

Participant P17

00:02:40 **Participant**
I have to change the color, so I will first check where the component is.

00:02:44 > **Viewing file** (public/index.html)

00:03:28 > **Viewing file** (src/todo/app.css)

00:03:36 **Participant**
I will try first, so now just identify where is
00:03:40 the part that changes the color of the box. Currently, it is red.
00:03:51 I'm just trying to check what is the correct configuration that.
00:03:56 Changes this color.

00:03:59 > **Viewing file** (src/todo/app.jsx)

00:04:04 > **Viewing file** (src/todo/constants.js)

00:04:04 > **Viewing file** (src/index.js)

00:04:06 > **Viewing file** (src/todo/reducer.js)

00:04:14 > **Viewing file** (src/todo/constants.js)

00:04:16 > **Viewing file** (src/todo/app.jsx)

00:04:29 > **Viewing file** (src/todo/components/footer.jsx)

00:04:34 > **Viewing file** (src/todo/components/header.jsx)

00:04:49 > **Viewing file** (src/todo/components/main.jsx)

00:05:01 > **Viewing file** (src/todo/components/header.jsx)

00:05:12 > **Viewing file** (src/todo/components/input.jsx)

00:05:22 > **Viewing file** (src/todo/components/item.jsx)

00:05:33 > **Viewing file** (src/todo/components/paginationControls.jsx)

00:05:37 > **Viewing file** (src/todo/app.css)

00:05:40 > **Viewing file** (src/todo/app.jsx)

00:05:45 > **Viewing file** (src/todo/components/header.jsx)

00:05:57 > **Viewing file** (src/todo/components/input.jsx)

00:06:12 > **Viewing file** (src/todo/components/main.jsx)

00:06:13 **Participant**
I tried to find the part that defines the text.
00:06:18 That shows the message.

00:06:27 **Researcher**
OK.
00:06:30 You can interact with the copilot, whenever you want.

00:07:05 > **Viewing file** (src/todo/app.css)

00:07:20 > **Viewing file** (src/todo/components/footer.jsx)

00:07:28 > **Viewing file** (src/todo/components/input.jsx)

00:07:30 > **Viewing file** (src/todo/components/item.jsx)

00:07:34 > **Viewing file** (src/todo/components/main.jsx)

00:07:41 **Participant**
The message I found here.
00:07:46 Then I identified that here in the CSS there is the class name being used to define
00:07:54 the color.
00:07:58 of the background box to blue.

00:08:02 > **Viewing file** (src/todo/app.css)

00:08:02 **Researcher**
ok.
00:08:06 > **Editing file** (src/todo/app.css)
- 00:08:30

00:08:35 **Participant**
Can I run the app?

00:08:39 **Researcher**
Yes, you can run the application. It is functional

00:08:50 > **Viewing file** (package.json)

00:09:12 > **Viewing file** (package-lock.json)

00:09:31 **Participant**
OK
it's blue, so the task is completed.

00:09:41 **Researcher**
OK, you can move on to another.

00:09:56 > **Viewing file** (src/todo/components/main.jsx)

00:10:01 > **Viewing file** (src/todo/components/input.jsx)

00:10:07 > **Viewing file** (src/todo/components/footer.jsx)

00:10:18 > **Editing file** (src/todo/components/header.jsx)
- 00:10:30

00:10:30 **Participant**
It's good. Task b is to change the place holder of the input field.
like in the first activity,
I had looked at the files, then I managed.
Now identify faster where it is necessary to make the change.
So now I'll just.
Check if it was really updated.
Sort tasks?

00:11:05 > **Viewing file** (src/todo/components/main.jsx)

00:11:23 > **Viewing file** (public/index.html)

00:11:42 > **Viewing file** (src/todo/app.css)

00:11:46 > **Viewing file** (src/todo/components/main.jsx)

00:11:53 > **Viewing file** (src/todo/components/item.jsx)

00:12:44 > **Viewing file** (src/todo/components/item.jsx)

00:13:31 **Participant**
Are the tasks being saved?

00:13:31 **Researcher**
Not at the moment.

00:13:59 > **Viewing file** (src/todo/components/item.jsx)

00:14:14 > **Viewing file** (src/todo/components/main.jsx)

00:14:18 > **Viewing file** (src/todo/components/header.jsx)

00:14:23 > **Viewing file** (src/todo/components/item.jsx)

00:14:26 > **Viewing file** (src/todo/components/header.jsx)

00:14:36 > **Viewing file** (src/todo/constants.js)

00:14:39 > **Viewing file** (src/todo/components/footer.jsx)

00:14:40 > **Viewing file** (src/todo/components/main.jsx)

00:15:38 **Researcher**
OK.

00:15:42 > **Viewing file** (src/todo/components/main.jsx)

00:16:30 > **Viewing file** (src/todo/app.jsx)

00:16:37 > **Viewing file** (src/todo/reducer.js)

00:17:10 **Participant**
A question. Do I really need to use the copilot?

00:17:24 **Researcher**
If you are not comfortable using the copilot. You can search the internet.

00:17:32 **Participant**

OK.

00:18:20

It's good, I was trying until now.

00:18:25

To understand how the code works.

00:18:35

but I don't know how it would be. The parameters to perform the sorting.

00:19:18

Chat (window: 0)

> **Agent**

github.copilot.workspace

> **Context**

workspace, viewport (src/todo/reducer.js)

> **Message**

@workspace How can I sort the results of the to-do items so that they are displayed in alphabetical order?

> **Response**

You can modify the [todoReducer \(src/todo/reducer.js\)](#) to sort the items in alphabetical order after each action that changes the to-do list. To do this, you can use the sort method on the array [newState \(src/todo/reducer.js\)](#) before saving it to localStorage. Here's how you can do it:

```
File: src/todo/reducer.js
export const todoReducer = (state, action) => {
  let newState;
  switch (action.type) {
    case ADD_ITEM:
      newState = state.concat({ id: generateUniqueID(), title:
action.payload.title, completed: false });
      break;
    case UPDATE_ITEM:
      newState = state.map((todo) => (todo.id ===
action.payload.id ? { ...todo, title: action.payload.title } : todo));
      break;
    case REMOVE_ITEM:
      newState = state.filter((todo) => todo.id !==
action.payload.id);
      break;
    case TOGGLE_ITEM:
      newState = state.map((todo) => (todo.id ===
action.payload.id ? { ...todo, completed: !todo.completed } : todo));
      break;
    case REMOVE_ALL_ITEMS:
      newState = [];
      break;
    case TOGGLE_ALL:
      newState = state.map((todo) => (todo.completed !==
action.payload.completed ? { ...todo, completed:
action.payload.completed } : todo));
      break;
    case REMOVE_COMPLETED_ITEMS:
      newState = state.filter((todo) => !todo.completed);
      break;
    case LOOD_STATE:
      newState = action.payload;
      break;
    default:
      throw Error(`Unknown action: ${action.type}`);
  }

  // Sort the items in alphabetical order by title
  newState = newState.sort((a, b) => a.title.localeCompare(b.title));

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

What was added:

1. After each change in the state ([newState \(src/todo/reducer.js\)](#)), the sort function is called to sort the items based on the [title \(src/todo/components/header.jsx\)](#) field.
2. The `localeCompare` function is used to ensure that the sorting is done correctly, considering differences in uppercase and lowercase and special characters.

Now, whenever the to-do list is changed, the items will automatically be displayed in alphabetical order.

00:19:26
- 00:19:38

> Response rendering

00:20:08

Participant

Let me see.

00:20:17

It's good, already answered, from what I saw, it should work.

00:20:34
- 00:20:35

> **Implementing Copilot code** by copying part of snippet (`src/todo/reducer.js`)

00:20:37

Participant

I think it's right.

00:20:41

I'll just check by testing the application then.

00:20:44

Researcher

OK.

00:20:47

Participant

Yeah, there's an error.

00:21:16

No.

00:21:38

It's right

00:22:25

Researcher

Did it work?

00:22:32

YES.

00:22:35

I just sent you the other tasks.

00:22:58

Participant

OK.

00:23:10

How much?

00:23:53

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

00:24:30

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

00:24:31

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

00:24:36

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

00:25:40

Participant

Hmm, Hmm.

00:25:43

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

00:25:46

Researcher

What are you thinking now?

00:25:52

Participant

I'm analyzing the task of persisting.

00:26:55

store in a JSON, something like that for later in each interaction?

00:27:00

keep fetching this JSON.

00:27:14

Researcher

You can record in the browser's `localStorage`.

00:27:19

To retrieve the data if you want.

00:28:11

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

00:28:18

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

00:28:27

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

00:28:35

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

00:29:17

Participant

Okay, I'll try to use the.

00:29:20

Use state to see if it will work.

00:29:29

Researcher

OK.

00:29:46

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

00:29:51

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

00:30:00

- 00:31:44

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

00:31:51

Participant

Well, I was researching here in another window.

00:31:54

It's what I could use in persistence.

00:31:56

but then the explanation I'm getting here on the site is kind of confusing.

00:32:01

So just to better understand the difference.

00:32:05

Chat (window: 0)

00:32:28

> **Agent**

github.copilot.default

> **Context**

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

> **Message**

useEffect x useState

> **Response**

[useState \(node_modules/@types/react/index.d.ts\)](#) and `useEffect` are two of the main React hooks, but they have different purposes:

[useState \(node_modules/@types/react/index.d.ts\)](#)

- **What it does:** Manages the local state of a component.
- **When to use:** When you need to store and update values within a component.
- **Example:**

```
const [count, setCount] = useState(0);

const increment = () => {
  setCount(count + 1);
};
```

useEffect

- **What it does:** Executes side effects in functional components, such as API calls, DOM manipulation, or data synchronization.
- **When to use:** When you need to perform actions after the component renders or when the state/props change.
- **Example:**

```
useEffect(() => {
  console.log("The component was mounted or the state changed!");
}, [count]); // Executes when count changes
```

Main differences:

Aspect	useState (node_modules/@types/react/index.d.ts)	<code>useEffect</code>
Purpose	Manage local state	Manage side effects
Executed when	Does not depend on renderings	After renderings or dependency changes
Return	State and function to update it	Nothing (it's a side effect function)

In the provided code, [useState \(node_modules/@types/react/index.d.ts\)](#) is being used to manage local state ([items \(src/todo/components/input.jsx\)](#)), while

00:32:30
- 00:32:46

00:33:20
- 00:35:02

00:35:02

00:35:13

00:35:18

00:35:34

00:35:59

useEffect could be used, for example, to synchronize the state with local storage or perform actions after the component is mounted.

> Response rendering

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

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

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

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

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

Chat (window: 0)

> Agent

github.copilot.default

> Context

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

> Message

For the case of data persistence after reload, which is the best?

> Response

To persist data after a page reload, the best option is to use a persistent storage solution, such as **localStorage**, **sessionStorage**, or even a backend API. In the case of a simple application like a task manager, **localStorage** is a common and efficient choice.

Why use localStorage?

- **Persistence:** Data remains stored even after closing the browser.
- **Easy to use:** Simple APIs to save and retrieve data.
- **No need for a backend:** Ideal for small applications or prototypes.

How to implement in your case

In the context of your code, you already have a `loadState` and `saveState` function in the `reducer.js` file. To ensure data is loaded and saved correctly, you can follow this flow:

1. Load data at the start:

- Ensure the initial state (`initialState`) is loaded from `localStorage`.

2. Save data whenever the state changes:

- Use [useEffect\(node_modules/@types/react/index.d.ts\)](#) in the [Main component\(src/todo/components/main.jsx\)](#) to monitor changes in the [todos state\(src/todo/components/main.jsx\)](#) and save to `localStorage`.

Implementation example

In the [Main component\(src/todo/components/main.jsx\)](#), add the following [useEffect\(node_modules/@types/react/index.d.ts\)](#):

```
useEffect(() => {  
  // Save todos to localStorage whenever they change  
  localStorage.setItem("todos", JSON.stringify(todos));  
}, [todos]);
```

Complete flow

1. **Load data at the start:** In `reducer.js`, you already have the `loadState` function to load data from `localStorage`.
2. **Automatically save data:** The [`useEffect`](#) ([`node\_modules/@types/react/index.d.ts`](#)) in the [`Main component`](#) ([`src/todo/components/main.jsx`](#)) ensures data is saved whenever the [`todos state`](#) ([`src/todo/components/main.jsx`](#)) changes.

Advantages

- Simple to implement.
- Does not depend on an internet connection.
- Data persists even after page reload.

If you need a more robust solution (such as synchronization between devices), you might consider using a backend API with a library like `axios` to manage requests.

00:36:01
- 00:36:12

> Response rendering

00:37:43

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

00:37:50

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

00:38:00

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

00:38:25

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

00:38:29 > **Viewing file** (src/todo/components/header.jsx)
00:38:34 > **Viewing file** (src/todo/components/input.jsx)
00:38:34 > **Viewing file** (src/todo/components/main.jsx)
00:38:54 > **Viewing file** (src/todo/reducer.js)
00:39:13 > **Viewing file** (src/todo/reducer.js)
00:40:09 **Participant**
I'm checking.
00:42:12 > **Viewing file** (src/todo/components/input.jsx)
00:42:16 > **Editing file** (src/todo/components/input.jsx)
- 00:42:52
00:42:34 **Participant**
I'll try to do the task
To persist the draft.
00:42:38 **Researcher**
00:42:43 OK.
00:42:51 [omitted] it's already been 40 minutes.
00:44:40 now I'm going to send you a post
00:44:43 study survey.
00:46:08 **Participant**
I usually don't use the copilot chat, so I don't know if this is the standard or
00:46:12 not, so just to also give the context,
00:46:15 because of my response.
00:46:22 He replied that it was necessary.
00:46:26 Didn't put much detail
00:46:29 because I think it also didn't need many details.
00:47:15 **Researcher**
OK.
00:50:24 **Participant**
I think if I had used it more, I would have a better performance
00:50:56 Searching for the solution on the internet ends up taking longer
00:51:03 So productivity would certainly increase.
00:51:10 Work effectiveness I will put in agree a little and as if I stayed there
00:51:16 also in studies.
00:51:18 In studies, I think it would hinder a little because
00:51:23 despite having the tool that helps you develop.
00:51:30 your studies, it can end up becoming rigid
00:51:34 you can become very dependent on copilot
00:51:38 The copilot is more of a help tool and not something that.
00:51:41 Should be used from the beginning of a project,
00:52:01 If you start using the copilot a lot,
00:52:05 it can end up hindering you when you try to tackle something more complex and
00:52:09 end up depending on the copilot.
00:52:28 **Researcher**
OK.
01:01:05 Now I will ask you a few more questions,
01:01:11 **Participant**
ok
01:01:12 **Researcher**
Were you able to orient yourself with the code
01:01:17 and how did the copilot influence you?
01:01:26 **Participant**
Well, first, I was more trying by myself,
01:01:31 to see how it would be, how the interaction between the codes of the
01:01:35 file, between the codes and the files themselves.
01:01:38 I was able to understand.
01:01:42 By asking a question to the copilot

01:01:47 I was also able to verify what I was thinking was working,
01:01:53 how the interactions between the codes were.
01:02:00 The copilot
01:02:03 helped me understand and verify what,
01:02:06 I had imagined.
01:02:15 **Researcher**
OK
01:02:24 do you feel you understood the basis of the code.
01:02:36 **Participant**
Well, in the simplest concept. I think so, because in the end,
01:02:41 let's say, everything is kind of Logic, regardless of the tool
01:02:46 we are using. But, as in react I ended up forgetting what
01:02:51 I had learned.
01:03:03 So to change the code the lack of knowledge in react
01:03:09 ended up hindering me, but.
01:03:12 In logic I was able to understand
01:03:15 the code ended up hindering me a bit.
01:03:23 **Researcher**
OK. And how would you have solved the tasks, without having the copilot available?
01:03:35 **Participant**
First I would use the
01:03:38 internet.
01:03:45 would search the internet using the terms
01:03:50 that were in the task description.
01:04:04 Would see real examples and something.
01:04:24 That would be the way.
01:04:32 **Researcher**
OK. And how did the copilot influence you in your strategy to solve the tasks
01:04:41 **Participant**
Well, in the first one.
01:04:46 I asked Him.
01:04:49 He suggested it would be the best alternative.
01:05:05 As I didn't know if it would work.
01:05:09 due to lack of knowledge of the language, I believed it would be the best option to use the
answer given by the copilot.
01:05:16 **Researcher**
OK.
01:05:20 What did you like and what didn't you like about your experience with the copilot?
01:05:29 **Participant**
From what I used, I liked it.
01:05:33 Chat layout. I have nothing to complain about, it's very intuitive,
01:05:38 the way to use it, I also didn't see any problem.
01:05:47 **Researcher**
OK. And during your use, did you change the way you interacted with the copilot?
01:05:58 **Participant**
No, I always kept the same way of prompts.
01:06:06 **Researcher**
Ok, thank you very much for your participation.