

Participant P18

00:00:59 **Researcher**
You can start.

00:01:35 **Participant**
OK

00:02:07 Change the color to blue.

00:02:15 I don't have much knowledge,

00:02:19 > **Viewing file** (src/todo/app.jsx)

00:02:20 **Participant**
but I am a little familiar, like the things that are working.

00:02:28 I will then have this structure

00:02:38 > **Viewing file** (src/todo/components/input.jsx)

00:02:43 > **Viewing file** (src/todo/components/item.jsx)

00:02:49 > **Viewing file** (src/todo/components/main.jsx)

00:02:56 **Participant**
The inputs are here, so I have this class that is here.

00:03:01 load

00:03:06 With the message.

00:03:20 > **Viewing file** (src/todo/app.css) through file search

00:03:26 **Participant**
As I said, I don't have much

00:03:30 knowledge.

00:03:47 **Researcher**
OK.

00:03:52 **Participant**
I need to change the box to blue

00:04:35 **Chat** (window: 0)
> **Agent**
github.copilot.default
> **Context**
viewport (src/todo/app.css)
> **Message**
Copilot, how to add an attribute for the blue color (#007bff) in CSS code
> **Response**
To add a blue color attribute (#007bff) in CSS code, you can use it in properties like color, background-color, border-color, among others. Here is an example of how to apply this color:

```
/* Example of using the blue color (#007bff) */  
.example-text {  
  color: #007bff; /* Sets the text color */  
}  
  
.example-background {  
  background-color: #007bff; /* Sets the background color */  
}  
  
.example-border {  
  border: 2px solid #007bff; /* Sets the border color */  
}
```

In the context of your file, you are already using #007bff in the class .pagination-button:not(.inactive):not(.active). If you need to add this color elsewhere, follow the same pattern.

00:04:36
- 00:04:38

00:04:53
- 00:05:00

00:05:29

> Response rendering

> Implementing Copilot code by copying part of snippet (src/todo/app.css)

Chat (window: 0)

> Agent

github.copilot.workspace

> Context

workspace, viewport (src/todo/app.css)

> Message

When updating the code in a running environment with the command `npm run dev`, is the web page automatically updated with the new modifications?

> Response

Yes, when executing the command `npm run dev`, the project uses the **Webpack Dev Server** configured in the [webpack.dev.js](#) file. This server includes the **Hot Module Replacement (HMR)** or **Live Reload** feature, which automatically updates the web page in the browser whenever there are modifications to the project files.

In your case, since the Webpack Dev Server is configured with the [devServer.static \(webpack.dev.js\)](#) property pointing to the `./dist` directory, it monitors changes in the files and recompiles the project, updating the page automatically.

Make sure the browser is accessing the local server started by the Webpack Dev Server (usually at `http://localhost:8080` by default). If the page does not update automatically, check if the browser is connected to the correct server and if there are no errors in the browser console or terminal.

00:05:33
- 00:05:36

00:05:39

> Response rendering

Participant

Principle hasn't changed.

00:05:55

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

00:05:57
- 00:06:12

> Editing file (src/todo/app.css)

00:06:47

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

00:07:05
- 00:07:11

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

00:07:20

Participant

Now I believe it's right.

00:07:25

Sorting task, alphabetically.

00:08:07

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

00:08:08

Participant

this might be related to the input,

00:08:10

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

00:08:13

Participant

I'll check the files.

00:08:28

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

00:08:32

Participant

It seems to me that it has no relation to the input message

00:08:37

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

00:08:42

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

00:08:44

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

00:09:39

Participant

It might be related to the main.jsx file

00:09:45

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

00:10:33

Participant

All the titles are doing a map

00:10:36

Title

00:10:46

00:10:57

users effect.

Chat (window: 0)

> **Agent**

github.copilot.default

> **Context**

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

> **Message**

What does this React code do? "useEffect(() => { if (todos.length > 0) { const lastItem = todos[todos.length - 1]; setLastAddedItem(lastItem); } }, [todos, lastAddedItem]);"

> **Response**

This piece of React code uses the **hook [useEffect](#)** ([node_modules/@types/react/index.d.ts](#)) to execute a side effect whenever the values of the dependencies ([todos \(src/todo/components/main.jsx\)](#) or [lastAddedItem \(src/todo/components/main.jsx\)](#)) change.

What it does:

1. Checks if there are items in the list [todos \(src/todo/components/main.jsx\)](#):
 - If the array [todos \(src/todo/components/main.jsx\)](#) has at least one item ([todos.length > 0 \(src/todo/components/main.jsx\)](#)), it proceeds.
2. Gets the last item from the list [todos \(src/todo/components/main.jsx\)](#):
 - The last item is accessed using [todos\[todos.length - 1\] \(src/todo/components/main.jsx\)](#).
3. Updates the state [lastAddedItem \(src/todo/components/main.jsx\)](#):
 - The state [lastAddedItem \(src/todo/components/main.jsx\)](#) is updated with the last item of the list [todos \(src/todo/components/main.jsx\)](#) using [setLastAddedItem\(lastItem\) \(src/todo/components/main.jsx\)](#).

Dependencies:

- The effect is executed whenever:
 - The array [todos \(src/todo/components/main.jsx\)](#) changes (for example, when a new item is added or removed).
 - The value of [lastAddedItem \(src/todo/components/main.jsx\)](#) changes.

Potential problem:

- **Unnecessary dependency:** Including [lastAddedItem \(src/todo/components/main.jsx\)](#) in the dependencies can cause unnecessary loops, since [setLastAddedItem \(src/todo/components/main.jsx\)](#) changes the state, which can trigger the effect again. This can be fixed by removing [lastAddedItem \(src/todo/components/main.jsx\)](#) from the dependencies.

Corrected code:

```
useEffect(() => {  
  if (todos.length > 0) {  
    const lastItem = todos[todos.length - 1];  
    setLastAddedItem(lastItem);  
  }  
}, [todos]); // Remove lastAddedItem from dependencies
```

Summary:

This code keeps the state [lastAddedItem \(src/todo/components/main.jsx\)](#) always updated with the last item of the array [todos \(src/todo/components/main.jsx\)](#).

00:10:58
- 00:11:07

> Response rendering

00:11:54

Participant

the copilot showed 2 ways to do this, the first, do this sorting here.

00:12:53

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

00:13:00

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

00:13:44

Chat (window: 0)

> Agent

github.copilot.default

> Context

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

> Message

Okay, so in the following code, perform alphabetical sorting of visibleTodos using the title criterion of the todo items in the list: "const visibleTodos = useMemo(() => todos.filter((todo) => { if (route === "/active") return !todo.completed; if (route === "/completed") return todo.completed; return todo; })), [todos, route]);"

> Response

To perform alphabetical sorting of [visibleTodos \(src/todo/components/main.jsx\)](#) based on the title attribute of the [todo \(src/todo/components/main.jsx\)](#) items, you can use the `.sort()` method after the filter. Here is the updated code:

```
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 alphabetically
  [todos, route]
);
```

What was added:

- `.sort((a, b) => a.title.localeCompare(b.title)):`
 - Sorts the [todo \(src/todo/components/main.jsx\)](#) items based on the title attribute in alphabetical order.
 - `localeCompare` is used to ensure a language-sensitive string comparison.

Now, the [visibleTodos \(src/todo/components/main.jsx\)](#) list will be filtered and sorted alphabetically by title before being rendered.

00:13:45
- 00:13:48

> Response rendering

00:13:53

Participant

This was alphabetical sorting.

00:14:13
- 00:14:21

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

00:14:34

Participant

Ready, it's in order.

00:14:39

Alphabetical

00:14:42

That's it.

00:14:45

Researcher

OK, I'll send you the next tasks

00:15:28

Participant

00:15:30

Currently, all to-dos disappear when the page is reloaded. Modify the application so that the to-dos remain even after a reload.

00:15:33

00:15:37

Ensure that the state of each to-do is preserved.

00:15:41

An attempt was made to solve the problem, but currently it is not working.

00:16:26

Chat (window: 0)

> Agent

github.copilot.default

> Context

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

> Message

In react, what is the first file to be loaded in the project?

> Response

In a React project, the first file to be loaded is the **main entry file**, usually called [index.js \(src/todo/components/main.jsx\)](#) or [src/todo/components/main.jsx](#), located in the [src/](#) directory. This file is responsible for rendering the root component of the application (usually App) in the DOM.

Typical example of [index.js \(src/todo/components/main.jsx\)](#):

```
import React from "react";
import ReactDOM from "react-dom/client";
import App from "../App";
import "../index.css"; // Global styles

const root = ReactDOM.createRoot(document.getElementById("root"));
root.render(
  <React.StrictMode>
    <App />
  </React.StrictMode>
);
```

Loading flow:

1. The file [index.js \(src/todo/components/main.jsx\)](#) is loaded first.
2. It imports the root component (App) and renders it in the HTML element with the ID root (usually defined in the file [public/index.html](#)).
3. From the App component, other components are loaded as needed.

If your project uses vite or another build tool, the entry file can be configured in the build file (such as vite.config.js or webpack.config.js).

00:16:28

- 00:16:31

> Response rendering

00:16:59

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

00:17:03

> Viewing file (src/index.js)

00:17:13

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

00:17:20

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

00:17:44

Chat (window: 0)

> Agent

github.copilot.default

> Context

selection (src/todo/app.jsx)

> Message

What is this React code doing? "return (<> <Header dispatch={dispatch} /> <Main todos={todos} dispatch={dispatch} /> <Footer todos={todos} dispatch={dispatch} /> </>);"

> Response

This React code is returning the main structure of the [App\(src/todo/app.jsx\)](#) component interface. It uses **JSX** to render three child components: [Header\(src/todo/components/header.jsx\)](#), [Main\(src/todo/components/main.jsx\)](#), and [Footer\(src/todo/components/footer.jsx\)](#). These components are organized within a fragment (`<>...</>`), which is a way to group multiple elements without adding an extra node to the DOM.

What is happening:

1. [Header\(src/todo/components/header.jsx\)](#):

- The [Header\(src/todo/components/header.jsx\)](#) component is rendered and receives the [dispatch\(src/todo/app.jsx\)](#) function as a **prop**.
- Probably, the [Header\(src/todo/components/header.jsx\)](#) uses this function to trigger actions related to the global state of the "todos" (tasks).

2. [Main\(src/todo/components/main.jsx\)](#):

- The [Main\(src/todo/components/main.jsx\)](#) component is rendered and receives two **props**:
 - [todos\(src/todo/app.jsx\)](#): The current state of the task list.
 - [dispatch\(src/todo/app.jsx\)](#): The function to manage the state of the tasks.
- The [Main\(src/todo/components/main.jsx\)](#) probably displays the task list and allows interactions with it.

3. [Footer\(src/todo/components/footer.jsx\)](#):

- The [Footer\(src/todo/components/footer.jsx\)](#) component is rendered and also receives the **props** [todos\(src/todo/app.jsx\)](#) and [dispatch\(src/todo/app.jsx\)](#).
- It may be responsible for displaying additional information, such as the number of remaining items or filtering options.

Context:

- **State managed with [useReducer\(node_modules/@types/react/index.d.ts\)](#):**
 - The state [todos\(src/todo/app.jsx\)](#) and the [dispatch\(src/todo/app.jsx\)](#) function are managed by the [useReducer\(node_modules/@types/react/index.d.ts\)](#) hook, which uses the [todoReducer\(src/todo/reducer.js\)](#) to update the state based on the triggered actions.
- **Fragment (`<>...</>`):**
 - The fragment is used to avoid adding an extra element in the DOM, like a `<div>`.

Summary:

This code organizes the application into three main parts ([Header\(src/todo/components/header.jsx\)](#), [Main\(src/todo/components/main.jsx\)](#), [Footer\(src/todo/components/footer.jsx\)](#)), passing the state and the state management function ([dispatch\(src/todo/app.jsx\)](#)) to the child components, allowing them to interact with the task list ([todos\(src/todo/app.jsx\)](#)).

> **Response rendering**

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

Chat (window: 0)

> **Agent**

00:17:51
- 00:18:01

00:18:20

00:19:40

github.copilot.default

> Context

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

> Message

Is the following React code just returning the HTML code for page display, or is it altering any object in the code, like `visibleTodos`? "return (<main className="main" data-testid="main"> {visibleTodos.length > 0 ? (<> <div className="toggle-all-container"> <input className="toggle-all" type="checkbox" id="toggle-all" data-testid="toggle-all" checked={visibleTodos.every((todo) => todo.completed)} onChange={toggleAll} /> <label className="toggle-all-label" htmlFor="toggle-all"> Toggle All Input </label> </div> <ul className={classnames("todo-list")} data-testid="todo-list"> {visibleTodos.map((todo, index) => (<Item todo={todo} key={todo.id} dispatch={dispatch} index={index} />))} </>) : (<div className="todo-list-empty-container"> No tasks here yet </div>) } <paginationControls currentPage={currentPage} itemsPerPage={itemsPerPage} setCurrentPage={setCurrentPage} totalItems={visibleTodos.length} /> </main>);"

> Response

The provided code **does not directly alter any object or state, like `visibleTodos` ([src/todo/components/main.jsx](#))**. It only returns the JSX to render the page interface based on the current state and provided properties.

What is happening:

1. [visibleTodos \(src/todo/components/main.jsx\)](#):

- It is a derived value, previously calculated using [useMemo \(node_modules/@types/react/index.d.ts\)](#). It is not altered within the return. The code only uses it to render the list of items (with <Item>).

2. Interactivity:

- The code includes interactive elements, like the checkbox <input>, which triggers the function [toggleAll \(src/todo/components/main.jsx\)](#) when changed. This function can alter the global state via [dispatch \(src/todo/components/main.jsx\)](#), but this occurs outside the scope of the return.

3. Conditional rendering:

- If [visibleTodos \(src/todo/components/main.jsx\)](#) has items ([visibleTodos.length > 0 \(src/todo/components/main.jsx\)](#)), it renders the task list.
- Otherwise, it displays a message indicating there are no tasks.

4. [paginationControls \(src/todo/components/main.jsx\)](#):

- This component is rendered based on the current state ([currentPage \(src/todo/components/main.jsx\)](#), [itemsPerPage \(src/todo/components/main.jsx\)](#), etc.), but it also does not directly alter any state or object at the time of rendering.

Summary:

The code in the return is only **returning the JSX for page display**. It does not directly alter [visibleTodos \(src/todo/components/main.jsx\)](#) or any other state/object. Changes in state or objects would occur in functions like [toggleAll \(src/todo/components/main.jsx\)](#) or in other events triggered by interactive components.

> Response rendering

00:19:42
- 00:19:47

00:20:39

Participant

And I'm thinking of the easiest way

00:21:02

No. Probably when a change is made to some file,

00:21:08

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

00:21:08

Participant

probably the app is not updated itself.

00:21:15

So, there could be some way to create a

00:21:18

variable.

00:21:33

The loading.

00:21:37

of the data that is saved here, but it's the logic that I think might

00:21:44

work..

00:21:51

Researcher

OK.

00:22:46

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

00:23:32

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

00:23:46

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

00:24:21

Chat (window: 0)

> **Agent**

github.copilot.default

> **Context**

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

> **Message**

In react, are parameters passed by value or by reference?

> **Response**

In React, the passing of parameters (or **props**) to components is done by **value** or **reference**, depending on the type of data being passed:

1. Primitive types (numbers, strings, booleans, etc.):

- Are passed by **value**.
- This means a copy of the value is passed to the child component, and changes to the value within the child component do not affect the original value in the parent component.

2. Objects, arrays, and functions:

- Are passed by **reference**.
- This means the child component receives a reference to the same object, array, or function. Changes made directly to these data (mutation) in the child component can affect the state in the parent component.

Example:

```
function Parent() {  
  const obj = { key: "value" };  
  const num = 42;  
  
  return <Child obj={obj} num={num} />;  
}  
  
function Child({ obj, num }) {  
  console.log(obj); // Reference to the same object  
  console.log(num); // Copy of the value  
}
```

Important:

Although objects and arrays are passed by reference, it is good practice in React to avoid direct mutations and work with **immutability** to ensure predictability and avoid unwanted side effects. This is especially important when dealing with state.

00:24:23
- 00:24:27

> Response rendering

00:24:33

Participant

The components are made by variable or reference

00:24:41

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

00:25:10

Chat (window: 0)

> Agent

github.copilot.default

> Context

viewport (src/todo/app.jsx)

> Message

In the following case, is "todos" a primitive type? "return (<main className="main" data-testid="main"> { visibleTodos.length > 0 ? (<> <div className="toggle-all-container"> <input className="toggle-all" type="checkbox" id="toggle-all" data-testid="toggle-all" checked={visibleTodos.every((todo) => todo.completed)} onChange={toggleAll} /> <label className="toggle-all-label" htmlFor="toggle-all"> Toggle All Input </label> </div> <ul className={classnames("todo-list")}> { visibleTodos.map((todo, index) => (<Item todo={todo} key={todo.id} dispatch={dispatch} index={index} />))} </>) : (<div className="todo-list-empty-container"> No tasks here yet </div>) } <paginationControls currentPage={currentPage} itemsPerPage={itemsPerPage} setCurrentPage={setCurrentPage} totalItems={visibleTodos.length} /> </main>);"

> Response

No, [todos \(src/todo/app.jsx\)](#) is not a primitive type.

In the provided code, [todos \(src/todo/app.jsx\)](#) is an **array** that contains objects representing the to-do list items (todos). Arrays in JavaScript are complex data types (non-primitive) and are passed by **reference** in React.

Context:

- [todos \(src/todo/app.jsx\)](#) is managed by the hook [useReducer \(node_modules/@types/react/index.d.ts\)](#) in the component [App \(src/todo/app.jsx\)](#):

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

Here, [todos \(src/todo/app.jsx\)](#) is initialized as an empty array ([]) and is updated by the [todoReducer \(src/todo/reducer.js\)](#).

- This array is passed as a **prop** to the component [Main \(src/todo/components/main.jsx\)](#):

```
<Main todos={todos} dispatch={dispatch} />
```

- Within the component [Main \(src/todo/components/main.jsx\)](#), [todos \(src/todo/app.jsx\)](#) is used to calculate visibleTodos:

```
const visibleTodos = useMemo(() => todos.filter(...), [todos, route]);
```

Summary:

[todos \(src/todo/app.jsx\)](#) is an array (complex type) and, therefore, is not a primitive type. It is passed by reference to child components, such as [Main \(src/todo/components/main.jsx\)](#).

00:25:12
- 00:25:21
00:25:43

> Response rendering

Chat (window: 0)

> Agent

github.copilot.default

> Context

viewport (src/todo/app.jsx)

> Message

In the following case, is "todos" a primitive type? "const todos: any"

> Response

In the case of [const todos: any \(src/todo/app.jsx\)](#), [todos \(src/todo/app.jsx\)](#) is **not necessarily a primitive type**, because the any type in TypeScript is a generic type that can represent **any value** — whether it is primitive (like number, string, boolean) or complex (like array, object, function).

Context:

- If [todos \(src/todo/app.jsx\)](#) is assigned a primitive value, like a number or string, it will be a primitive type.
- If [todos \(src/todo/app.jsx\)](#) is assigned an array or object (as in your project's code), it will be a complex type.

In your current code, [todos \(src/todo/app.jsx\)](#) is an **array** initialized as [] (an empty array) and managed by [useReducer \(node_modules/@types/react/index.d.ts\)](#). Therefore, [todos \(src/todo/app.jsx\)](#) **is not a primitive type**, but rather a complex type (array).

Summary:

Although the any type allows [todos \(src/todo/app.jsx\)](#) to be anything, in the context of your code, [todos \(src/todo/app.jsx\)](#) is an **array** and, therefore, **is not a primitive type**.

00:25:45
- 00:25:51
00:26:37

> Response rendering

Chat (window: 0)

> Agent

github.copilot.default

> Context

selection (src/todo/app.jsx)

> Message

So, everything is passed by reference to main, correct? "const [todos, dispatch] = useReducer(todoReducer, []); return (<> <Header dispatch={dispatch} /> <Main todos={todos} dispatch={dispatch} /> <Footer todos={todos} dispatch={dispatch} /> </>); "

> Response

Yes, [todos \(src/todo/app.jsx\)](#) is **passed by reference** to the component [Main \(src/todo/components/main.jsx\)](#) (and also to [Footer \(src/todo/components/footer.jsx\)](#)).

In the code, [todos \(src/todo/app.jsx\)](#) is an **array** managed by [useReducer \(node_modules/@types/react/index.d.ts\)](#), which is a complex type in JavaScript. When you pass [todos \(src/todo/app.jsx\)](#) as a prop to the component [Main \(src/todo/components/main.jsx\)](#) (or any other component), it is passed by reference, as arrays and objects in JavaScript are always passed this way.

Explanation:

- [todos \(src/todo/app.jsx\)](#) is an **array**:

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

Arrays are complex types and, when passed as props, React passes the reference to the array, not a copy.

- **Passing to [Main \(src/todo/components/main.jsx\)](#):**

```
<Main todos={todos} dispatch={dispatch} />
```

Here, the component [Main \(src/todo/components/main.jsx\)](#) receives the reference to the same array [todos \(src/todo/app.jsx\)](#). If the array is modified directly (which is not recommended), the modification will be reflected in all places that use this reference.

Summary:

Yes, [todos \(src/todo/app.jsx\)](#) is passed by reference to the component [Main \(src/todo/components/main.jsx\)](#) (and also to [Footer \(src/todo/components/footer.jsx\)](#)). This means that the component [Main \(src/todo/components/main.jsx\)](#) has access to the same instance of the array [todos \(src/todo/app.jsx\)](#) managed by [useReducer \(node_modules/@types/react/index.d.ts\)](#).

00:26:38
- 00:26:44
00:27:54

> Response rendering

Chat (window: 0)

> Agent

github.copilot.default

> Context

selection (src/todo/app.jsx)

> Message

Right. When reloading the page, is there any default file in react that is not "updated" in order to retain the state of objects/variables?

> Response

No, by default, when reloading the page in a React application, **the entire application state is lost**, because the browser reloads the main HTML file and restarts the JavaScript, including React. This means that the state managed by hooks like `useState` or [useReducer \(node_modules/@types/react/index.d.ts\)](#) will be reset to their initial values.

How to preserve the state when reloading the page?

If you want the state of objects/variables to be maintained even after reloading the page, you need to use some **persistence** mechanism. Here are some common options:

1. Local Storage / Session Storage:

- You can save the state in `localStorage` or `sessionStorage` and restore it when loading the application.

```
useEffect(() => {
  const savedTodos =
    JSON.parse(localStorage.getItem("todos")) || [];
  dispatch({ type: "SET_TODOs", payload: savedTodos });
}, []);

useEffect(() => {
  localStorage.setItem("todos", JSON.stringify(todos));
}, [todos]);
```

2. IndexedDB:

- For more complex data, you can use IndexedDB, which is a database solution in the browser.

3. Backend (API):

- Save the state on the server and restore it when reloading the page.

4. State Management with Persistence (Redux Persist, Zustand, etc.):

- Tools like `redux-persist` can be used to automatically save the state in local storage and restore it.

5. Service Workers:

- For PWA applications, you can use service workers to store data and maintain the state offline.

Summary:

By default, React does not maintain the state when reloading the page. To preserve the state, you need to implement a persistence solution, such as `localStorage`, `sessionStorage`, or save the data in a backend.

00:27:57
- 00:28:02

> Response rendering

00:28:00

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

00:29:10

Researcher

You can search the web if you prefer, or if you think the copilot's answers are not helping you.

00:29:14

Participant

OK

00:29:57

Chat (window: 0)

> Agent

github.copilot.default

> Context

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

> Message

Okay, then modify the code in main.jsx to add the use of local storage to store the state of the items in case the page is reloaded: "export function Main({ todos, dispatch }) { const { pathname: route } = useLocation(); const [currentPage, setCurrentPage] = useState(1); const [lastAddedItem, setLastAddedItem] = useState(null); const itemsPerPage = 5; 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 alphabetically [todos, route]); const toggleAll = useCallback((e) => dispatch({ type: TOGGLE_ALL, payload: { completed: e.target.checked } }), [dispatch]); useEffect(() => { if (todos.length > 0) { const lastItem = todos[todos.length - 1]; setLastAddedItem(lastItem); } }, [todos, lastAddedItem]); return (<main className="main" data-testid="main"> {visibleTodos.length > 0 ? (<> <div className="toggle-all-container"> <input className="toggle-all" type="checkbox" id="toggle-all" data-testid="toggle-all" checked={visibleTodos.every((todo) => todo.completed)} onChange={toggleAll} /> <label className="toggle-all-label" htmlFor="toggle-all"> Toggle All Input </label> </div> <ul className={classnames("todo-list")} data-testid="todo-list"> {visibleTodos.map((todo, index) => (<Item todo={todo} key={todo.id} dispatch={dispatch} index={index} />))} </>) : (<div className="todo-list-empty-container"> No tasks here yet </div>) } <paginationControls currentPage={currentPage} itemsPerPage={itemsPerPage} setCurrentPage={setCurrentPage} totalItems={visibleTodos.length} /> </main>); }"

> Response

Here is the modified code to include the use of **Local Storage** to store and restore the state of the items ([todos \(src/todo/components/main.jsx\)](#)) in case the page is reloaded:

```
File: src/todo/components/main.jsx
export function Main({ todos, dispatch }) {
  const { pathname: route } = useLocation();
  const [currentPage, setCurrentPage] = useState(1);
  const [lastAddedItem, setLastAddedItem] = useState(null);

  const itemsPerPage = 5;

  // Restore the state of todos from Local Storage when the page
  loads
  useEffect(() => {
    const savedTodos = JSON.parse(localStorage.getItem("todos")) ||
    [];
    if (savedTodos.length > 0) {
      dispatch({ type: "SET_TODO", payload: savedTodos }); //
  Ensure the reducer supports this action
    }
  }, [dispatch]);

  // Save the state of todos in Local Storage whenever they are
  updated
  useEffect(() => {
    localStorage.setItem("todos", JSON.stringify(todos));
  }, [todos]);

  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 alphabetically
        [todos, route]
    );

    const toggleAll = useCallback(
        (e) => dispatch({ type: TOGGLE_ALL, payload: { completed:
e.target.checked } }),
        [dispatch]
    );

    useEffect(() => {
        if (todos.length > 0) {
            const lastItem = todos[todos.length - 1];
            setLastAddedItem(lastItem);
        }
    }, [todos, lastAddedItem]);

    return (
        <main className="main" data-testid="main">
            {visibleTodos.length > 0 ? (
                <>
                    <div className="toggle-all-container">
                        <input
                            className="toggle-all"
                            type="checkbox"
                            id="toggle-all"
                            data-testid="toggle-all"
                            checked={visibleTodos.every((todo) =>
todo.completed)}
                            onChange={toggleAll}
                        />
                        <label className="toggle-all-label"
htmlFor="toggle-all">
                            Toggle All Input
                        </label>
                    </div>
                    <ul className={classnames("todo-list")} data-
testid="todo-list">
                        {visibleTodos.map((todo, index) => (
                            <Item todo={todo} key={todo.id} dispatch=
{dispatch} index={index} />
                        ))}
                    </ul>
                </>
            ) : (
                <div className="todo-list-empty-container">
                    <span className="todo-list-empty-message">No tasks
here yet</span>
                </div>
            )}

            <paginationControls
                currentPage={currentPage}
                itemsPerPage={itemsPerPage}
                setCurrentPage={setCurrentPage}
            />
        </main>
    );

```

```

        totalItems={visibleTodos.length}
      />
    </main>
  );
}

```

What was added:

1. Restore the state of [todos \(src/todo/components/main.jsx\)](#) from Local Storage:
 - o In the first [useEffect \(node_modules/@types/react/index.d.ts\)](#), the items saved in localStorage are loaded and sent to the [dispatch \(src/todo/components/main.jsx\)](#) with a custom action (SET_TODOS).
2. Save the state of [todos \(src/todo/components/main.jsx\)](#) in Local Storage:
 - o In the second [useEffect \(node_modules/@types/react/index.d.ts\)](#), whenever the state of [todos \(src/todo/components/main.jsx\)](#) is updated, it is saved in localStorage.

Note:

- Ensure that the todoReducer supports the SET_TODOS action to update the state with the restored items:

```

function todoReducer(state, action) {
  switch (action.type) {
    case "SET_TODOS":
      return action.payload;
    // Other actions...
    default:
      return state;
  }
}

```

With these changes, the state of the items will be preserved even after the page is reloaded.

00:29:59
- 00:30:23

> Response rendering

00:31:18
- 00:31:50

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

00:32:40

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

00:33:08

Chat (window: 0)

> Agent

github.copilot.default

> Context

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

> Message

What does this part of the code do? "const toggleAll = useCallback((e) => dispatch({ type: TOGGLE_ALL, payload: { completed: e.target.checked } }), [dispatch]);
useEffect(() => { if (todos.length > 0) { const lastItem = todos[todos.length - 1];
setLastAddedItem(lastItem); } }, [todos, lastAddedItem]);"

> Response

Code Explanation:

1. [toggleAll \(src/todo/components/main.jsx\)](#) with [useCallback \(node_modules/@types/react/index.d.ts\)](#):

```
const toggleAll = useCallback(
  (e) => dispatch({ type: TOGGLE_ALL, payload: { completed:
e.target.checked } }),
  [dispatch]
);
```

- **What it does:**

- Defines a function called [toggleAll \(src/todo/components/main.jsx\)](#) that is used to mark all items ([todos \(src/todo/components/main.jsx\)](#)) as complete or incomplete.
- This function triggers an action ([TOGGLE_ALL \(src/todo/constants.js\)](#)) using the [dispatch \(src/todo/components/main.jsx\)](#) from [useReducer](#), passing a [payload \(src/todo/components/main.jsx\)](#) with the value of [e.target.checked \(src/todo/components/main.jsx\)](#) (the state of the checkbox).
- If the checkbox is checked (true), all items will be marked as complete. Otherwise, all will be unchecked.

- **Why use [useCallback \(node_modules/@types/react/index.d.ts\)](#):**

- The [useCallback \(node_modules/@types/react/index.d.ts\)](#) memorizes the function [toggleAll \(src/todo/components/main.jsx\)](#) to prevent it from being recreated on each render, unless the dependency ([dispatch \(src/todo/components/main.jsx\)](#)) changes. This improves performance, especially if the component is rendered frequently.

2. [useEffect \(node_modules/@types/react/index.d.ts\)](#) to update [lastAddedItem \(src/todo/components/main.jsx\)](#):

```
useEffect(() => {
  if (todos.length > 0) {
    const lastItem = todos[todos.length - 1];
    setLastAddedItem(lastItem);
  }
}, [todos, lastAddedItem]);
```

- **What it does:**

- Monitors changes in the array [todos \(src/todo/components/main.jsx\)](#) and the state [lastAddedItem \(src/todo/components/main.jsx\)](#).
- Whenever [todos \(src/todo/components/main.jsx\)](#) is updated (e.g., when a new item is added), it takes the last item of the array ([todos\[todos.length - 1\] \(src/todo/components/main.jsx\)](#)) and stores it in the state [lastAddedItem \(src/todo/components/main.jsx\)](#) using [setLastAddedItem \(src/todo/components/main.jsx\)](#).

- **Why use [useEffect \(node_modules/@types/react/index.d.ts\)](#):**

- The [useEffect \(node_modules/@types/react/index.d.ts\)](#) is used to execute this side effect (update [lastAddedItem \(src/todo/components/main.jsx\)](#)) whenever the array [todos \(src/todo/components/main.jsx\)](#) or the state [lastAddedItem \(src/todo/components/main.jsx\)](#) changes.

- **Possible issue:**

- Including [lastAddedItem \(src/todo/components/main.jsx\)](#) as a dependency may cause unnecessary loops, as [setLastAddedItem \(src/todo/components/main.jsx\)](#) changes the state, which can trigger the effect again. This can be fixed by removing [lastAddedItem \(src/todo/components/main.jsx\)](#) from the dependencies:

```
useEffect(() => {
  if (todos.length > 0) {
    const lastItem = todos[todos.length - 1];
    setLastAddedItem(lastItem);
  }
}, [todos]); // Remove lastAddedItem from dependencies
```

Summary:

1. [toggleAll \(src/todo/components/main.jsx\)](#):

- Marks or unmarks all items ([todos \(src/todo/components/main.jsx\)](#)) as complete/incomplete based on the state of the checkbox.

2. [useEffect \(node_modules/@types/react/index.d.ts\)](#) for [lastAddedItem \(src/todo/components/main.jsx\)](#):

- Updates the state [lastAddedItem \(src/todo/components/main.jsx\)](#) with the last item of the array [todos \(src/todo/components/main.jsx\)](#) whenever it changes.

These parts of the code help manage the state and interaction with the list items efficiently.

> Response rendering

Chat (window: 0)

> Agent

github.copilot.default

> Context

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

> Message

Is the order in which a useEffect is present in the code of a function relevant, or are actions triggered according to the presence of some event?

> Response

The order in which the [useEffect \(node_modules/@types/react/index.d.ts\)](#) are present in the code **is not relevant** for their execution, as they are triggered based on the **dependencies** specified in the second argument (the dependency array). Each [useEffect \(node_modules/@types/react/index.d.ts\)](#) is independent and will execute when its dependencies change or at the appropriate moment in the component's lifecycle.

How [useEffect \(node_modules/@types/react/index.d.ts\)](#) works:

1. Initial execution:

00:33:12
- 00:33:27

00:34:32

- All [useEffect \(node_modules/@types/react/index.d.ts\)](#) are executed after the initial rendering of the component, as long as their dependencies are satisfied.

2. Subsequent execution:

- Each [useEffect \(node_modules/@types/react/index.d.ts\)](#) is re-executed whenever one or more of its dependencies change.

3. Independence:

- Each [useEffect \(node_modules/@types/react/index.d.ts\)](#) is treated independently. The order in the code does not affect its execution, as long as the dependencies are correctly set.

Example:

```
useEffect(() => {
  console.log("Effect 1");
}, [dependency1]);

useEffect(() => {
  console.log("Effect 2");
}, [dependency2]);
```

- **Effect 1** will be executed when dependency1 changes.
- **Effect 2** will be executed when dependency2 changes.
- The order in the code does not matter; each effect will be triggered according to its dependencies.

When the order might be relevant:

Although the execution of [useEffect \(node_modules/@types/react/index.d.ts\)](#) is independent, the order might be relevant in cases where one effect implicitly depends on another. For example: - If a [useEffect \(node_modules/@types/react/index.d.ts\)](#) modifies a state that is used as a dependency in another [useEffect \(node_modules/@types/react/index.d.ts\)](#), the logical order in the code can help with readability and avoid confusion.

Summary:

The order of [useEffect \(node_modules/@types/react/index.d.ts\)](#) in the code **is not relevant for their execution**, as they are triggered based on dependencies. However, to keep the code more readable and organized, it is good practice to place them in a logical order that reflects the flow of dependencies or expected actions.

00:34:35
- 00:34:40

> Response rendering

00:37:17

> Viewing file (src/todo/index.html) through file search

00:38:49

Chat (window: 0)

> Agent

github.copilot.default

> Context

selection (public/index.html)

> Message

The code to restore and save the state of the todos in local storage was added to the main.jsx file, however, when adding some todos in the web interface and reloading the page, it does not show the item listing page. It loads the message from a code in the index.html file. Should the local storage functionality store the data persistently, but do I also need to store the loaded page?

> Response

Yes, to ensure the loaded page (current route) is restored after reloading, you can also store the **current route** in localStorage and restore it when loading the application. This will ensure that, in addition to the items (todos), the interface is displayed on the same page (route) where the user was before reloading.

Solution:

1. Store the current route in localStorage:

- Save it in localStorage whenever the route changes.

2. Restore the route when loading the application:

- Before rendering the component, check if there is a saved route in localStorage and redirect the user to it.

Modification in main.jsx:

Add a useEffect to monitor the current route and save it in localStorage:

```
File: src/todo/components/main.jsx
import { useNavigate } from "react-router-dom";

export function Main({ todos, dispatch }) {
  const { pathname: route } = useLocation();
  const navigate = useNavigate(); // To redirect the user
  const [currentPage, setCurrentPage] = useState(1);
  const [lastAddedItem, setLastAddedItem] = useState(null);

  const itemsPerPage = 5;

  // Restore the saved route in Local Storage when loading the page
  useEffect(() => {
    const savedRoute = localStorage.getItem("currentRoute");
    if (savedRoute && savedRoute !== route) {
      navigate(savedRoute); // Redirects to the saved route
    }
  }, [navigate, route]);

  // Save the current route in Local Storage whenever it changes
  useEffect(() => {
    localStorage.setItem("currentRoute", route);
  }, [route]);

  // Restore the state of the todos from Local Storage when loading
  // the page
  useEffect(() => {
    const savedTodos = JSON.parse(localStorage.getItem("todos")) ||
    [];
    if (savedTodos.length > 0) {
      dispatch({ type: "SET_TODOS", payload: savedTodos }); //
    }
  }, [dispatch]);

  // Save the state of the todos in Local Storage whenever they are
```



```

updated
  useEffect(() => {
    localStorage.setItem("todos", JSON.stringify(todos));
  }, [todos]);

  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)), //
    [todos, route]
  );

  const toggleAll = useCallback(
    (e) => dispatch({ type: TOGGLE_ALL, payload: { completed:
e.target.checked } }),
    [dispatch]
  );

  useEffect(() => {
    if (todos.length > 0) {
      const lastItem = todos[todos.length - 1];
      setLastAddedItem(lastItem);
    }
  }, [todos, lastAddedItem]);

  return (
    <main className="main" data-testid="main">
      {visibleTodos.length > 0 ? (
        <>
          <div className="toggle-all-container">
            <input
              className="toggle-all"
              type="checkbox"
              id="toggle-all"
              data-testid="toggle-all"
              checked={visibleTodos.every((todo) =>
todo.completed)}
              onChange={toggleAll}
            />
            <label className="toggle-all-label"

```

```

htmlFor="toggle-all">
    Toggle All Input
  </label>
</div>
<ul className={classnames("todo-list")} data-
testid="todo-list">
    {visibleTodos.map((todo, index) => (
      <Item todo={todo} key={todo.id} dispatch=
{dispatch} index={index} />
    ))}
  </ul>
</>
) : (
  <div className="todo-list-empty-container">
    <span className="todo-list-empty-message">No tasks
here yet</span>
  </div>
)

  <paginationControls
    currentPage={currentPage}
    itemsPerPage={itemsPerPage}
    setCurrentPage={setCurrentPage}
    totalItems={visibleTodos.length}
  />
</main>
);
}

```

What was added:

1. Restore the saved route:

- In the first useEffect, the route saved in localStorage is restored using the navigate from React Router.

2. Save the current route:

- In the second useEffect, the current route (route) is saved in localStorage whenever it changes.

Result:

Now, when reloading the page: - The current route will be restored. - The items (todos) will be loaded from localStorage. - The interface will be correctly displayed on the same page (route) where the user was before reloading.

> Response rendering

00:38:51
- 00:39:08

00:40:28

Researcher

40 minutes have passed. I'll ask you to stop and answer the post-study questionnaire OK.

00:41:07

Participant

If I have doubts about what I need to do. I have to think very well about the question for the AI. I have doubts if the AI will

00:41:12

redirect me to very bad answers

00:41:16

So I try to be very descriptive and very verbose in the questions.

00:41:19

The example now, the local storage problem. I didn't know this term.

00:41:24

The App is persisting the data.

00:41:31

I thought the app was saving a json in some temporary file.

00:41:36

I thought of this solution.

00:41:44

Researcher

00:41:54

ok

00:42:23

Participant

I think it's excellent to give a very descriptive answer.

00:42:39

About the details. I think the more information the better,

00:42:42

so, excellent use of technical terms. Excellent the issue of local storage,

00:42:52

Researcher

OK.

00:42:53

Participant

Correction issue.

00:42:58

It's neutral, I believe the reason is because I couldn't write the problem for the AI.

00:43:18

Follow the instructions, I think it's probably excellent

00:43:41

Perform tasks more quickly for sure.

00:44:43

Would it be useful in my work or study? Certainly very useful.

00:45:40

Were the tasks physically demanding?

00:45:48

Physically, not so much

00:45:58

How rushed, accelerated the pace of the session,

00:46:15

So, it wasn't that successful, when you had to strive to

00:46:21

reach your performance level.

00:46:25

I thought a lot of time to ask the questions,

00:46:29

precisely because the quality of the input is relevant. For the copilot,

00:46:56

156.

00:47:43

Researcher

OK, Now I'm going to do some more,

00:47:46

Questions

00:47:51

Participant

HOK

00:47:51

Researcher

were you able to orient yourself with the code? and

00:47:55

how did the copilot influence that?

00:48:00

Participant

I think a lot. Intuitively, there was a file called main.JSX.

00:48:06

I thought, this must be the first file to be

00:48:09

loaded, I asked the copilot. it said there is no index file and the

00:48:15

app JS are the main ones normally.

00:48:20

So, the question of familiarization.

00:48:24

was good, but from the moment you mess with it

00:48:27

for 5, 10 minutes, I think you start to familiarize yourself.

00:48:32

syntax is not very friendly.

00:48:34

In my opinion, I think it's a matter of practice.

00:48:38

I think the copilot helped

00:48:41

For sure, a lot in relation to that

00:48:51

Your code is well structured.

00:48:58

Researcher

OK.

00:49:18

Cool, do you feel that you understood the basis of the

00:49:21

code and how the copilot influenced your understanding.

00:49:25

Participant

Totally.

00:49:26

I wouldn't say I understood.

00:49:33

Researcher

ok.

00:49:33

Participant

The parts I managed to understand were 90% because of the copilot

00:49:40

Researcher

I see. And how would you have solved these

00:49:44

issues without the help of the COPILOT?

00:49:50

Participant

00:49:57 I would try to follow the same logic. The problem is that I would have to spend more
00:50:05 time doing research in the sense that, for example, the persistence problem.
00:50:08 I would have to do more research to
00:50:13 identify the right term for me to be able to find
00:50:18 some post, site, that directs me to
00:50:32 this solution. I believe that's one of the advantages of having an AI model
Web searches need to have exact terms, AI with natural language processing can help with
that.
00:50:46 so there would be this.
00:50:47 Extra work.
00:51:04 **Researcher**
Got it.
00:51:04 And can you explain the aspects you liked most about the copilot and the least?
00:51:27 **Participant**
The functionality of writing code helps a lot, but no,
00:51:33 I wouldn't even say exactly that this is exactly its best attribute.
00:51:40 I think the whole context issue makes a big difference.
00:51:47 Sometimes you go to search for a solution,
00:51:52 you can find it, but the context issue makes a big
00:51:57 difference.
00:52:02 A disadvantage, I would say not only of the copilot, but also ChatGPT is the programmer
becoming dependent on AI, due to the ease with which it delivers good answers.
00:53:13 **Researcher**
During your interaction with the copilot
00:53:16 did you change the way. Or
00:53:20 Did you always try to follow the same line
00:53:26 **Participant**
In what sense?
00:53:28 **Researcher**
The approach of asking questions, style of prompts.
00:53:40 **Participant**
I always tried to be very consistent, think
00:53:44 a lot about the question and always giving the context.
00:54:20 **Researcher**
Ok, thank you very much for participating in our study.
00:55:05 See you later.