

Participant P27

00:02:48 **Researcher**
Can you start?

00:03:02 **Participant**
OK.

00:03:40 > **Viewing file** (src/todo/components/main.jsx) through file search

00:04:17 **Participant**
Is the application working?

00:04:20 **Researcher**
yes.

00:04:52 > **Viewing file** (src/todo/app.css)

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

00:05:16 > **Viewing file** (src/todo/app.css)

00:05:49 > **Editing file** (src/todo/app.css)
- 00:05:49

00:06:00 **Participant**
I'll save and test

00:06:09 **Researcher**
Ok, you can go to the next one.

00:06:30 > **Viewing file** (src/todo/components/header.jsx) through file search

00:06:50 > **Editing file** (src/todo/components/header.jsx)
- 00:06:50

00:08:46 > **Viewing file** (src/todo/components/item.jsx) through file search

00:09:14 > **Viewing file** (src/todo/components/main.jsx) through file search

00:09:18 **Researcher**
What are you thinking now?

00:09:20 **Participant**
Ah, sorry, it's not. My idea is to find where it's being done.
The for of the tasks.
for now the tasks are very simple.
for me to use the copilot
Maybe I'll use it in tasks that demand more repetitive things
Maybe in the sorting task

00:10:22 **Researcher**
ok

00:10:48 **Participant**
It's good here I am seeing what the code does
I'll take a look here to see if it's complex.

00:10:52 **Researcher**
ok.

00:10:59 **Participant**
Is it this one here?
let me see if this object is complex
I can ask the copilot to organize it for me.

00:11:04 > **Viewing file** (src/todo/components/item.jsx) through file search

00:11:22 **Researcher**
ok

00:11:51 > **Editing file** (src/todo/components/main.jsx)
- 00:12:26

00:12:52 **Chat** (window: 0)
> **Agent**
github.copilot.default
> **Context**

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

> Message

how to sort an object by attr text

> Response

To sort an array of objects by the text attribute, you can use the JavaScript `.sort()` method. Assuming each object has the text property, do it like this:

```
// Assuming an array of objects called todos
const sortedTodos = todos.slice().sort((a, b) =>
a.text.localeCompare(b.text));
```

- [todos.slice\(\)](#) (src/todo/components/main.jsx) creates a copy of the array so as not to modify the original.
- `localeCompare` correctly compares strings for alphabetical sorting.

If you want descending order, switch to `b.text.localeCompare(a.text)`.

> Response rendering

Participant

It's this here.

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

Participant

It does it by itself

Nothing.

I broke something here.

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

Participant

I can.

Search on Google

something faster.

Researcher

you can,

Participant

ok

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

Chat (window: 0)

> Agent

github.copilot.workspace

> Context

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

> Message

@workspace there is any sort function in the project?

> Response

Yes, there is a sort function used in the project. In the file [src/todo/components/main.jsx](#), within the component [Main](#) ([src/todo/components/main.jsx](#)), the following line uses the [sort](#) ([src/todo/components/main.jsx](#)) method:

```
{visibleTodos.sort((a, b) => a.text < b.text).map((todo, index) => (
  <Item todo={todo} key={todo.id} dispatch={dispatch} index={index}
/>
))}
```

However, it is worth noting that the comparison function passed to [sort](#) ([src/todo/components/main.jsx](#)) `((a, b) => a.text < b.text)` is not correct for sorting, as it returns a boolean value instead of a negative number, zero, or positive number. The correct approach would be something like:

00:12:54
- 00:12:56

00:13:05

00:13:15
- 00:13:15

00:13:29

00:13:37

00:13:52

00:14:06
- 00:14:14

00:14:10

00:14:21

00:14:24

00:14:25

00:14:29

00:14:52
- 00:14:54

00:17:22

```
(a, b) => a.text.localeCompare(b.text)
```

Additionally, there is no other custom sort function implemented in other files of the project.

00:17:27
- 00:17:33

> **Response rendering**

00:17:45

Participant

Yeah, it found exactly what I.

00:17:49

Researcher

I had done.

00:17:50

Participant

Yeah, but let's see. Actually, it was a little different.

00:17:52
- 00:18:43

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

00:18:00

Participant

This is now fine.

00:19:15

And 2 times to edit.

00:19:34
- 00:19:48

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

00:19:37

Participant

this comparison should already work. It was the same.

00:19:42

Researcher

OK.

00:19:46

Participant

That command, I don't know.

00:19:50

Let me see here.

00:20:04

I'm messing in the wrong place.

00:20:17
- 00:21:07

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

00:21:38

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

00:21:54

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

00:22:06

Participant

But where is it being used

00:22:14

It's here, let me see inside it.

00:22:15

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

00:22:42

Participant

Let me try to see if it helps me here.

00:23:11

I'll use it this way

00:23:28

Chat (window: 0)

> **Agent**

github.copilot.workspace

> **Context**

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

> **Message**

@workspace I need the task list to be sorted alphabetically

> **Response**

To sort the task list alphabetically, change the sorting function in the component [Main \(src/todo/components/main.jsx\)](#) to use `localeCompare` with the correct property ([title \(src/todo/components/header.jsx\)](#)). Here's how to do it:

```
File: src/todo/components/main.jsx
// ...existing code...
<ul className={classnames("todo-list")} data-testid="todo-list">
  {visibleTodos
    .sort((a, b) => a.title.localeCompare(b.title))
    .map((todo, index) => (
      <div key={todo.id}>
        <p>{todo.title}</p>
        <Item todo={todo} dispatch={dispatch} index={index} />
      </div>
    ))
  }
</ul>
```

```
    </div>
  )})
</ul>
// ...existing code...
```

This way, the tasks will be displayed in alphabetical order by title.

00:23:40
- 00:23:42

> Response rendering

00:24:13

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

00:24:14
- 00:24:19

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

00:26:06

Participant

Yes, just see if it worked.

00:26:39

Researcher

OK.

00:26:41

Thank you. Now I'll send you the next tasks

00:26:51

You can solve them in the order you think best. Move to the next task only after validating all the requirements,

00:27:17

Participant

ok

00:27:52

Right, persistence task

00:28:25

I'm thinking of using localStorage

00:28:47

I'll check how the implementation is

00:29:07

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

00:29:11

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

00:29:18

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

00:30:13

> Viewing file (src/index.js)

00:30:20

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

00:30:42

Participant

There's an error here

00:30:47

They are also, is it part of the evaluation or.

00:31:08

it's here.

00:31:26

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

00:31:32

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

00:32:12

Participant

I'm checking first where

00:32:14

it's coming from.

00:32:15

This variable

00:32:25

Researcher

OK.

00:32:27

Participant

Apparently, there isn't.

00:32:30

I didn't understand if there's something already implemented here, to persist.

00:32:39

Researcher

Right.

00:32:43

Participant

In this case, I like to use copilot to help me

00:32:51

I can get the answer faster

00:32:55

Researcher

ok.

00:33:50

Chat (window: 0)

> Agent

github.copilot.default

> Context

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

> Message

Currently, all tasks disappear when the page is reloaded. Modify the application so that the tasks remain even after a reload. Ensure that the state of each task (whether it is completed or active) is also preserved. An attempt to implement this functionality was made, but it is currently not working correctly. Fix or reimplement the functionality.

> Response

To ensure that the tasks remain after the page reloads, you must save and retrieve the state of the tasks from `localStorage`. Ideally, this should be done in the highest-level component that controls the state of the tasks (usually the App component or where the state [all \(src/todo/components/main.jsx\)](#) is defined).

Here is an example of how to correctly implement this functionality:

1. Save the tasks in `localStorage` whenever they change

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

2. Load the tasks from `localStorage` on initialization

In the component where the state [all \(src/todo/components/main.jsx\)](#) is defined (probably in your App.jsx):

```
File: src/App.jsx
import { useState, useEffect } from "react";
// ...existing code...

function App() {
  const [todos, setTodos] = useState(() => {
    const saved = localStorage.getItem("todos");
    return saved ? JSON.parse(saved) : [];
  });

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

  // ...rest of your code...
}
```

3. Ensure that the state of the tasks (completed/active) is being saved

If you are already saving the complete array of [todo objects](#) ([src/todo/components/main.jsx](#)), including the [completed property](#) ([src/todo/components/main.jsx](#)), this is already being done.

Summary:

- Initialize the state of the tasks by reading from `localStorage`. - Whenever the tasks change, save them again in `localStorage`.

If you need help locating the correct file or adapting it for your project, send the file where the state [all](#) ([src/todo/components/main.jsx](#)) is defined!

00:33:53
- 00:33:58

> Response rendering

00:34:42

Participant

2 clicks to edit

00:35:02

I'll see this in the code.

00:35:02

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

00:35:06

Participant

What did he say?

00:35:58

I think the copilot didn't use the context

00:36:04

Researcher

ok.

00:36:08

Participant

I don't know exactly how it works, but here it might cause a problem.

00:36:13

So I'm going to ask it to do it again.

00:36:17

Researcher

cool.

00:36:23

Chat (window: 0)

> Agent

github.copilot.workspace

> Context

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

> Message

@workspace Currently, all tasks disappear when the page is reloaded. Modify the application so that the tasks remain even after a reload. Ensure that the state of each task (whether it is completed or active) is also preserved. An attempt to implement this functionality was made, bu

> Response

To ensure that the tasks persist after the page reloads, you need to fix the state initialization in the [App\(src/todo/app.jsx\)](#) to use the state saved in localStorage. The file [reducer.js\(src/todo/reducer.js\)](#) already has the function [loadState\(src/todo/reducer.js\)](#), but it is saving and reading with different keys ("todoS" and "todos"). Additionally, the reducer is exporting [initialState\(src/todo/reducer.js\)](#), but it is not being used in the [useReducer\(node_modules/@types/react/index.d.ts\)](#).

Make the following changes:

1. Correct the localStorage key in [loadState\(src/todo/reducer.js\)](#) to "todos" (same as [saveState\(src/todo/reducer.js\)](#)).
2. In the [App\(src/todo/app.jsx\)](#), initialize the state of [useReducer\(node_modules/@types/react/index.d.ts\)](#) with [initialState\(src/todo/reducer.js\)](#) imported from the reducer.

Here's how it looks:

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

File: src/todo/app.jsx
// ...existing code...
import { todoReducer, initialState } from "../reducer";
// ...existing code...
export function App() {
  const [todos, dispatch] = useReducer(todoReducer, initialState); //
  initializes with initialState
  // ...existing code...
}
```

With this, the tasks and their states will be preserved after reloading the page.

> Response rendering

00:36:28

- 00:36:32

00:36:56

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

00:37:14

Participant

correct for all. There's a small error here.

00:37:21

let's see how it is

00:37:49

- 00:37:50

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

00:38:09

Participant

Let me just see what he did here.

00:38:14

Ah, I see.

00:38:16

Yeah, it's here. Usually, I try to understand what he did

00:38:21

Because since I don't have the context of the app

00:38:22

Researcher

OK

00:38:30

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

00:38:31

Participant

He helped a lot now, after reading these tips here I understood

00:38:38

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

00:38:39

Participant

This start of the 'state' was missing, which is what's inside the 'reducer'

00:38:45

So let's see if it works

00:38:54

It worked,

00:40:11

Chat (window: 0)

> **Agent**

github.copilot.workspace

> **Context**

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

> **Message**

@workspace how could I create the following tests: Add some tasks, reload the page, and confirm that the tasks are still there. Mark some tasks as completed, reload the page, and confirm that both the tasks and their states remain unchanged. Edit some tasks, reload the page, and confirm that the changes were retained. Delete some tasks, reload the page, and confirm that the deleted tasks remain absent.

> **Response**

You can create these persistence tests using a UI testing tool like Cypress or Playwright, as they allow you to simulate user interactions and reload the page. Here are examples of how these tests would look using Cypress:

```
describe('Task Persistence', () => {
  beforeEach(() => {
    cy.visit('/');
    cy.clearLocalStorage();
  });

  it('Adds tasks and checks after reloading', () => {
    cy.get('[data-testid="text-input"]').type('First task{enter}');
    cy.get('[data-testid="text-input"]').type('Second
task{enter}');
    cy.get('[data-testid="todo-item-label"]').should('have.length',
2);

    cy.reload();
    cy.get('[data-testid="todo-item-label"]').should('have.length',
2);

    cy.contains('First task');
    cy.contains('Second task');
  });

  it('Marks tasks as completed and checks after reloading', () => {
    cy.get('[data-testid="text-input"]').type('Completed
```

```

task{enter}');
    cy.get('[data-testid="todo-item-toggle"]').check();
    cy.reload();
    cy.get('[data-testid="todo-item-
toggle"]').should('be.checked');
  });

  it('Edits tasks and checks after reloading', () => {
    cy.get('[data-testid="text-input"]').type('Task to
edit{enter}');
    cy.get('[data-testid="todo-item-label"]').dblclick();
    cy.get('[data-testid="text-input"]').clear().type('Edited
task{enter}');
    cy.reload();
    cy.contains('Edited task');
  });

  it('Deletes tasks and checks after reloading', () => {
    cy.get('[data-testid="text-input"]').type('Task to
delete{enter}');
    cy.get('[data-testid="todo-item-button"]').click();
    cy.get('[data-testid="todo-item-label"]').should('not.exist');
    cy.reload();
    cy.get('[data-testid="todo-item-label"]').should('not.exist');
  });
});

```

These tests cover all the scenarios you described. Remember to install and configure Cypress in your project to run them.

00:40:17
- 00:40:23

> Response rendering

00:40:44

> Viewing file (package.json)

00:41:16

Researcher

Cool.

00:41:20

Participant

Let's see here.

00:41:21

Researcher

Uh-huh.

00:41:22

Participant

Add to the to-dos, load the page and confirm that the to-dos are there. Great that we've already tested confirming,

00:41:26

00:42:12

Researcher

Great. You can go to the next one

00:42:15

Participant

Great, so this complete.

00:42:26

If a user loads the page while typing,

00:42:40

it automatically saves drafts.

00:42:44

It's every few seconds

00:43:04

He has to continue, let me see.

00:43:18

And it's in the 'input'.

00:43:20

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

00:43:28

Participant

Here. Look, there's something here.

00:43:45

By the code here, I didn't identify anything.

00:43:53

It already does, so I don't know if it needs to be implemented,

00:43:57

but I'll use it here.

00:44:08

Chat (window: 0)

> Agent

github.copilot.workspace

> Context

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

> Message

@workspace If a user reloads the page while typing a new task, the text in the input field disappears. Modify the application so that the input field automatically saves drafts every few seconds and restores them when the page is reloaded.

> Response

To implement automatic draft saving for the input field and restore it when reloading the page, follow these steps in the component [Input](#) ([src/todo/components/input.jsx](#)):

1. Use a local state to control the input value.
2. Save the value in localStorage every few seconds (using setInterval).
3. When mounting the component, retrieve the saved value from localStorage.
4. Clear the draft from localStorage when submitting the task.

Here's how to do it:

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

const DRAFT_KEY = "todo-input-draft";

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

export function Input({ onSubmit, placeholder, label, defaultValue,
onBlur }) {
  const [inputValue, setInputValue] = useState("");
  const intervalRef = useRef();

  // Load the draft when mounting
  useEffect(() => {
    const draft = localStorage.getItem(DRAFT_KEY);
    setInputValue(draft ?? defaultValue ?? "");
  }, [defaultValue]);
```

```

// Save the draft every 2 seconds
useEffect(() => {
  intervalRef.current = setInterval(() => {
    localStorage.setItem(DRAFT_KEY, inputValue);
  }, 2000);
  return () => clearInterval(intervalRef.current);
}, [inputValue]);

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

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

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

      if (!hasValidMin(value, 1))
        return;

      onSubmit(value);
      setInputValue("");
      localStorage.removeItem(DRAFT_KEY); // Clear the draft
when submitting
      e.target.value = "";
    }
  },
  [onSubmit]
);

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

```

Thus, the typed text will be automatically saved and restored when the page is reloaded.

00:44:12

Researcher

Our 40-minute time is up, I'm going to ask you to stop coding now.

00:44:16
– 00:44:24

> Response rendering

00:45:43

Researcher

Now I'll send you the post-study questionnaire

00:45:54

after you answer this questionnaire I'll ask you a few more questions OK.

00:46:02

Participant

ok.

00:46:15

What would the completion be?

00:46:20

Researcher

Are the copilot's responses good, bad sizes.

00:46:27

Participant

ok.

00:46:30

I found it good.

00:46:32

Sometimes I use 'gemini', and it really.

00:46:36

Increases a lot of details, I found good use of technical terms,

00:46:43

it's neutral.

00:46:45

The correction is, it has already helped me a lot. Example the file

00:46:52

'reducer'.

00:46:54

Researcher

ok.

00:47:04

Participant

I found it neutral, nothing special.

00:47:07

Use of formatting, bold markers, I found it neutral.

00:47:13

Researcher

ok.

00:48:29

Participant

I agree.

00:48:34

Operating the copilot would be easy for me. I agree.

00:48:44 This one a little.
00:49:13 I agree.
00:49:17 More flexible to interact, I somewhat agree.
00:49:25 It would be easy for me to become skilled. Do you agree?
00:49:37 **Researcher**
ok.
00:49:44 **Participant**
How mentally demanding was the session.
00:49:51 I would say it was average
00:50:03 Rushed and fast-paced, that was the session's pace.
00:50:26 It's not much like that, because I work with this.
00:50:28 Maybe if it were with a language I wasn't so used to,
00:50:34 for example Java, maybe yes. I would find it a bit more difficult
00:50:43 I found it normal, it wasn't rushed at all, but I have time,
00:50:52 **Researcher**
ok.
00:50:55 **Participant**
When you had to strive to reach your performance level,
00:51:02 how insecure, unmotivated, and stressed,
00:51:05 did it bother you?
00:51:54 A realist. It's to persist.
00:51:58 It's moderately realistic, yes, that happens a lot.
00:52:06 I think it's very realistic for my work
00:52:11 It's to save the drafts. I think so
00:52:18 **Researcher**
Ok, thank you for answering.
00:52:23 I'll ask you a few more questions ok
00:52:28 **Participant**
Alright.
00:52:31 **Researcher**
Did you feel, do you think you oriented yourself there?
00:52:34 Were you able to orient yourself well with the code?
00:52:42 **Participant**
Yes, I think I needed a little more
00:52:47 time to understand the context.
00:53:00 to get used to the project files.
00:53:03 so I thought yes.
00:53:04 **Researcher**
Cool. And how did the copilot help you with that?
00:53:12 Did it influence you?
00:53:18 **Participant**
Well, when I had difficulty with the sorting task,
00:53:21 it helped me remember the details
00:53:31 **Researcher**
Right.
00:53:31 **Participant**
and a simple task too
00:53:35 when it was to find the problem in the code,
00:53:40 it helped me a lot. I would certainly take longer
00:53:45 to read the 'reducer' file code
00:53:59 **Researcher**
cool.
00:54:06 **Participant**
Sorting task also helped me a lot
00:54:07 **Researcher**
ok.

00:54:10 did you understand the code base well?

00:54:17 **Participant**
I believe so.

00:54:30 **Researcher**
How would you have solved these tasks
00:54:36 without the help of the copilot.

00:54:40 **Participant**
By searching on Google, stackoverflow and reading the code

00:54:49 **Researcher**
ok.
00:54:51 And how did the copilot influence you to solve the tasks?

00:55:02 **Participant**
When I put in the prompt the @workspace
00:55:09 it improved the response
00:55:26 After that I changed the strategy and started delegating more things to the copilot
00:55:46 After the sorting task
00:55:49 I started updating the copilot more

00:56:00 **Researcher**
ok
00:57:28 What did you like and dislike about the copilot?

00:57:41 **Participant**
Of course I liked it. It was.
00:57:43 the context. This command that was explained in the task helped a lot.

00:57:47 **Researcher**
ok.
00:57:59 **Participant**
And the buttons to apply the code generated by the copilot
00:58:06 Usually, I copied and pasted.

00:58:10 **Researcher**
Cool.
00:58:27 **Participant**
and the links to the files

00:58:41 **Researcher**
ok.
00:58:45 During the experiment, did you change your way of interacting with the copilot?

00:59:04 **Participant**
I changed
00:59:06 Yeah, first, I wrote in English, to train, that's it.
00:59:11 Then I started writing in Portuguese, to be more precise. And finally,
00:59:18 I just copied the task from the PDF, because it was more complete there.
00:59:24 The requirement.

00:59:27 **Researcher**
Cool. Thank you very much for participating in our study.