

Participant P09

00:09:09 **Participant**
Practice task A, change color. When, when there are no todos,
00:09:13 there is a red box with the message no todos here
00:09:17 yet. Change the color of the of this box to
00:09:20 blue, and then it's got the code saying what color.
00:09:24 That corresponds to.
00:09:27 OK so.
00:09:30 Let's go to the application and then I assume I'm looking in source in todo.
00:09:37 > **Viewing file** (src/todo/app.css)
00:09:38 **Participant**
Um and nice. I am not familiar with React, but I have used CSS. React or JavaScript
00:09:39 > **Viewing file** (src/todo/app.jsx)
00:09:41 > **Viewing file** (src/todo/constants.js)
00:09:43 > **Viewing file** (src/todo/app.css)
00:09:44 > **Viewing file** (src/todo/reducer.js)
00:09:45 > **Viewing file** (src/todo/constants.js)
00:09:46 > **Viewing file** (src/todo/reducer.js)
00:09:46 **Participant**
I'm not familiar with, but I see CSS so that's nice.
00:09:47 > **Viewing file** (src/todo/app.css)
00:09:51 **Participant**
And then.
00:09:54 I want to find.
00:09:58 When there are no todos, there is a red box with the message no todos here yet.
00:10:01 Yeah, change the color of this box to blue.
00:10:05 Um.
00:10:06 **Researcher**
Right. So just before you start.
00:10:08 **Participant**
OK.
00:10:09 **Researcher**
Just to make sure, was it clear what these what files Copilot can and cannot
00:10:14 see, and what the commands do?
00:10:16 **Participant**
Yes. So I can see Copilot if I ask a question
00:10:18 > **Viewing file** (src/todo/components/main.jsx)
00:10:19 **Participant**
now, it will answer a question about this main
00:10:21 **Researcher**
Mm hmm.
00:10:23 **Participant**
file and then if I change or answer a question about the header file is all I
00:10:24 > **Viewing file** (src/todo/components/header.jsx)
00:10:28 **Participant**
can ask.
00:10:30 I can also use the workspace command.
00:10:32 Maybe this is. I've not used this before, but maybe it's a nice time to use at
00:10:36 workspace to ask where.
00:10:41 This bit of code that I want to change is.
00:10:43 **Researcher**

00:10:45 Yeah.

00:10:49 **Participant**
So I can say.

00:10:53 Find the red box.

00:10:58 With the message.

00:11:00 No.

00:11:03 Todos here yet.

00:11:03 **Chat** (window: 0)

> Agent
 github.copilot.workspace

> Context
 workspace, viewport (src/todo/components/header.jsx)

> Message
 @workspace Find the red box with the message "No TODOs here yet"

> Response
 The red box with the message "No TODOs here yet" is rendered in the [Main \(src/todo/components/main.jsx\)](#) component. Specifically, it is displayed when there are no visible TODOs. The relevant code is:

```
File: src/todo/components/main.jsx
<div className="todo-list-empty-container">
  <span className="todo-list-empty-message">No TODOs here yet</span>
</div>
```

 This is styled in [app.css \(src/todo/app.css\)](#) with the following CSS:

```
File: src/todo/app.css
.todo-list-empty-container {
  padding: 5px;
  display: flex;
  justify-content: center;
}

.todo-list-empty-message {
  padding: 10px;
  background: #b83f45;
  color: white;
  border-radius: 5px;
  font-weight: bolder;
}
```

00:11:06 **Participant**
I think I would also if I was working normally do this which is going no todos yet.

00:11:06 - 00:11:09 **> Response rendering**

00:11:14 **Participant**
Oops, and be correct about my capitalization.

00:11:19 **> Viewing file** (src/todo/components/main.jsx) through file search

00:11:20 **Participant**
So this I imagine is what I want to find.

00:11:25 And then the red box with the message no todos here yet

00:11:28 is rendered in the main component.

00:11:30 I agree.

00:11:30 Specifically, it is displayed when there are no visible todos

00:11:33 The relevant code is this. That's good.

00:11:36 That's what I want to see. The app is styled with CSS with the

00:11:39 following CSS.

00:11:42 Let's do empty list container and it's red here.

00:11:46 Can I edit it in Copilot?

00:11:49 No, but I can go to CSS and I can search via.

00:12:00 Todo list empty container.

00:12:02 > **Viewing file** (src/todo/app.css) through file search

00:12:04 **Participant**

 And then I want to change this to what's in the PDF which is.

00:12:05 > **Editing file** (src/todo/app.css)

- 00:12:12

00:12:12 **Participant**

 007 BFF.

00:12:15 I wonder if I said what CSS is blue.

00:12:23 Give me.

00:12:26 **Chat** (window: 0)

 > **Agent**

 github.copilot.default

 > **Context**

 viewport (src/todo/app.css)

 > **Message**

 What .css is blue give the #...

 > **Response**

 The blue color in the CSS is defined by the hex code #007bff. It is used in the following classes:

 1. .todo-list-empty-message:

 background: #007bff;

 2. .pagination-button:not(.inactive):not(.active):

 background-color: #007bff;

00:12:27 > **Response rendering**

- 00:12:27

00:12:27 **Participant**

 It would help me the same number or a similar number that does give me the same number.

00:12:32 That's good. OK. But I didn't need it.

00:12:32 Nice. So that is the first thing I would do and

00:12:37 then verification is refresh the application confirm that the message

00:12:39 no todos here yet

00:12:42 shows up in a blue box.

00:12:43 Um, I haven't used React, so I don't know how you'd run the

00:12:46 > **Viewing file** (src/todo/app.jsx)

00:12:47

00:12:51 **Participant**

 application, but I will ask Copilot.

00:12:55 Because.

00:12:57 I guess that's the point.

00:12:59 How can I this application?

00:13:09 **Chat** (window: 1)

 > **Agent**

 github.copilot.workspace

 > **Context**

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

 > **Message**

 @workspace How can I run this application?

 > **Response**

 To run this application, use the dev script defined in the [package.json](#) file. Follow these steps:

 1. Open a terminal in Visual Studio Code.

 2. Install the dependencies by running:

- ```
npm install
```
3. Start the development server by running:

```
npm run dev
```

4. The application will open in your default browser. If it doesn't, you can manually navigate to `http://localhost:8080` (or the port specified in the terminal output).

00:13:12  
- 00:13:16

### > Response rendering

00:13:40

#### **Researcher**

So what are you thinking right now?

00:13:42

#### **Participant**

I'm concerned.

00:13:43

Why there's warnings showing up and whether I'm running this in the correct level of the directory if I'm working Copilot study to do.

00:13:47

And I feel like that wasn't what I needed to do in this situation.

00:13:58

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

00:14:05

#### **Participant**

Yeah.

00:14:06

I will try running `npm run dev` anyway because.

00:14:09

When I've worked in the past, it's given things.

00:14:12

And OK, that's what I wanted to see.

00:14:17

So no todos here.

00:14:19

Yeah, I should check that.

00:14:20

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

00:14:22

#### **Participant**

What I changed was actually.

00:14:22

Different in the first place.

00:14:25

So if I refresh, will it be?

00:14:28

Yeah. Good.

00:14:29

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

00:14:31

#### **Participant**

Nice. I can leave that up.

00:14:33

So this is what I'd wanted to see. Nice and that's practice task A, change colour.

00:14:38

I'm quite happy I've done that.

00:14:42

And then there's practice task B, which is change placeholder.

00:14:42

So this reads the placeholder text of the input field is currently what needs to be done change the text to enter.

00:14:45

So change the text to enter a task here.

00:14:50

So let's look at the app first.

00:14:54

What needs to be done?

00:14:56

And then I think I will search in the same way I did before, which.

00:14:59

Is searching twice basically.

00:15:01

Space.

00:15:04

And I'll be more descriptive based on the question.

00:15:09

00:15:23 Text from the.  
00:15:27 The input field.  
00:15:31 That reads, what needs to be done.  
00:15:37 Enter.  
00:15:44 **Chat** (window: 2)  
**> Agent**  
github.copilot.workspace  
**> Context**  
workspace, viewport (src/todo/app.css)  
**> Message**  
@workspace I want to change the text from the input field that reads what needs to be done to "Enter a task here"  
**> Response**  
To change the placeholder text of the input field, update the [placeholder \(src/todo/components/header.jsx\)](#) prop in the [Header \(src/todo/components/header.jsx\)](#) component. Here's the modified code:

```
File: src/todo/components/header.jsx
import { useCallback } from "react";
import { Input } from "../input";

import { ADD_ITEM } from "../constants";

export function Header({ dispatch }) {
 const addItem = useCallback((title) => dispatch({ type: ADD_ITEM,
 payload: { title } })), [dispatch]);

 return (
 <header className="header" data-testid="header">
 <h1>todos</h1>
 <Input onSubmit={addItem} label="New Todo Input"
placeholder="Enter a task here" />
 </header>
);
}
```

00:15:45 **Participant**  
Yeah.  
00:15:46 And then while that generates things, I'll also look via this which is.  
00:15:47 **> Response rendering**  
- 00:15:49  
00:15:52 **Participant**  
What needs to be done?  
00:15:55 **> Viewing file** (src/todo/components/header.jsx) through file search  
00:15:57 **Participant**  
And the placeholder could be.  
00:16:00 Oops, don't do that.  
00:16:02 **> Editing file** (src/todo/components/header.jsx)  
00:16:07 **Participant**  
This is the right file.  
00:16:08 This is what I would expect the change to be. Enter a task here.  
00:16:10 **> Editing file** (src/todo/components/header.jsx)  
- 00:16:22  
00:16:24 **Participant**  
I'm going to refresh the page because my clipboard's getting in the way.  
00:16:34 **> Editing file** (src/todo/components/header.jsx)  
- 00:16:59  
00:16:34 **Participant**  
So yeah, so we have this header file and we want  
00:16:36 to change this placeholder to enter.

00:16:39 Let me check the PDF again.  
00:16:40 Enter a task here.  
00:16:42 Seems simple enough.  
00:16:54 My clipboard is also still in the way.  
00:16:57 I don't know why.  
00:16:58 **Researcher**  
Are you able to copy and paste into this file?  
00:17:02 **Participant**  
Yeah, let's try that.  
00:17:04 So it's just triggering a notification which is annoying me,  
00:17:07 **Researcher**  
Right.  
00:17:08 **Participant**  
but maybe it's easier than typing it.  
00:17:10 **> Editing file** (src/todo/components/header.jsx)  
- 00:17:12  
00:17:10 **Researcher**  
What if you use control shift C and control shift V?  
00:17:17 Does this work?  
00:17:19 **Participant**  
Yes.  
00:17:21 This is what I want.  
00:17:24 New todo input, placeholder. Enter a task here and then if I refresh I  
00:17:28 get enter a task here, cool.  
00:17:33 So I'm happy that change has been made and that's practice task B done.  
00:17:38 And then I just want to.  
00:17:39 I didn't read this too much, but let's just check.  
00:17:43 That is what I wanted to do, update the placeholder prop.  
00:17:47 This I assume is a prop.  
00:17:50 To the header component. Here's code, yeah.  
00:17:54 New todo input, looks like what I wanted. Good.  
00:17:59 And then practice task C is sort todos, the todos are currently displayed in the  
00:18:04 order they were added.  
00:18:05 Modify the application so that the todos are sorted alphabetically instead.  
00:18:11 The following todos in this order. The quick 123 brown fox confirm that the  
00:18:16 todos are displayed in this order alphabetically. 123 brown fox quick,  
00:18:21 etc.  
00:18:23 OK so.  
00:18:25 I need to find where the todos are added.  
00:18:28 **> Viewing file** (src/todo/components/input.jsx)  
00:18:29 **Participant**  
Maybe input, not input?  
00:18:32 Umm.  
00:18:33 **> Viewing file** (src/todo/app.jsx)  
00:18:35 **> Viewing file** (src/todo/app.css)  
00:18:35 **Participant**  
This is more interesting because I don't know where I would start to look with the  
00:18:39 **> Viewing file** (src/todo/components/paginationControls.jsx)  
00:18:41 **Participant**  
string.  
00:18:42 **> Viewing file** (src/todo/components/main.jsx)  
00:18:42 **Participant**  
And I don't know in which file this is 'cause I'm not familiar with the  
00:18:47 application, so I can ask same way I did before with  
00:18:50 the workspace and I'll be more vague and say.  
00:18:56 Umm.

00:18:57 I guess something gets triggered here.  
00:18:59 So you can probably inspect.  
00:19:02 But we use Copilot first.  
00:19:05 I want to. Oh, oh, oh, I think I misinterpreted this.  
00:19:09 The todos is currently displayed in the order they were added.  
00:19:11 Modify the application so the todos are sorted alphabetically instead.  
00:19:13 So I'm adding the following todos in the application here.  
00:19:17 So I want to add the.  
00:19:18 Let's close this for now.  
00:19:33 Quick.  
00:19:35 123, brown.  
00:19:40 Fox.  
00:19:43 So the quick 123 round box and then something displays these and I want to  
00:19:47 change the order that's displayed.  
00:19:48 So I would just ask.  
00:19:51 How do you.  
00:19:52 Change the order todos.  
00:19:57 Are listed in my application.  
00:20:02 currently.  
00:20:04 I am not sure what.  
00:20:09 The order is. I do think the order is that it's  
00:20:12 displayed in the order I wrote them, but I wasn't paying too much attention.  
00:20:17 And I would like to know if Copilot will tell me.  
00:20:20 Can I, or how  
00:20:24 can I change this so that?  
00:20:29 The todos are displayed.  
00:20:35 To the user in an alphabetical.  
00:20:40 **Chat** (window: 3)  
**> Agent**  
github.copilot.workspace  
**> Context**  
workspace, selection (src/todo/components/main.jsx)  
**> Message**  
@workspace I want to change the order TODOs are listed in my application. Currently  
I am unsure what the order is. How can I change this so that TODOs are displayed to  
the user in an alphabetical order  
**> Response**  
To display TODOs in alphabetical order, you can sort the [visibleTodos](#)  
([src/todo/components/main.jsx](#)) array before rendering it. Modify the [visibleTodos](#)  
([src/todo/components/main.jsx](#)) computation in the [useMemo](#)  
([node\\_modules/@types/react/index.d.ts](#)) hook in  
[src/todo/components/main.jsx](#) to include a sorting step.  
  
Here's 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)), // Sort
```

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

This will ensure that the TODOs are displayed in alphabetical order based on their [title](#) ([src/todo/components/header.jsx](#)). The `localeCompare` method is used for case-insensitive string comparison.

00:20:41

### Participant

Alphabetical order.

00:20:43

And this workspace and then while that's running, I guess it's item.

00:20:44

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

00:20:44

> **Response rendering**

- 00:20:48

00:20:45

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

00:20:48

### Participant

And it gets updated toggle item.

00:20:52

Handle update.

00:20:58

Not here. And I'm being pointed somewhere else,

00:21:00

so to display todos in the alphabetical order, you can sort the visible todos array

00:21:04

before rendering it.

00:21:05

Modify visible todos array.

00:21:10

Modify the computation in the use memo hook in main dot JavaScript.

00:21:13

> **Viewing file** (`node_modules/%40types/react/index.d.ts`) through Copilot link

00:21:14

### Participant

I didn't know you could click this. That's quite nice.

00:21:18

This is not in main, but I guess it's used in main, that be

00:21:21

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

00:21:22

### Participant

correct.

00:21:25

Visible todos.

00:21:32

Visible todo computation is here. Yeah, so use memo is a hook.

00:21:36

I'm not too familiar with React, but I know you have hooks and they're

00:21:40

like importers, and I'm not going to worry about this.

00:21:43  
00:21:46  
00:21:49  
00:21:54  
00:21:58  
00:22:05  
00:22:08  
00:22:12  
- 00:22:17  
00:22:14  
00:22:15  
00:22:21  
00:22:23  
00:22:23  
00:22:25  
00:22:28  
00:22:31  
- 00:22:33  
00:22:48  
00:22:54  
00:22:57  
00:23:02  
00:23:07  
00:23:12  
00:23:17  
00:23:23  
00:23:25  
00:23:30  
00:23:33  
00:23:36

I'll just worry about visible todos.

So this use memo.

Is a function that filters todos.

If active, you return it if it's active.

If completed, return todo completed, sure.

And I'm not really familiar with what I'm looking at.

But Copilot has given me something to type in, so I see no harm.

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

**Participant**

In.

Typing in and testing whether it works.

Or copying it.

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

**Participant**

Because.

Maybe it's valuable.

And this comma is unneeded.

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

**Participant**

Fox.

123 brown fox quick the.

If I am not incorrect that's alphabetical order.

And I guess this is like a lambda function with the sort A to B.

I don't really know what locale compare or title is.

The locale compare method used for case insensitive string comparison.

I would like to know what the sort function is in this case.

Or maybe I'll be more broad.

How does this addition to the code?

Sort my items in.

Alphabetical.

**Chat** (window: 3)

> **Agent**

github.copilot.default

> **Context**

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

> **Message**

How does this addition to the code sort my items in alphabetical order?

> **Response**

The addition of the `.sort((a, b) => a.title.localeCompare(b.title))` method in the [visibleTodos \(src/todo/components/main.jsx\)](#) computation ensures that your TODO items are sorted alphabetically by their [title \(src/todo/components/main.jsx\)](#) property.

Here's how it works:

1. [sort \(src/todo/components/main.jsx\)](#) **Method:** The [sort \(src/todo/components/main.jsx\)](#) method is used to order the elements of an array. It takes a comparison function as an argument.
2. [localeCompare \(src/todo/components/main.jsx\)](#): The [localeCompare \(src/todo/components/main.jsx\)](#) method is called on the [title \(src/todo/components/main.jsx\)](#) property of each TODO item. It compares two strings ([a.title \(src/todo/components/main.jsx\)](#) and [b.title \(src/todo/components/main.jsx\)](#)) in a case-insensitive, locale-aware manner. It returns:

- A negative value if `a.title(src/todo/components/main.jsx)` comes before `b.title(src/todo/components/main.jsx)` alphabetically.
  - Zero if they are equal.
  - A positive value if `a.title(src/todo/components/main.jsx)` comes after `b.title(src/todo/components/main.jsx)`.
3. **Integration in `useMemo (node_modules/@types/react/index.d.ts)`:** The `sort(src/todo/components/main.jsx)` method is chained after filtering the `todos(src/todo/components/main.jsx)` array. This ensures that the filtered TODOs are sorted alphabetically before being stored in the `visibleTodos(src/todo/components/main.jsx)` variable.
- As a result, when the `visibleTodos(src/todo/components/main.jsx)` array is rendered in the `<ul>` element, the TODO items appear in alphabetical order based on their `title(src/todo/components/main.jsx)`.

00:23:36

**Participant**

The.

00:23:37  
- 00:23:43

**> Response rendering**

00:23:40

**Participant**

Maybe why instead of how, no how is good, I would like to know.

00:23:45

Computation ensures the todo items sorted, Blah blah blah. Here's how it works.

00:23:48

The sort method is used to order the elements of array.

00:23:51

It takes a comparison function as an argument.

00:23:57

So this.

00:24:01

Oh no.

00:24:02

This is the argument.

00:24:04

The local compare method is called on the title property,

00:24:07

each todo item. It compares two strings A and B.

00:24:12

And returns negative value if A comes before B, 0 if they are equal.

00:24:17

Positive value A comes after B.

00:24:27

Must I look at this.

00:24:27

**> Viewing file** (node\_modules/%40types/react/index.d.ts) through Copilot link

00:24:34

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

00:24:35

**Participant**

OK.

00:24:37

I'm not really sure why this works, but it works for now and I'm happy to

00:24:42

leave it and move on because it just seems like a sort function. Great.

00:24:44

**Researcher**

Great.

00:24:48

Perfect.

00:24:49

**Participant**

Cool. So that's practice tasks and I guess I

00:24:51

can move on to task tasks.

00:24:53

**Researcher**

Yes you can.

00:24:54

**Participant**

So.

00:24:55

**Researcher**

Thanks a lot for thinking aloud. The way you're doing, by the way.

00:24:58

That's helping a lot.

00:24:59

**Participant**

OK, good, good.

00:25:00

**Researcher**

For these tasks, no worries about narrating the questions.

00:25:04

I think these are quite a bit longer.

00:25:07

**Participant**

You don't want to hear them all multiple times. Fair enough.

00:25:09

**Researcher**

Well, you can narrate what you're thinking when

00:25:11

you're reading them.

00:25:12

**Participant**

Yeah. OK. OK.

00:25:13

**Researcher**

Yeah.

00:25:13

**Participant**

So, I'm going to read task A which is

00:25:16

persist todos.

00:25:22

Say supposedly the todos disappear when I refresh.

00:25:26

They do.

00:25:29

Sure that they that each todo state is preserved.

00:25:38

So supposedly this is already in the code somewhere.

00:25:42

And I don't want. I can't think of what string I would

00:25:45

search for or where it would be. Probably in main I guess, but.

00:25:50

We can ask.

00:25:52

The workspace and say.

00:25:56

Um.

00:25:59

There.

00:26:00

Is an attempt at functionality.

00:26:05

To create.

00:26:08

Persistent todos.

00:26:10

I'm not sure if that's clear enough, but let's try. Persistent todos.

00:26:16

For this application.

00:26:21

That remain after a refresh.

00:26:26

Find it.

00:26:28

Actually.

00:26:30

Yeah, find it.

00:26:32

And.

00:26:33

Tell me.

00:26:35

If it works, I'm interested to know if it works

00:26:39

because your document said it doesn't work.

00:26:40

**Chat (window: 4)**

**> Agent**

github.copilot.workspace

**> Context**

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

**> Message**

@workspace Their is an attempt at functionality to create persistent TODOs for this application, that remain after a refresh. Find it and tell me if it works

**> Response**

The functionality for creating persistent TODOs is implemented in

[src/todo/reducer.js](#). Specifically:

1. **Loading State:** The [loadState \(src/todo/reducer.js\)](#) function retrieves the saved TODOs from [localStorage \(node\\_modules/@types/node/globals.d.ts\)](#):

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

However, it attempts to load from the key "todoS", which is inconsistent with the key "todos" used in the [saveState \(src/todo/reducer.js\)](#) function.

2. **Saving State:** The [saveState \(src/todo/reducer.js\)](#) function saves the current state to [localStorage \(node\\_modules/@types/node/globals.d.ts\)](#):

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

3. **Reducer Integration:** The [saveState \(src/todo/reducer.js\)](#) function is called after every state update in the [todoReducer \(src/todo/reducer.js\)](#).

## Issue:

The mismatch between the keys "todoS" (used in [loadState \(src/todo/reducer.js\)](#)) and "todos" (used in [saveState \(src/todo/reducer.js\)](#)) prevents the application from correctly loading the saved TODOs after a refresh.

## Fix:

Update the [loadState \(src/todo/reducer.js\)](#) function to use the same key as [saveState \(src/todo/reducer.js\)](#):

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

00:26:42

### Participant

So let's see if Copilot agrees.

00:26:43  
- 00:26:58

### > Response rendering

00:26:44

### Participant

And I agree with your document because I see it doesn't work.

00:26:51

Umm.

00:26:52

Reducer.

00:26:53

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

00:26:55

### Participant

I don't what this is.

00:27:00

So you load a state and you save a state and.

00:27:05

That state has, is, that state is a todo.

00:27:11

Gosh okay.

00:27:17  
- 00:27:21

### > Editing file (src/todo/reducer.js)

00:27:17

### Participant

Lood state, I imagine.

00:27:21

Is an issue, but I won't touch that actually.

00:27:25

Yeah, because case lood state.

00:27:30 Update item.  
00:27:36 So I imagine this comes from somewhere else,  
00:27:39 but I don't know where it comes from.  
00:27:42 It is from up here and that's from constants.  
00:27:42 > **Viewing file** (src/todo/constants.js)  
00:27:47 > **Viewing file** (src/todo/reducer.js)  
00:27:47 **Participant**  
And then you can export this constant load state and.  
00:27:51 > **Editing file** (src/todo/reducer.js)  
- 00:27:56  
00:27:59 **Participant**  
Before I look at this, I'm gonna test if it works.  
00:28:02 Oops, I need to add something. Quick.  
00:28:05 Brown.  
00:28:08 Fox and then if you refresh it doesn't work.  
00:28:10 So there's something more required than load state.  
00:28:14 Um.  
00:28:17 Let's read this.  
00:28:18 Functionality for creating persistent todos is implemented in this file.  
00:28:24 Loading state. The load state function retrieves the saved todos from local  
00:28:29 storage, so that it's that, and I agree that's what this looks like it does.  
00:28:33 However, it attempts to load from the key todos  
00:28:36 which is inconsistent with the key todos used in  
00:28:39 the save state function. Ah I see.  
00:28:41 > **Editing file** (src/todo/reducer.js)  
00:28:42 **Participant**  
So it doesn't see anything, 'cause doesn't exist.  
00:28:46 Save state function saves the current state to local storage.  
00:28:51 The save state function is called after every state update in toreduce, todo  
00:28:56 reducer.  
00:28:58 That's this function here.  
00:29:01 Update load state to use the same key as save state, so maybe that's enough. Fox.  
00:29:07 Brown.  
00:29:09 No. Did I save this?  
00:29:14 Yes.  
00:29:16 Let's try again.  
00:29:22 No, that doesn't work.  
00:29:23 So there's something more to the problem.  
00:29:27 So let's see.  
00:29:29 I feel as if this had something to do with it and it wasn't mentioned in this  
00:29:34 response, so maybe.  
00:29:37 There is more information that's been missed by both of us.  
00:29:40 > **Viewing file** (src/todo/constants.js)  
00:29:44 **Participant**  
Here's a constants and they are used here.  
00:29:45 > **Viewing file** (src/todo/reducer.js)  
00:29:49 **Participant**  
And what does this do?  
00:29:51 New state action dot payload.  
00:29:56 Maybe this requires more information at the end, so I'd like to know if this.  
00:30:05 Is the todo.  
00:30:09 Reducer function.  
00:30:13 Complete.  
00:30:14 Specifically.  
00:30:21 The load.  
00:30:23 State, state case.

00:30:28

Chat (window: 4)

> Agent

github.copilot.default

> Context

viewport (src/todo/reducer.js)

> Message

Is the todoReducer function complete? Specifically the LOAD\_STATE case?

> Response

The [todoReducer \(src/todo/reducer.js\)](#) function is complete, but the [LOAD\\_STATE \(src/todo/constants.js\)](#) case is not fully functional in its current implementation. Here's why:

### Current [LOAD\\_STATE \(src/todo/constants.js\)](#) Case:

```
case LOAD_STATE:
 newState = action.payload;
 break;
```

### Issue:

The [LOADSTATE \(src/todo/constants.js\)](#) case assumes that the state to be loaded is passed as [action.payload \(src/todo/reducer.js\)](#). However, the [loadState \(src/todo/reducer.js\)](#) function is already defined to retrieve the state from [localStorage \(node\\_modules/@types/node/globals.d.ts\)](#). This means the [LOADSTATE \(src/todo/constants.js\)](#) case is redundant unless you explicitly dispatch an action with the saved state as the payload.

### Suggested Improvement:

If the [LOAD\\_STATE \(src/todo/constants.js\)](#) case is meant to initialize the state from [localStorage \(node\\_modules/@types/node/globals.d.ts\)](#), it should directly call the [loadState \(src/todo/reducer.js\)](#) function instead of relying on [action.payload \(src/todo/reducer.js\)](#). Here's how you can modify it:

```
case LOAD_STATE:
 newState = loadState(); // Load state directly from localStorage
 break;
```

Alternatively, if the [LOADSTATE \(src/todo/constants.js\)](#) case is meant to handle externally provided state (e.g., for testing or server-synced state), then it is fine as is, but you need to ensure that the [LOADSTATE \(src/todo/constants.js\)](#) action is dispatched with the correct payload.

### Conclusion:

The [todoReducer \(src/todo/reducer.js\)](#) function is complete, but the [LOAD\\_STATE \(src/todo/constants.js\)](#) case's functionality depends on how the action is dispatched. If the intent is to load the state from [localStorage \(node\\_modules/@types/node/globals.d.ts\)](#), it should call [loadState\(\) \(src/todo/reducer.js\)](#) directly.

> Response rendering

Participant

I'm not sure what it would expect to go here, but I get the impression there should be something here, so let's see if. Copilot agrees and I can see here that it's not picked up the whole function, which is a bit annoying. So.

00:30:30  
- 00:30:33

00:30:30

00:30:33

00:30:39

00:30:42

00:30:46

00:30:49  
00:30:53  
00:30:57  
00:31:00  
00:31:03  
00:31:07  
00:31:10  
00:31:15  
00:31:19  
00:31:24  
00:31:27  
00:31:40  
00:31:44  
  
00:31:47  
00:31:52  
00:31:54  
00:31:59  
00:32:00  
00:32:03  
  
00:32:06  
00:32:10  
00:32:14  
00:32:18  
00:32:26  
00:32:28  
00:32:38  
00:32:45  
00:32:49  
00:32:52  
00:32:56  
00:33:02  
00:33:05  
00:33:08  
00:33:20  
00:33:22  
00:33:25

Maybe.

Supposedly it's not functional in its current implementation, sure. Load state case assumes that the state to be loaded is passed as action dot payload. However the load state function is already defined to retrieve the state from local storage.

Sure, this means load state case is redundant unless you dispatch an action.

If the load state case is meant to initialize the state from local storage, it should directly call the load state function instead of relying on action dot payload, and it's referring to this function up here so we can try it.

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

**Participant**

And it built successfully, which is nice.

And we can just test if it works.

And it does not work.

That's OK.

So I would like to look at the whole. I would like to take this change back and

**> Editing file** (`src/todo/reducer.js`)

**Participant**

then look at the whole file.

And see.

There's also an initial state, maybe.

That overwrites it. Hmm, I'm not sure.

Constant, is this.

Referring.

To the whole file reducer JS.

Why is the?

Yeah. Why is the?

Lood.

Lood state functionality.

Not working as intended.

I had hoped.

For.

A persistent list of items but it is deleted every time my application.

Is refreshed.

And I'm going to copy this because at the moment I'm asking about a certain file

**Chat** (window: 5)

**> Agent**

github.copilot.workspace

**> Context**

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

**> Message**

Referring to the whole file "reducer.js" why is the loadState functionality not working as intended. I had hoped for a persistent list of items but it is deleted every time my application is refreshed

**> Response**

The issue lies in how the [initialState \(src/todo/reducer.js\)](#) is being used in the [todoReducer \(src/todo/reducer.js\)](#). While [loadState\(\)\(src/todo/reducer.js\)](#) correctly retrieves the persisted state from [localStorage \(node\\_modules/@types/node/globals.d.ts\)](#), the [useReducer \(node\\_modules/@types/react/index.d.ts\)](#) hook in [src/todo/app.jsx](#) initializes the state with an empty array (`[]`) instead of the [initialState \(src/todo/reducer.js\)](#) from [src/todo/reducer.js](#).

Here's the relevant part of [src/todo/app.jsx](#):

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

The second argument to [useReducer \(node\\_modules/@types/react/index.d.ts\)](#) is the initial state, and it is currently set to []. This overrides the [initialState \(src/todo/reducer.js\)](#) exported from [src/todo/reducer.js](#).

## Fix

To ensure the persisted state is loaded correctly, you need to pass [initialState \(src/todo/reducer.js\)](#) from [src/todo/reducer.js](#) to [useReducer \(node\\_modules/@types/react/index.d.ts\)](#). Update the [useReducer \(node\\_modules/@types/react/index.d.ts\)](#) call in [src/todo/app.jsx](#) as follows:

```
File: src/todo/app.jsx
const [todos, dispatch] = useReducer(todoReducer, initialState);
```

This ensures that the [todoReducer \(src/todo/reducer.js\)](#) starts with the state loaded from [localStorage \(node\\_modules/@types/node/globals.d.ts\)](#).

00:33:26

**Participant**

and maybe.

00:33:27  
- 00:33:33

**> Response rendering**

00:33:28

**Participant**

This file has nothing to do with it.

00:33:32

I'm not sure.

00:33:36

How the initial state is being used in the todo reducer,

00:33:40

while load state correctly, supposedly. So the issue lies in how the initial

00:33:44

state is being used in the todo reducer. That's here.

00:33:48

We saw this before. Load state correctly retrieves the

00:33:51

**> Viewing file** (node\_modules/%40types/node/globals.d.ts) through Copilot link

00:33:52

**Participant**

persisted state from local storage, which is here. The use reducer hook in app dot

00:33:55

**> Viewing file** (node\_modules/%40types/react/index.d.ts) through Copilot link

00:33:58

**Participant**

JSX initializes the state with an empty array.

00:34:02

Instead of the initial state from reducer dot JS.

00:34:07

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

00:34:07

**Participant**

Thus.

00:34:12

Second argument to use reducer is the initial state and it is currently set to this empty array.

00:34:17

00:34:19

This overrides the initial state exported from reducer dot JS.

00:34:23

To ensure the persist state is loaded correctly,

00:34:25

you need to pass your initial state from reducer JS to use reducer.

00:34:31

Update the use reducer call in app dot JSX as follows. This ensures.

00:34:38

That the todo reducer starts with the state loaded from local storage.

00:34:41

This is a very convincing answer, so I will try it.

00:34:43  
- 00:34:46

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

00:34:49

**Participant**

Although I'm not sure where this comes from and I imagine I have to import it.  
So use reducer comes from oh sorry, initial state comes from this file which

00:34:59

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

00:35:01

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

00:35:04

**Participant**

is reducer dot JS.

00:35:04

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

00:35:07  
- 00:35:10

**Participant**

Oops, not that one.

00:35:10

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

00:35:10

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

00:35:14  
- 00:35:15

**Participant**

Can I do this, initial state?

00:35:15

And then quick.

00:35:20

Fox, brown.

00:35:23

Nice. OK. It would have been nice if it had told me  
how to import things up here.

00:35:25

Up here, but I guess that's something that if you

00:35:30

knew React really well, you would look at straightaway and go, oh,

00:35:33

I have an import error and I guess I still managed to figure out without this.

00:35:35

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

00:35:39

**Participant**

You kind of don't need something to hold your hand the whole time,  
so that's good answer.

00:35:41

OK.

00:35:44

Good. I want to know if this isn't so relevant.

00:35:47

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

00:35:49

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

00:35:52

**Participant**

But it suggested I changed todos.

00:35:55

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

00:35:57

**Participant**

Let's find it before, it's todos as a variable.

00:35:58

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

00:35:59

**Participant**

Within this reducer dot JS and I'm interested to whether if I do this,

00:36:01

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

00:36:05

**Participant**

it's also the same.

00:36:05

OK.

00:36:06

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

00:36:09

**Participant**

That's just a variable, so it doesn't matter.

00:36:09

Fair enough.

00:36:19

I thought it was interesting that it gave me todo all lower case rather than todo  
all upper case, or change that to that, but yeah, makes sense.

00:36:19

Goo, there were more verification steps.

00:36:20

So add some todos, refresh the page, confirm the todos are  
still there.

00:36:21

I've done that. Mark some todos as completed, refresh the page and confirm both the todos

00:36:22

00:36:26

00:36:30

00:36:35

00:36:37

00:36:38

00:36:43  
00:36:46  
00:36:53  
00:36:55  
00:36:59  
00:37:00  
00:37:03  
00:37:07  
00:37:13  
00:37:18  
00:37:26  
00:37:30  
00:37:34

and their states remain unchanged.

So let's go back here and we will mark these as completed.

Refresh the page.

That doesn't work. Interesting. So let's go back.

And there were.

We want to find something about when we complete something, so maybe we can look here.

That's a lot of things, so I can also ask Copilot.

Let's go at workspace. What functionality?

Controls the marking of todos as completed.

How can I make it persist?

Between page refreshes.

**Chat** (window: 6)

**> Agent**

github.copilot.workspace

**> Context**

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

**> Message**

@workspace What functionality controls the marking of todos as completed. How can I make it persist between page refreshes

**> Response**

The functionality for marking todos as completed is controlled by the [TOGGLE\\_ITEM \(src/todo/constants.js\)](#) action in the [todoReducer \(src/todo/reducer.js\)](#). Here's the relevant part of the reducer:

```
case TOGGLE_ITEM:
 newState = state.map((todo) => (todo.id === action.payload.id ? {
 ...todo, completed: !todo.completed } : todo));
 break;
```

This updates the [completed \(src/todo/reducer.js\)](#) property of the todo item with the matching [id \(node\\_modules/@types/react/index.d.ts\)](#).

To make this functionality persist between page refreshes, the application already uses [localStorage \(node\\_modules/@types/node/globals.d.ts\)](#) to save and load the state. The [saveState \(src/todo/reducer.js\)](#) function is called after every state update in the reducer:

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

The [loadState \(src/todo/reducer.js\)](#) function retrieves the saved state when the application initializes:

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

Since this mechanism is already in place, marking todos as completed will persist between page refreshes as long as the [saveState \(src/todo/reducer.js\)](#) function is called after the state is updated. This is already happening in the reducer after every action. No additional changes are needed for persistence.

00:37:36

**Participant**

So I would like to know, I'd actually like to know this.

00:37:37  
- 00:37:42

**> Response rendering**

00:37:40

**Participant**

This is more important to figure out where the problem is, and then this is.

00:37:43 It would be nice if it told me how to do it, but if it can't tell me how to do it,  
00:37:46 that's fine too.  
00:37:48 Functionality for marketing todos is controlled by toggle item action in the  
00:37:50 > **Viewing file** (src/todo/constants.js) through Copilot link  
00:37:52 **Participant**  
todo reducer.  
00:37:53 > **Viewing file** (src/todo/reducer.js) through Copilot link  
00:37:54 **Participant**  
OK.  
00:37:55 So back here.  
00:37:56 Here's the relevant part of the todo reducer is toggle item.  
00:38:02 This updates the completed property which is here of the todo item with the matching  
00:38:09 ID which is there.  
00:38:12 To make this functionality persist between page refreshes,  
00:38:15 the application already uses local storage to save and load the state.  
00:38:19 The save state function is called after every state update in the reducer.  
00:38:25 Is it?  
00:38:29 It is.  
00:38:30 The load state function retrieves the saved state when the application  
00:38:33 initializes.  
00:38:37 Since this mechanism is already in place, marking todos as completed will persist  
00:38:41 between page refreshes as long as the save state function is called after the  
00:38:44 state is updated.  
00:38:45 This is already happening.  
00:38:46 The reducer after every action, no additional changes are needed for  
00:38:50 persistence.  
00:38:51 That's interesting.  
00:38:53 It is called after every action.  
00:38:59 So I.  
00:39:00 I'm very sure it didn't work before.  
00:39:02 Oh, maybe.  
00:39:04 Did I change something, 123?  
00:39:10 I don't think I changed anything.  
00:39:13 But maybe in the recording someone will look back and go, oh, he did.  
00:39:17 So that's helpful.  
00:39:18 And I guess in that case, I'm glad I asked. Very good.  
00:39:23 Umm.  
00:39:25 Edit some todos refresh the page and confirm the changes are still there.  
00:39:31 So I can edit this todo so. Let's go red.  
00:39:35 And that moves to the bottom, which is nice.  
00:39:38 And then I need to.  
00:39:41 Refresh the page and make sure that brown is still red.  
00:39:44 Brown is still red. Good.  
00:39:48 Delete some todos, refresh the page and confirm the deleted  
00:39:51 todos are still gone. Um.  
00:39:56 Ah, a cross.  
00:39:58 And a cross and then refresh and they're still gone.  
00:40:01 Good. So I'm happy that I've done everything in  
00:40:05 task A which is, add a todo just just for the sake of everything.  
00:40:11 Add a todo, green.  
00:40:15 Mark it as completed.  
00:40:18 Refresh.  
00:40:20 Edit it.  
00:40:25 And delete or refresh and delete.  
00:40:28 Nice. So I'm happy that's done.  
00:40:32 So then task B is to save drafts. If a user refreshes the page while typing

00:40:36 a new todo the text input, the text in the input field disappears.  
00:40:40 Is this true?  
00:40:44 Green.  
00:40:46 Then if I refresh, yes it's gone.  
00:40:49 Modify the application so that the input field automatically saves drafts every  
00:40:53 few seconds and restore them when the page reloads.  
00:40:57 So we modified a box at the start.  
00:40:59 **> Viewing file** (src/todo/components/main.jsx)  
00:41:02 **Participant**  
Which were, there's also an import here that doesn't do anything,  
00:41:05 **> Editing file** (src/todo/components/main.jsx)  
00:41:05 **Participant**  
so I'm going to remove it and hopefully it doesn't break anything.  
00:41:10 It doesn't look like it broke anything, but I see I don't need to.  
00:41:11 **> Editing file** (src/todo/components/main.jsx)  
00:41:13 **Participant**  
That's not the task. I'll keep it here and maybe it's relevant  
00:41:16 later.  
00:41:17 So I want to.  
00:41:21 Find the box.  
00:41:24 This box was the empty message container and then this is just a div.  
00:41:28 So somewhere else there's functionality.  
00:41:32 So I will ask Copilot because I don't know how you link the functionality of a  
00:41:37 div to the div itself in React.  
00:41:41 So I have a element.  
00:41:46 In my application.  
00:41:50 Todo.  
00:41:55 Todo list empty container.  
00:42:04 I want to save.  
00:42:11 Any information input to the container, every few seconds.  
00:42:20 Which part of the code must I change?  
00:42:25 I'm gonna go say that if a user refreshes.  
00:42:32 The page.  
00:42:34 I have an element.  
00:42:36 I have an element in my application todo list empty container.  
00:42:39 I want to save any information input to the container, seconds.  
00:42:41 So if user refreshes the page.  
00:42:45 That partially filled answer.  
00:42:50 **Chat** (window: 7)  
**> Agent**  
github.copilot.workspace  
**> Context**  
workspace, selection (src/todo/components/main.jsx)  
**> Message**  
@workspace I have an element in my application. "todo-list-empty-container" I want  
to save any information input to the container every few seconds, so that if a user  
refreshes the page that partially filled answer is still there. Which part of the code must  
I change  
**> Response**  
To save the information input into the "todo-list-empty-container" and persist it across  
page refreshes, you need to:  

1. Add an input field to the "todo-list-empty-container" for user input.
2. Use [localStorage \(node modules/@types/node/globals.d.ts\)](https://node_modules/@types/node/globals.d.ts) to save the  
input value periodically.

3. Load the saved value from [localStorage](#) ([node\\_modules/@types/node/globals.d.ts](#)) when the component mounts. You should modify the [Main\(src/todo/components/main.jsx\)](#) component in [src/todo/components/main.jsx](#). Here's how you can do it:

```
File: src/todo/components/main.jsx
import { useEffect, useState, useCallback } from "react";

export function Main({ todos, dispatch }) {
 // ...existing code...

 const [emptyMessageInput, setEmptyMessageInput] = useState(() => {
 // Load saved value from localStorage
 return localStorage.getItem("emptyMessageInput") || "";
 });

 const handleInputChange = useCallback((e) => {
 setEmptyMessageInput(e.target.value);
 }, []);

 useEffect(() => {
 // Save the input value to localStorage every few seconds
 const interval = setInterval(() => {
 localStorage.setItem("emptyMessageInput",
emptyMessageInput);
 }, 3000);

 return () => clearInterval(interval);
 }, [emptyMessageInput]);

 return (
 <main className="main" data-testid="main">
 {visibleTodos.length > 0 ? (
 // ...existing code...
) : (
 <div className="todo-list-empty-container">
 <input
 type="text"
 className="todo-list-empty-input"
 value={emptyMessageInput}
 onChange={handleInputChange}
 placeholder="Enter your message here"
 />
 </div>
)}
 { /* ...existing code... */ }
 </main>
);
}
```

## Key Changes:

1. **State Management:** Added emptyMessageInput state to manage the input value.
2. **Input Field:** Replaced the static message with an input field.
3. **Persistence:** Used [localStorage](#) ([node\\_modules/@types/node/globals.d.ts](#)) to save and load the input value.

This ensures that the user's input is saved and restored even after a page refresh.

00:42:50

## Participant

Is still there.

00:42:53  
- 00:43:03

## > Response rendering

00:42:55

## Participant

As an insight from me, I find sometimes I'm writing things to LLMs and I feel like I'm being too detailed, but having used them in my work, I think it is important to put the too detailed answer. If I said to a person, I wouldn't necessarily say so.

00:43:09

Much that I wanted to change it.

00:43:11

But to overemphasize I think is useful.

00:43:14

Maybe that's something other people think is strange, but I try to do it.

00:43:16

Add an input field.

00:43:22

For user input, there should already be an input field

00:43:24

here, so I'm surprised it told me to do this.

00:43:27

Use local storage to save the input.

00:43:32

I already don't trust the answer, so I'm going to be a bit more vague and

00:43:36

say.

00:43:40

How does the input for my?

00:43:42

Todo list empty.

00:43:47

Container work.

00:43:52

**Chat** (window: 8)

00:43:53

## > Agent

github.copilot.workspace

## > Context

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

## > Message

@workspace How does the input for my "todo-list-empty-container work"

## > Response

The [todo-list-empty-container \(src/todo/components/main.jsx\)](#) is a fallback UI that displays a message when there are no visible todos in the list. It works as follows:

1. **Condition for Rendering:** The [todo-list-empty-container \(src/todo/components/main.jsx\)](#) is rendered when the [visibleTodos \(src/todo/components/main.jsx\)](#) array is empty. This is determined by the conditional rendering logic in the [Main \(src/todo/components/main.jsx\)](#) component:

```
{visibleTodos.length > 0 ? (
 // Render the todo list
) : (
 <div className="todo-list-empty-container">
 No TODOs here
yet
 </div>
)}
```

2. **Empty State Message:** When [visibleTodos \(src/todo/components/main.jsx\)](#) is empty, the following JSX is rendered:

```
<div className="todo-list-empty-container">
 No TODOs here
yet
</div>
```

This displays a message (No TODOs here yet) inside a styled container.

3. **Styling:** The styles for the [todo-list-empty-container](#) ([src/todo/components/main.jsx](#)) and its message are defined in [src/todo/app.css](#):

```
.todo-list-empty-container {
 padding: 5px;
 display: flex;
 justify-content: center;
}

.todo-list-empty-message {
 padding: 10px;
 background: #007bff;
 color: white;
 border-radius: 5px;
 font-weight: bolder;
}
```

- The container is centered using `display: flex` and `justify-content: center`.
- The message has a blue background, white text, rounded corners, and bold font.

This ensures that when there are no todos to display, users see a clear and visually distinct message indicating the empty state.

00:43:56

### Participant

So I imagine it's it's looked at this and given an answer.

00:43:58  
- 00:44:12

### > Response rendering

00:44:02

### Participant

But I don't trust it.

00:44:03

Rendered when the visible todo array is empty.

00:44:19

I also don't know how I feel about this, going to look at the question again.

00:44:23

Sorry if you refresh the page typing you to do the text in the input field disappears.

00:44:26

00:44:26

Modify the application so the input field automatically saves drafts every few seconds, restore them when page reloads.

00:44:30

00:44:33

I don't know how to, oops, I've closed too many things. How to make it work in React.

00:44:43

### > Viewing file (src/index.js)

00:44:46

### Participant

I feel like there's, I think in React. This is what hooks do, but I also

00:44:47

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

00:44:48

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

00:44:52

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

00:44:53

### Participant

don't know.

00:45:04

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

00:45:06

### Participant

Input.

00:45:09

This looks more like what I want to change.

00:45:16

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

00:45:16

### Participant

Or this.

00:45:35

So I'm looking at the.

00:45:40

I always forget what this is called the developer thing.

00:45:45

On the website, just to see. So there is an input container and it has these things.

00:45:56

Data test ID.

00:45:57

Text input.

00:46:03

So maybe it'd be nice to see every instance of input container.

00:46:07

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

00:46:14

### Participant

And yeah, this is where we looked at before.

00:46:21

So I'd like to know kind of vaguely because I'm not an expert in Reacts what

00:46:27

this file, what does input dot JSX do?

00:46:29

**Chat** (window: 9)

### > Agent

github.copilot.workspace

### > Context

workspace, selection (src/todo/components/input.jsx)

### > Message

@workspace What does input.jsx do?

### > Response

The file [input.jsx \(src/todo/components/input.jsx\)](#) defines a reusable React component named [Input \(src/todo/components/input.jsx\)](#). This component is used to render an input field with specific behaviors and accessibility features. Here's what it does:

#### 1. Props:

- [onSubmit \(src/todo/components/header.jsx\)](#): A callback function triggered when the user presses the "Enter" key with valid input.
- [placeholder \(src/todo/components/header.jsx\)](#): A placeholder text displayed in the input field.
- [label \(src/todo/components/header.jsx\)](#): A visually hidden label for accessibility.
- [defaultValue \(node\\_modules/@types/react/index.d.ts\)](#): The default value of the input field.
- [onBlur \(src/todo/components/input.jsx\)](#): A callback function triggered when the input field loses focus.

#### 2. Validation:

- The [hasValidMin \(src/todo/components/input.jsx\)](#) function ensures that the input value meets a minimum length requirement (2 characters in this case).

#### 3. Event Handlers:

- [handleKeyDown \(src/todo/components/input.jsx\)](#): Handles the "Enter" key press. It validates the input, calls the [onSubmit \(src/todo/components/header.jsx\)](#) callback with the trimmed value, and clears the input field.
- [handleBlur \(src/todo/components/input.jsx\)](#): Calls the [onBlur \(src/todo/components/input.jsx\)](#) callback when the input loses focus.

#### 4. Rendering:

- The component renders a `<div>` with the class [input-container \(node\\_modules/@types/react/index.d.ts\)](#) containing:
  - An `<input>` element for user input, styled with the `new-todo` class and equipped with event handlers for [onBlur \(src/todo/components/input.jsx\)](#) and [onKeyDown \(node\\_modules/@types/react/index.d.ts\)](#).
  - A visually hidden `<label>` for accessibility, associated with the input field via the [htmlFor \(node\\_modules/@types/react/index.d.ts\)](#) attribute.

This component is likely used in the application to handle user input for creating or editing tasks in the to-do list.

00:46:32  
- 00:46:38

00:46:33

#### > Response rendering

##### Participant

I can make a guess based on the name, but sometimes it's helpful to know a bit more because this is may as well be Greek to me.

The file defines reusable React component named input.

This component is used to render an input field with specific behaviors and accessibility features.

I'll read it all, so on submit. These are all promises.

Displayed in the input field.

So this maybe is what I want to change.

A callback function.

No, I don't want to change that.

I want to save it or make something persist because that happens at the end.

Placeholder is not important.

Visual is not important.

Maybe I want to add a new prop.

This is not important for now and at the enter.

So I would like.

To make it and I'm going to look at the task,

do something very similar to what the task asked.

So.

I'd like to make it so that the input.

Field.

Automatically.

00:46:38

00:46:41

00:46:43

00:46:46

00:46:52

00:46:57

00:47:02

00:47:06

00:47:07

00:47:09

00:47:14

00:47:15

00:47:20

00:47:25

00:47:29

00:47:32

00:47:34

00:47:39

00:47:41

00:47:45

00:47:49

00:47:53  
00:47:59  
00:48:03  
00:48:05  
00:48:10  
00:48:13  
00:48:14

Saves drafts every few seconds.

I'd like to make it to the input field, automatic saves drafts every few seconds when I refresh.

The page.

I want the information.

To have persisted.

**Chat** (window: 9)

**> Agent**

github.copilot.default

**> Context**

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

**> Message**

I would like to make it so that the input field automatically saves drafts every few seconds. When I refresh the page. I want the information to have persisted

**> Response**

To implement automatic draft saving and persistence for the input field, you can use the browser's `localStorage` to store the draft and retrieve it when the component mounts. Here's how you can modify the [Input \(src/todo/components/input.jsx\)](#) component:

## Updated Code

```
File: src/todo/components/input.jsx
import { useCallback, useEffect, useState } from "react";

const isValidMin = (value, min) => {
 return value.length >= min;
};

export function Input({ onSubmit, placeholder, label, defaultValue,
onBlur }) {
 const [draft, setDraft] = useState(() => {
 // Load the draft from localStorage on initial render
 return localStorage.getItem("todo-draft") || defaultValue ||
 "";
 });

 // Save the draft to localStorage every few seconds
 useEffect(() => {
 const interval = setInterval(() => {
 localStorage.setItem("todo-draft", draft);
 }, 3000); // Save every 3 seconds

 return () => clearInterval(interval); // Cleanup on unmount
 }, [draft]);

 const handleBlur = useCallback(() => {
 if (onBlur) onBlur();
 }, [onBlur]);

 const handleKeyDown = useCallback(
 (e) => {
 if (e.key === "Enter") {
 const value = e.target.value.trim();

 if (!isValidMin(value, 2)) return;

 onSubmit(value);
 setDraft(""); // Clear the draft
 localStorage.removeItem("todo-draft"); // Remove the
draft from localStorage
 e.target.value = "";
 }
 },
);
```

```

 },
 [onSubmit]
);

 const handleChange = useCallback((e) => {
 setDraft(e.target.value); // Update the draft state on input
 }, []);

 return (
 <div className="input-container">
 <input
 className="new-todo"
 id="todo-input"
 type="text"
 data-testid="text-input"
 autoFocus
 placeholder={placeholder}
 value={draft}
 onChange={handleChange}
 onBlur={handleBlur}
 onKeyDown={handleKeyDown}
 />
 <label className="visually-hidden" htmlFor="todo-input">
 {label}
 </label>
 </div>
);
}

```

## Explanation of Changes

### 1. State for Draft:

- Added a draft state to manage the current value of the input field.
- Initialized the state with the value from `localStorage` or [defaultValue \(node\\_modules/@types/react/index.d.ts\)](#).

### 2. Auto-Save with `localStorage`:

- Used a `useEffect` hook to save the draft to `localStorage` every 3 seconds.
- Cleaned up the interval when the component unmounts.

### 3. Handle Input Changes:

- Added an `onChange` handler to update the draft state as the user types.

### 4. Clear Draft on Submit:

- Cleared the draft state and removed the saved draft from `localStorage` when the user submits the input.

### 5. Persist Draft on Refresh:

- Loaded the saved draft from `localStorage` when the component mounts.

## Result

- The input field automatically saves its content to `localStorage` every 3 seconds.
- When the page is refreshed, the saved draft is restored.
- Submitting the input clears the draft and removes it from `localStorage`.

00:48:17  
- 00:48:40

00:48:33

## > Response rendering

### Participant

What do you think, Copilot?

00:48:35

So it's only a reference input to JSX which if I am correct is all it needs.

00:48:43

And then to implement automatic draft saving and persistence for the input field, you can use local storage to store the draft and retrieve it when the component mounts.

00:48:46

00:48:50

Here's how you can modify the input component so it's giving me back my file with some changes.

00:48:51

00:48:55

Use effect and use state.

00:48:56

I don't know what these are, but I guess they're like Python normal libraries.

00:48:59

00:49:03

But we'll read everything first.

00:49:04

I see our `expers` function, `export` function input and that's the same.

00:49:06

Save the draft to local storage every few seconds.

00:49:11

So every three seconds making it save that, if that does what it says it does, that looks like I want what I want to find.

00:49:14

00:49:18

00:49:21 Key down is the same, handle blur.  
00:49:29 In fact, now do I know what Handleblur does?  
00:49:34 **Researcher**  
Alright, I see we've reached the 40 minute mark.  
00:49:36 **Participant**  
OK.  
00:49:38 **Researcher**  
Yeah, so great. Thanks a lots.  
00:54:59 **Participant**  
Generally, what do I think of the length?  
00:55:12 I think it's not too verbose. I think it answers the questions.  
00:55:17 I think if I I don't have an issue with it because for this one.  
00:55:22 Yeah, it was very short and I wanted a short  
00:55:24 answer. And then the other ones when I wanted  
00:55:26 something longer, it gave me more information.  
00:55:28 So I think the length is actually very good.  
00:55:30 I think good because it's not perfect. Detail, length and detail are funny  
00:55:36 because they're quite similar but different.  
00:55:41 Detail I think this is this is a nice example with the colour.  
00:55:46 Because I asked it a question and expected low length and low detail and I  
00:55:51 got low length and low detail but it answered my question and then for more.  
00:55:58 Example.  
00:56:05 I feel like sometimes the detail is too much and not really what I wanted.  
00:56:11 Because this gave me details about the wrong what I think is the wrong thing.  
00:56:16 Maybe it was the previous conversation.  
00:56:20 I think I think the detail is fine, but I think this is actually much more  
00:56:26 valuable as a detail. Is the links to the code.  
00:56:29 So a good use of technical terms.  
00:56:33 At no points that refer to hooks and props and I don't know what hooks or  
00:56:37 props are because I don't use React so often.  
00:56:41 But I understand that would be. I think it's best to call it that.  
00:56:45 Then let's find the right one.  
00:56:48 And.  
00:56:55 Was it this one.  
00:56:57 Was it the second to last one?  
00:57:02 That's not important.  
00:57:03 I saw hooks and props.  
00:57:03 I didn't know what they were, but I think that's good because I'd say  
00:57:07 excellent. Sure. Because I want the correct terminology  
00:57:11 even if I don't know what it means.  
00:57:13 And correctness I would describe as neutral.  
00:57:16 I think it's correct, but only as correct.  
00:57:22 As me, when I refer to it as the code say so.  
00:57:27 Well, they.  
00:57:31 It will give an answer that always seems convincing and corrects.  
00:57:35 You don't know how.  
00:57:39 Useful. It is until you validate it. I think in most cases it gave me the  
00:57:42 answer I wanted straight away like this.  
00:57:49 But less so in other cases and in some cases such as the.  
00:57:54 Alphabetical order, I think that was correct straight away  
00:57:57 the.  
00:58:00 There's one or two questions where it wasn't correct right away and that's fine,  
00:58:03 but.  
00:58:05 In that case, I keep it as neutral and following my  
00:58:07 instructions there were points where I wanted it to see the whole file or the whole  
00:58:11 the whole function, and it only looked at a portion of the

00:58:14 function.

00:58:17 And I think that's a bit confusing.

00:58:19 I think that's different to searching through the code for things.

00:58:26 And then tone, I would say the tone is good.

00:58:30 I guess it's very matter of fact.

00:58:33 It just tells me what I don't need, humorous or.

00:58:37 Anything confidence?

00:58:39 Fine. I even though I understand it's wrong and

00:58:41 can be confident, I still think that's useful and the

00:58:44 formatting is actually really helpful because.

00:58:47 This is in a code block.

00:58:48 This is a like numbered things are easy to use, then unnumbered text.

00:58:52 If it just gave me a block of text, I wouldn't read it.

00:58:56 And then before when it said props, it gave me a list of the props in bullet

00:59:00 points, which I think is very helpful.

00:59:02 So I actually really like formatting.

00:59:09 What extent do I agree with the following statements?

00:59:12 So I don't use Copilot in my job.

00:59:16 I will refer to things in these applications.

00:59:21 I normally use Python, so I'm a bit more.

00:59:24 I have a bit more knowledge of what's going on,

00:59:27 although I do use tools like this all the time and if.

00:59:31 I want to approach a problem from scratch.

00:59:34 I will use like Claude to say break this problem down for me.

00:59:38 What are the small steps that I need to do and if I have a syntax error I can

00:59:42 throw it in the LLM and it will generally give me a very good answer.

00:59:48 I don't use Copilot so much because the code we look at is on a scale much larger

00:59:54 than this.

00:59:55 So instead of having five files, it has thousands of files and then this

00:59:58 search doesn't work so well,

01:00:00 so to actually answer the question, it does enable me to

01:00:05 accomplished task more quickly.

01:00:08 I would.

01:00:08 I would agree, especially based on this session.

01:00:12 But I don't think it would improve my job or study performance that much.

01:00:17 Although I would replace using LLMs in the way that I use them does help me a

01:00:21 lot.

01:00:23 Using Copilot in.

01:00:23 My job would increase my performance and productivity is actually something that's

01:00:27 interesting.

01:00:28 I think productivity is higher.

01:00:33 I think they're linked, performance productivity.

01:00:37 Performance is how well you do your job.

01:00:38 And productivity, it's like how much?

01:00:41 I think I can write more code with Copilot.

01:00:45 I don't know if it will be as good a quality.

01:00:48 It hurts my effectiveness and then that comes into the same area.

01:00:53 I do think it makes I will give these neutral.

01:00:58 Yeah. And then this one higher.

01:01:01 Make it easier.

01:01:02 I think it does make it easier because I don't understand what hook or a prop is.

01:01:07 Yeah, I can change it.

01:01:08 That makes it a lot easier.

01:01:09 I don't have to look it up.

01:01:11 And it's useful.

01:01:15 Learning to operate Copilot is easy for me.

01:01:17 I think it is very intuitive, especially the chat function.  
01:01:21 I find it easy to get Copilot to do what I want it to do.  
01:01:24 I think with more practice, some of the things I found annoying would  
01:01:29 be quite clear.  
01:01:31 It's clear and understandable, I think, for the most part, yes,  
01:01:34 it's clear and understandable.  
01:01:36 Do I find it flexible?  
01:01:48 I less so much less so than other things I feel like in my mind I have a few.  
01:01:55 Ideas of when I should be using these tools and then I don't stray from it  
01:02:00 like a first draft of a function or finding some functionality.  
01:02:04 But then I don't think I apply it to many other use cases.  
01:02:09 It'd be easier for me to become skillful at using Copilot.  
01:02:14 I think so.  
01:02:15 I think it's quite easy to use.  
01:02:17 Yeah, I think it's easy to use.  
01:02:20 I think it's got a much lower barrier to entry compared to learning, React.  
01:02:24 That's definitely true.  
01:02:28 Mentally demanding was session?  
01:02:30 I actually I very much enjoyed it.  
01:02:34 I wouldn't consider it mentally demanding. I think it's quite nice to solve these  
01:02:37 puzzles.  
01:02:39 As different from my day-to-day work, but it's still thinking so a bit  
01:02:43 demanding, but not not really. How physically demanding.  
01:02:46 I don't feel very physically demanded. How hard or rushed I think.  
01:02:53 I.  
01:02:54 I was probably too.  
01:02:56 I didn't get very far in the questions and maybe it's because I was quite  
01:02:59 relaxed.  
01:03:01 So I didn't feel hurried or rushed, but maybe that's that would be good.  
01:03:05 Let's go with that.  
01:03:07 But that's also me.  
01:03:08 How successful you're accomplishing the tasks you asked to do, I think I did.  
01:03:13 The what? As far as I got in the task list I  
01:03:16 completed and I'm happy I completed them but I didn't get very far so I'd go.  
01:03:23 With with.  
01:03:25 I think I'm successful.  
01:03:26 Let's go two. I would have been much less successful  
01:03:29 if I hadn't had this because I wouldn't have understood what I was doing.  
01:03:34 How hard did you have to work to accomplish your level of performance?  
01:03:37 I feel like I didn't do a lot.  
01:03:40 Because I wasn't so much.  
01:03:45 It's funny.  
01:03:45 I feel like I'm not really writing code as much as I am getting somebody to look  
01:03:50 over my shoulder and help me, which I find.  
01:03:54 Like I'm not working as hard.  
01:03:56 Even though I'm still doing the same amount of work.  
01:04:00 How insecure or discouraged?  
01:04:02 Irritated, stressed or annoyed were you.  
01:04:04 I wouldn't really describe any of these emotions as coming out in the session. I  
01:04:08 quite enjoyed.  
01:04:11 Next.  
01:04:14 Todo I found this.  
01:04:18 I would say moderate.  
01:04:19 I I don't work with these tools on a daily basis,  
01:04:23 so I didn't really know how to approach it,  
01:04:26 but it all made sense once it was in place.

01:04:30 Save drafts I didn't complete, so I would describe it as difficult,  
01:04:34 although I think I understand the idea behind it and what needed to be done and  
01:04:38 I didn't do this one.  
01:04:41 Umm.  
01:04:44 How realistic is you find the tasks in reflecting real life programming  
01:04:47 situations.  
01:04:48 This is something that I have done in different.  
01:04:52 Umm this in web development in different frameworks,  
01:04:56 that's something that's come up for of making things persist between saves and  
01:05:01 state management.  
01:05:03 Saving drafts, I think is I haven't.  
01:05:06 I haven't done, but I also can imagine that would happen.  
01:05:08 And then pagination I've not done, but I know it does come up.  
01:05:14 And that's the end.  
01:05:15 **Researcher**  
Great. Thanks.  
01:05:17 So thanks for the narration as well.  
01:05:19 **Participant**  
Yeah, I realized the survey maybe didn't need  
01:05:21 to be narrated, did it?  
01:05:23 **Researcher**  
Well, actually these are things that I would  
01:05:25 ask some questions about, but I think you narrating them answered a  
01:05:29 lot of them already.  
01:05:30 So it's it's perfect actually.  
01:05:31 **Participant**  
OK, good, good.  
01:05:34 **Researcher**  
So I have some questions prepared and I'll ask some questions specific to your  
01:05:38 session.  
01:05:40 First of all, did you manage to find your way through  
01:05:44 the codebase and how did Copilot influence this?  
01:05:48 **Participant**  
I found.  
01:05:51 The, it makes it much easier to navigate a code base you've never seen.  
01:05:55 Normally I will use the search you saw me use it.  
01:05:58 The magnifying glass search bar to find things,  
01:06:01 but to find those things you need to know the exact string you're looking for.  
01:06:05 So it's like I need to find this todo box.  
01:06:07 I can find the todo box quite easily.  
01:06:10 The other situation was when it was about persistent storage and it's I don't know  
01:06:14 where I would even look for persistent storage at the start.  
01:06:18 I could make up some guesses in the magnifying glass,  
01:06:21 or I could read every file and make an assumption based off the off the name,  
01:06:25 but it's a lot easier for me to say where is this.  
01:06:28 And then Copilot does it for me and just tells me the box.  
01:06:32 So it makes it a lot easier.  
01:06:35 **Researcher**  
Right, so in a real life situation. Would that be your strategy going through  
01:06:38 it manually or would you have another tool for this maybe?  
01:06:43 **Participant**  
I would ask somebody, that's my my always go to thing is.  
01:06:50 I think.  
01:06:52 If the person who wrote it is around, it's much easier to ask them about that  
01:06:55 code, because they're the ones who actually  
01:06:57 know it.

01:06:57 That's the first thing I would do.  
01:07:00 If that, that doesn't always happen.  
01:07:01 So sometimes you have to do it yourself.  
01:07:04 I would.  
01:07:10 I would probably approach the problem as I described or did in that situation.  
01:07:14 I don't think I would use any external tool outside of it.  
01:07:18 **Researcher**  
Right. That makes sense.  
01:07:21 Sort of related.  
01:07:21 Do you feel like you understood the codebase?  
01:07:24 And how did Copilot influence your understanding?  
01:07:27 **Participant**  
Yeah, I have not used React really, but I think the codebase makes sense as  
01:07:32 you have sort of like the main files that run and then all of the elements and then  
01:07:37 the logic behind those elements is stored in different places.  
01:07:42 But I think.  
01:07:45 What I would like to know that I didn't ask is why some of it's JavaScript and  
01:07:49 some of its React and how the props work. And I think I because these concepts were  
01:07:54 all quite new.  
01:07:56 It's nice to have Copilot looking over my shoulder and saying the prop is the thing  
01:08:01 you need to change to, like, get the logic rather than the div  
01:08:05 because.  
01:08:07 Yeah, you don't need to touch the elements.  
01:08:11 Yeah, if I had been given this codebase  
01:08:14 without any assistance in that sense, I wouldn't have known what to do.  
01:08:18 **Researcher**  
All right. I think you mentioned that you liked that  
01:08:22 Copilot used the right technical terms in its answers,  
01:08:25 **Participant**  
Mm hmm.  
01:08:25 **Researcher**  
but that these terms were not familiar to you.  
01:08:31 Would it have been better for you if it used terms that would have been familiar  
01:08:36 to you?  
01:08:37 **Participant**  
I'm, so, I understand that React has props and  
01:08:41 hooks and I understand these are different.  
01:08:44 Whatever the whatever you have in Python it is because otherwise people wouldn't  
01:08:50 give it a different name.  
01:08:52 So I I think it makes sense that it uses different terminology unless you told me  
01:08:57 that this is the equivalent of a variable or this is the equivalent of a lambda  
01:09:01 function, in which case I would say oh, just call  
01:09:04 it a lambda function.  
01:09:06 But then that's not an issue really with Copilot.  
01:09:08 That's an issue with React. Does that make sense?  
01:09:11 **Researcher**  
OK. Yes.  
01:09:13 **Participant**  
I like how it uses the terms.  
01:09:14 I think it's. I've heard people talk about,  
01:09:16 React and say, oh, I spent all night writing,  
01:09:18 writing this book. And the props are really annoying.  
01:09:21 So I know that exists and then they would understand it.  
01:09:26 **Researcher**  
And then these these parallels to, well, domains or languages that you are  
01:09:31 familiar with to explain these terms. Is this something you would want out of

01:09:35

Copilot?

01:09:36

**Participant**

This is something I use.

01:09:38

I've done in my own time, which is I often have to look at C++ code and then I find that hard to interpret because C++.

01:09:42

01:09:47

Has you can do a lot more in C++ than in other languages,

01:09:52

and sometimes I can just give it to an LLM and say can you explain these

01:09:57

concepts in Python terms or is there anything I might recognize or why why when I've implemented something.

01:10:02

Like I would do in Python has this failed.

01:10:05

Due ot, just syntax differences or differences in

01:10:08

how languages handle things.

01:10:11

01:10:14

**Researcher**

Right.

01:10:18

So moving on, so do you have any insights on how you

01:10:22

would have solved these tasks without Copilot and did using Copilot

01:10:27

change your your strategy compared to how you'd normally do it?

01:10:32

**Participant**

Yeah. So.

01:10:35

In the times before Copilot, I would go on Stack Exchange and find

01:10:39

often where somebody has done something similar before,

01:10:43

especially if I've never seen that task. I imagine if I went on Stack Exchange now and said React make a persistent todo list, then there would be.

01:10:48

Something similar.

01:10:52

Then I would read what they have done and then try and reverse engineer it.

01:10:54

I think what you gain is that you don't have to go like browsing the internet.

01:10:59

You can just ask and get an implementation that is already.

01:11:04

That should work, and then even if you have to edit a

01:11:08

little, you're still faster than going online.

01:11:11

Yeah. And I think I do less.

01:11:18

I feel like I do less thinking than I did previously because some some of it's already solved.

01:11:21

But then if it's a problem that's already been solved, why would I want to solve it?

01:11:25

Maybe it's the logic there.

01:11:26

01:11:30

01:11:32

**Researcher**

Yeah, and this this delegation of thinking is

this purely a positive thing for you?

01:11:35

01:11:42

**Participant**

I worry that one day maybe I wouldn't have access to this tool and then I'd

forget what I was meant to be doing and I would struggle more.

01:11:46

So yeah, I guess I and I kind of enjoy and I do

01:11:53

enjoy the problem solving as an aspect of.

01:11:56

Of the programming.

01:12:00

I think that's the that's actually the fundament.

01:12:01

I think if you never got to do anything problem solving related then it would be much less interesting role.

01:12:03

01:12:08

01:12:13

**Researcher**

I guess maybe this connects with you saying using this tool you didn't feel like you were working as hard.

01:12:17

Also, is is this a positive or maybe a negative

01:12:20

experience for you? Or does it have multiple sides?

01:12:24

01:12:29

**Participant**

Probably a negative thing.

I feel like I'm more.

01:12:31

Yeah, I don't.

01:12:33

I don't know why, I can't really describe why in a sense,

01:12:37

01:12:41 but I do see it as a negative that I get that feeling,  
01:12:44 even though objectively I am solving the problem and being productive and  
01:12:48 efficient in the same way, even more so.  
01:12:55 **Researcher**  
I see you talked about some of the aspects of Copilot's responses in the survey.  
01:13:00 Did any of these stand out to you?  
01:13:04 **Participant**  
Yeah, I think I like the formatting that's  
01:13:06 really big.  
01:13:07 I remember when the formatting would be unhelpful and I think the fact that  
01:13:12 sources where the information comes from, I guess that fits under formatting is  
01:13:17 really important because I don't necessarily trust what the output is.  
01:13:23 I like, I read it and I observe it and I go.  
01:13:25 That's probably that's probably valuable, but I need to look at where it's getting  
01:13:29 information about these functions from and that it's not just making them up.  
01:13:34 **Researcher**  
Right. And how would you then validate these  
01:13:37 answers?  
01:13:39 **Participant**  
Uhm.  
01:13:41 By by testing I guess.  
01:13:44 So referring to where it comes from in the code and then reading it and trying  
01:13:48 to understand why the change has been made and then running the change and  
01:13:53 seeing if it works.  
01:13:55 So kind of the same way, where if I had, if I worked with a junior engineer or  
01:13:59 pair programmed, if somebody else wrote the code,  
01:14:02 I would look at the code.  
01:14:04 Go. Why have you done this?  
01:14:06 And then get them to run it and see if it works.  
01:14:09 **Researcher**  
That makes sense.  
01:14:11 I think it also said that sometimes you found Copilot annoying or that you were  
01:14:16 annoyed by Copilot.  
01:14:20 Can you maybe shed some light on this?  
01:14:26 **Participant**  
I think.  
01:14:28 Sometimes what annoys me is if I ask a question and then from the first.  
01:14:32 Sometimes you can see right from the start that it's incorrect or you search  
01:14:37 for the wrong thing and then it's still going and it's like oh why.  
01:14:42 I know this is wrong already and then it's just producing more incorrect  
01:14:45 information.  
01:14:47 Umm.  
01:14:52 But that's not so bad.  
01:14:55 **Researcher**  
And then in what cases?  
01:14:58 Do you think it it gave these incorrect answers?  
01:15:01 **Participant**  
There's one. So there was one answer that I thought  
01:15:04 was incorrect, but I didn't actually validate it  
01:15:06 was incorrect because I just assumed.  
01:15:09 But it was looking in.  
01:15:11 In answer, to be specific in answer two.  
01:15:17 I think you will see it talks about the UI element rather than the logic behind  
01:15:21 it and then tells you about the CSS and then these are things that I know  
01:15:26 wouldn't affect.  
01:15:28 It wouldn't implement the change that I wanted.

01:15:29 It just is going to change how it looks or whatever,  
01:15:33 and then because I know I can read it, I can see straight away that that's wrong.  
01:15:40 I just go, why do this again?  
01:15:42 But I couldn't be that annoyed, really, because then I asked the question in a  
01:15:45 different way.  
01:15:46 And didn't give up.  
01:15:48 So.  
01:15:49 Yeah, if I if I was that annoyed, I would turn it off.  
01:15:53 **Researcher**  
Fair enough.  
01:15:58 Also, you said that you liked.  
01:16:02 The confident tone.  
01:16:04 **Participant**  
Yeah.  
01:16:06 **Researcher**  
In cases where the answer isn't correct, do you still appreciate this confident  
01:16:10 tone?  
01:16:11 **Participant**  
Maybe that's why I find it annoying, but at the same time,  
01:16:14 I know there's nothing you can do about it, really.  
01:16:20 I think it's better to have. I've discussed this before.  
01:16:23 I think it's better to have always confident.  
01:16:27 Than always not confident or always unsure or always in the middle I think.  
01:16:33 Even if it's wrong, you should be able you can.  
01:16:36 You can figure it out despite the confidence, it doesn't matter so much.  
01:16:41 **Researcher**  
Right. So then why would be in confidence be  
01:16:44 better than not being confident in the answer?  
01:16:49 **Participant**  
I guess the way I interpret it is that the Copilot doesn't know how correct the  
01:16:54 answer.  
01:16:55 Is it just gives you an answer that it gives you the most likely answer.  
01:17:00 Based on your input and based on the the results just because because it's the  
01:17:05 most likely answer.  
01:17:08 Doesn't give you. Doesn't give the model any indication of  
01:17:11 how correct the answer is.  
01:17:12 The only way to figure out whether it's correct or not is to test it.  
01:17:17 Um.  
01:17:20 So I feel like from engineering perspective,  
01:17:23 if I was making Copilot because you can't measure how correct it thinks it is,  
01:17:28 you would just set confidence to high rather than confidence to to low.  
01:17:32 I think it's like two things that although as we as people want them to  
01:17:37 correlate.  
01:17:37 They don't correlate.  
01:17:41 **Researcher**  
Yeah, that makes sense.  
01:17:42 **Participant**  
It would be nice in an ideal world, it'd be nice if it wasn't sure that it  
01:17:46 said, oh, I'm not really sure, but maybe this, but I don't know how.  
01:17:49 Maybe in maybe in a couple years that'd be good.  
01:17:52 **Researcher**  
So it's mainly this not being able to know when it should not be confident  
01:17:57 that it should not just.  
01:18:00 Indicates when it's not confidence, yeah.  
01:18:05 I have two more questions.  
01:18:07 **Participant**

01:18:08 Yeah, please.  
**Researcher**  
You opened a new chat a few times.  
01:18:10 Can you explain to me why you did this?  
01:18:13 **Participant**  
Yeah, that is a good question.  
01:18:15 So I when I use these models I know that you have your like context window and  
01:18:21 that it's always reading the information from your new chat.  
01:18:27 So.  
01:18:30 I see it as like a noise thing like the more text you're giving at the start,  
01:18:34 the more random and unlikely and.  
01:18:39 Unexpected. Your final answer will be because if you  
01:18:43 have something completely unrelated and then your actual question,  
01:18:47 then your output answer is less likely to be related to that actual question.  
01:18:55 So I will.  
01:18:55 I will only follow up the chat if it's already something important.  
01:19:00 And then, yeah, if it's anything new, it's just adding noise to the system not  
01:19:04 to.  
01:19:06 Create a new chat and also it's like it's cheaper not to run all these tokens again.  
01:19:11 You're using more computation power. If you just have a big chat open.  
01:19:15 So if there's all, it could be environmental feelings.  
01:19:18 I don't know. You know, there's all sorts of reasons.  
01:19:19 **Researcher**  
That's fair.  
01:19:22 And then this first reason, the confusion and the noise that you  
01:19:26 talked about is this purely from your experience with other tools,  
01:19:31 or did you also observe this with Copilot?  
01:19:34 **Participant**  
So not from Copilot, but I guess I've used LLMs for the last  
01:19:38 like year and a half or two years or however long.  
01:19:42 They have since GPT 4 or 3.5 came out and it was a really big issue  
01:19:47 at the start of these models.  
01:19:49 Now I can use Claude and then have a massive long conversation where I paste  
01:19:54 in something huge and then ask a question about something else and it might not  
01:19:59 have as big an effect, but I think based on my old.  
01:20:04 Actions and like learned, what's the word, learned responses and activities.  
01:20:09 I will always just open a new chat, because even if they make the computation  
01:20:14 really fast then it doesn't.  
01:20:18 It makes less sense to me.  
01:20:20 **Researcher**  
Yeah, that's fair.  
01:20:22 So last but not least, do you feel like you changed the way you  
01:20:24 **Participant**  
Mm hmm.  
01:20:27 **Researcher**  
interacted with Copilot over time?  
01:20:34 **Participant**  
Yup, I think.  
01:20:36 So I've used it in the past and it didn't have that.  
01:20:40 What I learnt today is that has this thing where it sources the text and you  
01:20:43 can just click on it and it will take you to that line.  
01:20:48 That's to me amazing as a feature that you can just say, oh,  
01:20:50 I'll show it on my screen really briefly. So you know what I'm talking about.  
01:20:55 The fact that you have this thing here where it's todo list empty container and  
01:20:59 I can click on it and go oh there it is and I can directly verify this is CSS of  
01:21:04 it.

01:21:04 So it's a bad example. Visible todos so I can directly verify that this is the  
01:21:08 function it's talking about there.  
01:21:11 So.  
01:21:14 I used to look I guess at this text and now I actually look at the code because  
01:21:19 it tells me where in the code I should be looking and then I can read the  
01:21:24 information around it and then refer not to here but.  
01:21:32 But to here.  
01:21:34 **Researcher**  
Yeah.  
01:21:36 **Participant**  
Yeah.  
01:21:40 **Researcher**  
So did you at the start of the session, were you aware of this functionality?  
01:21:45 **Participant**  
No, not aware. I hadn't used the most recent  
01:21:47 version of Copilot, so that's very nice to me.  
01:21:49 **Researcher**  
So did this. Did this change your strategy of  
01:21:52 interacting with Copilot during the session?  
01:21:56 **Participant**  
Yes, it would do.  
01:21:57 Yeah, I wouldn't normally.  
01:22:00 I wouldn't normally trust it as much.  
01:22:03 Let's go with that.  
01:22:03 I wouldn't normally just go.  
01:22:04 Ah, great.  
01:22:05 That answer is correct. Because of this and this,  
01:22:08 I think it makes it makes the way I use it faster and my trust in it much greater.  
01:22:13 If I can say it's rendered when this and then when this array is empty and then  
01:22:13 **Researcher**  
OK.  
01:22:18 **Participant**  
look at it and find.  
01:22:19 The array that's in there, so yeah.  
01:22:23 **Researcher**  
Makes sense.  
01:22:24 Thanks. Those are my questions.  
01:22:27 Do you have anything to add?  
01:22:30 **Participant**  
Uh.  
01:22:31 I think.  
01:22:34 For I think for certain language what we've noticed.  
01:22:36 Maybe it's interesting to you, maybe not for certain languages.  
01:22:39 These LLMs and Copilot are way better.  
01:22:42 So for React, I know it's really good.  
01:22:44 For Python, it's really good.  
01:22:46 And then for other languages like C++.  
01:22:49 It's much more hit and miss.  
01:22:51 Whether that's due to less training data or whatever.  
01:22:55 So I feel like I trust in this situation.  
01:22:58 Partially 'cause I didn't know the language, and partially because I know other people  
01:23:02 I trust have told me that it's very good at React.  
01:23:04 I was much more happy to just, like, be like, yep, I'll throw it in.  
01:23:07 Whereas if I and if I was doing it with Python,  
01:23:10 I would probably be doing the same.  
01:23:12 But if I was using C++ then I would be much more.

01:23:17 Cautious about what it was up to.

01:23:20 **Researcher**  
Right. And and would you in that case use it  
01:23:23 less or try to validate the answers better?

01:23:27 **Participant**  
I would probably use the same amount and validate the answers more.

01:23:31 **Researcher**  
Yep, and. And do you feel like this is a problem  
01:23:34 with any LLM or any specific ones?

01:23:38 **Participant**  
So this is. I didn't even look at which one I was  
01:23:41 using.

01:23:41 I was using GPT 4 here.

01:23:45 I've noticed it with GPT 4 and with Claude.

01:23:49 I've not tested my hypothesis on other LLMs that I think it is kind of a

01:23:54 **Researcher**  
Yeah.

01:23:54 **Participant**  
hypothesis, but yeah.

01:23:57 **Researcher**  
That's fair enough. Thanks.