

Participant P19

00:01:44 **Researcher**
You can start

00:01:45 **Participant**
ok

00:01:47 First I will open the task file. And I will read what is being asked

00:02:36 Before I see the task, let me see what we have here.

00:02:41 **Researcher**
One thing I forgot to mention is that the dependencies are already installed.

00:02:44 **Participant**
Cool.

00:02:55 > **Viewing file** (src/todo/app.jsx)

00:02:55 > **Viewing file** (src/todo/components/main.jsx)

00:02:55 > **Viewing file** (src/index.js)

00:02:59 **Participant**
Ok.

00:03:34 > **Viewing file** (package.json)

00:04:22 > **Viewing file** (src/todo/components/main.jsx)

00:04:51 **Researcher**
What are you thinking now, Participant?

00:04:57 **Participant**
I'm analyzing the application and seeing what can be done currently,
now I will try to find where this component is
that looks like a button.

00:05:01

00:05:06

00:05:15 **Researcher**
To add the tasks, use the enter key,
alright.

00:05:17

00:05:20 **Participant**
Perfect.

00:05:37 I remembered to have standard CSS.

00:05:42 > **Viewing file** (src/todo/app.jsx)

00:05:49 > **Viewing file** (src/todo/components/paginationControls.jsx)

00:05:51 > **Viewing file** (src/todo/components/main.jsx)

00:05:56 > **Viewing file** (src/todo/app.css)

00:05:57 > **Editing file** (src/todo/app.css)

- 00:06:14

00:06:07 **Participant**
An information, might be somehow valuable that I don't
am not used to using Chat in the IDE.
in day to day and when I really need to use AI
I open it in the browser, separate tab from the project
I use it in the planning phase

00:06:11

00:06:17

00:06:21

00:06:33

00:06:37 **Researcher**
ok.

00:06:38 **Participant**
Good for ideas but bad for execution.

00:06:41 **Researcher**
Cool.

00:07:04 **Participant**
Now I will interact with the AI
I will ask how to change the label.

00:07:09

00:07:21 > **Viewing file** (src/todo/components/main.jsx)

00:08:04

Researcher

I forgot to mention that you can do web searches

00:08:10

If you feel that the copilot is not helping you.

00:08:21

> **Viewing file** (src/todo/app.css)

00:08:23

> **Viewing file** (src/todo/app.jsx)

00:08:26

> **Viewing file** (src/todo/components/header.jsx)

00:08:28

> **Editing file** (src/todo/components/header.jsx)

- 00:08:31

00:09:18

> **Viewing file** (src/todo/components/main.jsx)

00:09:36

Chat (window: 0)

> **Agent**

github.copilot.default

> **Context**

selection (src/todo/components/main.jsx)

> **Message**

What is useMemo?

> **Response**

The [useMemo \(node_modules/@types/react/index.d.ts\)](#) is a React *hook* that allows you to memorize the result of a calculation function to avoid it being recalculated unnecessarily on each render. It is useful for optimizing component performance, especially when the calculation is costly or when the result depends on values that change infrequently.

The signature of [useMemo \(node_modules/@types/react/index.d.ts\)](#) is:

```
const memoizedValue = useMemo(() => computeValue, [dependencies]);
```

How it works:

- **computeValue:** It is a function that returns the value you want to memorize.
- **dependencies:** It is an array of dependencies that determines when the value should be recalculated. React will only execute the function again if any of the dependencies change.

Example in your code:

In the snippet below from your file, [useMemo \(node_modules/@types/react/index.d.ts\)](#) is being used to calculate the [visibleTodos \(src/todo/components/main.jsx\)](#) based on the filters ([route \(src/todo/components/main.jsx\)](#)) and the list of [todos \(src/todo/components/main.jsx\)](#):

```
const visibleTodos = useMemo(  
  () =>  
    todos.filter((todo) => {  
      if (route === "/active")  
        return !todo.completed;  
  
      if (route === "/completed")  
        return todo.completed;  
  
      return todo;  
    }),  
  [todos, route]  
);
```

- **Calculation function:** Filters the [todos \(src/todo/components/main.jsx\)](#) based on the current route ([route \(src/todo/components/main.jsx\)](#)).

- **Dependencies:** [todos \(src/todo/components/main.jsx\)](#) and [route \(src/todo/components/main.jsx\)](#). The calculation will only be redone if the list of [todos \(src/todo/components/main.jsx\)](#) or the route changes.

Benefit:

Without [useMemo \(node_modules/@types/react/index.d.ts\)](#), this filtering would be recalculated on every render, even if [todos \(src/todo/components/main.jsx\)](#) and [route \(src/todo/components/main.jsx\)](#) have not changed. With [useMemo \(node_modules/@types/react/index.d.ts\)](#), React reuses the previously calculated value, improving performance.

00:09:38
- 00:09:44

00:10:34

> Response rendering

Chat (window: 0)

> Agent

github.copilot.default

> Context

selection (src/todo/components/main.jsx)

> Message

How can I sort this array: `const visibleTodos = useMemo( () => todos.filter((todo) => { if (route === "/active") return !todo.completed; if (route === "/completed") return`

```
todo.completed; return todo; }), [todos, route] );
```

> Response

You can sort the array [visibleTodos \(src/todo/components/main.jsx\)](#) by adding a call to the `.sort()` method after the filter in [useMemo \(node_modules/@types/react/index.d.ts\)](#). For example, if you want to sort the todos by a property like `title` in alphabetical order, or by `id` [\(node_modules/@types/react/index.d.ts\)](#) in ascending order, you can do this within the [useMemo \(node_modules/@types/react/index.d.ts\)](#).

Here's how the updated code would look:

```
File: src/todo/components/main.jsx
const visibleTodos = useMemo(
  () =>
    todos
      .filter((todo) => {
        if (route === "/active")
          return !todo.completed;

        if (route === "/completed")
          return todo.completed;

        return todo;
      })
      .sort((a, b) => a.title.localeCompare(b.title)), // Sorts
    by title in alphabetical order
    [todos, route]
);
```

Explanation:

1. [filter \(src/todo/components/main.jsx\)](#): Filters the todos based on the route ([route \(src/todo/components/main.jsx\)](#)).
2. **sort**: Sorts the filtered todos. In the example above:
 - `a.title.localeCompare(b.title)`: Sorts the items based on the `title` property in alphabetical order.
 - You can replace it with another sorting logic, like `a.id - b.id` to sort by [id \(node_modules/@types/react/index.d.ts\)](#) in ascending order.

Example of sorting by date:

If the todos have a `createdAt` property (creation date), you can sort by the most recent date:

```
.sort((a, b) => new Date(b.createdAt) - new Date(a.createdAt)) // Most recent first
```

Note:

Make sure that the property used for sorting (`title`, [id \(node_modules/@types/react/index.d.ts\)](#), `createdAt`, etc.) exists in each object of the array [todos \(src/todo/components/main.jsx\)](#).

> Response rendering

Participant

I'm going to test how the current sorting is going

> **Implementing Copilot code** by copying part of snippet (`src/todo/components/main.jsx`)

Participant

00:10:36
- 00:10:43

00:10:37

00:10:55
- 00:11:00

00:11:04

00:11:21
00:11:30
00:12:23

1223, no, wait, the quick brown 123.

Ready, quick, brown fox.

1123, also fox.

Chat (window: 0)

> Agent

github.copilot.default

> Context

selection (src/todo/components/main.jsx)

> Message

In this case, will the sort not be rendered?

> Response

In the current code, the `.sort()` method will be executed **only when the dependencies of `useMemo (node_modules/@types/react/index.d.ts)` change**, that is, when `todos (src/todo/components/main.jsx)` or `route (src/todo/components/main.jsx)` are updated. This happens because `useMemo (node_modules/@types/react/index.d.ts)` memorizes the returned value (the sorted array) and only recalculates when necessary.

Explanation:

- `useMemo (node_modules/@types/react/index.d.ts)` prevents the calculation (including `.filter()` and `.sort()`) from being redone on every component render.
- The `.sort()` will be executed **only** when:
 1. The list of `todos (src/todo/components/main.jsx)` changes (for example, a new item is added or removed).
 2. The route (`route (src/todo/components/main.jsx)`) changes (for example, from `/active` to `/completed`).

If none of these dependencies change, React will reuse the previously calculated value, and the `.sort()` will not be executed again.

When will the `.sort()` be re-executed?

If `todos (src/todo/components/main.jsx)` or `route (src/todo/components/main.jsx)` change, `useMemo (node_modules/@types/react/index.d.ts)` will recalculate the value of `visibleTodos (src/todo/components/main.jsx)`, which includes executing `.filter()` and `.sort()`.

How to ensure that `.sort()` is not re-executed unnecessarily?

`useMemo (node_modules/@types/react/index.d.ts)` is already correctly configured with the dependencies `[todos, route]`. This means that `.sort()` will be executed only when necessary. There is no need for additional optimizations unless the `todos (src/todo/components/main.jsx)` array is very large and the performance of `.sort()` becomes an issue.

If performance is a concern, you might consider: 1. **Sorting the data at the source** (for example, on the backend or when loading the data). 2. **Avoiding reordering if the data is already sorted**. This can be done by checking if the list is already in the desired order before calling `.sort()`.

> Response rendering

> Viewing file (src/todo/app.jsx)

> Viewing file (src/todo/reducer.js)

> Viewing file (src/todo/app.jsx)

> Viewing file (src/todo/components/main.jsx)

00:12:25
- 00:12:29

00:12:48

00:12:55

00:13:00

00:13:03

00:13:16
- 00:13:53
00:13:55
00:13:58
00:14:01
- 00:14:20
00:14:58
00:15:42

> **Editing file** (src/todo/components/main.jsx)

> **Viewing file** (src/todo/components/header.jsx)

> **Viewing file** (src/todo/components/main.jsx)

> **Editing file** (src/todo/components/main.jsx)

Participant

I'm going to test now.

Chat (window: 0)

> **Agent**

github.copilot.default

> **Context**

selection (src/todo/components/main.jsx)

> **Message**

Explain what "localeCompare" is in this line: `.sort((a, b) => a.title.localeCompare(b.title)), // Sorts by title in alphabetical order`

> **Response**

The [localeCompare \(src/todo/components/main.jsx\)](#) method is a JavaScript function used to compare two strings according to the sorting rules specific to a language or region (locale). It is particularly useful for correctly sorting strings alphabetically, taking into account accents, special characters, and case sensitivity (uppercase/lowercase).

Syntax:

```
string1.localeCompare(string2, [locales], [options])
```

- **string1**: The string being compared.
- **string2**: The string with which string1 will be compared.
- **locales** (optional): Specifies the language or region for the comparison (e.g., "pt-BR" for Brazilian Portuguese).
- **options** (optional): Allows configuring case sensitivity, accents, etc.

Return:

- **-1**: If string1 comes before string2 in alphabetical order.
- **0**: If string1 and string2 are equal.
- **1**: If string1 comes after string2 in alphabetical order.

In your code:

```
.sort((a, b) => a.title.localeCompare(b.title))
```

Here, the [localeCompare \(src/todo/components/main.jsx\)](#) method is being used to compare the values of the [title \(src/todo/components/main.jsx\)](#) property of two [todo \(src/todo/components/main.jsx\)](#) objects ([a \(src/todo/components/main.jsx\)](#) and [b \(src/todo/components/main.jsx\)](#)) and sort them in alphabetical order.

- [a.title \(src/todo/components/main.jsx\)](#): The title of the first item.
- [b.title \(src/todo/components/main.jsx\)](#): The title of the second item.
- The [localeCompare \(src/todo/components/main.jsx\)](#) method compares the two titles and returns:
 - **-1** if [a.title \(src/todo/components/main.jsx\)](#) should come before [b.title \(src/todo/components/main.jsx\)](#).
 - **1** if [a.title \(src/todo/components/main.jsx\)](#) should come after [b.title \(src/todo/components/main.jsx\)](#).
 - **0** if the two titles are equal.

Example:

If you have the following titles:

```
const todos = [  
  { title: "Banana" },  
  { title: "Árvore" },  
  { title: "abacaxi" }  
];
```

After sorting:

```
todos.sort((a, b) => a.title.localeCompare(b.title));
```

The result will be:

```
[  
  { title: "abacaxi" },  
  { title: "Árvore" },  
  { title: "Banana" }  
]
```

Why use `localeCompare` ([src/todo/components/main.jsx](#))?

- It respects the sorting rules of different languages (e.g., in Portuguese, "Árvore" comes before "Banana").
- It correctly handles accents and special characters.
- It is more robust than using operators like `<` or `>` to compare strings.

00:15:44
- 00:15:51

00:16:18

00:16:22
- 00:18:08

00:16:53

> Response rendering

> Viewing file (src/todo/components/main.jsx)

> Implementing Copilot code by copying part of snippet (src/todo/components/main.jsx)

Chat (window: 0)

> Agent

github.copilot.default

> Context

viewport (src/todo/components/main.jsx)

> Message

Why in the function visibleTodos does it not reflect in the sorted array

> Response

The reason why the function [visibleTodos \(src/todo/components/main.jsx\)](#) might not reflect the sorted array is that the `.sort()` method modifies the original array **in-place** (i.e., it alters the array [todos \(src/todo/components/main.jsx\)](#) directly). This can cause unexpected issues, especially in an environment like React, where data immutability is important to ensure that renders are triggered correctly.

Problem with `.sort()`:

- The `.sort()` method does not create a new array; it modifies the original array ([todos \(src/todo/components/main.jsx\)](#)).
- If the array [todos \(src/todo/components/main.jsx\)](#) is used elsewhere in the component or in other components, the sorting applied by `.sort()` can cause unwanted side effects.
- React may not detect changes in state or props if the original array is altered directly, which can lead to unexpected behaviors.

Solution:

To avoid this problem, you should create a copy of the array before sorting it. This ensures that the original array ([todos \(src/todo/components/main.jsx\)](#)) remains unchanged and that React correctly detects the changes.

Corrected Code:

Replace `.sort()` with an approach that copies the array before sorting it:

```
const visibleTodos = useMemo(
  () =>
    [...todos] // Creates a copy of the original array
      .filter((todo) => {
        if (route === "/active")
          return !todo.completed;

        if (route === "/completed")
          return todo.completed;

        return todo;
      })
);
```

```
    })
    .sort((a, b) => a.title.localeCompare(b.title)), // Sorts
by title in alphabetical order
    [todos, route]
);
```

What Changed:

1. [...**todos**]: Creates a shallow copy of the array [todos](#) ([src/todo/components/main.jsx](#)) before applying `.filter()` and `.sort()`. This ensures that the original array is not modified.
2. The rest of the code remains the same, but now React can correctly detect changes in the sorted array.

Why Does This Work?

- React relies on immutability to detect changes in state or props. By creating a new copy of the array, you ensure that React sees the array as a new value, triggering rendering when necessary.
- This also avoids unwanted side effects in other parts of the code that might be using the array [todos](#) ([src/todo/components/main.jsx](#)).

Conclusion:

Whenever you use methods that modify arrays in-place, like `.sort()`, `.splice()`, etc., remember to create a copy of the original array to preserve immutability and avoid issues in React.

00:16:55
- 00:17:08

> Response rendering

00:17:39

Participant

But in this case, then, I could simply make a change at this point.

00:17:57

No.

00:19:16

Is it working now?

00:19:18

Researcher

It's now, sorted.

00:19:28

Participant

Undeniably, the copilot helped a lot with the sorting, I needed to know how the pointer worked.

00:19:42

Researcher

Cool.

00:19:44

I'm going to send you the task file now

00:19:51

send it in the chat.

00:20:05

There are 3 tasks. You don't need to do them in order.

00:20:19

I'm going to ask you to validate all the requested points.

00:20:59

Participant

Alright.

00:21:01

So, persistence.

00:21:14

> Viewing file (src/todo/constants.js)

00:21:17

> Viewing file (src/todo/app.jsx)

00:21:21

> Viewing file (src/todo/components/main.jsx)

00:22:01

Chat (window: 0)

> Agent

github.copilot.workspace

> Context

workspace, viewport (src/todo/components/main.jsx)

> Message

@workspace where was there an attempt to implement local persistence in this application?

> Response

The attempt to implement local persistence in the application is in the file [src/todo/reducer.js](#). Specifically, the functions [loadState \(src/todo/reducer.js\)](#) and [saveState \(src/todo/reducer.js\)](#) were created to load and save the state of the tasks in [localStorage \(node_modules/@types/node/globals.d.ts\)](#).

Here are the relevant snippets:

```
File: src/todo/reducer.js
export const loadState = () => {
  const state = localStorage.getItem("todoS");
  return state ? JSON.parse(state) : [];
};

const saveState = (state) => {
  localStorage.setItem("todos", JSON.stringify(state));
};
```

- [loadState \(src/todo/reducer.js\)](#): Loads the saved state from [localStorage \(node_modules/@types/node/globals.d.ts\)](#) under the key "todoS". If nothing is saved, it returns an empty array.
- [saveState \(src/todo/reducer.js\)](#): Saves the current state in [localStorage \(node_modules/@types/node/globals.d.ts\)](#) under the key "todos".

Additionally, the function [todoReducer \(src/todo/reducer.js\)](#) calls [saveState \(src/todo/reducer.js\)](#) after each modification to the state:

```
File: src/todo/reducer.js
saveState(newState);
```

However, there is a problem: the key used to load the state ("todoS") is different from the one used to save the state ("todos"), which may cause persistence failures.

00:22:06
- 00:22:10

> **Response rendering**

00:22:19

> **Viewing file** (src/todo/reducer.js)

00:22:36
- 00:22:38

> **Implementing Copilot code** using apply button (src/todo/reducer.js)

00:23:08

Participant

I'm going to ask how to use localStorage.

00:23:13

> **Viewing file** (src/todo/reducer.js)

00:23:27

Chat (window: 0)

> **Agent**

github.copilot.default

> **Context**

viewport (src/todo/reducer.js)

> **Message**

How to use localStorage to save the todos?

> **Response**

The code already uses [localStorage \(node_modules/@types/node/globals.d.ts\)](#) to save and load the "todos" (tasks). Here is how it works and how you can ensure the "todos" are saved correctly:

1. Load the "todos" from [localStorage \(node_modules/@types/node/globals.d.ts\)](#)

The function [loadState \(src/todo/reducer.js\)](#) is used to load the saved "todos" from [localStorage \(node_modules/@types/node/globals.d.ts\)](#) when the application starts:

```
export const loadState = () => {
  const state = localStorage.getItem("todos");
  return state ? JSON.parse(state) : [];
};
```

- [localStorage.getItem\("todos"\) \(node_modules/@types/node/globals.d.ts\)](#): Retrieves the data saved under the key "todos".
- [JSON.parse\(state\) \(node_modules/typescript/lib/lib.es2015.symbol.wellknown.d.ts\)](#): Converts the string saved in [localStorage \(node_modules/@types/node/globals.d.ts\)](#) back into an array of objects.
- If nothing is saved, it returns an empty array ([]).

2. Save the "todos" in [localStorage \(node_modules/@types/node/globals.d.ts\)](#)

The function [saveState \(src/todo/reducer.js\)](#) is called after each state modification to save the "todos" in [localStorage \(node_modules/@types/node/globals.d.ts\)](#):

```
const saveState = (state) => {
  localStorage.setItem("todos", JSON.stringify(state));
};
```

- [JSON.stringify\(state\) \(node_modules/typescript/lib/lib.es2015.symbol.wellknown.d.ts\)](#): Converts the array of "todos" into a string to be stored in [localStorage \(node_modules/@types/node/globals.d.ts\)](#).

- `localStorage.setItem("todos", ...)`: Saves the string in [localStorage \(node_modules/@types/node/globals.d.ts\)](#) under the key "todos".

3. Automatically save after each action

In the [todoReducer \(src/todo/reducer.js\)](#), the function [saveState \(src/todo/reducer.js\)](#) is called after each state modification:

```
saveState(newState);
return newState;
```

This ensures that any changes to the "todos" (such as adding, removing, or updating) are immediately saved in [localStorage \(node_modules/@types/node/globals.d.ts\)](#).

4. Complete flow

- When the application is loaded, the initial state ([initialState \(src/todo/reducer.js\)](#)) is set with the data loaded from [localStorage \(node_modules/@types/node/globals.d.ts\)](#):


```
export const initialState = loadState();
```
- When an action is triggered (such as [ADDITEM \(src/todo/constants.js\)](#), [REMOVEITEM \(src/todo/constants.js\)](#), etc.), the state is updated in the [todoReducer \(src/todo/reducer.js\)](#) and saved in [localStorage \(node_modules/@types/node/globals.d.ts\)](#) with [saveState \(src/todo/reducer.js\)](#).

5. Usage example

If you add a new "todo", the flow will be: 1. The dispatch triggers the action [ADD_ITEM \(src/todo/constants.js\)](#). 2. The [todoReducer \(src/todo/reducer.js\)](#) adds the new "todo" to the state. 3. The [saveState \(src/todo/reducer.js\)](#) saves the updated state in [localStorage \(node_modules/@types/node/globals.d.ts\)](#).

6. Checking in the browser

You can check the data saved in [localStorage \(node_modules/@types/node/globals.d.ts\)](#) in the browser console:

```
console.log(localStorage.getItem("todos"));
```

7. Error correction in the code

There is a typo in the [todoReducer \(src/todo/reducer.js\)](#):

```
case L00D_STATE:
```

It should be:

```
case LOAD_STATE:
```

With this corrected, the local persistence flow will work correctly.

00:23:29
- 00:23:42

> Response rendering

00:25:52
- 00:25:54

> Implementing Copilot code using apply button (src/todo/reducer.js)

00:26:02

> Viewing file (src/todo/reducer.js)

00:27:15

Participant

I'm going to check where this is being used.

00:27:16

> Viewing file (src/todo/app.jsx)

00:27:23

Participant

Send the reducer directly.

00:27:37

Chat (window: 0)

> Agent

github.copilot.default

> Context

selection (src/todo/app.jsx)

> Message

How does the useReducer hook work

> Response

The [useReducer\(node_modules/@types/react/index.d.ts\)](#) is a React *hook* that allows you to manage a component's state in a more structured way, especially when the state is complex or involves multiple transitions. It is an alternative to `useState` and is inspired by the Redux state management pattern.

Syntax:

```
const [state, dispatch] = useReducer(reducer, initialState, init);
```

- **reducer**: A function that defines how the state should be updated based on an action. It receives the current state and an action as arguments and returns the new state.
- **initialState**: The initial state of the component.
- **init** (optional): A function to initialize the state, useful for complex initialization logic.
- **state**: The current state managed by [useReducer \(node_modules/@types/react/index.d.ts\)](#).
- **dispatch (src/todo/app.jsx)**: A function used to trigger actions that update the state.

How it works:

1. The [useReducer \(node_modules/@types/react/index.d.ts\)](#) receives a reducer and an initialState.
2. The reducer is automatically called whenever the [dispatch \(src/todo/app.jsx\)](#) is used.
3. The [dispatch \(src/todo/app.jsx\)](#) accepts an "action" (an object that describes what should be done) and passes this action to the reducer.
4. The reducer processes the action and returns the new state, which replaces the previous state.

In your code:

In the file [src/todo/app.jsx](#), the [useReducer \(node_modules/@types/react/index.d.ts\)](#) is being used to manage the state of "todos":

```
const [todos, dispatch] = useReducer(todoReducer, []);
```

- [todoReducer \(src/todo/reducer.js\)](#): It is the function that defines how the state of "todos" will be updated based on actions (such as adding, removing, or toggling a "todo").
- `[]`: It is the initial state, which starts as an empty array.
- [todos \(src/todo/app.jsx\)](#): It is the current state, which contains the list of "todos".
- [dispatch \(src/todo/app.jsx\)](#): It is the function used to trigger actions that update the state.

Example flow:

1. When the user adds a new "todo", the component triggers an action:

```
dispatch({ type: "ADD_ITEM", payload: { title: "New Todo" } });
```

2. The [todoReducer \(src/todo/reducer.js\)](#) processes this action:

```
case ADD_ITEM:
  newState = state.concat({ id: generateUniqueID(), title:
  action.payload.title, completed: false });
  break;
```

3. The state is updated with the new "todo" and the component is re-rendered.

Benefits of [useReducer \(node_modules/@types/react/index.d.ts\)](#):

- **Organization:** Centralizes the state update logic in a single function (reducer).
- **Scalability:** Facilitates the management of complex states with multiple transitions.
- **Predictability:** The state is updated predictably based on actions.

When to use:

- Use [useReducer \(node_modules/@types/react/index.d.ts\)](#) when the component's state is complex or involves multiple actions.
- For simple states, `useState` is generally sufficient.

00:27:41
- 00:27:49

> Response rendering

00:27:43

> Viewing file (src/todo/reducer.js)

00:27:50

> Viewing file (src/todo/app.jsx)

00:27:54

> Viewing file (src/todo/app.jsx)

00:28:01

Participant

No.

00:28:25

Check this line.

00:28:52

> Viewing file (src/todo/reducer.js)

00:29:15

Chat (window: 0)

> Agent

github.copilot.default

> Context

selection (src/todo/app.jsx)

> Message

So in this line, isn't useReducer receiving the initial state? const [todos, dispatch] = useReducer(todoReducer, []);

> Response

In the line const [todos, dispatch] = useReducer(todoReducer, []);, the [useReducer \(node_modules/@types/react/index.d.ts\)](#) is receiving an initial state, which is the empty array []. However, this is not taking advantage of the initial state persisted in localStorage that was implemented in the reducer.js file.

The correct initial state should be the value returned by the loadState function, which loads the saved "todos" from localStorage. To fix this, you can pass the persisted initial state as an argument to the [useReducer \(node_modules/@types/react/index.d.ts\)](#).

Correction:

Replace the initial state [] with the value returned by loadState:

```
import { todoReducer, loadState } from "../reducer";

export function App() {
  const [todos, dispatch] = useReducer(todoReducer, loadState());

  return (
    <>
      <Header dispatch={dispatch} />
      <Main todos={todos} dispatch={dispatch} />
      <Footer todos={todos} dispatch={dispatch} />
    </>
  );
}
```

Explanation:

- **loadState():** Loads the initial state of the "todos" from localStorage. If nothing is saved, it returns an empty array ([]).
- Now, the [useReducer \(node_modules/@types/react/index.d.ts\)](#) will start with the "todos" saved in localStorage, ensuring that local persistence works

correctly.

Alternative with init function:

Another approach would be to use the `init` function of [useReducer](https://react-redux.js.org/api/useReducer) (node_modules/@types/react/index.d.ts) to initialize the state. This is useful if you want to separate the initialization logic:

```
const init = () => loadState();

export function App() {
  const [todos, dispatch] = useReducer(todoReducer, [], init);

  return (
    <>
      <Header dispatch={dispatch} />
      <Main todos={todos} dispatch={dispatch} />
      <Footer todos={todos} dispatch={dispatch} />
    </>
  );
}
```

In this case: - The third argument (`init`) is a function that will be called to initialize the state based on the initial value (`[]`). - The `init` function calls `loadState()` to load the "todos" from `localStorage`.

Both approaches ensure that the initial state is correctly loaded from `localStorage`.

00:29:16
- 00:29:25

00:30:37
- 00:30:44

00:31:18

> **Response rendering**

> **Implementing Copilot code** using apply button (src/todo/app.jsx)

Participant

The code got worse.

Right?

I'll try to do the task of saving the draft

How will the pagination task work?

Researcher

Today the app is not working with pagination

All tasks are added in order without showing the navigation pagination options. The functionality is incomplete in the app.

Participant

ok

> **Viewing file** (src/todo/components/main.jsx)

> **Viewing file** (src/todo/components/paginationControls.jsx)

> **Viewing file** (src/todo/components/main.jsx)

> **Viewing file** (src/todo/components/paginationControls.jsx)

> **Viewing file** (src/todo/components/main.jsx)

> **Viewing file** (src/todo/components/paginationControls.jsx)

> **Viewing file** (src/todo/components/main.jsx)

> **Viewing file** (src/todo/components/paginationControls.jsx)

Chat (window: 0)

> **Agent**

github.copilot.default

> **Context**

selection (src/todo/components/paginationControls.jsx)

> **Message**

Evaluate this pageChange implementation

> **Response**

The implementation of the function [pageChange](#) ([src/todo/components/paginationControls.jsx](#)) has some issues that may cause unexpected behaviors. Let's analyze it:

Issues in the implementation:

1. Lack of handling for the "prev" direction:

- The function only handles the case where the direction is "next". There is no logic to decrement the page when the direction is "prev".
- This means that clicking the "Previous" button will have no effect.

2. undefined return in unhandled cases:

- If the direction is not "next", the function [setCurrentPage](#) ([src/todo/components/paginationControls.jsx](#)) will return undefined, which may cause issues in the page state.

3. Unnecessary dependency on [setClickedButton](#) ([src/todo/components/paginationControls.jsx](#)):

- Although [setClickedButton](#) ([src/todo/components/paginationControls.jsx](#)) is used to store the clicked button, it does not seem essential for the pagination functionality. This can be simplified or removed, depending on the use.

Corrected Code:

Here is a corrected and more robust version of the function [pageChange \(src/todo/components/paginationControls.jsx\)](#):

```
const pageChange = (direction) => {
  setClickedButton(direction);
  setCurrentPage((prevPage) => {
    if (direction === "next") {
      return prevPage + 1;
    } else if (direction === "prev" && prevPage > 1) {
      return prevPage - 1;
    }
    return prevPage; // Returns the current page if no condition is
    met
  });
};
```

Improvements made:

1. Added support for "prev":

- Now, if the direction is "prev", the page will be decremented, but only if [prevPage > 1 \(src/todo/components/paginationControls.jsx\)](#) (to avoid negative or zero pages).

2. Safe return:

- If no condition is met (e.g., an invalid direction), the function returns the current page ([prevPage \(src/todo/components/paginationControls.jsx\)](#)), ensuring that the state is not changed to undefined.

3. Maintained the logic of [setClickedButton \(src/todo/components/paginationControls.jsx\)](#):

- The logic of [setClickedButton \(src/todo/components/paginationControls.jsx\)](#) was maintained, but it can be removed if not essential for functionality.

Additional improvement suggestion:

If you want to avoid passing invalid directions to the function, you can validate the input before calling [setCurrentPage \(src/todo/components/paginationControls.jsx\)](#):

```
const pageChange = (direction) => {
  if (direction !== "next" && direction !== "prev") {
    console.error("Invalid direction:", direction);
    return;
  }

  setClickedButton(direction);
  setCurrentPage((prevPage) => {
    if (direction === "next") {
      return prevPage + 1;
    } else if (direction === "prev" && prevPage > 1) {
      return prevPage - 1;
    }
    return prevPage;
  });
};
```

Final considerations:

- The corrected logic now correctly handles the "next" and "prev" cases.
- Ensure that the "Previous" and "Next" buttons are correctly disabled based on the limits (as already implemented in the component).
- If [setClickedButton \(src/todo/components/paginationControls.jsx\)](#) is not used for something important (like styling or tracking), it can be removed to simplify the code.

00:36:28
- 00:36:37

> Response rendering

00:36:39

> Viewing file (src/todo/app.jsx)

00:36:42

> Viewing file (src/todo/components/main.jsx)

00:37:18
- 00:37:21

> Editing file (src/todo/components/main.jsx)

00:37:25

> Viewing file (src/todo/components/main.jsx)

00:37:31

> Viewing file (src/todo/components/paginationControls.jsx)

00:37:33

> Viewing file (src/todo/components/main.jsx)

00:38:50
- 00:41:25

> Implementing Copilot code using apply button (src/todo/components/main.jsx)

00:40:09

Chat (window: 0)

> Agent

github.copilot.default

> Context

selection (src/todo/components/main.jsx)

> Message

paginationControls is not being imported and yet IntelliJ does not recognize that there is an error, why is this happening on this page

> Response

The reason why IntelliJ (or any other editor/IDE) is not marking an error for [paginationControls \(src/todo/components/main.jsx\)](#) is that JSX allows you to use identifiers that are not explicitly defined or imported, as long as they are not used directly as valid functions or components. This happens because JSX is converted into

function calls during compilation, and the error will only be detected at runtime when React tries to render something that does not exist.

Why does this happen?

1. JSX does not automatically validate identifiers:

- JSX does not check if [paginationControls \(src/todo/components/main.jsx\)](#) is a valid function or component at the time of writing the code. It just assumes it is something that will be resolved at runtime.
- If [paginationControls \(src/todo/components/main.jsx\)](#) is not defined or imported, the error will only be detected when React tries to render the component and cannot find it.

2. No direct use of [paginationControls \(src/todo/components/main.jsx\)](#):

- Since [paginationControls \(src/todo/components/main.jsx\)](#) is not being called as a function or used directly, the editor cannot determine that it should be imported.

3. Possible IntelliJ configuration:

- Depending on IntelliJ's settings, it may not be configured to automatically check if all JSX components are defined or imported.

How to fix:

You need to import the `PaginationControls` component correctly in the [src/todo/components/main.jsx](#) file. Replace the part where [paginationControls \(src/todo/components/main.jsx\)](#) is being used with a valid import and correct use of the component.

Fix:

Add the import at the beginning of the file:

```
import { PaginationControls } from "./paginationControls";
```

And replace the use of [paginationControls \(src/todo/components/main.jsx\)](#) with the correct component:

```
<PaginationControls  
  currentPage={currentPage}  
  setCurrentPage={setCurrentPage}  
  itemsPerPage={itemsPerPage}  
  totalItems={visibleTodos.length}  
>
```

Explanation:

- [currentPage \(src/todo/components/main.jsx\)](#) and [setCurrentPage \(src/todo/components/main.jsx\)](#): Are used to control the current page.
- [itemsPerPage \(src/todo/components/main.jsx\)](#): Defines the number of items per page.
- [totalItems](#): Passes the total number of items (in this case, the length of [visibleTodos \(src/todo/components/main.jsx\)](#)).

Conclusion:

Although IntelliJ does not mark this as an error, React will throw a runtime error because [paginationControls \(src/todo/components/main.jsx\)](#) is not defined. Make sure to import and use the component correctly to avoid issues.

00:40:11
- 00:40:16

> Response rendering

00:41:36

> **Viewing file** (src/todo/components/paginationControls.jsx)

00:42:29

> **Viewing file** (src/todo/app.jsx)

00:42:32

> **Viewing file** (src/todo/components/paginationControls.jsx)

00:42:40

> **Viewing file** (src/todo/components/main.jsx)

00:43:03

Researcher

Participant, time is up, we passed our 40.

00:43:06

Participant

OK

00:43:07

Researcher

I'll pass it to you now.

00:43:12

The post-study questionnaire.

00:43:17

Next, I have a few more questions to ask

00:43:46

And if you have any questions about answering the form or want to make any observations, feel free

00:43:50

00:43:53

Participant

OK

00:46:44

I thought the answers were what I needed..

00:46:51

We have technical, I don't remember him not using technical terms efficacy and improve productivity?

00:48:17

00:48:22

Researcher

It's a level of efficacy, level of response being assertive.

00:48:28

Do you believe that the copilot can help increase productivity.

00:48:37

Participant

Ah, yes I believe so.

00:48:55

documentation and it helps a lot.

00:51:16

did you feel?

00:51:20

Just a little anxious, nothing.

00:51:24

It's too much.

00:51:28

Researcher

In this part, answer how far you got in the tasks

00:52:17

OK. Thank you. Now I'm going to ask you for just a little more time of yours to ask some questions

00:52:20

00:52:25

Participant

OK?

00:52:28

Researcher

Did you feel that you managed to orient yourself through the code and how did the COPILOT help you?

00:52:32

00:52:39

Participant

At first, the code is in an architecture

00:52:43

I'm used to, but really with the copilot tool

00:52:49

in the workspace it's much easier to orient through the code.

00:52:55

even in a messy code, you can find things.

00:53:02

Researcher

Cool.

00:53:04

And did you feel that you understood the code base?

00:53:10

Participant

I believe so.

00:53:13

If I. If I spent more, some time with it,

00:53:17

I believe I would understand the entire code smoothly,

00:53:21

but some time, maybe about 20 minutes to get everything there and do the other tasks

00:53:28

Researcher

00:53:29 Cool.
00:53:33 Did the copilot influence your understanding?
00:53:35 **Participant**
Yes.
00:53:35 **Researcher**
Cool.
00:53:38 How would you have solved the problems without access to the copilot?
00:53:46 **Participant**
Possibly reading much more than I need to read from the documentation.
00:53:52 For the local storage task, of persistence.
00:53:57 And reading much more of the code itself.
00:54:03 **Researcher**
OK. And having the copilot available, do you think it influenced
00:54:07 your strategy to solve the tasks?
00:54:10 **Participant**
Yes, a lot.
00:54:15 **Researcher**
Cool.
00:54:34 What aspect did you like, and what didn't you like about your experience with the copilot?
00:54:44 **Participant**
I liked the aspect that it has access to the entire code.
00:54:48 **Researcher**
OK
00:54:48 **Participant**
helps to locate things.
00:54:50 I also liked the fact that it is very good at generally finding and summarizing text,
00:54:55 It can read documentation or create
00:54:57 documentation much faster with it.
00:55:01 I don't like when I really need to create something from scratch with it
00:55:06 because I don't think it works well.
00:55:10 **Researcher**
Cool.
00:55:12 And during the experiment, did you change the way you interacted with the copilot?
00:55:26 **Participant**
I think so, I believe I became much more.
00:55:28 Willing to use it.
00:55:30 **Researcher**
Cool. Do you think that with time using the copilot you became more confident?
00:55:38 **Participant**
Yes.