

Participant P01

00:04:44 **Participant**
OK.

00:04:45 Sure. So.

00:04:53 Supposed to do.

00:05:01 Let's see. How do we check this?

00:05:12 Just try

00:05:46 **Chat** (window: 0)
> **Agent**
github.copilot.workspace
> **Context**
workspace
> **Message**
@workspace how do I change the color of the no todo's messafge?
> **Response**
To change the color of the "No TODOs here yet" message, you can modify the `.todo-list-empty-message` class in the [src/todo/app.css](#) file. For example, to change the color to blue, update the `color` property:

```
File: src/todo/app.css
.todo-list-empty-message {
  padding: 10px;
  background: #b83f45;
  color: blue; /* Change this to your desired color */
  border-radius: 5px;
  font-weight: bolder;
}
```

00:05:58
- 00:06:00 > **Response rendering**

00:06:05 > **Viewing file** (src/todo/app.css)

00:06:06 **Participant**
Over here.

00:06:10 That's it.

00:06:12 Somewhere be.

00:06:20 Yeah, the empty message.

00:06:22 And we can change this color to the color you mentioned here.

00:06:30 > **Editing file** (src/todo/app.css)

00:06:35 **Participant**
Now it should.

00:06:38 The change and we could change this by.

00:06:44 Let's see. Can we run this application somewhere?

00:06:48 > **Viewing file** (src/todo/app.jsx)

00:06:56 > **Viewing file** (src/todo/app.css)

00:06:58 > **Viewing file** (public/index.html)

00:06:59 **Participant**
Is there some way of running the application or is it just?

00:07:03 I figure that out.

00:07:05 **Researcher**
Yes. So this is also something that we ask you

00:07:06 **Participant**
OK.

00:07:06 Sure, sure.

00:07:07 **Researcher**

00:07:20

to figure out.

Chat (window: 0)

> **Agent**

github.copilot.workspace

> **Context**

workspace, viewport (public/index.html)

> **Message**

@workspace How do I run this app?

> **Response**

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

1. Open a terminal in your workspace.

2. Install the dependencies by running:

```
npm install
```

3. Start the development server by running:

```
npm run dev
```

4. The app will open in your default browser, or you can access it at

<http://localhost:8080>.

> **Response rendering**

00:07:32

- 00:07:34

00:07:48

Participant

Let's try this, I suppose.

00:07:53

Umm.

00:08:26

OK, this one says it's installed.

00:08:30

Suppose I can run this as

00:08:35

well.

00:08:40

It indeed says no todos here.

00:08:42

Yes, in not very readable, no, but it's a yeah so.

00:08:50

Practice Task 2, B.

00:08:54

Researcher

Could you double check the verification step please?

00:09:04

Participant

Oh, indeed.

00:09:05

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

00:09:10

Participant

I suppose the box should be.

00:09:13

- 00:09:20

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

00:09:17

Researcher

No worries, by the way. It's completely fine to make a mistake,

00:09:18

Participant

Yeah.

00:09:20

Researcher

right?

00:09:22

Participant

Yeah, so this should be.

00:09:26

Stop this one.

00:09:31

Try to run the same program again.

00:09:36

And now it does work.

00:09:38

Like it's supposed to be.

00:09:40

Umm.

00:09:43

Blue box, OK.

00:09:47

Placeholder text of the input field is currently what needs to be done.

00:09:52

Change the text to enter task here. OK, so let's.

00:09:57 > **Viewing file** (public/index.html)

00:10:05 > **Viewing file** (src/todo/components/input.jsx)

00:10:13 > **Viewing file** (src/todo/components/main.jsx)

00:10:18 **Participant**
OK.
This is too much code.
Let's just ask this thing again.
No.
Chat (window: 0)

00:10:47 > **Agent**
github.copilot.workspace

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

> **Message**
@workspace where do i change the placeholder of what needs to be done?

> **Response**
You can change the placeholder text "What needs to be done?" in the [Header](#) (src/todo/components/header.jsx) component. Specifically, update the [placeholder](#) (src/todo/components/header.jsx) prop of the [Input](#) (src/todo/components/input.jsx) component in [src/todo/components/header.jsx](#):

```
File: src/todo/components/header.jsx
<Input onSubmit={addItem} label="New Todo Input" placeholder="Your new
placeholder text here" />
```

00:11:00 > **Response rendering**
- 00:11:02

00:11:11 **Participant**
Somewhere here.

00:11:14 > **Viewing file** (src/todo/components/header.jsx) through Copilot link

00:11:17 **Participant**
Oh, here it is. Yep, found it.

00:11:20 > **Editing file** (src/todo/components/header.jsx)
- 00:11:28

00:11:22 **Participant**
Change it to.
"Enter a task here".
OK so.
Let's refresh this.
Save mode.
And that also says "Enter a task here" so.

00:11:53 **Researcher**
Also, just so you know, you don't need to restart the the process
for changes to show up.
It should automatically refresh.

00:12:00 **Participant**
Oh yeah, of course. That was the fun thing about React.
OK.
C, sort todos.
are currently displayed in the order they were added.
Modify the application, are sorted alphabetically instead.
OK.

00:12:08

00:12:10

00:12:13

00:12:15

00:12:28 > **Viewing file** (src/todo/components/item.jsx)

00:12:33 **Researcher**
So what is your thought process now?
How are you approaching this?

00:12:35

00:12:36 > **Viewing file** (src/todo/components/main.jsx)
00:12:36 **Participant**
OK this I'm just first going to look through the code if I can anywhere find
00:12:42 where the todos are added, but as my React skills are not that good
00:12:46 > **Viewing file** (src/todo/components/paginationControls.jsx)
00:12:49 > **Viewing file** (src/todo/components/main.jsx)
00:12:49 **Participant**
yet.
00:12:51 Probably I will not find it.
00:12:59 > **Viewing file** (src/todo/app.jsx)
00:13:00 **Participant**
No, that still does not say a lot.
00:13:06 > **Viewing file** (src/todo/components/main.jsx)
00:13:06 > **Viewing file** (src/todo/components/header.jsx)
00:13:07 > **Viewing file** (src/todo/components/input.jsx)
00:13:07 > **Viewing file** (src/todo/components/item.jsx)
00:13:47 **Chat** (window: 0)
> **Agent**
github.copilot.workspace
> **Context**
workspace, viewport (src/todo/components/item.jsx)
> **Message**
@workspace Where do I find the part about displaying all todos?
> **Response**
The part responsible for displaying all TODOs is in the [Main](#)
([src/todo/components/main.jsx](#)) component, specifically in the [visibleTodos.map](#)
([src/todo/components/main.jsx](#)) function. This is located in
[src/todo/components/main.jsx](#):


```
File: src/todo/components/main.jsx
<ul className={classnames("todo-list")} data-testid="todo-list">
  {visibleTodos.map((todo, index) => (
    <Item todo={todo} key={todo.id} dispatch={dispatch} index=
{index} />
  ))}
</ul>
```


This iterates over the [visibleTodos](#) ([src/todo/components/main.jsx](#)) array and
renders each TODO item using the [Item](#) ([src/todo/components/item.jsx](#))
component.

00:13:59 > **Response rendering**
- 00:14:02
00:14:03 > **Viewing file** (src/todo/components/main.jsx) through Copilot link
00:14:08 **Participant**
OK.
00:14:13 todos.
00:14:25 **Researcher**
So what are you thinking right now?
00:14:26 **Participant**
OK.
00:14:27 Now I'm trying to understand the code here,
00:14:29 what it does and this only seems to filter it whether.
00:14:34 todos are completed or not, and.
00:14:41 Because this should return something.
00:14:47 Just a list of todos.
00:14:48 > **Editing file** (src/todo/components/main.jsx)
- 00:15:35

00:14:49

Participant

But I suppose we can just say.

00:14:54

Should be some kind of an order function.

00:15:02

And it does not like me typing here. Why not?

00:15:07

Let me see.

00:15:19

But it's the sort function.

00:15:30

Access should already work.

00:15:36

Back, let's try to add some things.

00:15:41

This wasn't direct.

00:15:54

So we got something.

00:15:58

Two different ones.

00:16:02

Let me just add "ABC" for now.

00:16:10

How do we do this?

00:16:24

OK.

00:16:27

Let's just ask this thing and.

00:16:34

Chat (window: 0)

> Agent

github.copilot.default

> Context

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

> Message

How do I add todo's?

> Response

To add TODOs, you need to look at the component responsible for handling new TODO input, which is likely the Header component. This component typically contains an input field where users can type a new TODO and submit it.

In the Header component (likely located in [src/todo/components/header.jsx](#)), there should be a function that dispatches an action to add a new TODO to the list. For example:

```
<Input
  onSubmit={addItem}
  label="New Todo Input"
  placeholder="What needs to be done?"
/>
```

The addItem function would dispatch an action like this:

```
const addItem = useCallback(
  (title) => {
    if (title.trim().length > 0) {
      dispatch({ type: ADD_ITEM, payload: { title } });
    }
  },
  [dispatch]
);
```

To add a TODO:

1. Type the TODO text in the input field (placeholder: "What needs to be done?").
2. Press Enter to submit the TODO.

If you want to test this functionality, ensure the ADD_ITEM action is handled in your reducer to update the [todos \(src/todo/components/main.jsx\)](#) state.

00:16:37
- 00:16:45

> Response rendering

00:16:49

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

00:17:23

Participant

Now this work OK.

00:17:35

Now just.

00:17:38

I want them to sort.

00:17:39

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

00:17:40

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

00:17:42

Participant

So.

00:17:43

And it doesn't seem to be doing that.

00:17:43

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

00:17:45

Participant

So let's see where.

00:17:49

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

00:17:51

Participant

Look here again.

00:18:03

Chat (window: 0)

> **Agent**

github.copilot.default

> **Context**

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

> **Message**

How di I get the todo's to be sorted?

> **Response**

To sort the TODOs, you can modify the [visibleTodos](#) ([src/todo/components/main.jsx](#)) logic in the [useMemo](#) ([node_modules/@types/react/index.d.ts](#)) hook within the [Main](#) ([src/todo/components/main.jsx](#)) component. You can use JavaScript's [sort](#) ([src/todo/components/main.jsx](#)) method to define a sorting order. For example, to sort TODOs alphabetically by their title, you can update the [useMemo](#) ([node_modules/@types/react/index.d.ts](#)) hook like this:

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

alphabetically by title

If you want to sort by another property (e.g., creation date or completion status), you can adjust the [sort](#) ([src/todo/components/main.jsx](#)) function accordingly. For example:

- **Sort by creation date** (assuming [todo.createdAt](#) ([src/todo/components/main.jsx](#)) exists):

```
.sort((a, b) => new Date(a.createdAt) - new Date(b.createdAt))
```

- **Sort by completion status (completed first):**

```
.sort((a, b) => a.completed - b.completed)
```

Place this updated [useMemo](#) ([node_modules/@types/react/index.d.ts](#)) logic in your [Main](#) ([src/todo/components/main.jsx](#)) component to ensure the TODOs are displayed in the desired order.

> **Response rendering**

00:18:06
- 00:18:19

00:18:10

Participant

00:18:15 OK.
00:18:25 So maybe.
00:18:25 Let's see.
00:18:29 I suppose we have to sort them by name.
00:18:36 Yeah.
00:18:38 Creation date, in here it says.
00:18:51 But there was this function that just.
00:18:55 **> Implementing Copilot code** using insert button (src/todo/components/main.jsx)
00:18:56 **Participant**
00:18:56 Alright, so this one.
00:19:04 All right, now, that wasn't what I wanted.
00:19:05 **> Editing file** (src/todo/components/main.jsx)
00:19:06 **Participant**
00:19:06 Let's look.
00:19:13 OK.
00:19:14 Let's just add it in here.
00:19:27 **> Implementing Copilot code** by copying part of snippet
(src/todo/components/main.jsx)
00:19:40 **Participant**
00:19:40 Sort alphabetically by title, no.
00:19:44 Let's add some.
00:19:52 Like I said, it doesn't work.
00:19:54 **Researcher**
00:19:54 You might need to refresh the page for changes to show up.
00:19:59 **Participant**
00:19:59 OK.
00:20:00 **Researcher**
00:20:00 I'm not 100% sure on this, but you could see if that works.
00:20:03 **Participant**
00:20:03 OK.
00:20:07 Yeah. Now it does work.
00:20:10 Seems to work with these ones and we had to do verification so.
00:20:20 Let's remove them all.
00:20:23 The.
00:20:32 Now, it is correctly sorted.
00:20:35 **Researcher**
00:20:35 Perfect. Thanks.
00:20:36 **Participant**
00:20:36 OK, nice.
00:20:37 **Researcher**
00:20:37 All right, I'll be sending you the the tasks then.
00:20:41 By the way, thanks for thinking aloud the way you're
00:20:41 **Participant**
00:20:41 Yeah. OK.
00:20:44 **Researcher**
00:20:44 doing.
00:20:44 That's really helping a lot.
00:20:46 **Participant**
00:20:46 OK, nice.
00:20:48 **Researcher**
00:20:48 Also, we have 30 minutes left for the actual
00:20:51 tasks.
00:20:53 Yeah, and don't worry about it.
00:20:54 All it's getting them all finished.
00:20:57 They're in a random order.
00:20:58 **Participant**

00:20:58 **Researcher**
Feel free to start with any one you want.

00:21:03 **Participant**
OK, sure.

00:21:09 These are in the same same working space?

00:21:13 **Researcher**
Yes.

00:21:14 **Participant**
Yeah. OK, good.

00:21:17 So let's see.

00:21:19 Here.

00:21:21 Refresh page, modify the application so that todos remain
00:21:26 even after refresh, ensure that each todo's state.

00:22:07 **Researcher**
So what are you looking for now?

00:22:10 **Participant**
Let's see, I'm trying to find something about where
00:22:14 people have been trying to.
00:22:18 Do something with refreshing but.
00:22:19 > **Viewing file** (src/todo/app.css)
00:22:19 > **Viewing file** (src/todo/app.jsx)
00:22:21 **Participant**
Now it does. I suppose because someone was making an
00:22:22 > **Viewing file** (src/todo/components/main.jsx)
00:22:25 **Participant**
attempt. It should be here as this one is marked
00:22:28 right.
00:22:31 But I I'm trying to see.
00:22:36 Something about the todos or where they are saved.
00:22:38 Or anything.
00:22:46 Let's see if I can't find anything but.
00:22:54 Anything that's where the todos are stored.
00:22:57 > **Viewing file** (src/todo/components/input.jsx)
00:23:01 **Participant**
Somewhere.
00:23:13 Now we'll just use this again.
00:23:29 **Chat** (window: 0)
> **Agent**
github.copilot.workspace
> **Context**
workspace, viewport (src/todo/components/input.jsx)
> **Message**
@workspace Where are the todo's stored?
> **Response**
The TODOs are stored in the [state \(src/todo/reducer.js\)](#) managed by the [todoReducer \(src/todo/reducer.js\)](#) function in [src/todo/reducer.js](#). The initial state is loaded from the browser's [localStorage \(node_modules/@types/node/globals.d.ts\)](#) using the [loadState \(src/todo/reducer.js\)](#) function:

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

When the state is updated, it is saved back to [localStorage](#) ([node_modules/@types/node/globals.d.ts](#)) using the [saveState](#) ([src/todo/reducer.js](#)) function:

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

The [todoReducer](#) ([src/todo/reducer.js](#)) handles actions like adding, updating, toggling, and removing TODOs, and ensures the state is kept in sync with [localStorage](#) ([node_modules/@types/node/globals.d.ts](#)).

00:23:43
- 00:23:46

> Response rendering

00:23:51

Participant

Todo reducer

00:23:56

Local storage using the load state function.

00:23:58

> **Viewing file** ([node_modules/%40types/node/globals.d.ts](#)) through Copilot link

00:24:02

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

00:24:04

Participant

Maybe not entirely, but I was looking for it, yeah.

00:24:08

This look a lot better.

00:24:15

Save state. Yeah, so.

00:24:18

It is not working correctly at the moment.

00:24:23

So we have save state and.

00:24:33

Yeah.

00:24:36

Well, this is new state.

00:24:45

OK. Yeah.

00:24:54

Oh wait, I see capital S.

00:24:54

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

00:24:55

Participant

Let's maybe this works.

00:25:10

Refreshes.

00:25:13

No.

00:25:17

That did not do as I expected.

00:25:21

Umm.

00:25:28

state.

00:25:31

Where do we load the state?

00:25:50

I think that's fine. That's where.

00:26:35

Chat (window: 0)

> Agent

github.copilot.default

> Context

selection ([src/todo/reducer.js](#))

> Message

Where are the todo's loaded form local storage into the page?

> Response

The TODOs are loaded from [localStorage \(src/todo/reducer.js\)](#) into the page through the [initialState \(src/todo/reducer.js\)](#) in the [todoReducer \(src/todo/reducer.js\)](#). This [initialState \(src/todo/reducer.js\)](#) is set by calling the [loadState \(src/todo/reducer.js\)](#) function:

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

The [initialState \(src/todo/reducer.js\)](#) is then passed to the `useReducer` hook in the component where the state management is initialized, likely in a higher-level component like `App` or `Main`. For example:

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

This initializes the `todos` state with the data loaded from [localStorage \(src/todo/reducer.js\)](#). The `todos` state is then passed down as a prop to components like `Main`, which renders the TODOs on the page.

00:26:36
- 00:26:41

00:26:51

00:26:56

00:27:17

> Response rendering

Participant

Now do @.

workspace.

Chat (window: 0)

> Agent

github.copilot.workspace

> Context

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

> Message

@workspace Where are the todo's loaded from local storage into the page?

> Response

The TODOs are loaded from [localStorage \(src/todo/reducer.js\)](#) into the page through the [initialState \(src/todo/reducer.js\)](#) in the [todoReducer \(src/todo/reducer.js\)](#) function. This [initialState \(src/todo/reducer.js\)](#) is passed to the [useReducer \(node_modules/@types/react/index.d.ts\)](#) hook in the [App \(src/todo/app.jsx\)](#) component:

```
File: src/todo/app.jsx
import { todoReducer } from "../reducer";

export function App() {
  const [todos, dispatch] = useReducer(todoReducer, []);
  ...
}
```

The [todoReducer \(src/todo/reducer.js\)](#) initializes its state using the [loadState \(src/todo/reducer.js\)](#) function:

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

export const initialState = loadState();
```

This ensures that the TODOs stored in [localStorage \(node_modules/@types/node/globals.d.ts\)](#) are loaded into the app when it starts.

00:27:25

Researcher

So why did you ask this question again?

00:27:26

Participant

Yeah.

00:27:28

Yeah, I wanted to figure out because I saw a

00:27:30

- 00:27:33

> Response rendering

00:27:31

Participant

little mistake here that they use the capital S and stuff small as somewhere there was difference.

00:27:37

But it still does not seem to be.

00:27:40

That does not seem to be the only problem.

00:27:42

So now I'm trying to find out where somewhere in the whole code.

00:27:45

This function is being called because I suppose.

00:27:52

Researcher

Alright.

00:27:54

Participant

Load state.

00:27:57

Or at least save state is called here.

00:27:59

To, in this function.

00:28:05

So I suppose this does work.

00:28:06

Yeah.

00:28:22

Oh, this is also a little thing.

00:28:28

A little typo.

00:28:35

Yeah, let's see.

00:28:37

[unintelligible] now we have something.

00:28:39

Refresh it now.

00:28:41

Still does not work.

00:28:47

So.

00:28:53

[unintelligible]

00:29:25

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

00:29:31

Participant

And why isn't this one?

00:29:36

In here.

00:29:42

OK, let's just. Find out why.

00:29:47

Chat (window: 0)

00:30:11

> Agent

github.copilot.default

> Context

viewport (src/todo/reducer.js)

> Message

It does not seem to do that properly when i refresh the page, how can I fix this?

> Response

If the TODOs are not properly loaded from [localStorage](#) ([node_modules/@types/node/globals.d.ts](#)) when you refresh the page, it could be due to how the [initialState](#) ([src/todo/reducer.js](#)) is being used in the app. Here's how you can fix this:

1. Ensure [initialState](#) ([src/todo/reducer.js](#)) is passed correctly to `useReducer`

Verify that the [initialState](#) ([src/todo/reducer.js](#)) is passed as the second argument to `useReducer` in the component where the reducer is used (e.g., App or a similar component):

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

If [initialState](#) ([src/todo/reducer.js](#)) is not being used, replace the empty array (`[]`) with [initialState](#) ([src/todo/reducer.js](#)).

2. Debug the [loadState \(src/todo/reducer.js\)](#) Function

Ensure that the [loadState \(src/todo/reducer.js\)](#) function is correctly retrieving and parsing the data from [localStorage \(node_modules/@types/node/globals.d.ts\)](#). Add a `console.log` to check if the data is being loaded:

```
export const loadState = () => {
  const state = localStorage.getItem("todos");
  console.log("Loaded state from localStorage:", state); // Debugging
  return state ? JSON.parse(state) : [];
};
```

If the [state \(src/todo/reducer.js\)](#) is `null`, it means there is no data in [localStorage \(node_modules/@types/node/globals.d.ts\)](#). This could happen if the [saveState \(src/todo/reducer.js\)](#) function is not saving the data correctly.

3. Verify the [saveState \(src/todo/reducer.js\)](#) Function

Ensure that the [saveState \(src/todo/reducer.js\)](#) function is being called and saving the state to [localStorage \(node_modules/@types/node/globals.d.ts\)](#) after every action. Add a `console.log` to confirm:

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

If the [saveState \(src/todo/reducer.js\)](#) function is not being called, check the [todoReducer \(src/todo/reducer.js\)](#) to ensure it is invoked after every action.

4. Check for [localStorage \(src/todo/reducer.js\)](#) Issues

Ensure that [localStorage \(src/todo/reducer.js\)](#) is not being cleared or overwritten elsewhere in your app. Also, confirm that your browser allows [localStorage \(src/todo/reducer.js\)](#) access and that no errors are being thrown in the console.

5. Fallback to [LOAD_STATE \(src/todo/reducer.js\)](#) Action

If the [initialState \(src/todo/reducer.js\)](#) is not working as expected, you can explicitly dispatch a [LOAD_STATE \(src/todo/reducer.js\)](#) action when the app initializes:

```
useEffect(() => {
  const savedState = loadState();
  dispatch({ type: LOAD_STATE, payload: savedState });
}, []);
```

This ensures that the state is loaded from [localStorage \(src/todo/reducer.js\)](#) when the app starts.

Final Steps

After making these changes, refresh the page and check if the TODOs persist. If the issue persists, share any errors or unexpected behavior you observe for further

debugging.

00:30:12
- 00:30:34

> Response rendering

00:30:38

Participant

What?

00:30:44

Researcher

What are you thinking now?

00:30:47

Participant

I'm sure it's about what it's saying.

00:30:50

Let's see [unintelligible].

00:30:53

Researcher

What's confusing you now?

00:30:56

Participant

Because I have no idea where this code is located anywhere.

00:31:02

So.

00:31:06

And use reducer.

00:31:10

Is this one made anywhere?

00:31:16

It's just a.

00:31:18

OK.

00:31:44

Anywhere else.

00:32:01

Chat (window: 0)

> Agent

github.copilot.default

> Context

viewport (src/todo/reducer.js)

> Message

step 1, where can I find that code?

> Response

The code for initializing the useReducer hook with the [initialState](#) ([src/todo/reducer.js](#)) is typically found in the main application component, such as App.jsx or a similar high-level component. This is where the state management for the TODOs is set up.

Look for a file named something like App.jsx or App.js in your project, likely located in the [src/todo/](#) directory. Inside that file, you should find something like this:

```
import React, { useReducer } from "react";
import { todoReducer, initialState } from "../reducer";

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

  return (
    <div>
      { /* Other components */ }
    </div>
  );
}
```

```
    );  
  }
```

If you can't find it, let me know, and I can guide you further!

00:32:02
- 00:32:04

> **Response rendering**

00:32:12

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

00:32:14

Participant

Yeah.

00:32:17

Use reducer.

00:32:22

Yeah.

00:32:27

Find this. Yeah, it seems too.

00:32:30

True, but it does not contain the initial state here.

00:32:34

So how do we get that?

00:32:35

Right.

00:32:40

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

00:32:44

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

00:32:51

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

00:32:54

Participant

[unintelligible] we can just use.

00:32:57

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

00:32:58
- 00:33:01

00:33:02

Participant

So state here.

00:33:11

Yeah, now it is, does works.

00:33:15

I think we can mark this one complete.

00:33:17

We can also.

00:33:21

Edit something about this.

00:33:31

And still reload it.

00:33:34

Seems to work and deleting also still works, yeah.

00:33:40

Researcher

Perfect. Thanks.

00:33:40

Participant

OK.

00:33:44

The second one.

00:33:47

If the user refreshes the page, [unintelligible].

00:33:54

every few seconds, and restore them when the page reloads.

00:34:00

Yeah. OK.

00:34:03

So gotta find where the app.

00:34:06

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

00:34:09

Participant

Is where we're typing.

00:34:14

Something about where it gets the updates, so.

00:34:20

Umm.

00:34:22

Suppose here it uses the title to find out.

00:34:31

If length is 0.

00:34:32

Then it just removes it and otherwise it updates it.

00:34:49

Yeah.

00:35:01

Let's see.

00:35:15

OK.

00:35:18

Suppose we can.

00:35:23

Somewhere add.

00:35:28

Some function that calls this every few seconds so.

00:35:33

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

00:35:34

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

00:35:35 > **Viewing file** (src/todo/components/header.jsx)
00:35:40 > **Viewing file** (src/todo/components/item.jsx)
00:35:45 **Participant**
OK, sure.
00:35:52 OK.
00:35:55 This isn't [unintelligible].
00:35:55 > **Viewing file** (src/index.js)
00:36:05 > **Viewing file** (src/todo/constants.js)
00:36:07 > **Viewing file** (src/todo/reducer.js)
00:36:09 **Researcher**
So can you tell me what you're trying to do right now?
00:36:11 **Participant**
No, I'm. I'm trying to find a place where I can
00:36:15 still look at the base itself because I know there's this function that can make,
00:36:17 > **Viewing file** (src/todo/constants.js)
00:36:19 > **Viewing file** (src/todo/components/paginationControls.jsx)
00:36:23 **Participant**
some kind of a? Yeah, what's called?
00:36:28 An interval based, yeah, function call and I want to put it
00:36:34 somewhere.
00:36:38 > **Viewing file** (src/todo/components/input.jsx)
00:36:39 **Participant**
That suppose I can.
00:36:41 Maybe I can just put it in here.
00:36:44 > **Viewing file** (src/todo/components/item.jsx)
00:36:45 **Participant**
No.
00:36:47 **Researcher**
And what do you want
00:36:48 this interval interval for?
00:36:51 **Participant**
We need the interval because the somewhere out here it says that.
00:36:58 Automatically saves draft every few seconds.
00:37:02 And reshows them on the page reload. So I suppose that should be the case.
00:37:04 **Researcher**
Right.
00:37:10 > **Editing file** (src/todo/components/item.jsx)
- 00:37:24
00:37:12 **Participant**
So that would have to make some kind of.
00:37:23 Second step along and we wanted to.
00:37:28 Let's see what function does this need to call.
00:37:34 No update.
00:37:37 With this.
00:37:44 And updates.
00:37:52 > **Editing file** (src/todo/components/item.jsx)
- 00:37:56
00:38:01 **Participant**
Can I find what title it is?
00:38:12 > **Editing file** (src/todo/components/item.jsx)
- 00:38:29
00:38:15 **Participant**
Maybe just.
00:38:20 No idea if this work will try.
00:38:24 Every.
00:38:27 This is milliseconds and every few seconds so.

00:38:32
00:38:39
00:38:49
00:39:42

Let's see.

No, that doesn't seem to work.

OK.

Chat (window: 0)

> Agent

github.copilot.default

> Context

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

> Message

How can I make sure the input that is being typed is updated into the local storage every 2 seconds?

> Response

To ensure that the input being typed is updated into localStorage every 2 seconds, you can use a combination of useEffect and setTimeout or [setInterval](#) ([node_modules/@types/node/timers.d.ts](#)). Here's how you can modify your code:

Steps to Implement Auto-Save to localStorage

1. Track the Current Input Value

Use a [useState](#) ([node_modules/@types/react/index.d.ts](#)) hook to track the current value of the input while the user is typing.

2. Use useEffect for Auto-Save

Set up a useEffect that listens to changes in the input value and updates localStorage every 2 seconds.

3. Clear the Interval on Component Unmount

Ensure you clean up the interval when the component unmounts to avoid memory leaks.

Updated Code

Here's how you can modify your [Item](#) ([src/todo/components/item.jsx](#)) component:

```
import { memo, useState, useCallback, useEffect } from "react";
import classNames from "classnames";

import { Input } from "../input";

import { TOGGLE_ITEM, REMOVE_ITEM, UPDATE_ITEM } from "../constants";

export const Item = memo(function Item({ todo, dispatch }) {
  const [isWritable, setIsWritable] = useState(false);
  const [currentTitle, setCurrentTitle] = useState(todo.title); //
  Track the current input value
  const { completed, id } = todo;

  const toggleItem = useCallback(() => dispatch({ type: TOGGLE_ITEM,
    payload: { id } }), [dispatch]);
  const removeItem = useCallback(() => dispatch({ type: REMOVE_ITEM,
    payload: { id } }), [dispatch]);
  const updateItem = useCallback((id, title) => dispatch({ type:
    UPDATE_ITEM, payload: { id, title } }), [dispatch]);

  const handleDoubleClick = useCallback(() => {
    setIsWritable(true);
  }, []);

  const handleBlur = useCallback(() => {
    setIsWritable(false);
  }, []);
```

```

const handleUpdate = useCallback(
  (title) => {
    if (title.length === 0) {
      removeItem(id);
    } else {
      updateItem(id, title);
    }
    setIsWritable(false);
  },
  [id, removeItem, updateItem]
);

// Auto-save to localStorage every 2 seconds
useEffect(() => {
  const interval = setInterval(() => {
    if (currentTitle !== todo.title) {
      updateItem(id, currentTitle); // Dispatch the update
    }
  }, 2000);

  return () => clearInterval(interval); // Cleanup on unmount
}, [currentTitle, todo.title, id, updateItem]);

return (
  <li className={classnames({ completed: todo.completed })} data-
testid="todo-item">
    <div className="view">
      {isWritable ? (
        <Input
          onSubmit={handleUpdate}
          label="Edit Todo Input"
          defaultValue={currentTitle}
          onBlur={handleBlur}
          onChange={(e) =>
setCurrentTitle(e.target.value)} // Update the current input value
        />
      ) : (
        <>
          <input
            className="toggle"
            type="checkbox"
            data-testid="todo-item-toggle"
            checked={completed}
            onChange={toggleItem}
          />
          <label data-testid="todo-item-label"
onDoubleClick={handleDoubleClick}>
            {currentTitle}
          </label>
          <button className="destroy" data-testid="todo-
item-button" onClick={removeItem} />
        </>
      )}
    </div>
  </li>
);
});

```

Key Changes

1. **currentTitle State**

Added a [useState](https://node_modules/@types/react/index.d.ts) (node_modules/@types/react/index.d.ts) hook to track the current value of the input (currentTitle).

2. `useEffect` for Auto-Save

Used `useEffect` with [`setInterval` \(node_modules/@types/node/timers.d.ts\)](#) to dispatch the [`UPDATE\_ITEM` \(src/todo/constants.js\)](#) action every 2 seconds if the `currentTitle` has changed.

3. Input `onChange` (node_modules/@types/react/index.d.ts) Handler

Updated the `currentTitle` state whenever the user types in the input field.

4. Cleanup Interval

Cleared the interval in the `useEffect` cleanup function to avoid memory leaks.

Result

With this implementation, the input value will be saved to `localStorage` every 2 seconds while the user is typing. This ensures that the TODOs remain up-to-date even if the page is refreshed.

00:39:44
- 00:40:06

> Response rendering

00:39:51

Participant

Yeah.

00:39:58

It's just giving the whole code, sure.

00:40:15

No, I mean sure it gives me everything.

00:40:18

[unintelligible]

00:40:29

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

00:40:41

Participant

It did not like that.

00:40:46

OK.

00:40:46

What don't you like?

00:40:59

Of course it's a bit bigger.

00:41:04

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

00:41:10

Participant

Oh fine.

00:41:11

I'll just copy everything.

00:41:12

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

00:41:31

Participant

Oh wait, there's something about two seconds.

00:41:47

No.

00:42:08

I'm just going to put this whole thing in here.

00:42:12

Chat (window: 0)

> Agent

github.copilot.default

> Context

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

> Message

If a user refreshes the page while typing a new TODO, the text in the input field disappears. Modify the application so that the input field automatically saves drafts every few seconds, and restore them when the page reloads

> Response

To implement draft-saving functionality for the input field, you can use `localStorage` to save the draft text every few seconds and restore it when the page reloads. Here's how you can modify your [Item](#) (src/todo/components/item.jsx) component:

Updated Code

```

import { memo, useState, useCallback, useEffect } from "react";
import classNames from "classnames";

import { Input } from "../input";

import { TOGGLE_ITEM, REMOVE_ITEM, UPDATE_ITEM } from "../constants";

export const Item = memo(function Item({ todo, dispatch }) {
  const [isWritable, setIsWritable] = useState(false);
  const [currentTitle, setCurrentTitle] = useState(() => {
    // Load draft from localStorage if it exists, otherwise use the
    todo title
    const draft = localStorage.getItem(`draft-${todo.id}`);
    return draft || todo.title;
  });
  const { completed, id } = todo;

  const toggleItem = useCallback(() => dispatch({ type: TOGGLE_ITEM,
    payload: { id } }), [dispatch]);
  const removeItem = useCallback(() => dispatch({ type: REMOVE_ITEM,
    payload: { id } }), [dispatch]);
  const updateItem = useCallback((id, title) => dispatch({ type:
    UPDATE_ITEM, payload: { id, title } }), [dispatch]);

  const handleDoubleClick = useCallback(() => {
    setIsWritable(true);
  }, []);

  const handleBlur = useCallback(() => {
    setIsWritable(false);
  }, []);

  const handleUpdate = useCallback(
    (title) => {
      if (title.length === 0) {
        removeItem(id);
      } else {
        updateItem(id, title);
      }
      localStorage.removeItem(`draft-${id}`); // Clear the draft
      when the TODO is updated
      setIsWritable(false);
    },
    [id, removeItem, updateItem]
  );

  // Auto-save drafts to localStorage every 2 seconds
  useEffect(() => {
    const interval = setInterval(() => {
      if (currentTitle !== todo.title) {
        localStorage.setItem(`draft-${id}`, currentTitle); //
        Save the draft
      }
    }, 2000);

    return () => clearInterval(interval); // Cleanup on unmount
  }, [currentTitle, todo.title, id]);

  return (
    <li className={classNames({ completed: todo.completed })} data-
      testid="todo-item">
      <div className="view">
        {isWritable ? (
          <Input

```



```

        onSubmit={handleUpdate}
        label="Edit Todo Input"
        defaultValue={currentTitle}
        onBlur={handleBlur}
        onChange={(e) =>
setCurrentTitle(e.target.value)} // Update the current input value
    />
    ) : (
    <>
        <input
            className="toggle"
            type="checkbox"
            data-testid="todo-item-toggle"
            checked={completed}
            onChange={toggleItem}
        />
        <label data-testid="todo-item-label"
onDoubleClick={handleDoubleClick}>
            {currentTitle}
        </label>
        <button className="destroy" data-testid="todo-
item-button" onClick={removeItem} />
    </>
    )}
</div>
</li>
);
});

```

Key Changes

1. Load Draft on Initialization

The [useState \(node_modules/@types/react/index.d.ts\)](#) hook initializes [currentTitle \(src/todo/components/item.jsx\)](#) by checking if a draft exists in `localStorage` for the specific TODO (`draft-${todo.id}`). If a draft exists, it is used; otherwise, the TODO's title is used.

```

const [currentTitle, setCurrentTitle] = useState(() => {
    const draft = localStorage.getItem(`draft-${todo.id}`);
    return draft || todo.title;
});

```

2. Save Draft Every 2 Seconds

A [useEffect \(node_modules/@types/react/index.d.ts\)](#) hook with [setInterval \(node_modules/@types/node/timers.d.ts\)](#) saves the current input value ([currentTitle \(src/todo/components/item.jsx\)](#)) to `localStorage` every 2 seconds if it differs from the original TODO title.

```
useEffect(() => {
  const interval = setInterval(() => {
    if (currentTitle !== todo.title) {
      localStorage.setItem(`draft-${id}`, currentTitle);
    }
  }, 2000);

  return () => clearInterval(interval);
}, [currentTitle, todo.title, id]);
```

3. Clear Draft on Update

When the TODO is updated (e.g., when the user presses Enter), the draft is removed from localStorage.

```
localStorage.removeItem(`draft-${id}`);
```

Behavior

1. While Typing

The input field's value is saved to localStorage every 2 seconds as a draft.

2. On Page Refresh

If the page is refreshed, the draft is restored from localStorage and displayed in the input field.

3. On Update

When the user submits the TODO (e.g., by pressing Enter), the draft is cleared from localStorage.

This ensures that the user's input is preserved even if the page is refreshed, providing a seamless experience.

00:42:14
- 00:42:28

> Response rendering

00:42:38

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

00:42:41

Participant

See if this works better.

00:42:56

OK, 2 seconds. I don't want to have it 2 seconds.

00:43:09

No.

00:43:56

I don't see.

00:44:37

Chat (window: 0)

> Agent

github.copilot.workspace

> Context

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

> Message

@workspace this does not work, maybe on an other file?

> Response

The TODOs are displayed in the [Main \(src/todo/components/main.jsx\)](#) component, specifically in the [visibleTodos.map \(src/todo/components/main.jsx\)](#) function. This iterates over the [visibleTodos \(src/todo/components/main.jsx\)](#) array and renders each TODO item using the [Item \(src/todo/components/item.jsx\)](#) component.

Here is the relevant code in [src/todo/components/main.jsx](#):

```
File: src/todo/components/main.jsx
<ul className={classnames("todo-list")} data-testid="todo-list">
  {visibleTodos.map((todo, index) => (
    <Item todo={todo} key={todo.id} dispatch={dispatch} index=
{index} />
  ))}
</ul>
```

The [visibleTodos \(src/todo/components/main.jsx\)](#) array is filtered and sorted based on the current route (`/`, `/active`, or `/completed`) and is derived from the [todos \(src/todo/app.jsx\)](#) prop passed to the [Main \(src/todo/components/main.jsx\)](#) component.

> Response rendering

Researcher

So what do you think?

Participant

It's not giving me anything, I suppose.

Because this is already in there.

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

Participant

So.

I'm a little confused because it's, right.

Yeah this is code right here.

This is basically exactly.

The same code.

This one.

Chat (window: 0)

> Agent

github.copilot.workspace

> Context

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

> Message

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

> Response

To implement auto-saving drafts for the input field and restoring them when the page reloads, you can modify the [Input \(src/todo/components/input.jsx\)](#) component to save the current input value to [localStorage \(node_modules/@types/node/globals.d.ts\)](#) and load it when the component mounts.

Here's how you can modify the [Input \(src/todo/components/input.jsx\)](#) component:

```
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 if it exists
    return localStorage.getItem("todo-draft") || defaultValue ||
    ""
  });

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

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

```

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

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

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

// Save the draft to localStorage every 2 seconds
useEffect(() => {
    const interval = setInterval(() => {
        localStorage.setItem("todo-draft", draft);
    }, 2000);

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

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 \(src/todo/components/item.jsx\)](#) state to manage the current value of the input field.
2. **Load Draft:** On component mount, the draft is loaded from [localStorage \(node_modules/@types/node/globals.d.ts\)](#) if it exists.
3. **Save Draft:** The draft is saved to [localStorage \(node_modules/@types/node/globals.d.ts\)](#) every 2 seconds using a [setInterval \(node_modules/@types/node/timers.d.ts\)](#) inside a [useEffect \(node_modules/@types/react/index.d.ts\)](#).
4. **Clear Draft:** The draft is cleared from both the input field and [localStorage \(node_modules/@types/node/globals.d.ts\)](#) when the user submits the TODO.

This ensures that the input field retains its value even after a page refresh.

00:46:15
- 00:46:25

> Response rendering

00:46:29

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

00:46:30

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

00:46:31

Participant

[unintelligible]

00:46:57

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

00:47:06

Participant

Let's just try if this works.

00:47:09

I'm not sure.

00:47:18

Two seconds thing.

00:47:43

[unintelligible]

00:47:52

It still doesn't seem to work.

00:47:53

I'm just going to read what it

00:47:57

said it would change.

00:48:17

Let's look.

00:48:32

It thinks it's still here.

00:48:38

The input text is not.

00:48:50

But it also doesn't seem to be updating at all.

00:48:57

Researcher

All right, I see it's time.

00:48:59

Participant

Yeah.

00:49:00

Researcher

So let's let's stop here.

00:54:51

So yeah, first of all, could you find your way through the

00:54:51

Participant

Yeah, sure.

00:54:56

Researcher

codebase?

00:54:59

Participant

Like yes, I mean.

00:55:03 It looked a lot like Visual Studio code, so I could kind of find my way around
00:55:08 this.
00:55:10 Had some trouble with the whole react part because I'm not really using that,
00:55:15 but.
00:55:18 I think I could find my way, yeah.
00:55:22 **Researcher**
OK.
00:55:23 It's Copilot help you in finding your way.
00:55:25 Or did it?
00:55:27 Did it make it harder for you to find your way?
00:55:30 **Participant**
Yeah. Well, this at some moments I it was just.
00:55:35 I just used it to find my way. [unintelligible]. And it helps at the start.
00:55:45 But eventually it also got I did not know what to do at all and.
00:55:51 I well, I gave a whole tasks to Copilot, and it eventually I did not know what it
00:55:56 was doing, and I could only say it doesn't work.
00:56:00 And basically I did not know what it was doing at all. So.
00:56:06 **Researcher**
Right.
00:56:08 So do you feel like you understood the code well?
00:56:11 **Participant**
The first task? Yes.
00:56:15 But the second one I eventually did not really understand what it was doing.
00:56:22 **Researcher**
Right.
00:56:23 **Participant**
I was just like, OK.
00:56:24 Let's try to see where it goes.
00:56:25 And eventually I was like, OK.
00:56:28 Let's go back to where the original code and try to fix it from there.
00:56:34 **Researcher**
And did using Copilot improve your understanding of the code?
00:56:41 **Participant**
Yeah, at the start. Yeah, it did.
00:56:47 Well, usually, at task B.
00:56:48 I was.
00:56:50 I did not really understand how to do that so.
00:56:55 **Researcher**
Yeah.
00:56:55 **Participant**
In that case, it did not really help me because I was
00:56:59 not.
00:56:59 My intentions were not to understand because when we tried to figure it out
00:57:03 myself at the moment too.
00:57:06 Yeah, do the task.
00:57:08 **Researcher**
Yeah.
00:57:10 Yeah. So I noticed that you initially tried to
00:57:15 solve the task yourself.
00:57:18 **Participant**
Yeah.
00:57:19 **Researcher**
And.
00:57:21 Only when you at some point became a bit stuck, you would ask Copilot.
00:57:26 So why did you approach it like this? Or do you recognize this?
00:57:32 **Participant**

00:57:36 Well, it's because I well, when I'm writing my own code,
00:57:38 I do like it a lot.
00:57:40 That's that.
00:57:45 I know what my own code means and what it does.
00:57:49 And I, as I noticed this time again.
00:57:54 Yeah. If I let Copilot do everything.
00:57:59 Then I won't know what what my code actually is and what it does and where to find
00:58:01 specific things.
00:58:03 Uhm.
00:58:05 So yeah.
00:58:06 Then yeah.
00:58:10 **Researcher**
00:58:14 So if you if you didn't have Copilot.
00:58:17 Would you have solved the tasks in a different way?
00:58:22 **Participant**
00:58:26 Yeah, definitely.
00:58:30 And I would first take like half an hour, figuring out exactly what.
00:58:32 But just reading through the code and then.
00:58:34 As I understood it, maybe tried to do some tasks.
00:58:36 **Researcher**
00:58:38 Right.
00:58:40 All right, that makes sense.
00:58:42 **Participant**
00:58:44 Yeah.
00:58:46 It helps because Copilot just reads through it a lot faster than it feels
00:58:48 like. Already knows what I'm dealing with.
00:58:50 **Researcher**
00:58:52 Right. So you want to.
00:58:54 Not let's Copilot take over this part of the of programming for you.
00:58:56 **Participant**
00:58:58 No, because as I just also saw.
00:59:00 I get literally gave you the task that was in the PDF and did not solve the
00:59:02 problem.
00:59:04 So for that reason I tried to fix it myself first.
00:59:06 Yeah.
00:59:08 **Researcher**
00:59:10 All right. Thanks.
00:59:12 So in the survey you were talking about which aspects of Copilot responses you
00:59:14 liked or did not like.
00:59:16 Were there any of these aspects that stood out to you that you particularly liked or
00:59:18 disliked?
00:59:20 **Participant**
00:59:22 I the part where it just told me where I could find something,
00:59:24 and it usually is really good at that kind of things.
00:59:26 And also just that it gives me a small link, like
00:59:28 the API immediately to the file or exactly where
00:59:30 all of these parts are.
00:59:32 So that I don't have to first look through all the folders where the file is
00:59:34 and specifically where all the functions are. So yeah.
00:59:36 **Researcher**
00:59:38 OK.
00:59:40 And at some point you said you were a bit confused by an answer.
00:59:42 Do you remember what made this answer confusing to you?
00:59:44 **Participant**
00:59:46 But it was like I don't know which one it was anyway.
00:59:48 Yeah. Well, then it just gives me a whole lot of code

01:00:25 and.
01:00:29 But a lot of different things it mentioned that I don't know of the code
01:00:34 because I didn't write it initially myself.
01:00:39 So it talks about functions.
01:00:41 Use state or.
01:00:45 What else did it mention before.
01:00:48 Initial state of different.
01:00:51 Functions that were already that already existed and I did not know them.
01:00:56 **Researcher**
OK.
01:00:57 So is this for you? Mostly about Copilot giving a huge piece
01:01:02 of code? Or is it about it not explaining well
01:01:05 enough what it did or is it both?
01:01:08 Or something else?
01:01:10 **Participant**
It feels like it's, you know, how do I describe this?
01:01:16 I think it's also that Copilot already understands it and.
01:01:23 It refers to specific parts of the code that I don't know yet.
01:01:30 But Copilot, as it already can just read through it
01:01:34 and understands it basically.
01:01:37 It assumes I already understand that, so I would have to ask a lot more
01:01:41 questions to.
01:01:44 Get.
01:01:46 **Researcher**
Yeah.
01:01:46 **Participant**
Understandable.
01:01:47 Yes.
01:01:48 **Researcher**
That makes sense.
01:01:50 So.
01:01:53 Yeah, I think in the survey you also indicated
01:01:55 that at some point.
01:01:57 You felt a bit frustrated and you felt like you were not successful at the tasks.
01:02:01 Can you explain why you felt this way?
01:02:01 **Participant**
Yeah.
01:02:04 Well, I always like it when
01:02:08 any kind of AI chatbots just gives me an answer, and it works immediately and then.
01:02:15 It's always really nice, but now it did not so.
01:02:19 I'll ask something of
01:02:21 something and it does not get me what I want.
01:02:23 So yeah, that get's me,
01:02:25 gets me a bit frustrated.
01:02:29 **Researcher**
Did you feel like when you ask follow up questions like.
01:02:33 It would explain what it did wrong or it would then improve its answer.
01:02:41 **Participant**
Yeah.
01:02:43 Sometimes it does, but in this case that was not really the
01:02:46 case.
01:02:47 Umm.
01:02:49 Usually when already from my own experience,
01:02:53 if I have to ask a follow up question because the code it gave me did not work
01:02:59 then usually that's follow up questions will not solve it either.
01:03:06 **Researcher**

01:03:06 **Participant**
Right.
So yeah.

01:03:08 **Researcher**
OK.

01:03:10 So last question I have for you.

01:03:12 **Participant**
Yeah.

01:03:13 **Researcher**
So.
You mentioned that.
Some of the responses from Copilot were not what you wanted.
So did you change the way you interacted with Copilot compared to how ideally you would interact based on the kind of kinds of responses that it gave?

01:03:29 **Participant**
I don't know.

01:03:38 **Participant**
Yeah.
But usually I do not let it write my whole code.
So I would ask it like how can I fix this?
And then sometimes it would give me like a few steps on how to fix it.
And now it did just give me all all of the code itself.

01:03:42 **Researcher**
Right.

01:03:45 **Participant**
It's it's a bit difficult to go through and check if everything is correct.
And before that, like it's one of the first tasks,
it literally gave me an idea, but it gave me a small piece of code and
I already recognized it from somewhere else that that could be replaced.
And I kind of understood why.

01:03:54 **Participant**
So yeah.

01:03:59 **Researcher**
So ideally you'd like step by step guidance and not.

01:04:04 **Participant**
Yeah.

01:04:04 **Researcher**
A full answer, so to say.

01:04:04 **Participant**
Yeah, so not full take over, but I just like it more as a tool.

01:04:11 **Researcher**
Yeah.

01:04:15 **Participant**
Than something that does my work.

01:04:20 **Researcher**
That makes sense. Thanks.

01:04:25 **Participant**
Because it does not do my work.

01:04:28 **Researcher**
Great.

01:04:30 **Participant**
OK.