

Participant P11

00:10:29 **Participant**
The first thing I'm going to try to do is see.

00:10:35 How this codebase interface mentioned in the first pdf works.

00:10:45 Like any interface, here it is in English, including the buttons.

00:10:48 Some things are in English, others are in Italian because I'm

00:10:53 in Italy.

00:10:56 I imagine the copilot responds well in any language I type here,

00:11:01 so I'll try.

00:11:04 To do it in English.

00:11:07 **Researcher**
, Portuguese,

00:11:25 **Chat** (window: 0)
> **Agent**
github.copilot.default
> **Context**
codebase
> **Message**
#codebase please find a file for me containing a string message "Nenhum afazer aqui
ainda"
> **Response**
Il file che contiene la stringa "Nenhum afazer aqui ainda" è:

/workspaces/copilot-study-todo/src/todo/components/main.jsx

Ecco il frammento di codice rilevante:


```
<div className="todo-list-empty-container">  
  <span className="todo-list-empty-message">Nenhum afazer aqui  
ainda</span>  
</div>
```

00:11:27 **Participant**
To experiment with what it does this search,

00:11:30
- 00:11:31 > **Response rendering**

00:11:31 **Participant**
since I don't know the project

00:11:50 > **Viewing file** (src/todo/components/main.jsx)

00:11:56 **Participant**
Well, it doesn't tell me the line, but okay.

00:12:02 What am I going to do now?

00:12:05 So, theoretically, we're talking there at 56 to see the

00:12:11 Switch to this 007 BFF. for this style definition.

00:12:28 > **Viewing file** (src/todo/app.css)

00:12:32 > **Viewing file** (src/todo/components/main.jsx)

00:12:42 **Participant**
I'm trying to find the class here

00:12:45 > **Viewing file** (src/todo/app.css)

00:12:49 **Participant**
This one background

00:12:50 > **Editing file** (src/todo/app.css)

00:13:20 **Participant**
Okay, apparently OK.

00:13:21 **Researcher**
Because of this?

00:13:35 **Participant**
In the verification, it says to check, well,
00:13:41 but I don't know if I have the application running on any link.

00:13:45 **Researcher**
Can you run the app through the terminal
00:14:04 > **Viewing file** (src/todo/package.json)

00:14:22 **Participant**
I'm going to do it here still. Wonderful.
00:14:26 Assuming it was red. I didn't run it before.
00:14:29 But, anyway, patience

00:14:30 **Researcher**
ok.

00:14:32 **Participant**
let's keep going.
00:14:42 Change the text to, type a do here.
00:14:46 this placeholder text, I imagine it's in the same file.
00:14:51 > **Viewing file** (src/todo/components/main.jsx)

00:15:17 **Participant**
I'll ask for help from the copilot again.
00:15:26 How is it what needs to be done?
00:15:35 > **Viewing file** (src/todo/components/header.jsx) through file search

00:15:37 **Participant**
It was really another component.
00:15:42 Police of any situation really.
00:15:46 Type a do here too.
00:15:51 > **Editing file** (src/todo/components/header.jsx)

00:16:40 **Participant**
sort the tasks
00:16:47 modify the application so that they are sorted alphabetically.
00:16:59 So, if I add, the, quick. 123 brown fox will stay.
00:17:03 Theoretically in this order. Let me see.
00:17:23 Well, it seems it's not.

00:17:31 **Researcher**
Is it not adding?

00:17:33 **Participant**
No, then I don't know if this is part of the things
00:18:20 I'll take the opportunity then and ask he's looking, Oh he is. He changes when he's,
00:18:24 when I change the focus of the screen here he already changes automatically any file
00:18:29 that he is understanding. Like Crew of fire.
00:18:33 But I don't know if it's just in this mine, so let's put it.
00:18:39 This is #codeBase.

00:19:00 **Chat** (window: 0)
> **Agent**
github.copilot.default
> **Context**
codebase
> **Message**
#codebase please check whether the callback for adding new items is connected to the
input
> **Response**
Sì, il callback per aggiungere nuovi elementi è collegato all'input. Ecco come funziona:

1. Nel file header.jsx, il componente [Header](#) ([src/todo/components/header.jsx](#)) riceve una funzione [dispatch](#)

([src/todo/components/header.jsx](#)) come prop.

2. La funzione [addItem\(src/todo/components/header.jsx\)](#) è definita utilizzando [useCallback\(node_modules/@types/react/index.d.ts\)](#) per inviare un'azione di tipo [ADD_ITEM\(src/todo/constants.js\)](#) al reducer:

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

3. Questa funzione [addItem\(src/todo/components/header.jsx\)](#) viene passata come prop [onSubmit\(src/todo/components/header.jsx\)](#) al componente [Input\(src/todo/components/input.jsx\)](#):

```
<Input onSubmit={addItem} label="Novo afazer" placeholder="Digite
um afazer aqui" />
```

4. Nel file `input.jsx`, il componente [Input\(src/todo/components/input.jsx\)](#) utilizza la funzione [onSubmit\(src/todo/components/header.jsx\)](#) quando l'utente preme il tasto "Enter":

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

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

      onSubmit(value);
      e.target.value = "";
    }
  },
  [onSubmit]
);
```

Quindi, il callback per aggiungere nuovi elementi è correttamente collegato all'input tramite il flusso [Header -> Input\(src/todo/components/header.jsx\)](#).

00:19:03

Participant

Let's see what he tells me here.

00:19:05
- 00:19:09

> Response rendering

00:19:06

Participant

Meanwhile, I'll take a look.

00:19:17

Uhum. From the header.

00:19:21

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

00:19:22

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

00:19:26

Participant

Makes it seem like an event-oriented thing,
apparently that makes this Ed item.

00:19:31

00:19:35

Which is the type of event.

00:20:00

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

00:20:01

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

00:20:03

Participant

Then, apparently, it should.

00:20:11

Please. Ah, it was fine.

00:20:48

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

00:20:52

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

00:21:03

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

00:21:54

Participant

He's getting the task list here.

00:22:25

Chat (window: 0)

> **Agent**

github.copilot.default

> **Context**

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

> **Message**

Does JS Array.sort work in place or does it return a new array?

> **Response**

Il metodo `Array.sort()` in JavaScript modifica l'array originale in modo **in-place**. Non restituisce una nuova copia dell'array ordinato, ma piuttosto ordina l'array esistente e lo restituisce.

Esempio:

```
const arr = [3, 1, 4, 1, 5];
arr.sort((a, b) => a - b); // Ordina in-place
console.log(arr); // Output: [1, 1, 3, 4, 5]
```

Se vuoi mantenere l'array originale intatto, devi creare una copia prima di ordinarlo:

```
const arr = [3, 1, 4, 1, 5];
const sortedArr = [...arr].sort((a, b) => a - b); // Crea una copia e
la ordina
console.log(arr); // Output: [3, 1, 4, 1, 5]
console.log(sortedArr); // Output: [1, 1, 3, 4, 5]
```

00:22:26

- 00:22:29

> **Response rendering**

00:22:26

Participant

I'm taking the opportunity to research.

00:22:29

It's been a while since I worked with JavaScript, I never, know the sort operation.

00:22:31

Researcher

00:22:35

Cool.

00:22:37

Participant

So I'm researching here for him, it modifies the original.

00:22:39

Researcher

Nice.

00:22:50

Participant

And returns there, Cool? So I can do a sort

00:22:52

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

00:22:55

Participant

here.

00:23:00

Done

00:23:07

it will reset the page. We type there again or

00:23:35

He expected it to come out as 123, brown, the, quick, fox.

00:23:39

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

00:23:40

Participant

It didn't work.

00:23:57

So, since I made a change that didn't

00:24:01

work, I'll research

00:24:06

and see what it tells me.

00:24:11

It might be eventually wrong here.

00:24:14

I'll explain to the copilot, so I put this method to

00:24:17

00:24:19

00:25:02

ensure.

That if I put it with sort

Chat (window: 0)

> **Agent**

github.copilot.default

> **Context**

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

> **Message**

On line 19, I added the `.sort()` method to ensure lexicographical sorting of the todo list, but apparently it doesn't work. What could I do differently? Consider that I might be wrong about the positioning of the method in the file

> **Response**

Il problema è che il metodo `.sort()` modifica l'array originale **in-place**, e poiché [todos \(src/todo/components/main.jsx\)](#) è probabilmente un array passato come prop, modificarlo direttamente può causare comportamenti inaspettati o bug. Inoltre, `.sort()` senza un comparatore non garantisce un ordinamento corretto per oggetti complessi come i tuoi [todo \(src/todo/components/main.jsx\)](#).

Ecco cosa puoi fare per risolvere il problema:

1. Crea una copia dell'array prima di ordinarlo

Invece di ordinare direttamente [todos \(src/todo/components/main.jsx\)](#), crea una copia utilizzando lo spread operator (`...`) o il metodo `.slice()`:

```
const visibleTodos = useMemo(
  () => [
    ...todos // Crea una copia dell'array
    .sort((a, b) => a.title.localeCompare(b.title)) // Ordina
    per titolo in modo lessicografico
    .filter((todo) => {
      if (route === "/active") return !todo.completed;
      if (route === "/completed") return todo.completed;
      return todo;
    }),
    [todos, route]
  ],
  [todos, route]
);
```

2. Assicurati che il comparatore sia corretto

Se vuoi ordinare i [todo \(src/todo/components/main.jsx\)](#) in base a una proprietà specifica (ad esempio `title`), usa un comparatore come `localeCompare` per stringhe o una funzione personalizzata per altri tipi di dati.

3. Posizionamento del `.sort()`

Il metodo `.sort()` deve essere posizionato **prima** del `.filter()` per garantire che l'array venga ordinato prima di essere filtrato. Questo è già corretto nel tuo codice.

4. Evita di modificare direttamente i props

Se [todos \(src/todo/components/main.jsx\)](#) è un prop, non modificarlo direttamente. Creare una copia come mostrato sopra risolve questo problema.

Codice aggiornato

Ecco il codice corretto per [visibleTodos \(src/todo/components/main.jsx\)](#):

```
const visibleTodos = useMemo(  
  () =>  
    [...todos] // Crea una copia per evitare di modificare l'array  
    originale  
    .sort((a, b) => a.title.localeCompare(b.title)) // Ordina  
    per titolo  
    .filter((todo) => {  
      if (route === "/active") return !todo.completed;  
      if (route === "/completed") return todo.completed;  
      return todo;  
    }),  
  [todos, route]  
);
```

Con questa modifica, l'ordinamento dovrebbe funzionare correttamente senza effetti collaterali.

00:25:03
- 00:25:08

> Response rendering

00:25:09

Participant

Apparently, that was the point

00:25:23

Ah, OK.

00:25:40

Second alternative here, apparently has more potential.

00:25:47

ok

00:25:58

I thought the default sort would already do it,

00:26:08

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

00:26:20

Participant

Alright, now it's working.

00:26:48

Researcher

To go to the other tasks of the other PDF.

00:26:53

Participant

So, the first thing I saw here, that I can do in the order I find
most appropriate, let me take a quick look first.

00:26:57

00:26:59

Researcher

ok.

00:27:00

Participant

general way, then.

00:27:30

So, I imagine the functionality that comes to

00:27:33

mind is to use the browser's own local store as a kind of cache.

00:27:46

To-dos can be checked, put something, reload the page,

00:27:52

ensure they are there, edit, reload, alright, so theoretically,

00:27:58

here it's telling me that add, edit and delete all have to trigger this

00:28:04

repository update trigger

00:28:07

We'll have to.

00:28:10

take a look.

00:28:11

How is this implementation attempt

00:28:13

Things explore there a little bit.

00:28:16

Save draft if the user reloads for me when typing,

00:28:20

I won't make the text in the input field disappear.

00:28:27

this also triggers the new field. Same thing type, reload, it has.

00:28:31

To stay there?

00:28:34

And when it's good, when creating a new to-do,

00:28:38

you have to clear there and clear the Cache

00:28:45

If I have a lot of stuff, it will paginate 13 or 5.

00:28:59

Alright.

00:29:01

I think honestly, I won't leave pagination for last.

00:29:06

Researcher

ok

00:29:07

Participant

I'll try to start by saving the draft,

00:29:12

because as I, I don't have experience with this part of
persistence, maybe it's the simplest part.

00:29:16

a single field as a task. Need to do more stuff.

00:29:20

My eye went straight

00:29:38

to the directories to take a look if there was any.

00:29:41

file that could already tell me something?

00:29:47

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

00:29:48

00:30:03

Participant

Here, I think it would be this first attempt,

00:30:07

of implementation.

00:30:12

How are we going to mess with the Header? Yeah.

00:30:15

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

00:30:18

Participant

That's where we type the task.

00:30:23

task b wants to save the draft. I imagine it will be in it that the

00:32:17

so to save the drafts, I am imagining that a good

00:32:22

strategy would be to do, just like a Callback on on submit

00:32:26

maybe do a callback on on update, something like that?

00:32:32

But I confess that with react I don't have much certainty

00:32:41

If it already gives me this for these components, like this input component,

00:32:45

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

00:32:46

Participant

or if I need to go down into the input component to define this

00:32:50

behavior as a kind of prop or something like that.

00:32:55

I think the first thing I do at this moment, for .

00:32:58

Ease is to ask here for copilot

00:33:01

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

00:33:14

Participant

Let's go. First, I will explain to him what

00:33:16

I want to do.

00:34:12

Chat (window: 1)

> **Agent**

github.copilot.default

> **Context**

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

> **Message**

I'd like to add a new callback to the input on line 12 that, for each keystroke (different than a submit with Enter) it does some action I define externally to the component.

How to do so?

> **Response**

To add a callback for each keystroke in the [Input \(src/todo/components/input.jsx\)](#) component, you can pass a new prop (e.g., onChange) to the [Input \(src/todo/components/input.jsx\)](#) component. This prop will handle the action you define externally. Here's how you can modify the [Header \(src/todo/components/header.jsx\)](#) component:

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

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

export function Header({ dispatch, onChange }) { // Add
  onChange as a prop
  const addItem = useCallback((title) => dispatch({ type: ADD_ITEM,
    payload: { title } })), [dispatch]);

  return (
    <header className="header" data-testid="header">
      <h1>Afazeres</h1>
      <Input
        onSubmit={addItem}
        onChange={onChange} // Pass the onChange
callback to Input
        label="Novo afazer"
        placeholder="Digite um afazer aqui"
      />
    </header>
  );
}
```

Explanation:

1. **Prop Addition:** Add a new `onInputChange` prop to the [Header](#) (`src/todo/components/header.jsx`) component. This prop will be passed down to the [Input](#) (`src/todo/components/input.jsx`) component.
2. **Input Component Update:** Ensure the [Input](#) (`src/todo/components/input.jsx`) component supports an `onChange` prop and calls it for each keystroke. If it doesn't already, you may need to modify the [Input](#) (`src/todo/components/input.jsx`) component to handle this.

If you need help updating the [Input](#) (`src/todo/components/input.jsx`) component, let me know!

00:34:13
- 00:34:16

> Response rendering

00:34:53

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

00:36:48

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

00:36:56

Participant

I will put a prompt.

00:37:49

asking copilot to explore here

00:37:52

the repository to see if my guess is right,

00:37:55

because if it is, I will continue exploring.

00:37:58

If not, I will try to find out.

00:38:05

Researcher

OK.

00:38:14

Chat (window: 1)

> **Agent**

github.copilot.default

> **Context**

codebase

> Message

#codebase As it seems, there is a websocket running communications between a persistence component and the interface. It seems that the communication with the socket happens with this dispatch function. Can you check the project to see if that is correct for the addItem callback?

> Response

Based on the provided codebase, the [dispatch \(src/todo/components/header.jsx\)](#) function in the [Header \(src/todo/components/header.jsx\)](#) component is used to handle actions related to the application's state. Specifically, in the [addItem \(src/todo/components/header.jsx\)](#) callback, it dispatches an action of type [ADD_ITEM \(src/todo/constants.js\)](#) with a payload containing the [title \(src/todo/components/header.jsx\)](#) of the new item. This action is processed by the [todoReducer \(src/todo/reducer.js\)](#) in `reducer.js`, which updates the state accordingly.

However, there is no direct evidence in the codebase that [dispatch \(src/todo/components/header.jsx\)](#) communicates with a WebSocket. The [dispatch \(src/todo/components/header.jsx\)](#) function appears to be part of a [useReducer \(node_modules/@types/react/index.d.ts\)](#) setup, as seen in `app.jsx`:

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

The [todoReducer \(src/todo/reducer.js\)](#) updates the state locally and persists it to [localStorage \(node_modules/@types/node/globals.d.ts\)](#) using the [saveState \(src/todo/reducer.js\)](#) function in `reducer.js`. There is no indication of WebSocket communication in the reducer or elsewhere in the provided codebase.

If WebSocket communication is intended, it is not implemented in the current codebase. You would need to integrate a WebSocket client (e.g., WebSocket API or a library like `socket.io-client`) and modify the [dispatch \(src/todo/components/header.jsx\)](#) function or the [addItem \(src/todo/components/header.jsx\)](#) callback to send data to the WebSocket server.

Let me know if you'd like help adding WebSocket functionality!

> Response rendering

Participant

So it makes sense to be here.

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

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

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

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

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

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

Participant

I am thinking that actually, it's better to go back to doing the task, to familiarize myself better with these operations, because, apparently,

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

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

Participant

they are already implemented.

I will ask copilot.

Chat (window: 1)

> Agent

github.copilot.default

> Context

selection (src/todo/reducer.js)

00:38:19
- 00:38:24

00:38:45

00:38:50

00:39:34

00:39:34

00:39:43

00:40:03

00:40:18

00:40:38

00:40:43

00:40:44

00:40:46

00:40:47

00:41:04

00:41:52

> Message

Would you say these operations implemented in the reducer covers solely the CRUD of todo items? Or it goes further and has also code for drafting the input before the user hits enter?

> Response

The operations implemented in the [todoReducer \(src/todo/reducer.js\)](#) focus solely on the **CRUD operations** for managing the todo items and some additional state management for toggling and filtering items. Here's a breakdown:

CRUD Operations:

1. **Create:** [ADD_ITEM \(src/todo/constants.js\)](#) adds a new todo item with a unique ID, title, and [completed \(src/todo/reducer.js\)](#) status.
2. **Read:** The state is loaded from [localStorage \(node_modules/@types/node/globals.d.ts\)](#) using [loadState \(src/todo/reducer.js\)](#) and saved back using [saveState \(src/todo/reducer.js\)](#).
3. **Update:** [UPDATE_ITEM \(src/todo/constants.js\)](#) updates the title of an existing todo item.
4. **Delete:** [REMOVE_ITEM \(src/todo/constants.js\)](#) removes a specific todo item by its ID.

Additional State Management:

- **Toggle:** [TOGGLE_ITEM \(src/todo/constants.js\)](#) toggles the [completed \(src/todo/reducer.js\)](#) status of a specific todo item.
- **Bulk Operations:**
 - [REMOVEALLITEMS \(src/todo/constants.js\)](#) clears all items.
 - [TOGGLE_ALL \(src/todo/constants.js\)](#) toggles the [completed \(src/todo/reducer.js\)](#) status of all items.
 - [REMOVECOMPLETEDITEMS \(src/todo/constants.js\)](#) removes all completed items.
- **Load State:** [LOAD_STATE \(src/todo/constants.js\)](#) replaces the current state with a new one (e.g., from an external source).

Drafting Input:

There is **no code in the reducer** for handling drafting input before the user hits Enter. Drafting input is likely managed locally in the Header component (e.g., via the `updateDraft` callback) or within the Input component itself. The reducer does not handle transient states like drafts, as it focuses on persistent state changes.

If you want to manage the draft state globally, you would need to add a new action (e.g., `UPDATE_DRAFT`) and corresponding logic in the reducer. Let me know if you'd like help with that!

> Response rendering

Participant

I asked him if he covered only the CRUD.

or if he already had something to

save the draft

from what he is saying, no, there is nothing to,

Save the draft in this BUILD.

I will go back to the code I had done and I will try to correct this

attempt here that already exists in the task. AI let's see what happens.

Researcher

ok.

00:41:56
- 00:42:00

00:42:14

00:42:20

00:42:26

00:42:30

00:42:34

00:42:38

00:42:44

00:42:50

00:42:54 > **Viewing file** (src/todo/components/header.jsx)
00:42:59 **Participant**
See you.
00:43:05 > **Viewing file** (src/todo/reducer.jsx)
00:43:24 > **Editing file** (src/todo/reducer.jsx)
00:43:29 **Participant**
I will see if this
00:43:32 Is enough to solve
00:43:34 Got it.
00:43:47 So something is missing. I think this REDUCE is only dealing with
00:43:49 this
00:43:55 > **Viewing file** (src/todo/index.jsx)
00:44:07 > **Viewing file** (src/todo/app.jsx)
00:45:06 **Participant**
Ask him here how do I open the local store through the
00:45:10 browser?
00:45:15 **Chat** (window: 1)
> **Agent**
github.copilot.default
> **Context**
viewport (src/todo/reducer.js)
> **Message**
I'd like to have a view on the localStorage state during the execution using the browser
developer tools
> **Response**
To view the [localStorage \(node_modules/@types/node/globals.d.ts\)](#) state during
execution using the browser developer tools, follow these steps:

Steps:

1. Open Developer Tools:

- In your browser, press F12 or Ctrl+Shift+I (Windows/Linux) or Cmd+Option+I (Mac) to open the developer tools.

2. Go to the "Application" Tab:

- In the developer tools, navigate to the "Application" tab.

3. Locate [localStorage \(node_modules/@types/node/globals.d.ts\)](#):

- In the left-hand sidebar, under **Storage**, expand the "Local Storage" section.
- Select your application's domain (e.g., http://localhost:3000 if running locally).

4. View and Edit [localStorage \(node_modules/@types/node/globals.d.ts\)](#):

- You will see a table with key-value pairs stored in [localStorage \(node_modules/@types/node/globals.d.ts\)](#).
- Look for the key todos (used in your [loadState \(src/todo/reducer.js\)](#) and [saveState \(src/todo/reducer.js\)](#) functions).
- The value will be a JSON string representing the current state of your todo list.

5. Inspect Changes in Real-Time:

- Perform actions in your application (e.g., adding, updating, or removing todos).
- Observe how the todos value in [localStorage \(node_modules/@types/node/globals.d.ts\)](#) updates in real-time.

6. Manually Edit [localStorage \(node_modules/@types/node/globals.d.ts\)](#) (Optional):

- Double-click the value of the todos key to edit it.
- Modify the JSON string and press Enter to save changes.
- Refresh the page to see how your application reacts to the updated state.

Debugging Tip:

If you want to log the [localStorage \(node_modules/@types/node/globals.d.ts\)](#) state to the console during execution, you can add a `console.log` statement in your reducer:

```
const saveState = (state) => {  
  console.log("Saving state to localStorage:", state); // Debugging  
  log  
  localStorage.setItem("todos", JSON.stringify(state));  
};
```

This will print the state to the console every time it is saved.

00:45:16
- 00:45:23

00:45:51

00:46:17

00:47:16

00:47:36

00:48:11

> Response rendering

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

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

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

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

Chat (window: 1)

> Agent

github.copilot.default

> Context

codebase

> Message

#codebase how can I grab the initial state of the reducer and set it as the initial list in the page?

> Response

To grab the initial state of the reducer and set it as the initial list on the page, you can use the [initialState \(src/todo/reducer.js\)](#) exported from the reducer file and pass it to the [useReducer \(node_modules/@types/react/index.d.ts\)](#) hook in the [App \(src/todo/app.jsx\)](#) component. Here's how you can ensure this is done:

Steps:

1. **Verify the [initialState \(src/todo/reducer.js\)](#) Export:** The [initialState \(src/todo/reducer.js\)](#) is already exported from the reducer file and is loaded from [localStorage \(node_modules/@types/node/globals.d.ts\)](#):

```
export const initialState = loadState();
```

2. **Pass [initialState \(src/todo/reducer.js\)](#) to [useReducer \(node_modules/@types/react/index.d.ts\)](#):** In the [App \(src/todo/app.jsx\)](#) component, ensure that [initialState \(src/todo/reducer.js\)](#) is passed as the second argument to [useReducer \(node_modules/@types/react/index.d.ts\)](#):

```
// filepath: /workspaces/copilot-study-todo/src/todo/app.jsx
import { useReducer } from "react";
import { Header } from "../components/header";
import { Main } from "../components/main";
import { Footer } from "../components/footer";

import { todoReducer, initialState } from "../reducer"; // Import initialState

import "../app.css";

export function App() {
  const [todos, dispatch] = useReducer(todoReducer,
    initialState); // Use initialState here

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

3. **Ensure [loadState \(src/todo/reducer.js\)](#) Works Correctly:** The [loadState \(src/todo/reducer.js\)](#) function in the reducer already retrieves the state from [localStorage \(node_modules/@types/node/globals.d.ts\)](#). If [localStorage \(node_modules/@types/node/globals.d.ts\)](#) is empty, it initializes an empty array:

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

4. **Test the Application:**

- Add some todos to the list.

- Refresh the page to confirm that the initial state is correctly loaded from [localStorage\(node_modules/@types/node/globals.d.ts\)](#) and displayed on the page.

This ensures that the initial state of the reducer is used as the initial list on the page. Let me know if you encounter any issues!

> Response rendering

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

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

Participant

I can't understand why it's not working, so in this case I'll ask here to the copilot.

Researcher

[omitted], I'm going to ask you to stop now.

Participant

ok.

Researcher

40 minutes have passed

Now I'm going to send you a link.

please, respond to this questionnaire

Participant

OK?

Researcher

ok, now I'm going to ask you a few more questions

Were you able to orient yourself through the code, and how did the copilot influence you in this?

Participant

So, in a very objective way, yes.

This was one of the first things I thought the copilot could be very useful,

because although I know a little about javascript,

but I've been away for a while from react. My knowledge is

at least, 3 years old.

Being a new project, the first thing I asked was

for the copilot. More than doing a textual search,

being able to explain some things to me, so exploring the code,

I think it was essential. I actually didn't even hesitate.

First thing.

that I thought, I'll go straight to it. In the second search I did,

it was manual.

I think AI was very important in this first

stage

Researcher

OK. And when did you feel that you understood the basis of the

code and how did the copilot influence your understanding?

Friend of the code base.

Participant

I don't

clearly remember how the interaction was in detail, but

I think that the interaction with copilot

sped up.

Made it faster for me to understand some files,

the division of some things in the project.

Researcher

OK, third question.

what would be your approach to

solve the tasks without using copilot?

01:00:54

Participant

I think it would be a bit more laborious,
because as I mentioned about React,
it's been at least 3 years since I worked with it. So, as it's a language that is constantly evolving

01:01:17

I would have to research the documentation.

01:01:20

maybe look for a tutorial for example,

01:01:36

Anyway, I would probably open and do a search using Google

01:01:42

Or maybe see a smaller project, or even create a new one,

01:01:53

to better understand the React interface.

01:02:21

I'm not the type of programmer who actively participates in communities

01:02:25

so really my, my first resource would be to look for things

01:02:30

on Google, because I wouldn't have a support

01:02:33

community. So, eventually

01:02:37

at a later time

01:02:38

I would join a more specific community.

01:02:40

If the difficulty persisted

01:02:42

in understanding how some part of the code works.

01:03:10

Researcher

And In the questionnaire, there is a question that talked about the
aspects you liked or didn't like about Copilot?

01:03:16

can you explain better what you

01:03:21

liked, and what you didn't like?

01:03:25

01:03:36

Participant

I think that maybe what immediately stands out

01:03:40

to me is the interaction. When you do a search,

01:03:45

I think the interaction interface is quite.

01:03:50

Clear, and it's quite simple to understand what

01:03:53

is happening in which code I am altering.

01:03:54

Researcher

OK.

01:04:06

Participant

The answers are structured in a visual language that the programmer is used to
as if it were a tutorial, it uses Markdown.

01:04:09

So it creates code boxes, this makes a difference.

01:04:13

It's visual

01:04:17

It shows the reasoning that was used, different from what is a piece of
code.

01:04:20

01:04:24

The highlights of Markdown I think are quite interesting.

01:04:27

The chat interface itself.

01:04:30

It. It is very natural to understand what are

01:04:32

the contexts being considered. So,

01:04:35

for me it was very easy to see what is the file he is using to give

01:04:39

the context or if I used that hashtag #codebase.

01:04:44

And then these changes. The settings seemed very natural.

01:04:53

The interaction with it, given that, at least for me, I,

01:04:59

tend to keep these windows a bit smaller for my code,

01:05:03

always having the largest possible space.

01:05:07

I think the volume of text was also reasonable.

01:05:09

For the interfaceinterface.

01:05:12

in the sense that it is not too verbose

01:05:14

to the point of always having to scroll the chat to see.

01:05:17

but it is also not just. Oh.

01:05:21

The idea of conducting reasoning. I think it is also very interesting for

01:05:34

me. I can have a critical analysis

01:05:39

of what it is doing, because from my experience with these

01:05:41

01:05:45 GPTS, LLMs.
01:05:51 I have a feeling that sometimes they lose focus a bit
01:05:59 the prompt needs to be well written so that the AI
01:06:03 can give me a good answer
01:06:06 Maybe my prompt is shallow. And then the answer sometimes becomes very
01:06:10 broad or or strays a bit from the focus that I am wanting
01:06:45 From the interface of the interaction.
01:06:49 I really liked that.
01:09:19 A way I use the chat in my daily life is that after 3 interactions where I don't get the answer
I'm looking for. I reverse roles with the AI.
01:09:22 **Researcher**
Ok. During the interaction with the copilot, you changed the approach,
01:09:34 **Participant**
I think I ended up.
01:09:35 changing a bit.
01:09:39 In the sense that, again, maybe it's a matter of my very specific
01:09:43 profile. I think at the beginning. I was very proud,
01:09:47 trying to ask it. To explain things to me,
01:09:51 but I wanted to feel that I was in control.
01:09:58 in the middle of the interaction, I was thinking.
01:10:01 I could have copied the task and put it there in the prompt and told it to
01:10:05 solve it
01:10:11 I didn't do that.
01:10:39 I was asking.
01:10:42 Help me with this function. What do I need to do?
01:10:44 How does it work?
01:10:45 Where do I define it?
01:10:46 How do I pull this data? But I could have thrown the problem directly to it
01:10:50 already
01:10:57 I feel that throughout the in.
01:10:59 I was leaning more towards this side of throwing the problem for it to
01:11:02 solve.
01:11:04 But still with some resistance.
01:11:22 But later towards the end, maybe if we had,
01:11:24 instead of 40 minutes, had, I don't know, 1 hour and a half.
01:11:27 I think I would end up migrating.
01:11:29 to another approach. Passing the entire problem to the copilot to solve
01:11:32 was going in that direction.
01:11:35 **Researcher**
OK, thank you.