} catch (error) {

console.error("Failed to load todos from localStorage:", error);

return [];

}

});

**5. Consider adding input validation:**

const addTodo = (text) => {

const trimmedText = text.trim();

if (!trimmedText) return; *// Don't add empty todos*

setTodos((prevTodos) => [

...prevTodos,

{

id: crypto.randomUUID(),

text: trimmedText,

completed: false

},

]);

setNewTodo("");

};

Using array indices as IDs is problematic because **indices change when the array is modified**, which can cause serious issues in React applications.

**The Problems with Using Index as ID**

**1. React's Reconciliation Gets Confused**

*// Initial state: [todo1, todo2, todo3]*

*// Indices: [0, 1, 2]*

*// After deleting the first item: [todo2, todo3]*

*// Indices: [0, 1]*

React sees index 0 and thinks it's still the same item (todo1), but it's actually todo2 now. This causes React to:

* Update the wrong components
* Lose component state unexpectedly
* Show incorrect data

**2. Input Focus and State Issues**

*// If you have editable todos with input fields:*

todos.map((todo, index) => (

<input key={index} value={todo.text} />

))

*// When you delete the first todo, React thinks the input at index 0*

*// is the same input, so it keeps its internal state (like cursor position)*

*// but shows different content*

**3. Performance Problems** React can't efficiently determine which items actually changed, so it might re-render more components than necessary.

**Real Example of the Bug**

*// BAD: Using index as key*

{todos.map((todo, index) => (

<TodoItem

key={index} *// Don't do this!*

todo={todo}

onToggle={() => toggleTodo(index)}

/>

))}

*// What happens:*

*// 1. You have 3 todos*

*// 2. Each TodoItem has some internal state (like hover effects, input focus)*

*// 3. You delete the first todo*

*// 4. The TodoItem that was at index 1 is now at index 0*

*// 5. React thinks it's the same component and keeps the old state*

*// 6. You get weird behavior like wrong hover states or focus issues*

**The Solution: Stable, Unique IDs**

*// GOOD: Using stable unique IDs*

{todos.map((todo) => (

<TodoItem

key={todo.id} *// This never changes for a given todo*

todo={todo}

onToggle={() => toggleTodo(todo.id)}

/>

))}

**Why this works:**

* Each todo keeps the same ID throughout its lifetime
* React can correctly track which component is which
* Component state is preserved correctly
* Better performance through proper reconciliation

**Quick Rule of Thumb**

* Use index as key: Only for static lists that never change
* Don't use index as key: For dynamic lists where items can be added, removed, or reordered

**What Actually Happens**

*// Initial render:*

*// todos = [{text: "Buy milk"}, {text: "Walk dog"}, {text: "Code"}]*

*// Keys: [0, 1, 2]*

*// After deleting first item:*

*// todos = [{text: "Walk dog"}, {text: "Code"}]*

*// Keys: [0, 1]*

*// React thinks:*

*// - Key 0 = "Buy milk"* → *now shows "Walk dog" (wrong!)*

*// - Key 1 = "Walk dog"* → *now shows "Code" (wrong!)*

*// - Key 2 = "Code"* → *component deleted*

Even if you stored the index in a variable, **the index values themselves change when the array is modified**. React uses the key to determine component identity - if the key changes meaning, React gets confused.

**The Real Solution**

You need **stable identifiers** that don't change when the array is modified:

*// Each todo keeps its ID forever*

const addTodo = (text) => {

setTodos(prev => [...prev, {

id: crypto.randomUUID(), *// or Date.now() + Math.random()*

text,

completed: false

}]);

};

*// Keys remain stable regardless of array changes*

{todos.map((todo) => (

<TodoItem

key={todo.id} *// This never changes for a given todo*

todo={todo}

onToggle={() => toggleTodo(todo.id)}

/>

))}

The key insight is that **the problem isn't how you access the index - it's that indices are inherently unstable in dynamic arrays**. You need identifiers that survive array operations like adding, removing, and reordering items.