

Participant P03

00:04:55 **Participant**
All right.

00:04:57 So this is a project.

00:05:00 And it could be used in copilot, but

00:05:05 my initial focus would be to just look at this source because that's what I would

00:05:09 > **Viewing file** (src/todo/components/footer.jsx)

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

00:05:10 **Participant**
normally do. And I see that we are in

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

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

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

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

00:05:13 **Participant**
this framework.

00:05:13 > **Viewing file** (src/todo/constants.js)

00:05:14 > **Viewing file** (src/todo/app.jsx)

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

00:05:17 > **Viewing file** (src/todo/app.jsx)

00:05:18 > **Viewing file** (src/todo/reducer.js)

00:05:19 > **Viewing file** (src/todo/constants.js)

00:05:19 **Participant**
So we're building some app.

00:05:20 > **Viewing file** (.eslintrc)

00:05:20 > **Viewing file** (src/index.js)

00:05:24 **Participant**
React's components in there.

00:05:26 > **Viewing file** (src/todo/components/item.jsx)

00:05:27 > **Viewing file** (src/todo/components/main.jsx)

00:05:27 > **Viewing file** (src/todo/components/paginationControls.jsx)

00:05:27 > **Viewing file** (src/todo/app.css)

00:05:28 > **Viewing file** (src/todo/constants.js)

00:05:29 > **Viewing file** (package.json)

00:05:33 **Participant**
A todo MVC written in React, yeah.

00:05:38 Normally what I would do is you can just

00:05:42 run the script.

00:05:44 I'm not sure how this works in Copilot web or in the web interface.

00:05:50 No, I can just. OK.

00:05:54 I can just

00:05:57 run this.

00:06:03 Yeah, run dev.

00:06:07 We don't want to have the debugger.

00:06:11 We don't need the debugger per se

00:06:21 The task, first task, change the color.

00:06:26 When there are no todos is a red box change color of this box to blue.

00:06:29 OK, so I would like to

00:06:32 see this thing running.

00:06:41 Well we can ask Copilot.

00:06:45
00:06:51
00:06:54
00:06:56
00:06:56

Let's.

Run this program.

Uh.

GitHub dev web.

Chat (window: 0)

> Agent

github.copilot.workspace

> Context

workspace, viewport (package.json)

> Message

@workspace how can I run this program in github dev web

> Response

To run this program in GitHub Codespaces (or any web-based development environment like GitHub Dev), follow these steps:

1. **Install Dependencies:** Open the integrated terminal in GitHub Dev and run:

```
npm install
```

2. **Start the Development Server:** Run the following command to start the development server:

```
npm run dev
```

3. **Access the Application:** Once the development server starts, it will provide a URL. Open this URL in your browser to view the application.

Make sure the node and npm versions in the environment meet the requirements specified in the engines field of the [package.json](#) file (node >= 18.13.0 and npm >= 8.19.3).

00:07:12
- 00:07:15

> Response rendering

00:07:18

Participant

Yeah.

00:07:34

So it's apparently installing all these dependencies.

00:07:40

Thought it worked automatically, but apparently not. OK, run dev.

00:07:47

Yeah, now it's installed,

00:07:48

so then it's opening. Yeah. Great.

00:07:54

There's no todos yet.

00:07:56

All right, since we have Copilot

00:08:00

let's just do it quickly.

00:08:07

Where is the red box defined when I

00:08:14

Chat (window: 0)

> Agent

github.copilot.default

> Context

viewport (package.json)

> Message

Where is the red box defined when I dont have any todos yet

> Response

The red box or placeholder UI that appears when there are no todos is likely defined in the React component responsible for rendering the main application view. This could be a message or visual element displayed when the todo list is empty.

To locate it:

1. Look for a component that renders the todo list, such as `App.js`, `TodoList.js`, or similar.
2. Search for conditional rendering logic that checks if the todo list is empty. For example:

```
{todos.length === 0 && <div className="empty-state">No todos yet</div>}
```

3. Check the CSS file (e.g., todomvc-app-css or a custom CSS file) for styles related to the class used for the red box.

If you share the relevant React component code, I can help pinpoint the exact location.

00:08:14

Participant

don't have any todos yet.

00:08:15
- 00:08:20

> Response rendering

00:08:22

Participant

At workspace.

00:08:28

Chat (window: 0)

> Agent

github.copilot.workspace

> Context

workspace, viewport (package.json)

> Message

@workspace Where is the red box defined when I dont have any todos yet

> Response

The red box that appears when there are no todos is defined in the CSS file [src/todo/app.css](#). Specifically, it is styled using the `.todo-list-empty-container` and `.todo-list-empty-message` classes:

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

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

These styles are applied in the [Main\(src/todo/components/main.jsx\)](#) component in [src/todo/components/main.jsx](#) when there are no visible todos:

```
File: src/todo/components/main.jsx
<div className="todo-list-empty-container">
  <span className="todo-list-empty-message">No TOD0s here yet</span>
</div>
```

00:08:40

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

00:08:40
- 00:08:43

> Response rendering

00:08:40

Participant

For me, it's taking too long, so I will find it myself,

00:08:40

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

00:08:41

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

00:08:44

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

00:08:44

Participant

which I know that it's because it's. I'm using the whole code base as input.

00:08:49

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

00:08:52

Participant

We'll probably even do with, ah, this doesn't work.

00:08:57

What I normally do is

00:09:00

just search for for no todos.

00:09:02

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

00:09:04

Participant

Yes, here it is.

00:09:07

Todo list empty message.

00:09:11

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

00:09:11

Participant

That one is defined here.

00:09:14

Now I want to change this color to

00:09:18

this newly defined color. Because this is the easiest way to just

00:09:22

find it.

00:09:23

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

00:09:28

Participant

Let me do that [unintelligible] because you just did.

00:09:30

Is that auto-reloading? Yes it does. OK.

00:09:34

So when there's no todo, there's red box, verification,

00:09:37

refresh the application, confirm that the box is blue.

00:09:40

Yes, we did.

00:09:41

The placeholder text of the input field is what needs to be done. That's probably in the same folder,

00:09:46

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

00:09:46

Participant

where was it?

00:09:49

There, [unintelligible] wasn't there.

00:09:53

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

00:09:54

Participant

Should be done and that should become

00:09:59

enter a task here.

00:10:04

OK.

00:10:05
- 00:10:11

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

00:10:05

Participant

Enter a task here.

00:10:12

Then we can reload the application.

00:10:16

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

00:10:18

Participant

Yeah.

00:10:20

And.

00:10:22

Practice task C, the todos are currently displayed in the order they were added, modify so they're alphabetically ordered.

00:10:25

So we go to.

00:10:30

Where was it again.

00:10:35

00:10:37
00:10:38

00:10:42
00:10:46
00:10:48
00:10:54
00:11:03
00:11:03
00:11:04
00:11:08
00:11:14
00:11:16
00:11:20
00:11:23

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

Participant

In comp main.

Visible todos, todo filter.

Umm.

Use memo.

[unintelligible] something here.

Yeah, so

here we are

defining the todos and I know I'm not yet using Copilot,

but I will probably do that in a minute for this.

So I can probably

Yeah, at main.jsx.

To order the todos alphabetically.

Chat (window: 0)

> Agent

github.copilot.default

> Context

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

> Message

I want to order the todos alphabetically

> Response

To order the todos alphabetically, you can modify the [visibleTodos](#) ([src/todo/components/main.jsx](#)) logic in the [useMemo](#) ([node_modules/@types/react/index.d.ts](#)) hook to sort the todos by their title or any other property before filtering them. Here's how you can do it:

Updated Code

```
File: src/todo/components/main.jsx
const visibleTodos = useMemo(
  () =>
    todos
      .slice() // Create a shallow copy to avoid mutating the
original array
      .sort((a, b) => a.title.localeCompare(b.title)) // Sort
alphabetically by title
      .filter((todo) => {
        if (route === "/active") return !todo.completed;
        if (route === "/completed") return todo.completed;
        return true;
      }),
  [todos, route]
);
```

Explanation

1. **slice()**: Creates a shallow copy of the [todos](#) ([src/todo/components/main.jsx](#)) array to avoid mutating the original array.
2. **sort()**: Sorts the todos alphabetically using `localeCompare` on the `title` property.
3. **filter()** ([src/todo/components/main.jsx](#)): Filters the todos based on the current route (`/active`, `/completed`, or `all`).

Result

This will ensure that the todos are displayed in alphabetical order within the filtered list.

00:11:25
- 00:11:29

> Response rendering

00:11:28

Participant

It should be faster.

00:11:31

Yeah.

00:11:33

[unintelligible]

00:11:42

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

00:11:46

Participant

Did this fully work?

00:11:57

Yeah, that works.

00:11:59

And that's the practice tasks.

00:12:02

Researcher

Great. Thanks.

00:12:03

Participant

Stop the time.

00:12:06

Researcher

There we go, very speedy.

00:12:09

Alright, I'll send you

00:12:12

the actual tasks then.

00:12:14

Participant

Yep.

00:12:14

Researcher

Thanks for thinking out loud the way you're doing, by the way.

00:12:16

That's helping a lot.

00:12:25

Great. I'll send it in the chat here.

00:12:31

Let's see. You have 33 minutes left.

00:12:35

Participant

All right.

00:12:35

Researcher

So yeah, don't feel hurried.

00:12:38

Participant

I won't.

00:12:40

But I think you are going to stop me. Or not.

00:12:40

Researcher

Great.

00:12:43

Participant

Maybe I will be done.

00:13:10

Reading task A right now.

00:13:13

Not yet

00:13:13

B and C. We will do stepwise.

00:13:17

Keep everything in my own working memory.

00:13:22

Right.

00:13:26

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

00:13:27

Participant

Let's check myself, so yeah.

00:13:29

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

00:13:30

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

00:13:30

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

00:13:36

Participant

Each memo was used memoed.

00:13:41

OK.

00:13:42

That's something memorized.

00:13:47

So maybe interesting to know for the record I have never done anything in React yet.

00:13:51

00:13:53

I'm I'm quite an advanced programmer but

00:13:57

mostly backend stuff and no no React framework or

00:14:02
00:14:05
00:14:08
00:14:12

hardly any frontend frameworks whatsoever.

What does

use memo?

Chat (window: 0)

> **Agent**

github.copilot.default

> **Context**

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

> **Message**

What does `useMemo` do?

> **Response**

[useMemo \(node_modules/@types/react/index.d.ts\)](#) is a React Hook that memoizes the result of a computation, ensuring that the computation is only re-executed when its dependencies change. It is used to optimize performance by avoiding unnecessary recalculations.

Syntax

```
const memoizedValue = useMemo(() => computeValue(), [dependencies]);
```

Key Points

1. **Purpose:** It caches the result of a function so that the function is not re-run on every render unless one of its dependencies changes.
2. **Dependencies:** The second argument is an array of dependencies. If any dependency changes, the function is re-executed.
3. **Use Case:** It is useful for expensive calculations or derived data that doesn't need to be recalculated on every render.

Example in [src/todo/components/footer.jsx](#)

```
const activeTodos = useMemo(() => todos.filter((todo) => !todo.completed), [todos]);
```

- **What it does:** Filters the [todos \(src/todo/components/footer.jsx\)](#) array to get only the active (not completed) todos.
- **Why use [useMemo \(node_modules/@types/react/index.d.ts\)](#):** Without [useMemo \(node_modules/@types/react/index.d.ts\)](#), this filtering would happen on every render, even if [todos \(src/todo/components/footer.jsx\)](#) hasn't changed. By using [useMemo \(node_modules/@types/react/index.d.ts\)](#), the filtering is only recalculated when the [todos \(src/todo/components/footer.jsx\)](#) array changes.

When to Use [useMemo \(node_modules/@types/react/index.d.ts\)](#)

- When you have expensive computations that don't need to run on every render.
- When you want to avoid unnecessary re-renders caused by derived data.

00:14:13
- 00:14:16

> **Response rendering**

00:14:17

Participant

Because that's what the todos

00:14:19

are coming from.

00:14:23

That memorizes the result. OK.

00:14:30

The caching thing.

00:14:30

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

00:14:32

Participant

Oh.

00:14:36 > **Viewing file** (src/todo/components/item.jsx)
00:14:36 **Participant**
What files we have here.
00:14:39 This is just the way an item looks like [unintelligible].
00:14:44 > **Viewing file** (src/todo/components/paginationControls.jsx)
00:14:44 **Participant**
Pagination.
00:14:47 > **Viewing file** (src/todo/components/main.jsx)
00:14:57 > **Viewing file** (src/todo/components/input.jsx)
00:14:58 > **Viewing file** (src/todo/components/footer.jsx)
00:15:00 > **Viewing file** (src/todo/components/header.jsx)
00:15:02 > **Viewing file** (src/todo/components/footer.jsx)
00:15:02 **Participant**
I'm looking what is the
00:15:06 the attempt that they already did.
00:15:09 They already programmed to keep this in state so.
00:15:10 > **Viewing file** (src/todo/components/header.jsx)
00:15:11 > **Viewing file** (src/todo/components/input.jsx)
00:15:14 > **Viewing file** (src/todo/components/main.jsx)
00:15:14 **Participant**
It's done here on the main JS.
00:15:19 You have the current page and the last item added. But nothing like dates.
00:15:25 Now I would be like, OK, sure.
00:15:29 Maybe at workspace, maybe. Oh, wait, we,
00:15:30 > **Viewing file** (src/index.js)
00:15:31 > **Viewing file** (src/todo/reducer.js)
00:15:33 **Participant**
here's something interesting.
00:15:48 Load state loads [unintelligible].
00:15:51 > **Viewing file** (src/todo/app.jsx)
00:15:51 > **Viewing file** (src/todo/constants.js)
00:15:53 > **Viewing file** (src/todo/app.jsx)
00:15:54 > **Viewing file** (src/todo/app.css)
00:15:58 **Participant**
I would like to
00:16:03 store
00:16:05 all todo items to my local
00:16:10 storage.
00:16:16 Yes.
00:16:20 **Chat** (window: 0)
> **Agent**
github.copilot.workspace
> **Context**
workspace, viewport (src/todo/app.css)
> **Message**
@workspace I would like to store all todo items to my local storage
> **Response**
To store all todo items in local storage, you can use the [saveState](#)
([src/todo/reducer.js](#)) function already defined in your [src/todo/reducer.js](#). This
function saves the current state of todos to local storage whenever the state changes.

Here's how you can ensure that todos are saved to local storage:

1. **Update the [todoReducer \(src/todo/reducer.js\)](#):** The [todoReducer \(src/todo/reducer.js\)](#) already calls [saveState \(src/todo/reducer.js\)](#) after every state update. This ensures that any changes to the todos are automatically saved to local storage.

```
// filepath: /workspaces/copilot-study-todo/src/todo/reducer.js
const saveState = (state) => {
  localStorage.setItem("todos", JSON.stringify(state));
};

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

  saveState(newState); // Save the updated state to local
storage
  return newState;
};
```

2. **Load Todos from Local Storage on App Initialization:** The [loadState \(src/todo/reducer.js\)](#) function in the reducer already retrieves todos from local storage. This is used in the [App \(src/todo/app.jsx\)](#) component in [src/todo/app.jsx](#) to initialize the state.

```
// 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 "../app.css";

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

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

```

With these changes, your app will automatically save todos to local storage whenever they are added, updated, or removed, and it will load them from local storage when the app initializes.

00:16:24

Participant

Takes even longer.

00:16:32
- 00:16:39

> Response rendering

00:16:43

Participant

[unintelligible]

00:16:47

[unintelligible]

00:16:57

It updates any changes to todos to local storage.

00:17:15

Yeah, sure.

00:17:17

Can I automatically apply those? I think.

00:17:23

It's too bad.

00:17:26

Was in the reducer.

00:17:28

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

00:17:32

Participant

We use the Git integration to see what was actually changed.

00:17:44

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

00:17:47

Participant

Where did it remove the generate unique UID?

00:17:53

Possibly that myself, but sure, we have the todo reducer.

00:17:58

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

00:18:00
- 00:18:14

00:18:00

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

Participant

This one.

00:18:06

It's fixing the typo.

00:18:15

Doesn't seem like anything particularly changed except for the load state.

00:18:23

Weird, but.

00:18:32

Load todos from local storage on app

00:18:34

initiation, initialization.

00:18:38

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

00:18:38

Participant

Basics.

00:18:39

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

00:18:40
- 00:18:44

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

00:18:46

Participant

Sure.

00:18:53

It's saying that this should be enough. Test.

00:19:04

It doesn't work.

00:19:17

That's that's very stupid.

00:19:19

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

00:19:24

Participant

Made the changes you suggested.

00:19:30

Chat (window: 0)

> **Agent**

github.copilot.default

> **Context**

viewport (src/todo/reducer.js)

> **Message**

I made the changes you suggested but it isnt working

> **Response**

The issue seems to be a typo in your code. Specifically:

1. Typo in [loadState \(src/todo/reducer.js\)](#):

- You are trying to load the state from [localStorage \(src/todo/reducer.js\)](#) using the key "todoS", but in [saveState \(src/todo/reducer.js\)](#), you are saving it with the key "todos". This mismatch will cause the app to fail to load the saved state.

Fix: Update the key in [loadState \(src/todo/reducer.js\)](#) to match the key used in [saveState \(src/todo/reducer.js\)](#).

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

2. Typo in the [LOAD_STATE \(src/todo/reducer.js\)](#) case:

- In your reducer, you have a typo in the action type [LOADSTATE \(src/todo/reducer.js\)](#). It should be [LOADSTATE \(src/todo/reducer.js\)](#).

Fix: Correct the typo in the case statement.

```
case LOAD_STATE: // Fix the typo here  
  newState = action.payload;  
  break;
```

Updated Code

Here is the corrected version of your [src/todo/reducer.js](#):

```
File: src/todo/reducer.js
import { ADD_ITEM, UPDATE_ITEM, REMOVE_ITEM, TOGGLE_ITEM,
REMOVE_ALL_ITEMS, TOGGLE_ALL, REMOVE_COMPLETED_ITEMS, LOAD_STATE } from
"./constants";

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

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

function generateUniqueID() {
  return Date.now().toString() + Math.floor(Math.random() *
1000).toString();
}

export const initialState = loadState();

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

  saveState(newState); // Save the updated state to localStorage
}
```

```
    return newState;  
};
```

Steps to Verify

1. Save the updated file.
2. Restart your application to ensure the changes are applied.
3. Add some todos, refresh the page, and verify that the todos persist in local storage.

If the issue persists, let me know!

00:19:31

Participant

Isn't working.

00:19:34
- 00:19:49

> Response rendering

00:19:38

Participant

Very much vibe coding, sure.

00:19:45

Load state.

00:19:51
- 00:19:52

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

00:19:52

Participant

Yeah.

00:19:54

That's weird, thst

00:19:55

it didn't suggest this earlier, but sure.

00:20:03
- 00:20:04

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

00:20:04

Participant

Oh, it didn't save it.

00:20:08

Oh, this is VS code, you need to save everything explicitly.

00:20:12

I always forget that.

00:20:12 So here's the updated code.
00:20:14 Yeah. Thanks.
00:20:15 Are there any more changes?
00:20:20 Yeah.
00:20:22 I'm curious to see them.
00:20:25 And I need to go to this, it's only here.
00:20:33 So normally I really like the Git view.
00:20:44 Cancel that.
00:20:48 And since it only did this here, it's probably also still wrong here.
00:20:50 > **Viewing file** (src/todo/app.jsx)
00:20:53 **Participant**
No, it is here.
00:20:57 > **Viewing file** (src/todo/components/main.jsx)
00:21:01 **Participant**
Yeah.
00:21:03 Nice.
00:21:14 It's.
00:21:16 That's good.
00:21:19 Yeah. Mark as completed.
00:21:21 Good one, didn't do that.
00:21:26 Work.
00:21:29 Edit, we already did that and delete we also already did that, so that's good.
00:21:34 And we got to task B.
00:21:38 If the user refreshes while typing, modify the application so the input field automatically saves every few seconds.
00:21:50 Yeah.
00:21:54 It's probably in the,
00:21:56 > **Viewing file** (src/todo/components/item.jsx)
00:22:00 > **Viewing file** (src/todo/components/header.jsx)
00:22:00 **Participant**
[unintelligible], in the input field we'd need it.
00:22:00 > **Viewing file** (src/todo/components/input.jsx)
00:22:07 **Participant**
Change the code to
00:22:13 store the inputted text
00:22:21 every two seconds to
00:22:26 local states.
00:22:29 Restore
00:22:32 it when reopen
00:22:38 initialize if any text was entered.
00:22:43 **Chat** (window: 0)
> **Agent**
github.copilot.default
> **Context**
viewport (src/todo/components/input.jsx)
> **Message**
Change the code to store the input field text every 2 seconds to local state, and restore it when I initialize if any text was entered
> **Response**
To store the input field text every 2 seconds and restore it when the component initializes, you can use the useEffect hook along with useState. Here's the updated code:

Updated Code

```

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

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

export function Input({ onSubmit, placeholder, label, defaultValue,
onBlur }) {
  const [inputValue, setInputValue] = useState(() => {
    // Restore saved input value from localStorage
    return localStorage.getItem("todo-input") || "";
  });

  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);
        setInputValue(""); // Clear local state
        localStorage.removeItem("todo-input"); // Clear saved
input
        e.target.value = "";
      }
    },
    [onSubmit]
  );

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

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

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

  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}
        onBlur={handleBlur}
        onKeyDown={handleKeyDown}
      />
      <label className="visually-hidden" htmlFor="todo-input">

```

```
        {label}  
      </label>  
    </div>  
  );  
}
```

Explanation of Changes

1. State Initialization:

- The `inputValue` state is initialized with the value from `localStorage` (if it exists) or an empty string.

2. Handle Input Change:

- The `handleChange` function updates the `inputValue` state whenever the user types in the input field.

3. Save to Local Storage:

- A `useEffect` hook is used to save the `inputValue` to `localStorage` every 2 seconds.

4. Restore on Initialization:

- When the component initializes, the saved value from `localStorage` is restored into the `inputValue` state.

5. Clear on Submit:

- When the user submits the input (presses Enter), the `inputValue` is cleared from both the state and `localStorage`.

Behavior

- The input field text is saved to `localStorage` every 2 seconds.
- If the user refreshes the page or reopens the app, the saved text is restored in the input field.
- When the user submits the input, the saved text is cleared.

> Response rendering

Participant

Oh, we're gonna need more.

And I don't know enough about the code base if this is the nice way to do it, because probably it should go through the reducer.

But, anyway,

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

Participant

I'm a vibe coder now.

Let's just try immediately if it works.

No doesn't work yet, because I need to store it first, save it first. Probably.

Not yet.

Oh, now I can see the Git thingies. Oh, I need to wait two seconds.

Now let's see what the assignment was. Every few seconds,

yeah, not when I stop typing.

I need to do submit it.

It works.

All right. Then we go to task C, pagination.

Implement pagination to limit number of displayed todos.

Modify the application to display a fixed number of. Yeah, alright. With options to show one, three or five.

The next button should be disabled and grey.

Yeah.

Very nice explanation.

This is inconvenient.

Todo 1.

OK.

This is just interesting,

I want to know.

Why

can't I make a to do that starts

Chat (window: 0)

> Agent

`github.copilot.default`

> Context

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

> Message

Why can't I make a todo that starts with a number

> Response

The issue lies in the [hasValidMin \(src/todo/components/input.jsx\)](#) function, which is used to validate the input. It only checks the length of the input but does not account

00:22:44
- 00:22:59

00:22:50

00:23:13

00:23:18

00:23:23

00:23:24

00:23:25

00:23:32

00:23:38

00:23:49

00:23:54

00:24:08

00:24:11

00:24:20

00:24:34

00:24:37

00:24:42

00:24:59

00:25:05

00:25:11

00:25:21

00:25:24

00:25:40

00:25:51

00:25:54

00:25:55

00:25:57

00:25:59

00:26:02

00:26:06

for inputs starting with numbers. However, there is no restriction in the provided code that explicitly prevents creating todos starting with numbers.

If you are unable to create todos starting with numbers, it might be due to one of the following reasons:

1. Trimming the Input

- The [handleKeyDown \(src/todo/components/input.jsx\)](#) function trims the input value using [e.target.value.trim\(\) \(src/todo/components/input.jsx\)](#). If the input contains only spaces or is empty after trimming, it will not pass validation.

2. Validation Logic

- The [hasValidMin \(src/todo/components/input.jsx\)](#) function only checks if the input length is greater than or equal to 2. If the input is valid (e.g., "12"), it should pass this validation.

Debugging Steps

To debug the issue: 1. Add a `console.log` inside the [handleKeyDown \(src/todo/components/input.jsx\)](#) function to check the value of [e.target.value \(src/todo/components/input.jsx\)](#):

```
const handleKeyDown = useCallback(
  (e) => {
    if (e.key === "Enter") {
      const value = e.target.value.trim();
      console.log("Input value:", value); // Debugging

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

      onSubmit(value);
      setInputValue(""); // Clear local state
      localStorage.removeItem("todo-input"); // Clear saved
input      e.target.value = "";
    }
  },
  [onSubmit]
);
```

2. Test creating a todo starting with a number (e.g., "123") and check the console output.

Fix (If Needed)

If you want to ensure that todos starting with numbers are allowed, you can modify the [hasValidMin \(src/todo/components/input.jsx\)](#) function to include additional checks (if necessary). However, based on the current code, there is no restriction preventing numbers.

If the issue persists, it might be caused by other parts of the application (e.g., the reducer or UI logic). Let me know if you'd like to investigate further!

00:26:06

Participant
with a number.

00:26:07
- 00:26:13

> Response rendering

00:26:09

Participant
Just curious.

00:26:17

Oh.

00:26:19

OK.

00:26:19

It's just it's that yeah, sure.

00:26:22

Todo 3, todo 4, todo 5, todo 6,

00:26:28

todo 7, todo 8, todo 9, todo 10.

00:26:33

And now we're getting a problem, yeah.

00:26:40

01.

00:26:44

02.

00:26:47

Todo 03.

00:26:49

04.

00:26:53

05. Todo 06.

00:26:59

07.

00:27:02

Todo 08, todo 09.

00:27:08

Todo 10, todo 11, todo 12. Should be enough.

00:27:17

And let's just try and copy the whole.

00:27:21

Vibe code it, code it style.

00:27:28

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

00:27:28

Participant

It should go to main as context.

00:27:34

Yeah.

00:27:35

Chat (window: 0)

> Agent

github.copilot.default

> Context

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

> Message

Currently, the TODO application displays all tasks in a single list. Implement pagination to limit the number of displayed TODOs per page, allowing users to easily navigate through their tasks. Modify the application to display a fixed number of TODOs per page, with options to show 1, 3, or 5 (the default value) TODOs at a time. Also implement navigation controls to allow users to move between pages of TODOs. The application should have "Next" and "Previous" buttons to navigate between pages. The "Next" button should be disabled and grey when the user is on the last page, and the "Previous" button should be disabled and grey when the user is on the first page. The page that the user is currently on should also be indicated. An attempt to implement the navigation functionality was made, but it is not working correctly at this moment. Please fix or reimplement the navigation and implement the other requirements.

> Response

To implement pagination with the specified requirements, we need to modify the [Main \(src/todo/components/main.jsx\)](#) component to handle the following:

1. **Limit the number of TODOs displayed per page.**
2. **Allow users to select the number of TODOs per page (1, 3, or 5).**
3. **Add "Next" and "Previous" buttons for navigation.**
4. **Disable navigation buttons when on the first or last page.**
5. **Indicate the current page number.**

Here's the updated implementation:

Updated Code for [Main.jsx \(src/todo/components/main.jsx\)](#)

```

File: src/todo/components/main.jsx
import { Item } from "../item";
import classNames from "classnames";
import { TOGGLE_ALL } from "../constants";
import { useLocation } from "react-router-dom";
import { useMemo, useCallback, useState, useEffect } from "react";

export function Main({ todos, dispatch }) {
  const { pathname: route } = useLocation();
  const [currentPage, setCurrentPage] = useState(1);
  const [itemsPerPage, setItemsPerPage] = useState(5); // Default
items per page
  const [lastAddedItem, setLastAddedItem] = useState(null);

  const visibleTodos = useMemo(
    () =>
      todos
        .slice() // Create a shallow copy to avoid mutating the
original array
        .sort((a, b) => a.title.localeCompare(b.title)) // Sort
alphabetically by title
        .filter((todo) => {
          if (route === "/active") return !todo.completed;
          if (route === "/completed") return todo.completed;
          return true;
        }),
    [todos, route]
  );

  const paginatedTodos = useMemo(() => {
    const startIndex = (currentPage - 1) * itemsPerPage;
    const endIndex = startIndex + itemsPerPage;
    return visibleTodos.slice(startIndex, endIndex);
  }, [visibleTodos, currentPage, itemsPerPage]);

  const totalPages = Math.ceil(visibleTodos.length / itemsPerPage);

  const toggleAll = useCallback(
    (e) => dispatch({ type: TOGGLE_ALL, payload: { completed:
e.target.checked } }),
    [dispatch]
  );

  const handleNextPage = () => {
    if (currentPage < totalPages) {
      setCurrentPage((prevPage) => prevPage + 1);
    }
  };

  const handlePreviousPage = () => {
    if (currentPage > 1) {
      setCurrentPage((prevPage) => prevPage - 1);
    }
  };

  const handleItemsPerPageChange = (e) => {
    setItemsPerPage(Number(e.target.value));
    setCurrentPage(1); // Reset to the first page when items per
page changes
  };

  useEffect(() => {
    if (todos.length > 0) {
      const lastItem = todos[todos.length - 1];

```



```

        setLastAddedItem(lastItem);
    }
}, [todos, lastAddedItem]);

return (
    <main className="main" data-testid="main">
        {visibleTodos.length > 0 ? (
            <>
                <div className="toggle-all-container">
                    <input
                        className="toggle-all"
                        type="checkbox"
                        id="toggle-all"
                        data-testid="toggle-all"
                        checked={visibleTodos.every((todo) =>
todo.completed)}
                        onChange={toggleAll}
                    />
                    <label className="toggle-all-label"
htmlFor="toggle-all">
                        Toggle All Input
                    </label>
                </div>
                <ul className={classnames("todo-list")} data-
testid="todo-list">
                    {paginatedTodos.map((todo, index) => (
                        <Item todo={todo} key={todo.id} dispatch=
{dispatch} index={index} />
                    ))}
                </ul>
                <div className="pagination-controls">
                    <button
                        className="pagination-button"
                        disabled={currentPage === 1}

```

```

        onClick={handlePreviousPage}
      >
        Previous
      </button>
      <span className="pagination-info">
        Page {currentPage} of {totalPages}
      </span>
      <button
        className="pagination-button"
        disabled={currentPage === totalPages}
        onClick={handleNextPage}
      >
        Next
      </button>
    </div>
    <div className="items-per-page">
      <label htmlFor="items-per-page">Items per page:

      <select
        id="items-per-page"
        value={itemsPerPage}
        onChange={handleItemsPerPageChange}
      >
        <option value={1}>1</option>
        <option value={3}>3</option>
        <option value={5}>5</option>
      </select>
    </div>
  </>
) : (
  <div className="todo-list-empty-container">
    <span className="todo-list-empty-message">No TODOs
here yet</span>
  </div>
)}
</main>
);
}

```

Explanation of Changes

1. Pagination Logic:

- paginatedTodos is derived from [visibleTodos](#) ([src/todo/components/main.jsx](#)) using the [currentPage](#) ([src/todo/components/main.jsx](#)) and [itemsPerPage](#) ([src/todo/components/main.jsx](#)) values.
- totalPages is calculated based on the total number of [visibleTodos](#) ([src/todo/components/main.jsx](#)) and [itemsPerPage](#) ([src/todo/components/main.jsx](#)).

2. Navigation Controls:

- "Next" and "Previous" buttons are added to navigate between pages.
- Buttons are disabled when on the first or last page.

3. Items Per Page Selector:

- A dropdown allows users to select the number of items per page (1, 3, or 5).
- Changing the value resets the current page to 1.

4. UI Updates:

- The current page and total pages are displayed.

CSS Updates (Optional)

You can add styles for the pagination controls and dropdown in your `app.css` file:

```
.pagination-controls {
  display: flex;
  justify-content: center;
  align-items: center;
  gap: 10px;
  margin-top: 20px;
}

.pagination-button {
  padding: 10px;
  margin: 5px;
  border: none;
  cursor: pointer;
}

.pagination-button:disabled {
  background-color: #ccc;
  cursor: not-allowed;
}

.pagination-info {
  font-weight: bold;
}

.items-per-page {
  display: flex;
  justify-content: center;
  align-items: center;
  margin-top: 10px;
}

.items-per-page label {
  margin-right: 5px;
}
```

Testing the Implementation

1. Add more than 5 TODOs to test pagination.
2. Use the "Next" and "Previous" buttons to navigate between pages.
3. Change the "Items per page" dropdown to verify the functionality.

Let me know if you encounter any issues!

> Response rendering

Participant

So now,

just asking

Copilot to fix it for me, because I think the explanation was

already very

good. Let's just see what it does, because it's not too complicated and

there's lots of work already been done.

So probably all the work that has already been done is steering enough in the right direction

that it will just work.

00:27:37
- 00:28:01

00:27:48

00:27:50

00:27:51

00:27:56

00:27:59

00:28:03

00:28:05

00:28:11

00:28:13

00:28:16 I think. And otherwise, we will
00:28:17 **> Implementing Copilot code** by copying full snippet (src/todo/components/main.jsx)
00:28:17 **Participant**
rollback.
00:28:23 Get some CSS [unintelligible]. Yeah, let's do that.
00:28:29 **> Viewing file** (src/todo/app.css)
00:28:29 **Participant**
Busy with it.
00:28:34 [unintelligible] is already.
00:28:49 That works.
00:28:57 Yeah, we do need to add CSS because the styling is
00:29:03 not correct.
00:29:05 Pagination controls until item [unintelligible].
00:29:08 **> Implementing Copilot code** by copying full snippet (src/todo/app.css)
00:29:16 **Participant**
Restart.
00:29:23 Yeah. Interesting choice, that it's indeed making the boxes grey
00:29:30 when we're on the the first and last page.
00:29:34 Probably not
00:29:35 what the user wants, but it was in the prompt
00:29:37 and it's also in the description, so I'm good with that.
00:29:41 Add 10 or more todos, give them sequential numbers, yeah.
00:29:43 Confirm that the users can selection options 1, 3, or 5 todos to show at a time, yeah, we have
done so.
00:29:50 Select different page size options, yeah, that works. Confirm that the current page
00:29:55 is indicated clearly, yeah.
00:29:58 Next button, previous button work.
00:30:03 Is it also disabled?
00:30:05 Yes, it is. Confirm that the todo count on each page matches the selected page size.
00:30:15 Yeah, yeah.
00:30:20 At least one and at most the selected and confirm that they're all present.
00:30:24 It is.
00:30:24 And what if I delete this one and this one?
00:30:29 Then we're in the weird situation,
00:30:30 but sure, now it's gone. Yeah.
00:30:36 That is fine by me.
00:30:46 Yeah, I think that's it.
00:30:49 **Researcher**
That's it indeed. Thanks a lot.
00:35:07 So, then, to talk about your experience.
00:35:10 **Participant**
Yep.
00:35:11 **Researcher**
How did you manage to find your way through the code base?
00:35:14 Did this work for you?
00:35:19 **Participant**
Yeah. So it was a bit new because I'm a
00:35:21 JetBrains user and not very much VS code. So that was a bit of a thing.
00:35:26 Yeah, it's almost the same.
00:35:30 Yeah, works fine.
00:35:33 **Researcher**
And how did your navigation then differ from the way you'd ideally do this in
00:35:33 **Participant**
Yeah.
00:35:38 **Researcher**
your default IDE?

00:35:43

Participant

Yeah, it's regarding the,
the chat interface that's, JetBrains doesn't,
well, it does have the chat interface, but I'm not using that one.
So, for lack of better alternative, I mostly use
a different window with Claude at my one screen.
and my default IDE on my other screen.
But I actually like this and the fact that there's just a chat window on the
right, that's actually the same for JetBrains if I were to use that one.
So the the the problems I mentioned is mostly with everything non-chat about
Copilot.

00:36:25

Researcher

Right. What so the did you like?

00:36:26

Participant

[unintelligible]

00:36:28

Researcher

What did you like exactly about having this chat window in the in the IDE?

00:36:35

Participant

So it's just easier because you don't have to switch all the time.
And what I did like, and I would need to get used to it a bit more
is the fact that it's following you, like a files that you have open.
So normally what I do in a project is I copy paste all files that I'm using or
all significant files to
Claude. And then I have one big context window in
which I can work, one big project.
And here you already have that. Not entirely because
when you want it to actually use the whole repository that you need to do at
workspace and then it's getting slower.
But it appears like if I'm opening one file, it's using that file as context,
but it is also able to
load files that you reference.
I have the idea.
Because it's suggesting the [unintelligible] changes, which is a different file. So it, it did
have some map from me that it works. So that that's nice.
I would have loved it to be able to hit a button here and just insert the code.
Maybe it is there, but I couldn't find it yet.

00:37:58

Researcher

Right, I think you copied and pasted the code
mainly right.

00:38:01

Participant

Yeah, I did.

00:38:03

Yeah, this inserting thing I, I used that
I've used it once or twice, and it's also always goes wrong.

00:38:06

Researcher

How does this go wrong?

00:38:12

Participant

So yeah.

00:38:14

It's just inserting at little weird place. And you never want to insert it,
you always want to replace it.
You never want to insert it. No, also, when you're writing a new application,
you never want to insert things because you already have a a structure,
so that doesn't work.

00:38:17

00:38:19

00:38:24

00:38:29

00:38:31

I love the Git interface.

00:38:34

Researcher

The what interface?

00:38:35

Participant

00:38:40 Like like with Git so that you can just you have almost like a commit that you
00:38:43 can just apply a diff.
Researcher
Right.
00:38:44 **Participant**
Because that's probably interesting for you.
00:38:47 I really.
00:38:49 The productivity of vibe coding like this, especially the last task that I did.
00:38:55 It's great.
00:38:56 You're very fast, but you have no idea what's happening and
00:38:59 probably it's, or in 50% of the cases is gonna be rubbish.
00:39:03 So I want to know what it's doing and whether it's following the correct style.
00:39:09 And just looking at the diff, is the only way and it's a perfect way to
00:39:12 see what you're doing.
00:39:14 So that's why I did all the copy paste and look for what was changed.
00:39:20 Yeah.
00:39:21 **Researcher**
So you feel like the way the the generated code was presented now impacted your
00:39:25 understanding of the code if I'm correct?
00:39:29 **Participant**
Like this?
00:39:30 Yeah, it could be better.
00:39:32 **Researcher**
OK.
00:39:32 **Participant**
It would be better to show me the the file. I'm currently working on or let me go
00:39:38 through every file.
00:39:42 And present. Hey, I would do these changes,
00:39:45 these deletions and these changes, these insertions.
00:39:50 Like you're applying a commit.
00:39:53 **Researcher**
That makes a lot of sense.
00:39:54 **Participant**
It. Yeah, and it's also other tools that I used.
00:39:57 So I've used Aider for a while and that's actually working with with Git,
00:40:02 but it's a command line application. What it is doing,
00:40:06 it's just applying code.
00:40:10 And it it's keeping track with Git commits and then in your normal IDE you
00:40:14 can roll back.
00:40:15 But I mean, that's not really that the ideal either,
00:40:19 because I,
00:40:21 I want to be more in control, so I don't want it to make a commit for
00:40:24 **Researcher**
OK.
00:40:25 **Participant**
me and me being able to roll back.
00:40:27 No, I want to go through all changes before I
00:40:31 agree with committing, but that's a different way of using it.
00:40:33 **Researcher**
Right.
00:40:36 **Participant**
That's that's my style.
00:40:37 **Researcher**
And so you can't see these precise changes in this way of interacting, right?
00:40:43 **Participant**
Mm hmm.

00:40:44

Researcher

But do you still have this feeling of control in the way you're interacting right now?

00:40:48

00:40:51

Participant

In control, but I decide to give away the control.

00:40:56

Researcher

Right.

00:40:56

Participant

In, in this this last task

00:40:57

I just copied and pasted everything and it's nice that it's saying like this is the updated code for this whole file so I can just put it in.

00:41:01

00:41:08

Yeah. And then I hand away control.

00:41:12

It would be nice to see with a diff like I'm doing this and then I can scroll through and then I would be more in control.

00:41:17

00:41:20

Researcher

Right. And of course, this is a an experimental setting.

00:41:27

Would you have approached this the same in a work environment?

00:41:36

Participant

Really depends on the type of project.

00:41:40

So in the work environment I'm working in, for my [omitted] library.

00:41:45

No.

00:41:46

Maybe if it's some very weird edge case for inspiration, but yeah.

00:41:52

In the work environment where I'm using a script

00:41:55

or library to make some integrations in applications. Definitely yes.

00:42:00

[unintelligible] we have an upload surface where we, it's, it's very easy,

00:42:04

you just need to have a web page where you can upload a file. And then afterwards it's doing some very interesting [omitted] stuff with it.

00:42:09

00:42:12

But making the the bootstrap web page boilerplate code,

00:42:16

we already did that. That's all

00:42:18

AI generated and that's fine.

00:42:20

And then I scroll through it and I change a bunch.

00:42:23

So, normally.

00:42:23

Researcher

So does this depend on the difficulty of the task or the impacts of errors?

00:42:30

So what does this?

00:42:30

Participant

Yeah, the yeah, the the importance, the impact of errors. Yeah, yeah, definitely.

00:42:35

00:42:39

Yeah. Also the difficulty if it's if it's not a

00:42:42

difficult job

00:42:45

then I would do it.

00:42:49

Mm.

00:42:51

And it is impactful because if there's an error in the in the file upload service, then that would be problematic.

00:42:55

00:42:59

But not really, now I think about it.

00:43:03

Like the [omitted] library is much more important and [omitted] stuff.

00:43:10

And I'm I'm more the author of the [omitted] library and the upload service is not really my, it's my work, but it's just an application.

00:43:15

00:43:20

Researcher

So there's a sense of authorship here as well then.

00:43:23

Participant

Yeah, perhaps. Yeah.

00:43:27

Though I must say that I also have projects in between that are

00:43:31

significantly programmed using Claude in this case

00:43:36

that I feel really feel the author of because

00:43:43

I decide which code is gonna make it through to the repository and in more than

00:43:48 half of the cases, we're not doing it because it's rubbish.
00:43:53 So I'm still deciding that, and in many, many cases I'm writing the whole
00:43:57 boilerplate and giving it structure and then make it fill in the blanks exercise.
00:44:04 **Researcher**
Right. That makes sense.
00:44:08 So in in your in this survey, at the end you said you wouldn't feel
00:44:11 **Participant**
Mm hmm.
00:44:14 **Researcher**
more productive or effective at work using Copilot.
00:44:21 **Participant**
Did I say that? At least I think I said I am a lot more productive,
00:44:25 but not necessarily better.
00:44:28 **Researcher**
Right. OK.
00:44:32 So.
00:44:32 **Participant**
Or just a bit bit better.
00:44:35 **Researcher**
So is that about the quality then of the code or?
00:44:39 **Participant**
Yeah.
00:44:41 Yeah.
00:44:44 No.
00:44:47 Yeah, it makes me a lot more productive, but I am not getting smarter from it
00:44:50 **Researcher**
Mm.
00:44:52 **Participant**
myself.
00:44:53 **Researcher**
Right.
00:44:53 **Participant**
It's just doing stupid stuff,
00:44:56 dumb stuff that I don't want to think about.
00:45:02 It does help me learn a framework more.
00:45:05 That's maybe the way I value my job.
00:45:08 Yes, I am the [omitted] that is doing a significant
00:45:12 part of programming, but I'm not a software developer.
00:45:16 So being able to understand a programming framework isn't something I value
00:45:22 my, it's not really a part of my job.
00:45:25 So if I if I'm very able to
00:45:27 learn React,
00:45:29 yeah, that doesn't make me better at my job.
00:45:31 It does make me a lot more productive.
00:45:34 **Researcher**
That's fair.
00:45:35 That's fair.
00:45:36 So tying into that, do you felt like you understood this this
00:45:38 **Participant**
Mm hmm.
00:45:40 **Researcher**
code base well?
00:45:43 **Participant**
Meh.
00:45:45 But that comes with time.
00:45:47 So if I would do 10 more tasks with this then probably yes.

00:45:54 And using Copilot makes it faster
00:46:00 to do that.
00:46:03 **Researcher**
00:46:06 So you think Copilot would make you
00:46:07 understand this code base faster,
00:46:09 is what you're saying.
00:46:12 **Participant**
00:46:12 Uhm.
00:46:17 Yeah, but not directly, more indirectly.
00:46:17 Because if I were to spend,
00:46:21 like I I would understand this code base if I'm doing 10 tasks with it, 10 changes.
00:46:28 And using Copilot makes me twice as fast in making these
00:46:33 changes than if I were to do it without Copilot.
00:46:37 But I still need to do the 10 tasks.
00:46:40 **Researcher**
00:46:42 OK.
00:46:42 So you would you, you would need the same process,
00:46:45 but maybe in a different time span.
00:46:47 **Participant**
00:46:51 Yeah. And the individual steps are just faster.
00:46:51 **Researcher**
00:46:51 OK.
00:46:51 That makes sense.
00:46:54 Then.
00:46:56 So you rated some aspects of the responses.
00:46:59 **Participant**
00:47:01 Yep.
00:47:01 **Researcher**
00:47:07 Were, did any of these stand out to you?
00:47:07 **Participant**
00:47:08 No, not really.
00:47:08 Like, it was, I wouldn't say spot on, but maybe yeah, maybe I would say pretty spot
00:47:14 on.
00:47:16 It,
00:47:18 yeah well it didn't take too long.
00:47:21 Like in some cases it did take a while, but sure.
00:47:26 Fine.
00:47:28 They were.
00:47:28 **Researcher**
00:47:30 And I saw you,
00:47:34 you rated length and tone a bit lower than the rest.
00:47:34 Was there a reason for this?
00:47:38 **Participant**
00:47:41 I'm not sure what the others were, but maybe the others were more standing
00:47:41 out.
00:47:41 I thought there was nothing wrong with tone per se, and the length,
00:47:50 it was complete in a way.
00:47:51 Not too long.
00:47:52 It was long sometimes, but then you read through to us and now it's
00:47:55 just complete.
00:47:58 It's very different a little bit. So like for example this last thing is
00:48:03 pretty spot on.
00:48:06 And
00:48:09 I think here in the beginning. Yeah, this one.
00:48:14 Basically, no here.
00:48:18 This was weird because it's suggesting the code that was already there.

00:48:25 Yeah.
00:48:30 I didn't.
00:48:31 No, oh no, here. Now did fix the typo.
00:48:36 But then it's weird,
00:48:37 like just say there's a typo in here and don't give the whole block.
00:48:42 **Researcher**
Right.
00:48:43 **Participant**
Also it's it's it's missing some code that should be in between here.
00:48:49 And the same here this it's giving the whole file while you just fix this line.
00:48:53 So that was a bit long.
00:48:55 **Researcher**
So so is this.
00:48:56 Are these issues that will be fixed by the by your suggestion of having these
00:49:01 file diffs?
00:49:02 **Participant**
Yeah, yeah. Now I think about it, yes.
00:49:05 **Researcher**
OK.
00:49:08 Let me see what else do we want to, do
00:49:10 we want to know.
00:49:13 So OK.
00:49:16 Say you wouldn't have this Copilot chat.
00:49:18 How would you have solved these tasks?
00:49:20 Could you talk me through your strategy?
00:49:22 **Participant**
Probably I can't use Claude either.
00:49:25 **Researcher**
I,
00:49:25 I mean, would that be your your your second
00:49:28 choice then?
00:49:29 **Participant**
Yeah, because that's what I'm using right now.
00:49:33 **Researcher**
Mm hmm.
00:49:34 **Participant**
But yeah.
00:49:36 So then then I would have.
00:49:41 Yeah, thinking about the practice tasks, by the way,
00:49:44 that's my default strategy and that's also what I defaulted to right now
00:49:49 because.
00:49:50 I don't know.
00:49:51 I'm a bit in between. Like I started vibe coding like this last
00:49:56 few weeks, maybe months.
00:49:59 If there's a nice if if it's a small stuff like there should be a file where
00:50:03 this color code is defined and should be this color codes that that's easy.,
00:50:07 that, would do it like that. Probably,
00:50:13 it's, it's what I started the whole
00:50:14 session with so,
00:50:17 we are in this repository.
00:50:19 We have a package.json,
00:50:21 so we are in a MVC framework.
00:50:23 We have some dependencies.
00:50:24 That's nothing interesting. Then there's the source folder. In the
00:50:28 source folder, there is an index.js.
00:50:30 There we see it's React thing.

00:50:34 There's the todo folder with some components.
00:50:37 Well, main is always a nice thing.
00:50:39 Main defines a page and just just go through the files like that.
00:50:46 **Researcher**
So there is this this step that you always do yourself, by orienting yourself
00:50:51 in the code base.
00:50:52 **Participant**
Yeah, almost always, yeah.
00:50:57 Yeah, just yeah. Start with the package.json, or a cargo.toml or whatever.
00:51:03 And then you look into the source folder.
00:51:08 And look for
00:51:10 names like module or main or index or whatever and then start working from
00:51:15 there.
00:51:17 **Researcher**
Mm hmm.
00:51:17 **Participant**
Look at some folders where interesting file names are.
00:51:21 Try to inspect their structure a bit.
00:51:23 So then I'm actually building up this mind map in my own head.
00:51:27 Where things
00:51:30 are approximately. Where are we storing stuff?
00:51:33 Where are we displaying stuff?
00:51:35 Where are we communicating stuff or whatever?
00:51:40 **Researcher**
And so what are then the the tasks that you usually use Claude for?
00:51:53 **Participant**
To, either write completely new
00:51:56 stuff, that's not really interesting to myself,
00:51:59 like scripting some stuff or things like that.
00:52:04 Obscure bugs.
00:52:08 If something weird happens and I can't really
00:52:13 reproduce it or find it or something like that.
00:52:15 Or like a new feature that's really changing
00:52:21 some
00:52:21 architectural stuff.
00:52:24 That's maybe the same like like a new project.
00:52:25 So if we have something and I want to add something new and I don't want to think
00:52:30 about architecture, then I could say hey, how would you do this?
00:52:33 And then that probably gives me some inspiration.
00:52:40 Filling the blanks.
00:52:41 So if if I have this
00:52:44 API that I need to write with and
00:52:45 I'll probably implement one or two steps myself and then give it to Claude and say
00:52:51 finish it for me.
00:52:56 **Researcher**
And how do you usually?
00:52:56 **Participant**
That's the things.
00:52:58 **Researcher**
How do you reason about whether what you're, what the LLM is doing,
00:53:03 or the tool is doing is correct?
00:53:05 How do you verify this?
00:53:06 Or do you verify this?
00:53:11 **Participant**
So if if you would for example have this this diff interface and I think
00:53:15 Claude is doing that nicely.

00:53:18 It's not giving the diff interface but, I don't know.
00:53:21 I I generally see what is changed.
00:53:26 Then I'm doing a code review. Most of the times,
00:53:28 like I'm doing it with a different author. So
00:53:34 see whether it makes sense.
00:53:34 **Researcher**
All right.
00:53:38 **Participant**
Yeah, just check your changes, I say
00:53:40 oh yeah
00:53:41 oh, I understand this or, yeah.
00:53:44 **Researcher**
Yep, that makes sense.
00:53:46 To get back to a term, you used, vibe coding.
00:53:49 I've heard of this before.
00:53:50 **Participant**
Yeah, yeah.
00:53:51 **Researcher**
Can you describe to me what you mean with this?
00:53:54 **Participant**
Yeah, so this isn't.
00:53:57 I think this is the what everyon is calling it last time.
00:54:01 It's not completely letting go of the code itself and just say make it more
00:54:07 like this and just give a complex
00:54:12 task to AI model related coding and let it fix it. And
00:54:19 surprisingly, in the last few months
00:54:22 this is very much possible.
00:54:23 And you will get working code.
00:54:28 So for some applications it's great.
00:54:32 **Researcher**
And in this case, did you experience like you were vibe
00:54:36 coding?
00:54:37 **Participant**
Yeah, the last.
00:54:38 The last one I did, yeah. With the pagination.
00:54:42 It's it's it's borderline,
00:54:43 I would say.
00:54:44 So real vibe coding would be, I want to have a todo app, make it.
00:54:51 That would be real vibe coding, but this [unintelligible]
00:54:53 a little bit like that.
00:54:56 The last task did we just paste in the assignments.
00:55:02 **Researcher**
Right.
00:55:03 **Participant**
And just copy paste the responses without thinking about it.
00:55:07 **Researcher**
So if I if I see it correctly, this is about relinquishing this control
00:55:12 whenever you want?
00:55:13 **Participant**
Yeah, yeah.
00:55:15 Yeah, which is fine in specific cases.
00:55:20 Yeah, it's fine. If in specific cases, but I wouldn't,
00:55:26 you don't expect quality.
00:55:30 **Researcher**
Yeah.
00:55:30 **Participant**

00:55:33 So if you are making a script
00:55:37 to, for example with what I did a weekend ago,
00:55:42 an automatic script to inquire some some tickets for some
00:55:43 concerts or something.
00:55:43 That's fine.
00:55:44 I'm using it myself and that's completely fine if I'm writing a [omitted] library,
00:55:50 maybe not. If I'm making a very replacable software application that's
00:55:55 very simple,
00:55:58 like we do at [omitted] for a file uploads,
00:56:01 that's fine. Also because I am reading through it one more time in the end and I
00:56:06 don't really care about a lot of stuff.
00:56:09 But if you're making a piece of
00:56:13 government infrastructure that should be maintainable and it's is really
00:56:18 complex,
00:56:19 yeah that'll be,
00:56:20 you don't want this.
00:56:22 **Researcher**
00:56:25 So if I hear it correctly, you're still choosing and controlling
00:56:29 when you're letting go of this control.
Participant
00:56:33 Yeah, I decided when I'm vibe coding, yeah.
Researcher
00:56:37 Right, OK. I have one last question for you.
Participant
00:56:38 Yeah, sure.
Researcher
00:56:40 So.
00:56:40 Over time, or as you used it
00:56:42 **Participant**
00:56:44 Mm hmm.
Researcher
00:56:49 maybe even before the study, if you've used Copilot before,
00:56:53 did you change the way you interacted with Copilot based on things you noticed
00:56:55 about its responses?
00:56:59 So either things you didn't like about it that you had to work around or things
00:57:06 that you did like that you tried to reinforce?
Participant
00:57:10 Yeah. So.
00:57:12 I didn't use Copilot before.
00:57:16 Yeah, I did the inline suggestions but nothing
00:57:19 else. I did not that.
00:57:23 Just the general sense that
00:57:26 it's,
00:57:33 the models are becoming smarter, but it's like over time over months.
00:57:35 So yeah, and then I start using it differently
00:57:40 because I know that I can ask more complex questions or larger things like
00:57:42 that.
00:57:42 That you are probably [unintelligible].
Researcher
00:57:47 So so how would your questions change then?
Participant
00:57:51 And.
00:57:55 Yeah. I would just give a more vaguely defined
00:57:57 task.
00:58:00 So, no for example,
00:58:00 what I did here is,

00:58:03 at some time I said
00:58:04 I want you to store this to local state and do this and this and gets me much more
00:58:10 detailed instructions and now I can also say hey I want to have this effect and
00:58:16 you fix it and then it probably will figure it out itself.
00:58:22 Now I have the feeling that, it's just a feeling, that with the the older models
00:58:27 it would just do very weird stuff, make very weird suggestions.
00:58:35 Yeah.
00:58:35 **Researcher**
So and yeah, what? What makes you?
00:58:40 What would make you want to not give this context?
00:58:44 If it's possible not to give it.
00:58:49 **Participant**
Yeah. If I don't give the context, it's just less work.
00:58:53 But you have less control.
00:58:56 So you have to have some trust that it's
00:59:03 finding the correct solution.
00:59:05 **Researcher**
All right.
00:59:07 And would you say you have this trust now?
00:59:07 **Participant**
It's regarding.
00:59:10 And.
00:59:15 Yeah, more, but more on the in the sense of
00:59:20 getting to the end result in an efficient way.
00:59:25 I'm still gonna check it first, verify it so I'm I'm still checking
00:59:30 it but
00:59:32 with the smaller models, the older models
00:59:37 I wouldn't do it because I know that it's entering a path with no ending.
00:59:43 Right now I have more faith in the quality so.
00:59:49 It's faster.
00:59:50 then.
00:59:52 **Researcher**
Yep, that makes a lot of sense.
00:59:54 **Participant**
Yep.
00:59:55 But this is all about like the the the older models like 4o against,
01:00:00 4, against 3. 5, against 3, stuff like that.
01:00:03 So nothing with
01:00:06 memory or
01:00:08 stuff like that.
01:00:10 Because I I I use Claude and Claude doesn't have any memory.
01:00:15 ChatGPT does, but it's very limited.
01:00:20 **Researcher**
So this lack of memory, how do you notice this?
01:00:23 **Participant**
Mm hmm.
01:00:27 Not. I don't notice it myself.
01:00:30 I,
01:00:30 I don't care about it, like.
01:00:37 Yeah. No, no experience with it.
01:00:40 **Researcher**
Fair enough.
01:00:41 Fair enough.
01:00:42 **Participant**
Yes.
01:00:42 **Researcher**

Alright, those are my questions.