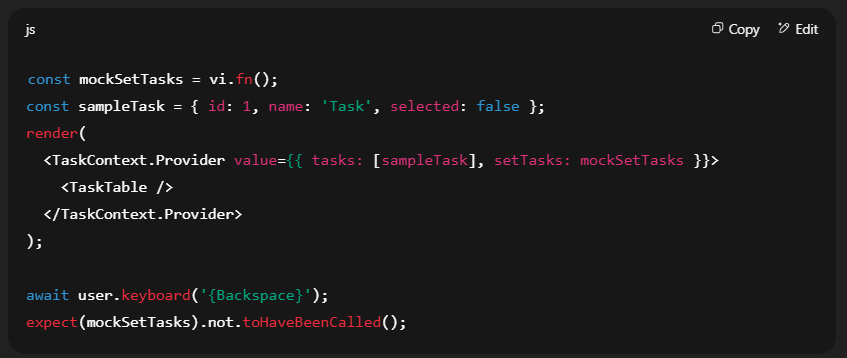
# 🧪 TaskTable Integration Test Quiz

## Multiple Choice

1. 1. What does `screen.getByRole('button', { name: /career/i })` do?  
    A. Selects a button with exact text "career"  
    B. Selects a button containing the word "Career", case-insensitive  
    C. Selects a span with role "button"  
    D. Finds a button by its CSS class name

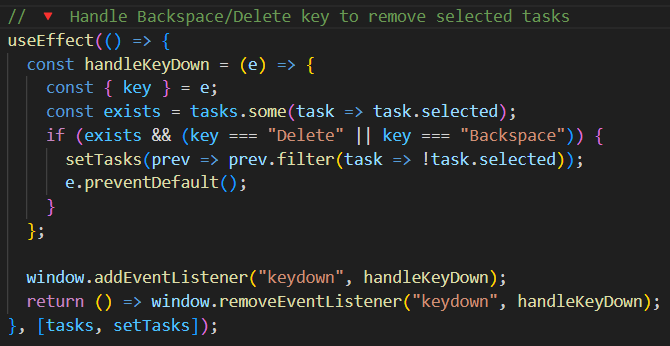
2. Why does the following test pass even though the `keyboard('{Backspace}')` action is called? (From TaskTable.test.jsx)  


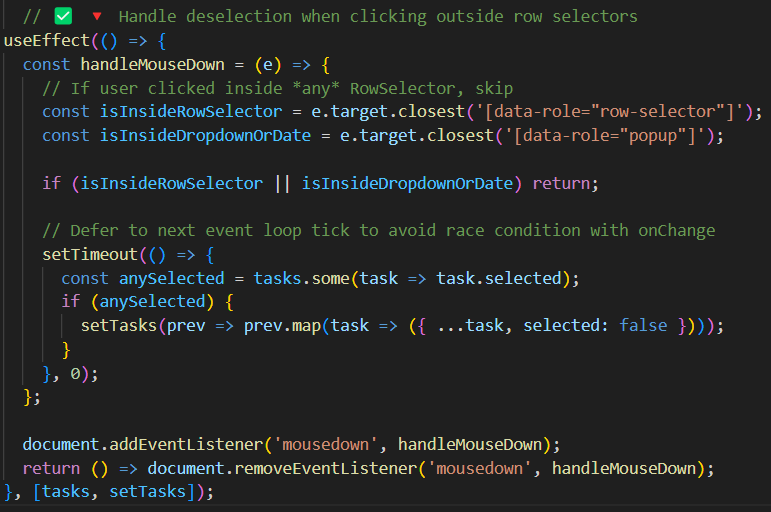
A. The mock function fails silently  
 B. The component unmounts before the call happens  
 C. The deletion logic only triggers if a task is marked as `selected: true`  
 D. `keyboard` is not supported by the testing library

Reference the images below for the following questions

A black background with blue and white text

AI-generated content may be incorrect.





3. What is the primary purpose of the `useMemo` hook in this context?

A. To make `calcSum` recompute every time any key is pressed  
B. To improve performance by memoizing the computed sum unless `tasks` changes  
C. To avoid re-rendering the table when tasks are deleted  
D. To initialize task estimation with a default value

4. What does `(Number(task.time\_estimation) || 0)` accomplish inside the reducer?

A. Converts falsy values to null  
B. Ensures only string values are added  
C. Guards against `undefined`, `null`, or non-numeric values  
D. Throws an error if `time\_estimation` is missing

5. Why is `e.preventDefault()` used after detecting a Delete or Backspace keypress?

A. To prevent the browser from navigating back or deleting a focused input field  
B. To stop the task from being added  
C. To block React from re-rendering the component  
D. To simulate a database call

6. Which hook adds and cleans up a global keyboard event listener?

A. `useReducer`  
B. `useCallback`  
C. `useEffect`  
D. `useMemo`

7. What does `e.target.closest('[data-role="row-selector"]')` check for in the second `useEffect`?

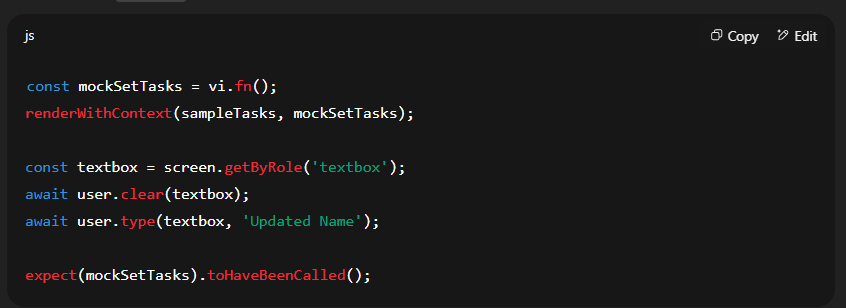
A. Whether the event came from a button element  
B. Whether the clicked element is within a dropdown  
C. Whether the clicked target is inside a row selector element  
D. Whether the document was clicked at all

8. Why is `setTimeout(..., 0)` used before deselecting tasks?

A. To wait 1 second  
B. To debounce the deselection logic  
C. To defer execution until after onChange finishes to prevent a race condition  
D. To allow React to cancel the action

## Short Answer

9. Explain what `vi.fn()` is and how it's used in this context (From TaskTable.test.jsx):



10. Why is `{ name: /career/i }` written with slashes and the `i` flag in the role query?

11. What does the following test verify? (From TaskTable.test.jsx)A computer screen shot of a program code

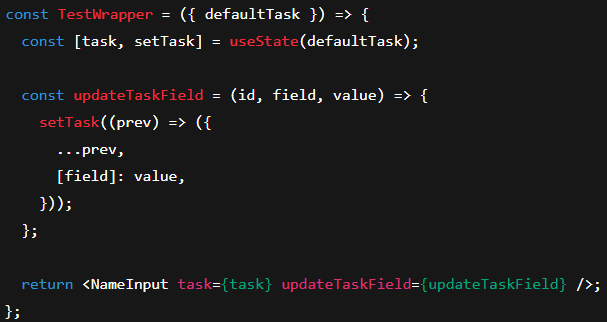
AI-generated content may be incorrect.

12. Explain why `[tasks, setTasks]` appears in the dependency array of both `useEffect` hooks.  
What would happen if you omitted them?

13 What side effects do each `useEffect` implement in plain language?  
Give one sentence for each `useEffect` block summarizing what they accomplish.

14 Describe how deselection is handled when clicking outside a row. Why not just use `onBlur` instead?

15. What behavior would break if you forgot to remove the event listeners in the cleanup function of either `useEffect`?  
Explain how this might affect performance or user interaction over time.

**16.** In the test file, the NameInput component is rendered through the following wrapper:  


Explain why the TestWrapper component is necessary for testing NameInput. Additionally, explain why the updateTaskField function must accept (id, field, value) as parameters and use dynamic property access like [field]: value. What would break if either of these patterns were changed?

## Coding Exercises

## A black background with white text AI-generated content may be incorrect.

## A screenshot of a black screen AI-generated content may be incorrect.

A screenshot of a computer program

AI-generated content may be incorrect.

A screenshot of a computer program

AI-generated content may be incorrect.

A screenshot of a phone

AI-generated content may be incorrect.

## Section 2: Answer Key

**1.** **B** – Selects a button containing the word “Career”, case-insensitive  
screen.getByRole('button', { name: /career/i }) uses a case-insensitive regex to find a button labeled “Career”.

**2.** **C** – The deletion logic only triggers if a task is marked as selected: true  
The test verifies that nothing is deleted unless a task is selected.

**3.** **B** – To improve performance by memoizing the computed sum unless tasks changes  
useMemo is used to avoid recalculating the sum on every render unless tasks has changed.

**4.** **C** – Guards against undefined, null, or non-numeric values  
Number(...) || 0 ensures a valid number is added to the total even if a task has an invalid or missing time\_estimation.

**5.** **A** – To prevent the browser from navigating back or deleting a focused input field  
e.preventDefault() blocks native behavior that could interfere with custom task deletion.

**6.** **C** – useEffect  
This hook is responsible for managing side effects, including adding/removing event listeners.

**7.** **C** – Whether the clicked target is inside a row selector element  
This ensures we don’t deselect tasks if the user clicked inside a multi-select checkbox area.

**8.** **C** – To defer execution until after onChange finishes to prevent a race condition  
Using setTimeout(..., 0) lets UI events complete before deselection logic runs.

**9.**  
vi.fn() is a mock function from Vitest.  
It’s used to spy on setTasks to confirm it’s called when the user interacts with the input (like typing a new task name).

**10.**  
{ name: /career/i } uses a regular expression with the i flag for case-insensitive matching.  
This allows the test to match "Career", "career", or "CAREER" consistently.

**11.**  
The test verifies that selected tasks are removed when Delete or Backspace is pressed.  
It inspects whether setTasks filters out all task.selected === true entries.

**12.**  
Including [tasks, setTasks] in the dependency array ensures the useEffect hooks respond to changes.  
Omitting them could cause the event listeners to reference stale state or behave inconsistently.

**13.**

* First useEffect: Adds a global keyboard listener for Delete/Backspace to remove selected tasks.
* Second useEffect: Adds a mousedown listener that deselects all tasks when clicking outside a selector or popup.

**14.**  
Deselection is triggered by detecting clicks outside any row selector or popup using .closest().  
onBlur wouldn’t work reliably across dropdowns or calendar popups, as focus changes too easily between elements.

**15.**  
Forgetting to clean up event listeners would cause memory leaks or repeated handler calls.  
Over time, tasks might be deleted multiple times or undesired state might persist due to outdated event logic.

16.

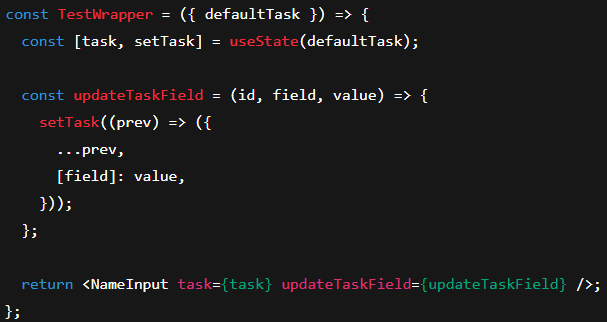
The TestWrapper component is necessary because NameInput relies on receiving a task object and an updateTaskField function via props. Since NameInput is a controlled input component, it doesn't manage its own internal state—instead, it expects its parent to pass updated props when the input changes. TestWrapper fulfills this role by:

1. **Providing local state (task)** using useState, which allows the test to simulate and track changes.
2. **Implementing the updateTaskField function**, which updates the specific field (name) in the task when the input is edited.

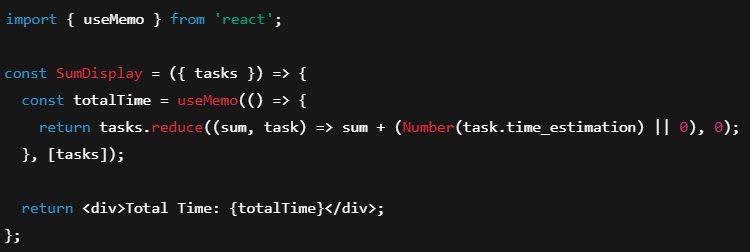
The reason updateTaskField must have the signature (id, field, value) and use dynamic property access ([field]: value) is because:

* It is designed to be a **generic field updater** for any task field (e.g., name, date, time\_estimation), not just the name field.
* The NameInput component calls updateTaskField(task.id, "name", e.target.value). If the function didn’t accept these three parameters or didn’t use [field]: value, the test would break or fail silently because:
  + The state wouldn't update correctly,
  + The component wouldn't re-render with the new input value,
  + And the input box would appear non-editable or stale in tests.

In summary, the wrapper simulates parent behavior, and the field-based update logic mirrors the app’s actual dynamic state update patterns, making the test meaningful and realistic.

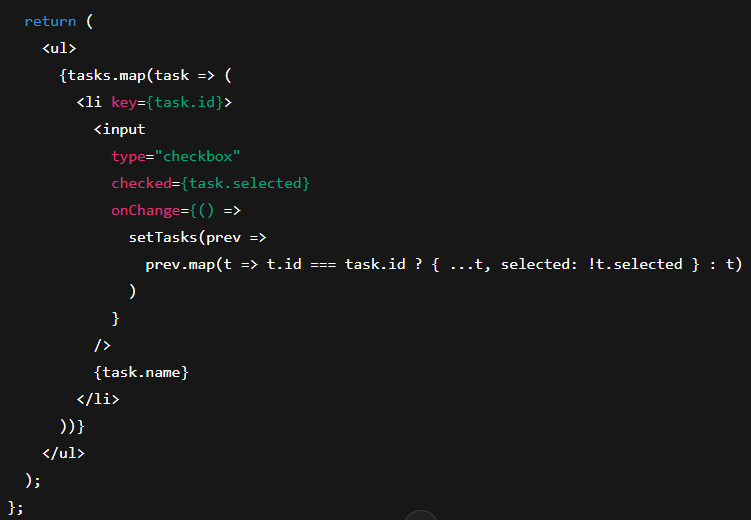


Coding Exercises

1.

2





3

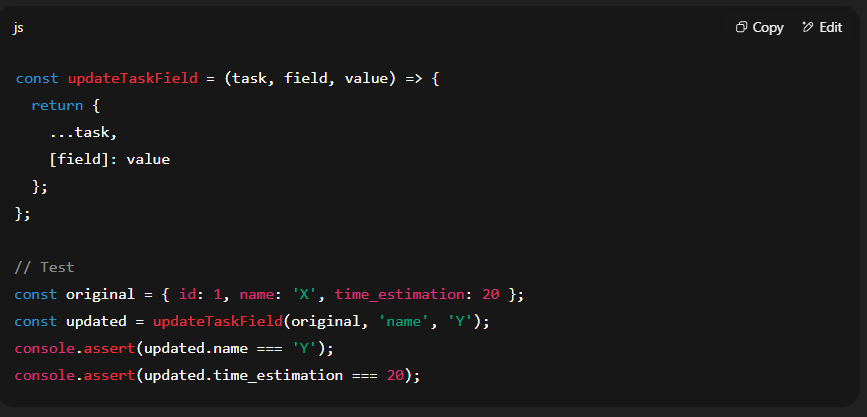
A computer screen shot of a program code

AI-generated content may be incorrect.

4

A computer screen shot of a program code

AI-generated content may be incorrect.

5. 

6.