

Participant P06

00:04:25 **Participant**
Yeah, I don't know how to.

00:04:27 I don't really use the online.

00:04:30 Editor so I don't know how if running is even is running even a possibility.

00:04:33 > **Viewing file** (package.json)

00:04:38 **Participant**
Is this, I would just do this.

00:04:46 OK.

00:04:59 Is this running on my machine or in like a virtual machine.

00:05:03 **Researcher**
So this is on a virtual machine.

00:05:05 **Participant**
Oh, OK.

00:05:09 OK.

00:05:12 So I guess with that I can just start with the first task or?

00:05:15 **Researcher**
Yes, feel free.

00:05:17 **Participant**
Change the color to blue, OK.

00:05:28 Uh, so first thing I'm doing is I'm looking

00:05:31 at which element.

00:05:32 Needs to be changed so that will be this span, and I know that it has this class,

00:05:37 so I assume that if I look for this class I can find the style related to.

00:05:45 This this object, say todo list empty message.

00:05:53 > **Viewing file** (src/todo/app.css) through file search

00:05:54 **Participant**
There it is.

00:05:55 OK. And I need to change it to this color.

00:06:02 > **Editing file** (src/todo/app.css)
- 00:06:11

00:06:05 **Participant**
I know what's happening.

00:06:06 **Researcher**
Sorry if you could allow the this pop up, they can copy and paste.

00:06:09 **Participant**
Oh, I see. I see.

00:06:13 Right. OK.

00:06:15 I press save and now.

00:06:18 Do I need to, it says that it did something, but I guess I need to reload this. Yeah,

00:06:23 OK.

00:06:26 Refreshed application. OK, well, I think that that's complete.

00:06:32 Let's see, task B. Change your placeholder, what needs to be done.

00:06:37 So that's this part.

00:06:38 This is called.

00:06:42 ID todo input, so I'll look for that again.

00:06:48 > **Viewing file** (src/todo/components/input.jsx) through file search

00:06:50 **Participant**
And here it says the placeholder which is

00:06:55 an argument given to this. I'll look for

00:06:59 where that is assigned, it's assigned in wherever this input

00:07:03 thing is used.

00:07:06 I'm used to click control, clicking and then I can see where it is

00:07:09
00:07:12
00:07:17
00:07:20
00:07:28
00:07:29
00:07:33
00:07:35
00:07:36

00:07:37
00:07:40
00:07:46
- 00:07:52

00:08:05

00:08:07
00:08:12
00:08:15
00:08:20
00:08:22
00:08:26
00:08:31
00:08:37
00:08:38
00:08:42
00:08:44

00:08:59
00:09:03
00:09:21
00:09:22

used. I don't know.

Find the implementations.

OK.

I guess that would be a good, good possibility to ask where this.

Or no, I could just.

I could just search for the what needs to be done part.

That's actually easier for me.

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

Participant

There we go.

What needs to be done and that needs to be, change

the text to enter a task here.

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

Participant

Enter a task here.

So I would say that that's also done.

And then task C.

The todos are displayed in the order they were added. Modify so that the todos are sorted alphabetically instead.

Add the following todos in this order.

The quick, does it save the todos if I?

No, OK, it does not save them.

OK.

So this would be, let's take a look at the actual code.

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

Participant

Visible todos seems to be what shows up, so I would.

I guess if I was, yeah, I guess you could just say.

Like this, is it at workspace or at workspaces?

I'm mainly doing this because.

Chat (window: 0)

> Agent

github.copilot.workspace

> Context

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

> Message

@workspace Where can I sort the todos alphabetically

> Response

To sort the todos alphabetically, you can modify the [visibleTodos](#) ([src/todo/components/main.jsx](#)) computation in the [useMemo](#) ([node_modules/@types/react/index.d.ts](#)) hook in [src/todo/components/main.jsx](#). You can add a sorting step after filtering the todos based on the route.

Here's how you can update the [useMemo](#) ([node_modules/@types/react/index.d.ts](#)) hook:

```
File: src/todo/components/main.jsx
const visibleTodos = useMemo(
  () =>
    todos
      .filter((todo) => {
        if (route === "/active")
          return !todo.completed;

        if (route === "/completed")
          return todo.completed;
```

```

        return todo;
    })
    .sort((a, b) => a.title.localeCompare(b.title)), // Sort
    alphabetically by title
    [todos, route]
);

```

This will ensure that the [visibleTodos \(src/todo/components/main.jsx\)](#) array is sorted alphabetically by the [title \(src/todo/components/header.jsx\)](#) property before being rendered.

00:09:24
- 00:09:26

> Response rendering

00:09:24

Participant

There's, you know, you can probably sort them over here, but this route thing is confusing me a little bit.

00:09:27

Makes me think that there that this is not doing what I think it's doing.

00:09:30

So I think I'll just take the lazy approach.

00:09:38

I could probably look for it, but this is really easier.

00:09:40

OK.

00:09:43

Well, it's telling me exactly that I can do it over here.

00:09:44

So.

00:09:46

Comes after here.

00:09:48

Let me get this part, this line.

00:09:50

Sure.

00:09:54

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

00:09:59

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

00:10:01

00:10:02

Participant

And uh, let's see.

00:10:05

The quick 123 brown fox.

00:10:08

123 brown fox quick the.

00:10:19

Yeah, I think that's that's done.

00:10:20

Researcher

Great. Thanks a lot.

00:10:22

I'll send you the actual task right now.

00:10:25

Thanks for thinking aloud the way you're doing, by the way, you're doing great.

00:10:28

Participant

00:10:32

00:10:35 OK.
00:10:38 I I kinda forgot that that I thought it was done.
00:10:38 **Researcher**
Yeah.
00:10:40 **Participant**
OK.
00:10:44 OK.
00:10:48 So all todos disappear when you refresh the page.
00:10:51 Modify the application so todos remain after refresh.
00:10:53 So my first thought would be to use local storage.
00:11:01 An attempt to implement this functionality was made, it's not working, OK.
00:11:07 So where would that be made?
00:11:13 Also don't know if this error is something I should be worried about,
00:11:17 but maybe that's for later task.
00:11:26 Let's see.
00:11:31 Where is the input.
00:11:33 I had the input earlier.
00:11:39 > **Viewing file** (src/todo/components/header.jsx) through file search
00:11:40 **Participant**
It's in the header OK.
00:11:42 On submit add item.
00:11:49 Where the header is used.
00:12:02 It's not really working.
00:12:05 When I was.
00:12:07 > **Viewing file** (src/todo/components/input.jsx)
00:12:07 **Participant**
It's not bringing me to where the header is being used,
00:12:09 > **Viewing file** (src/todo/components/header.jsx)
00:12:10 > **Viewing file** (src/todo/components/input.jsx)
00:12:10 **Participant**
but I guess I can just search for this.
00:12:16 > **Viewing file** (src/todo/app.jsx) through file search
00:12:19 **Participant**
Dispatch dispatch.
00:12:22 I'm looking for where the the the implementation was used,
00:12:25 but I can't find it yet.
00:12:27 I don't really know.
00:12:28 Use reducer that well.
00:12:34 OK, I guess.
00:12:37 This is seems like a good.
00:12:42 This is a good how you say opportunity to use the chat again.
00:12:58 It should probably tell me where.
00:12:59 **Chat** (window: 0)
> **Agent**
github.copilot.workspace
> **Context**
workspace, viewport (src/todo/app.jsx)
> **Message**
@workspace How can I persist the to-dos across page refreshes?
> **Response**
To persist the to-dos across page refreshes, you can use the browser's [localStorage](#) ([node_modules/@types/node/globals.d.ts](#)). Your [todoReducer](#) ([src/todo/reducer.js](#)) in [src/todo/reducer.js](#) already includes logic to save and load the state using [localStorage](#) ([node_modules/@types/node/globals.d.ts](#)). However, the [App](#) ([src/todo/app.jsx](#)) component in [src/todo/app.jsx](#) needs to initialize the [todos](#) ([src/todo/app.jsx](#)) state with the persisted data.

Here's how you can modify the [App\(src/todo/app.jsx\)](#) component to load the initial state from [localStorage\(node_modules/@types/node/globals.d.ts\)](#):

```
File: 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, loadState } from "../reducer"; // Import loadState

import "../app.css";

export function App() {
  const [todos, dispatch] = useReducer(todoReducer, loadState()); //
  Initialize with persisted state

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

This ensures that the [todos\(src/todo/app.jsx\)](#) state is initialized with the data saved in [localStorage\(node_modules/@types/node/globals.d.ts\)](#). The [todoReducer\(src/todo/reducer.js\)](#) will handle saving the state to [localStorage\(node_modules/@types/node/globals.d.ts\)](#) whenever the state changes.

00:13:01

Participant

The code is being, can be found.

00:13:02

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

00:13:05
- 00:13:09

> **Response rendering**

00:13:08

Participant

In reducer dot JS. Uh. Huh, there we go.

00:13:17

And then we have.

00:13:24

Save state. There we go. Local storage, set item todos.

00:13:30

But there's no load state.

00:13:31

Ah, because there is a typo.

00:13:32

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

00:13:34

Participant

And maybe that should fix it.

00:13:36

Maybe not.

00:13:43

Does not yet OK.

00:13:46

Maybe load state is never.

00:13:49

See.

00:13:53

I should probably read what the actual chat says, I was.

00:13:55

I just used it to get to the file.

00:14:05

Uh, huh. That makes sense.

00:14:09

Is this initial state used anywhere then?

00:14:13

Or not.

00:14:17 No, it's just used here, which does nothing.

00:14:19 > **Viewing file** (src/todo/components/input.jsx)

00:14:19 **Participant**

I suppose you can go to where we just were.

00:14:20 > **Viewing file** (src/todo/app.css)

00:14:20 > **Viewing file** (src/todo/reducer.js)

00:14:21 > **Viewing file** (src/todo/components/header.jsx)

00:14:22 > **Viewing file** (src/todo/components/main.jsx)

00:14:26 > **Viewing file** (src/todo/components/header.jsx)

00:14:26 > **Viewing file** (src/todo/components/main.jsx)

00:14:26 > **Viewing file** (src/todo/reducer.js)

00:14:26 > **Viewing file** (src/todo/components/input.jsx)

00:14:26 **Participant**

Which was where, which file is that.

00:14:27 > **Viewing file** (src/todo/app.css)

00:14:29 > **Viewing file** (src/todo/components/input.jsx)

00:14:29 **Participant**

Not this one.

00:14:32 > **Viewing file** (src/todo/components/header.jsx)

00:14:33 **Participant**

Not this one. In the header. Right.

00:14:36 Oh no, it wasn't the header. Add item that's the one.

00:14:43 > **Viewing file** (src/todo/components/main.jsx)

00:14:44 > **Viewing file** (src/todo/components/header.jsx)

00:14:46 > **Viewing file** (src/todo/components/main.jsx)

00:14:50 **Participant**

I just had to file open where the the this code was. Oh, it's app.

00:14:55 > **Viewing file** (src/todo/app.jsx)

00:14:57 > **Editing file** (src/todo/app.jsx)

- 00:15:01

00:14:58 **Participant**

And this is initial states.

00:15:03 See.

00:15:06 Yes, that works.

00:15:07 So then if I add another one.

00:15:09 It stays.

00:15:10 If I check it's also stays.

00:15:14 Let's see. Verification. Add some todos, refresh the page

00:15:17 confirm the todos are still there.

00:15:19 Well, we did that. Mark

00:15:20 some as completed, refresh the page and

00:15:21 confirm both the todos and their states remain unchanged.

00:15:23 I did that. Edit some todos and confirm the changes are still there.

00:15:27 Let's see.

00:15:28 How do I edit a todo?

00:15:34 Refresh, changes are still there. Delete some todos.

00:15:40 Yeah, that all works.

00:15:43 Unless I missed something.

00:15:47 OK, task B save drafts. If a user refreshes the page, while typing

00:15:51 a new todo the text in the input field disappears,

00:15:54 modify the application so that the input automatic field automatically saves

00:15:58 drafts every few seconds and restores them when the page reloads. Interesting.

00:16:07 I mean, I'm interested about the every few

00:16:10 seconds part because the way I would do it would just be to use like a debounce

00:16:13
00:16:17
00:16:18
00:16:21
00:16:25
00:16:28
00:16:30
00:16:35
00:16:40
00:16:42
00:16:45
00:16:47
00:16:53
00:16:55
00:17:04
00:17:20

on the input field but.

Umm.

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

Participant

I feel like normally if I were to do a task like this, I would.

Put the functionality you know the way the same way that's in the header.

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

Participant

You have the on submit at item, but I suppose and then you add take this

as like a parameter using this dispatch, but for the sake of this you can probably just hard code it into.

The component.

Umm.

Yeah, or can be lazy again.

I mean, yeah.

This is the easiest way.

The contents of the input.

Chat (window: 0)

> **Agent**

github.copilot.workspace

> **Context**

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

> **Message**

@workspace I would like to persist the contents of the input field across page refreshes

> **Response**

To persist the contents of the input field across page refreshes, you can use [localStorage](#) ([node_modules/@types/node/globals.d.ts](#)) to save the input value whenever it changes and load it when the component mounts.

Here's how you can modify the [Header](#) ([src/todo/components/header.jsx](#)) component in [src/todo/components/header.jsx](#):

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

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

export function Header({ dispatch }) {
  const [inputValue, setInputValue] = useState("");

  // Load the input value from localStorage when the component mounts
  useEffect(() => {
    const savedValue = localStorage.getItem("todoInput");
    if (savedValue) {
      setInputValue(savedValue);
    }
  }, []);

  // Save the input value to localStorage whenever it changes
  const handleInputChange = useCallback((e) => {
    const value = e.target.value;
    setInputValue(value);
    localStorage.setItem("todoInput", value);
  }, []);

  const addItem = useCallback(
    (title) => {
      dispatch({ type: ADD_ITEM, payload: { title } });
      setInputValue(""); // Clear the input field after adding
```

```

the item
    localStorage.removeItem("todoInput"); // Clear the saved
value
    },
    [dispatch]
  );

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

```

Explanation:

1. **State Initialization:** The `inputValue` state is initialized with an empty string.
2. **Load from [localStorage \(node_modules/@types/node/globals.d.ts\)](#):** On component mount, the saved input value is loaded from [localStorage \(node_modules/@types/node/globals.d.ts\)](#) and set to `inputValue`.
3. **Save to [localStorage \(node_modules/@types/node/globals.d.ts\)](#):** The `handleInputChange` function updates `inputValue` and saves it to [localStorage \(node_modules/@types/node/globals.d.ts\)](#) whenever the input changes.
4. **Clear on Submit:** When a new item is added, the input field is cleared, and the saved value in [localStorage \(node_modules/@types/node/globals.d.ts\)](#) is removed.

00:17:20

Participant

See what it says.

00:17:22
- 00:17:26

> **Response rendering**

00:17:26

Participant

Let's see. What does it say?

00:17:31

On Blur, I would not do it.

00:17:33

On an Blur I would do it on an on change or an on input event but.

00:17:42

Also, I don't think necessary that the whole

00:17:44

use callback thing is necessary.

00:17:45

I mean, I I don't program in React that often,

00:17:48

but I would.

00:17:50

I don't know if it's necessary to always use those wrapper functions,

00:17:54

but it's probably a good practice and it's already used here.

00:17:57
- 00:17:18

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

00:18:02

Participant

Uh, this is something that I can probably

00:18:04

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

00:18:04

Participant

just copy paste.

00:18:10

And.

00:18:14

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

00:18:14

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

00:18:19
- 00:18:29

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

00:18:21

Participant

Let's see.

00:18:26

And on.

00:18:30

Oh, the on blur is not defined on this

00:18:31

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

00:18:32

Participant

input element.

00:18:35

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

00:18:35

Participant

I actually don't know how on blur works. If it works when you only work if you

00:18:36

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

00:18:36

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

00:18:39

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

00:18:40

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

00:18:40

Participant

click away from it, I think.

00:18:41

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

00:18:41

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

00:18:43

Participant

Hmm.

00:18:49

I I want.

00:18:49

I I think I need to just introduce the on change event or the on input event on

00:18:55
00:19:00
00:19:04
00:19:08
00:19:10
00:19:21
00:19:26
00:19:28
00:19:31
00:19:32

the input element.

But I'm just too lazy type everything out.

I will say normally when I use like Copilot I will I will I just have like

inline completions and I press tab.

But so this this is different for me asking it explicitly.

To use on blur, I think on input or.

On change.

Would be better.

And uh, I'll see if it can take the hint.

Chat (window: 0)

> Agent

github.copilot.workspace

> Context

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

> Message

@workspace I don't want to use onBlur, I think onInput or onChange would be better

> Response

You can replace the [onBlur \(src/todo/components/input.jsx\)](#) handler with onInput or [onChange \(node_modules/@types/react/index.d.ts\)](#) to handle input changes in real-time. Here's how you can modify the [Input \(src/todo/components/input.jsx\)](#) component in [src/todo/components/input.jsx](#):

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

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

export function Input({ onSubmit, placeholder, label, defaultValue,
onInput }) {
  const handleKeyDown = useCallback(
    (e) => {
      if (e.key === "Enter") {
        const value = e.target.value.trim();

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

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

  return (
    <div className="input-container">
      <input
        className="new-todo"
        id="todo-input"
        type="text"
        data-testid="text-input"
        autoFocus
        placeholder={placeholder}
        defaultValue={defaultValue}
        onInput={onInput} // Use onInput for real-time updates
        onKeyDown={handleKeyDown}
      />
      <label className="visually-hidden" htmlFor="todo-input">
        {label}
      </label>
    </div>
  );
}
```

```

    );
  }

```

Explanation:

1. **Replaced `onBlur` ([src/todo/components/input.jsx](#)) with `onInput`:** The `onInput` event is triggered whenever the value of the input changes, making it suitable for real-time updates.
2. **Removed `handleBlur` ([src/todo/components/input.jsx](#)):** Since `onBlur` ([src/todo/components/input.jsx](#)) is no longer used, the `handleBlur` ([src/todo/components/input.jsx](#)) function is removed.

You can now pass an `onInput` handler to the `Input` ([src/todo/components/input.jsx](#)) component to handle input changes in real-time. For example, in the `Header` ([src/todo/components/header.jsx](#)) component, you can update the `handleInputChange` ([src/todo/components/header.jsx](#)) function to use `onInput`:

```

<Input
  onSubmit={addItem}
  defaultValue={inputValue}
  label="New Todo Input"
  placeholder="Enter a task here"
  onInput={handleInputChange} // Pass the real-time input handler
/>

```

00:19:34
- 00:19:39

> Response rendering

00:19:39

Participant

Let's see.

00:19:42

On input on input.

00:19:45

Where is this coming from?

00:19:48
- 00:19:51

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

00:19:54

Participant

I'm also not used to not having TypeScript.

00:19:57
- 00:20:03

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

00:20:04

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

00:20:05

Participant

OK.

00:20:06

Then we can go back here.

00:20:07

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

00:20:08

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

00:20:11
- 00:20:13

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

00:20:16

Participant

I don't suppose you have.

00:20:20

Umm.

00:20:22

> **Viewing file** (package.json)

00:20:23

Participant

Lodash or anything in this projects?

00:20:27

> **Viewing file** (package-lock.json)

00:20:28

Participant

No. OK, so.

00:20:29

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

00:20:29

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

00:20:30

> **Viewing file** (package-lock.json)

00:20:30

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

00:20:35

Participant

Give me a debounce function similar to lodash.

00:20:43

Chat (window: 0)

> Agent

github.copilot.default

> Context

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

> Message

give me a debounce function similar to lodash

> Response

Here's a simple debounce function similar to Lodash's debounce:

```
function debounce(func, wait) {  
  let timeout;  
  
  return function (...args) {  
    const context = this;
```

```

        clearTimeout(timeout);
        timeout = setTimeout(() => func.apply(context, args), wait);
    };
}

```

How it works:

1. **func**: The function you want to debounce.
2. **wait**: The delay in milliseconds before the function is executed.
3. **timeout**: A timer that resets every time the function is called within the delay period.

Example Usage:

You can use this debounce function in your [Header \(src/todo/components/header.jsx\)](#) component to debounce the [handleInputChange \(src/todo/components/header.jsx\)](#) function:

```

import { useCallback, useState } from "react";
import { Input } from "../input";
import { ADD_ITEM } from "../constants";

function debounce(func, wait) {
    let timeout;
    return function (...args) {
        const context = this;
        clearTimeout(timeout);
        timeout = setTimeout(() => func.apply(context, args), wait);
    };
}

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

    // Debounced input change handler
    const handleInputChange = useCallback(
        debounce((e) => {
            const value = e.target.value;
            setInputValue(value);
            localStorage.setItem("todoInput", value);
        }, 300), // 300ms debounce delay
        []
    );

    return (
        <header className="header" data-testid="header">
            <h1>todos</h1>
            <Input
                onSubmit={addItem}
                defaultValue={inputValue}
                label="New Todo Input"
                placeholder="Enter a task here"
                onInput={handleInputChange} // Use the debounced
handler
            />
        </header>
    );
}

```

Explanation:

- The debounce function ensures that [handleInputChange \(src/todo/components/header.jsx\)](#) is only executed after the user stops typing for 300ms.
- This reduces the frequency of updates to [localStorage \(node_modules/@types/node/globals.d.ts\)](#) and improves performance.

00:20:44
- 00:20:49

> Response rendering

00:20:51

Participant

Because I am used to handling input with a debounce.

00:20:53

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

00:20:53

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

00:20:57

Participant

So and then this can be.

00:21:01

Uh, now.

00:21:09

Chat (window: 0)

> Agent

github.copilot.default

> Context

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

> Message

use debounce to handle the input change

> Response

Here's how you can use the [debounce \(src/todo/components/header.jsx\)](#) function to handle the input change in the [Header \(src/todo/components/header.jsx\)](#) component:

```
File: src/todo/components/header.jsx
const handleInputChange = useCallback(
  debounce((e) => {
    const value = e.target.value;
    setInputValue(value);
    localStorage.setItem("todoInput", value);
  }, 300), // 300ms debounce delay
  []
);
```

Explanation:

1. **Debounce Function:** The [debounce \(src/todo/components/header.jsx\)](#) function ensures that the [handleInputChange \(src/todo/components/header.jsx\)](#) logic is executed only after the user stops typing for 300ms.

2. **Performance Improvement:** This reduces the frequency of updates to [localStorage\(node_modules/@types/node/globals.d.ts\)](#) and improves performance.

Updated Usage in the Component:

Make sure to pass [handleInputChange\(src/todo/components/header.jsx\)](#) to the [onInput\(src/todo/components/header.jsx\)](#) prop of the [Input\(src/todo/components/input.jsx\)](#) component:

```
<Input
  onSubmit={addItem}
  defaultValue={inputValue}
  label="New Todo Input"
  placeholder="Enter a task here"
  onInput={handleInputChange} // Use the debounced handler
/>
```

00:21:11
- 00:21:14

> Response rendering

00:21:12

Participant

Maybe I already did that.

00:21:18

Yeah, I think my style is very much.

00:21:20

I take what I need and don't read anything else.

00:21:25

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

00:21:26

Participant

And let's verify if this makes sense.

00:21:33

And now I need to.

00:21:43

Uh.

00:21:46

Chat (window: 0)

> Agent

github.copilot.default

> Context

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

> Message

load the todoInput from localStorage on page load

> Response

To load the `todoInput` value from [localStorage](#) ([node_modules/@types/node/globals.d.ts](#)) on page load, you can use the `useEffect` hook to initialize the `inputValue` ([src/todo/components/header.jsx](#)) state when the component mounts. Here's the updated code:

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

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

function debounce(func, wait) {
  let timeout;

  return function (...args) {
    const context = this;

    clearTimeout(timeout);
    timeout = setTimeout(() => func.apply(context, args), wait);
  };
}

export function Header({ dispatch }) {
  const [inputValue, setInputValue] = useState("");

  // Load the input value from localStorage when the component mounts
  useEffect(() => {
    const savedValue = localStorage.getItem("todoInput");
    if (savedValue) {
      setInputValue(savedValue);
    }
  }, []);

  const addItem = useCallback((title) => {
    dispatch({ type: ADD_ITEM, payload: { title } });
    setInputValue(""); // Clear the input field after adding the
item
    localStorage.removeItem("todoInput"); // Clear the saved value
  }, [dispatch]);

  // Save the input value to localStorage whenever it changes
  const handleInputChange = useCallback(
    debounce((e) => {
      const value = e.target.value;
      setInputValue(value);
      localStorage.setItem("todoInput", value);
    }, 300), // 300ms debounce delay
    []
  );

  return (
    <header className="header" data-testid="header">
      <h1>todos</h1>
      <Input
        onSubmit={addItem}
        defaultValue={inputValue}
        label="New Todo Input"
        placeholder="Enter a task here"
        onInput={handleInputChange} // Use the debounced
handler
      />
    </header>
  );
}
```

Changes Made:

1. `useEffect` for Initialization:

- Added a `useEffect` hook to load the `todoInput` value from [localStorage \(node_modules/@types/node/globals.d.ts\)](#) when the component mounts.
- If a value exists in [localStorage \(node_modules/@types/node/globals.d.ts\)](#), it sets the [inputValue \(src/todo/components/header.jsx\)](#) state.

2. Clear [localStorage \(node_modules/@types/node/globals.d.ts\)](#) on Submit:

- When a new item is added, the `todoInput` value is removed from [localStorage \(node_modules/@types/node/globals.d.ts\)](#) to reset the saved state.

This ensures that the input field is pre-filled with the saved value from [localStorage \(node_modules/@types/node/globals.d.ts\)](#) on page load.

00:21:46

Participant

Assign this.

00:21:47
- 00:21:52

> Response rendering

00:21:53

Participant

OK.

00:21:54

I don't need all all the extra stuff, but.

00:22:00

Oh, it's already did that, OK.

00:22:02

Never mind.

00:22:04

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

00:22:06

Participant

See, get input, if saved value.

00:22:07

Yeah, that makes sense.

00:22:08

Let's see if it works.

00:22:16

Yeah, I broke something, I guess.

00:22:22
- 00:22:30

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

00:22:32

Participant

It's still not working.

00:22:41

I must have made a mistake somewhere I don't see.

00:22:47

Oh no.

00:22:49

I don't know what I just did.

00:22:51

I tried to.

00:22:52

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

00:22:52

Participant

00:22:55 Control Y everything.
00:22:56 Where are the?
00:22:56 Where's the?
00:22:58 Compilation coming from.
00:22:59 I don't see anything in the in the Visual Studio code that signifies.
00:23:02 > **Viewing file** (src/todo/components/main.jsx)
00:23:05 > **Viewing file** (src/todo/app.jsx)
00:23:05 **Participant**
There's an error, maybe. Maybe it knows.
00:23:05 > **Viewing file** (src/todo/components/input.jsx)
00:23:19 **Chat** (window: 0)
> **Agent**
github.copilot.workspace
> **Context**
workspace, viewport (src/todo/components/input.jsx)
> **Message**
@workspace the code is not compiling, can you find where the problem is
> **Response**
The issue lies in the [Header \(src/todo/components/header.jsx\)](#) component in [src/todo/components/header.jsx](#). Specifically, the [onInput \(src/todo/components/header.jsx\)](#) prop passed to the [Input \(src/todo/components/input.jsx\)](#) component is left empty:

```
<Input onSubmit={addItem} defaultValue={inputValue} label="New Todo Input" placeholder="Enter a task here" onInput={} />
```


This is invalid syntax because the [onInput \(src/todo/components/header.jsx\)](#) prop is not assigned a value. To fix this, you need to pass the [handleInputChange \(src/todo/components/header.jsx\)](#) function (or any valid handler) to the [onInput \(src/todo/components/header.jsx\)](#) prop.

Here's the corrected code for the [Header \(src/todo/components/header.jsx\)](#) component:


```
File: src/todo/components/header.jsx  
<Input  
  onSubmit={addItem}  
  defaultValue={inputValue}  
  label="New Todo Input"  
  placeholder="Enter a task here"  
  onInput={handleInputChange} // Pass the input handler here  
>
```


Explanation:

- The [onInput \(src/todo/components/header.jsx\)](#) prop is now assigned the [handleInputChange \(src/todo/components/header.jsx\)](#) function, which ensures the input changes are handled correctly.
- This resolves the syntax error and allows the code to compile.

> **Response rendering**
> **Viewing file** (src/todo/components/header.jsx) through Copilot link
00:23:22 **Participant**
- 00:23:24 That makes sense.
00:23:24 > **Editing file** (src/todo/components/header.jsx)
00:23:26 **Participant**
OK.
00:23:27 Still not really looking the way it should.
- 00:23:28
00:23:33
00:23:35

00:23:41
00:23:42
00:23:42
00:23:43
00:23:47

00:23:47
00:23:49
00:23:52
00:23:56
00:24:00
- 00:24:03

00:24:01

00:24:12
00:24:14
00:24:17
00:24:18
00:24:25
00:24:27
00:24:32
00:24:39
00:24:48

Oh no, I really broke it.

> **Viewing file** (package-lock.json)

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

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

Participant

OK.

That makes sense as well.

How do I import it?

Which which key bind is that?

Oh.

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

Participant

Use callback. Use state, use effect.

That works.

I guess the only thing OK does that also work?

No, that does not work.

So once you add an item, it should actually clear it from the.

I I think well, it doesn't say in the task,

but I think if if we're gonna do this, we're gonna should do it the right way.

So it should clear the input when you add an item.

Let's see.

Chat (window: 0)

> **Agent**

github.copilot.default

> **Context**

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

> **Message**

clear the input when you add an item

> **Response**

To clear the input field after adding an item, you can update the [addItem \(src/todo/components/header.jsx\)](#) function to reset the [inputValue \(src/todo/components/header.jsx\)](#) state and remove the saved value from [localStorage \(node_modules/@types/node/globals.d.ts\)](#). Here's the updated code:

```
File: src/todo/components/header.jsx
const addItem = useCallback((title) => {
  dispatch({ type: ADD_ITEM, payload: { title } });
  setInputValue(""); // Clear the input field
  localStorage.removeItem("todoInput"); // Remove the saved value
  from localStorage
}, [dispatch]);
```

Updated Component:

Here's how the [Header \(src/todo/components/header.jsx\)](#) component looks with the updated [addItem \(src/todo/components/header.jsx\)](#) function:

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

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

function debounce(func, wait) {
  let timeout;

  return function (...args) {
    const context = this;
```

```

        clearTimeout(timeout);
        timeout = setTimeout(() => func.apply(context, args), wait);
    };
}

export function Header({ dispatch }) {
    const [inputValue, setInputValue] = useState("");

    const addItem = useCallback((title) => {
        dispatch({ type: ADD_ITEM, payload: { title } });
        setInputValue(""); // Clear the input field
        localStorage.removeItem("todoInput"); // Remove the saved value
from localStorage
    }, [dispatch]);

    // Load the input value from localStorage when the component mounts
    useEffect(() => {
        const savedValue = localStorage.getItem("todoInput");
        if (savedValue) {
            setInputValue(savedValue);
        }
    }, []);

    // Save the input value to localStorage whenever it changes
    const handleInputChange = useCallback(
        debounce((e) => {
            const value = e.target.value;
            setInputValue(value);
            localStorage.setItem("todoInput", value);
        }, 300), // 300ms debounce delay
        []
    );

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

```

Explanation:

1. **Clear Input Field:** The `setInputValue("")` ([src/todo/components/header.jsx](#)) call resets the input field after an item is added.
2. **Remove from localStorage (node_modules/@types/node/globals.d.ts):** The `localStorage.removeItem("todoInput")` ([node_modules/@types/node/globals.d.ts](#)) call ensures that the saved input value is cleared after submission.

00:24:51
- 00:24:59

> Response rendering

00:24:59

Participant

How does this work, apply?

00:25:02

Oh, you can just click apply. How does that work?

00:25:03
- 00:25:06

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

00:25:06

Participant

Oh wow.

00:25:09

And now I guess I just keep this change, OK? I've never seen this before, but this is pretty neat.

00:25:13

00:25:15

I have used the chat in IntelliJ but I don't think it has this feature.

00:25:24

And then.

00:25:33

Wait, oh OK.

00:25:39

Well, I think I think it works.

00:25:41

Type a new todo without submitting.

00:25:47

Type a new todo, submit it, refresh the page.

00:25:50

Yeah. OK.

00:25:51

So that's task B done.

00:25:55

Task C. It's a lot of text.

00:25:59

You know what?

00:26:01

I'm gonna be really bad.

00:26:06

I'm not reading all this.

00:26:09

Chat (window: 0)

> Agent

github.copilot.workspace

> Context

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

> Message

@workspace 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 ability to display 1, 3, or 5 TODOs per page, we need to modify the [Main\(src/todo/components/main.jsx\)](#) component in [src/todo/components/main.jsx](#) and ensure the [PaginationControls\(src/todo/components/paginationControls.jsx\)](#) component in [src/todo/components/paginationControls.jsx](#) works correctly.

Here's the updated implementation:

Update [Main\(src/todo/components/main.jsx\)](#) Component

We will add a dropdown to select the number of items per page and fix the pagination logic.

```
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";
import { PaginationControls } from "../paginationControls";

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

  const visibleTodos = useMemo(() => {
    const filteredTodos = todos.filter((todo) => {
      if (route === "/active") return !todo.completed;
      if (route === "/completed") return todo.completed;
      return true;
    });

    return filteredTodos.sort((a, b) =>
a.title.localeCompare(b.title));
  }, [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 handleItemsPerPageChange = (e) => {
    setItemsPerPage(Number(e.target.value));
    setCurrentPage(1); // Reset to the first page when items per
page changes
  };
}
```



```

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>
        <div className="items-per-page-container">
          <label htmlFor="items-per-page">Items per page:
        </label>
          <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>
        <ul className={classnames("todo-list")} data-
          testid="todo-list">
          {paginatedTodos.map((todo, index) => (
            <Item todo={todo} key={todo.id} dispatch=
              {dispatch} index={index} />
          ))}
        </ul>
        <PaginationControls
          currentPage={currentPage}
          setCurrentPage={setCurrentPage}
          itemsPerPage={itemsPerPage}
          totalItems={visibleTodos.length}
        />
      </>
    ) : (
      <div className="todo-list-empty-container">
        <span className="todo-list-empty-message">No TODOs
          here yet</span>
      </div>
    )}
  </main>
);
}

```

Fix [PaginationControls](#)

[\(src/todo/components/paginationControls.jsx\)](#) Component

We will fix the navigation logic to ensure the "Next" and "Previous" buttons work correctly.

```

File: src/todo/components/paginationControls.jsx
import React from "react";
import classNames from "classnames";

export function PaginationControls({ currentPage, setCurrentPage,
itemsPerPage, totalItems }) {
  const totalPages = Math.ceil(totalItems / itemsPerPage);

  const handlePageChange = (direction) => {
    setCurrentPage((prevPage) => {
      if (direction === "next" && prevPage < totalPages) {
        return prevPage + 1;
      }
      if (direction === "prev" && prevPage > 1) {
        return prevPage - 1;
      }
      return prevPage;
    });
  };

  return (
    <div className="pagination-controls">
      <button
        className={classNames("pagination-button", { inactive:
currentPage === 1 })}
        onClick={() => handlePageChange("prev")}
        disabled={currentPage === 1}
      >
        Previous
      </button>
      <span className="current-page-indicator">
        Page {currentPage} of {totalPages}
      </span>
      <button
        className={classNames("pagination-button", { inactive:
currentPage === totalPages })}
        onClick={() => handlePageChange("next")}
        disabled={currentPage === totalPages}
      >
        Next
      </button>
    </div>
  );
}

```

Explanation of Changes

1. **Dropdown for Items Per Page:** Added a dropdown in the [Main](#) ([src/todo/components/main.jsx](#)) component to allow users to select the number of items per page (1, 3, or 5).
 2. **Pagination Logic:** Updated the [PaginationControls](#) ([src/todo/components/paginationControls.jsx](#)) component to correctly handle "Next" and "Previous" button states and navigation.
 3. **Page Reset on Items Per Page Change:** When the user changes the number of items per page, the current page resets to the first page.
 4. **Page Indicator:** Added a page indicator to show the current page and total pages.
-

CSS (Optional)

You can add the following CSS to style the dropdown and page indicator:

```
File: src/todo/app.css
.items-per-page-container {
  margin: 10px 0;
  display: flex;
  align-items: center;
  gap: 10px;
}

.current-page-indicator {
  margin: 0 10px;
  font-weight: bold;
}
```

This implementation ensures that the TODO application supports pagination with the required features.

00:26:11

Participant

Let's see how far I can push this.

00:26:11

- 00:26:25

> Response rendering

00:26:27

Participant

I I did.

00:26:28

I would normally never use it like this, I think.

00:26:32

It seems really fun to just try it.

00:26:33

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

00:26:34

- 00:26:43

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

00:26:38

Participant

I hope I can easily undo this. Actually doesn't work.

00:26:48

I'll just keep everything. Why not and?

00:26:52

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

00:26:54

- 00:26:56

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

00:26:54

Participant

Here as well.

00:27:01

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

00:27:02

Participant

Sure.

00:27:02

- 00:27:03

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

00:27:04

Participant

I completely trust you.

00:27:09

Let's see.

00:27:12

Three items left. OK. Items per page 1, 3, 5, OK.

00:27:17

How do I verify. Add 10 or more todos. OK, let's just uh.

00:27:25

1. Oh no, it's not working anymore. OK, it is just not.

00:27:28

Taking the number one for some reason.

00:27:34

That's very interesting.

00:27:38

But it's not really part of.

00:27:41

The task.

00:27:51

OK, 10 todos. Confirm that the user can select option show 1, 3 or 5 todos

00:27:57

at a time. 1, 3 or 5. OK.

00:28:02

Confirm that the next select different page size option, confirm that only.

00:28:06

Yeah, OK.

00:28:07

I did that. Confirm that the current page is indicated clearly. Page one of ten, page 2.

00:28:12

Yeah, and now if I.

00:28:19

Think that's indicated clearly.

00:28:21

TakClicke the next current yes, navigate to the first page and confirm

00:28:25 it's previous button is gray.
00:28:27 Yep, navigate the last page confirm the next
00:28:30 button is gray.
00:28:33 Yes.
00:28:36 confirm the todo count on each page matches the selected page size.
00:28:39 Well, here it obviously doesn't, but that's because there's ten and you
00:28:44 can't divide it by three.
00:28:46 But if you go to, I have confirmed it on the other ones,
00:28:49 I think.
00:28:54 It's still sorting them alphabetically.
00:28:57 And confirm that the last page contains at least one and at most OK, confirm all todos added
are present.
00:29:01 I think I think that's it.
00:29:03 I don't know why it won't let me make a todo.
00:29:05 That's just a single.
00:29:08 I also don't like what this is supposed to do,
00:29:11 but I I did not change anything there.
00:29:13 I think I think I'm done, unless you say that this really is a bug
00:29:17 that I introduced.
00:29:19 **Researcher**
No, that's a feature.
00:29:21 Not a bug.
00:29:22 Let's say so.
00:29:22 **Participant**
OK.
00:29:24 **Researcher**
Thanks a lot for that.
00:29:25 That's indeed all of the tasks.
00:30:38 **Participant**
I'm gonna. Yeah. This is difficult because I did not
00:30:41 really look at all of the like.
00:30:44 I don't really look at all this text.
00:30:45 I just look at the code.
00:30:48 So I don't know.
00:31:21 I like my answers to be very short and to the point,
00:31:24 and I feel like the chat apps by default don't do that.
00:31:28 But I know that I can just ask for it probably, but I always forget.
00:31:33 But I guess if you don't ask for it then I don't know.
00:33:29 A little bit 'cause. I broke it at some point.
00:34:08 OK.
00:34:10 **Researcher**
Great. So let's start with some questions.
00:34:15 **Participant**
OK.
00:34:17 **Researcher**
So navigating through the codebase, did you manage to find your way?
00:34:27 **Participant**
No. I'm very used that I can hold control and click on things and it will take me to
00:34:30 where they are referenced and things like that. And every time I tried it,
00:34:33 it just didn't happen.
00:34:34 So that was annoying.
00:34:36 **Researcher**
OK.
00:34:37 **Participant**
I mainly used like text search to find something or I I guess I asked the chat
00:34:41 to just fix it for me and it would automatically bring me to where it needed

00:34:45 to be fixed.

00:34:46 So that was probably quite nice, I'd say.

00:34:49 **Researcher**
OK.

00:34:49 So in this situation, Copilot helped you navigate your way.

00:34:55 **Participant**
To an, to an extent.

00:34:56 Yeah. If I had been using, you know, IntelliJ or something like that,

00:35:00 I I would have found my way without the chat, I'd say.

00:35:03 **Researcher**
OK.

00:35:05 So understanding the codebase, do you?

00:35:09 Did you feel like you understood where everything was and how everything worked

00:35:12 together as well?

00:35:14 **Participant**
Uh.

00:35:17 Not really.

00:35:18 I kind of just, I kind of just went for it and changed

00:35:21 what needed to be changed without really looking at the surround the surrounding

00:35:25 context of it.

00:35:27 I kind of just in my head.

00:35:28 I guess I just sort of sped through it without really stopping to really look

00:35:32 around.

00:35:34 **Researcher**
Yeah, that makes sense.

00:35:36 Did using Copilot impact your understanding in any way?

00:35:42 **Participant**
Well, I mean, if I wouldn't have had it, I probably would have needed to take the

00:35:47 time to look at everything.

00:35:48 Yeah. So yes.

00:35:51 **Researcher**
OK. And in a real life situation, would you have taken this time?

00:35:59 **Participant**
That really depends on, you know the the project.

00:36:02 If it's like a hobby project or something, probably not.

00:36:06 If it's, you know, for my job and it's really my job to

00:36:09 really understand the code, then yeah, I would take that time.

00:36:14 **Researcher**
Fair enough.

00:36:16 So so you were talking about taking what you need from Copilot's responses.

00:36:23 **Participant**
Mm hmm.

00:36:25 **Researcher**
Did you manage to locate what aspects of the responses were were useful to you?

00:36:32 **Participant**
The code blocks. I don't really care about everything

00:36:35 around it.

00:36:37 Especially especially at the end when I find that button that could just do

00:36:40 everything automatically.

00:36:42 Yeah, that quickly.

00:36:43 I quickly stopped reading.

00:36:46 **Researcher**
And and and. Did it intuitively make sense to you?

00:36:49 What these code blocks meant what they were doing?

00:36:53 **Participant**
I would say so, yes.

00:36:57 I think you know for the first I think for the practice test also, but.
00:37:02 Yeah, I know what I want to do and I know how I
00:37:03 should do it.
00:37:04 It's just why would I do it when I can just ask the chat and it'll do it for me?
00:37:09 So it's not like I have no clue what it's doing, it's just.
00:37:14 I'm just, yeah, I said. Too lazy to type everything out.
00:37:18 But I know.
00:37:18 **Researcher**
So.
00:37:18 **Participant**
I know what it's doing.
00:37:21 **Researcher**
So for you, using Copilot is purely a way of saving
00:37:24 time or effort?
00:37:27 **Participant**
Yeah, I'd say so.
00:37:29 I don't know.
00:37:31 I.
00:37:31 I.
00:37:31 I barely looked at the pagination stuff, of course, obviously,
00:37:35 so I don't know how I would have fared on on that task if I had to do it without
00:37:40 the Copilot.
00:37:43 But for all the other tasks I would say I could easily achieve that without Copilot,
00:37:49 which taken longer.
00:37:51 **Researcher**
Right. And then about this pagination task that
00:37:55 you fully implemented with with Copilot, what was your impression of this?
00:38:02 **Participant**
I thought it worked really well.
00:38:05 I don't know how like I I know that there was already the pagination.
00:38:09 Pagination was already there were already, like there was already code for it.
00:38:14 So I I did not know how how much code there was before I let the chat do it.
00:38:22 So I don't know how it the code looked before.
00:38:25 But if did everything from scratch, that's very impressive.
00:38:30 But I would say that it's pretty, pagination.
00:38:33 You know it's not.
00:38:37 How I say this?
00:38:39 It's it's, it's a pretty common problem, so it would make sense that it knows how
00:38:43 to deal with it very well.
00:38:45 But I think once your problems become really like niche,
00:38:48 it might be more difficult for the chat.
00:38:51 So I I would.
00:38:52 I would still be hesitant to use.
00:38:55 It with, I'd say, more serious or more complex things.
00:39:01 **Researcher**
OK so.
00:39:01 **Participant**
I would.
00:39:02 I would definitely try, but I would take bigger I would.
00:39:09 Be more careful and actually read what it's doing instead of just copying it.
00:39:16 **Researcher**
That makes.
00:39:16 That makes a lot of sense.
00:39:17 That actually answers my my follow up question here.
00:39:21 So in what way would you verify responses from Copilot?
00:39:27 **Participant**

00:39:30 Well, like I said, normally I use the the like the tap
00:39:33 completion you tap it, it completes the lines for you.
00:39:35 So that's pretty simple.
00:39:39 To verify, because it's always just a little bit.
00:39:43 And it's pretty much it's pretty much just finishes it the way I would have
00:39:44 finished it already.
00:39:46 It's just that.
00:39:46 It's all in my head, yet all in my head and not typed out yet,
00:39:50 so that's very easy to verify. With the chat.
00:39:54 I guess it's just read through its see if I understand what it's doing.
00:39:59 If I don't understand it then I can maybe ask more questions.
00:40:04 I've done that in the past.
00:40:07 And if not, I would maybe ask it to generate
00:40:09 something that I would understand.
00:40:11 Like just try again, but do it a different way.
00:40:14 **Researcher**
OK.
00:40:16 Are there, so you're talking about using the auto complete usually.
00:40:21 **Participant**
Yeah.
00:40:21 **Researcher**
Do you see any advantages or disadvantages of the chat compared to
00:40:26 the auto complete?
00:40:28 **Participant**
Yeah, I think the chat of course has.
00:40:30 I don't know how what the auto complete has access to,
00:40:33 so I don't know if it has an at workspace where you can scan your entire workspace.
00:40:39 That's very useful, of course.
00:40:40 I don't know how good it would work when you have a really big workspace.
00:40:47 I can imagine that.
00:40:51 How do you say?
00:40:52 It becomes too much.
00:40:55 For a very small project like this, it works great.
00:40:59 **Researcher**
OK.
00:41:03 So you were talking about which in the survey you were indicating which aspects
00:41:09 of Copilot responses you liked or disliked,
00:41:11 **Participant**
Mm hmm.
00:41:12 **Researcher**
and I saw you ranked length and detail a bit lower than the rest.
00:41:17 Can you explain why?
00:41:18 **Participant**
I think the chat, especially the chat, always wants to explain things and I
00:41:23 don't need them explained 'cause I I. Like I said, I already have it in my head.
00:41:28 I know what I need to do.
00:41:29 I'm just too lazy to do it.
00:41:31 I don't need the explanation, so just leave those out or you know,
00:41:36 I feel like also this is more about ChatGPT,
00:41:39 but when you ask for a little bit of code then.
00:41:44 I've done in the past where you take your code and you just copy it into ChatGPT
00:41:48 and then you say like this method is not working correctly.
00:41:50 So I give the entire file for context, but I only really care about one method
00:41:54 and it will give you the entire the entire file back with just the changes.
00:41:58 And I don't need. I just need to. I just need the method and I'll copy back
00:42:02 the method.

00:42:04 So in general, I would like these tools to be more
00:42:07 concise and just brief and don't bother with explaining.
00:42:12 **Researcher**
Yeah.

00:42:13 **Participant**
Unless I ask for it, of course.
00:42:14 If I ask you, can you explain it and then they should.
00:42:18 **Researcher**
So so how do you usually deal with this? Or does this?
00:42:27 So when this happens, when you get a a very long or well overly
00:42:31 detailed response, how do you usually prevent this or or
00:42:35 deal with this? If it does happen?
00:42:38 **Participant**
I don't prevent it.
00:42:38 I probably should.
00:42:39 I just gloss over it.
00:42:41 Don't don't even bother reading it. I I know what I need and that's what I go
00:42:46 for.
00:42:46 And once I recognize that I I just copy paste.
00:42:47 **Researcher**
So does it.
00:42:50 So would you say it takes you a lot of effort to take out what you want?
00:42:58 **Participant**
That's a good question.
00:43:01 Well.
00:43:02 It would.
00:43:03 No, probably not. It would.
00:43:04 It would take me less effort if I only got what I want,
00:43:08 but you know the overall effort is still very low.
00:43:13 **Researcher**
Right. That makes sense.
00:43:14 So it will be less effort to just gloss over the irrelevant parts than to,
00:43:21 for example instruct.
00:43:24 An assistant to just give a very concise answer.
00:43:28 **Participant**
Well, it depends.
00:43:29 It depends if if I could have like global settings where I can say for every
00:43:34 response be very brief and just, you know give the give only the changed
00:43:38 code for example only the changed method.
00:43:41 That'd be great. If I have to do it for every prompt, every problem.
00:43:43 That becomes very annoying very quickly.
00:43:46 **Researcher**
Yeah, that makes sense.
00:43:49 OK, so I think one more question.
00:43:53 So you've been using the Copilot chat throughout the throughout the tasks.
00:43:59 Probably. Or maybe you noticed some things that it
00:44:02 was good at or not so good at that you liked or disliked. So over time,
00:44:06 do you think you changed the way you interacted with Copilot based on what you
00:44:10 noticed?
00:44:12 **Participant**
Definitely. Once I realized that at first I would
00:44:15 just use it in a way that I think I would be used to like.
00:44:20 I know that I need to use local storage so.
00:44:24 I would ask it for something like that.
00:44:25 I think I'm not sure what I asked it, but I'm probably in in a in a practical
00:44:30 application, say something like. Oh, use local storage to save.

00:44:36 And I so part of the solution is already dictated by me, I'd say.
00:44:40 And then once I realized, oh, this is actually really smart,
00:44:43 I could just say fix pagination and it does it.
00:44:45 It definitely takes away.
00:44:48 I don't need to think about the implementation anymore.
00:44:50 I know that if I had to, I could.
00:44:53 It's just not now.
00:44:54 I don't have to, so I it makes me happy to not think about
00:44:58 **Researcher**
Right.
00:44:59 **Participant**
the complex code.
00:45:02 **Researcher**
So for you this is a.
00:45:03 That's a purely positive sort of experience to have.
00:45:10 **Participant**
Yeah, I do think that if I were to do this in,
00:45:13 like a professional setting.
00:45:16 They probably wouldn't think very highly of me.
00:45:22 **Researcher**
And why not?
00:45:24 **Participant**
'Cause it doesn't really demonstrate that you know how to solve the problems
00:45:28 yourself, I'd say.
00:45:31 And I guess.
00:45:31 **Researcher**
Is that for you?
00:45:32 Relevant if you have the same output, let's say.
00:45:38 **Participant**
Well, I can't, in my job.
00:45:39 I'm not allowed to use like things like this,
00:45:41 so for me it is important that I actually know how to do things.
00:45:44 **Researcher**
Yeah, that makes sense. OK.
00:45:47 Sorry, one more question. At some point you you had an error
00:45:49 **Participant**
No problem.
00:45:51 **Researcher**
somewhere I think in the survey you indicated that you had some frustration
00:45:56 during this moment and that you broke the code.
00:45:57 **Participant**
Mm hmm.
00:46:00 **Researcher**
So.
00:46:03 How did Copilot affect this moment?
00:46:06 Was it in your eyes the cause of this confusion or frustration,
00:46:10 or did it help in solving it?
00:46:14 **Participant**
No, I think at that point I still wanted to
00:46:16 sort of do it myself a little bit.
00:46:19 So typing everything manually and that's where I introduced the bug.
00:46:23 But I couldn't see the bug because normally, I'm just so used to like the IntelliJ
00:46:29 interface and how it looks where the how it shows you where the bugs are.
00:46:33 So I just couldn't find it in the in the UI of of the Visual Studio actually.
00:46:39 So I figured why waste time trying to look for it when the chat can probably
00:46:43 tell me where it is.

00:46:45

Researcher

All right. And in real life programming situations in your work?

00:46:47

00:46:51

Do these moments happen as well?

00:46:59

Participant

No, I'd say because so the chat is like the separate window. So it definitely takes away a bit of your attention to the actual code.

00:47:02

00:47:06

I think that's also part of the reason why.

00:47:10

Also, because I didn't really know the codebase right, I was just going from file to file

00:47:12

00:47:15

without really taking in the meaning of everything.

00:47:17

So I think the combination of that made it so that I missed the the bug.

00:47:20

Researcher

OK.

00:47:26

Participant

Just but of course, in like in my use case where I would use the auto complete feature, I'm in the code as I'm getting those suggestions.

00:47:27

00:47:31

00:47:35

I feel like that wouldn't really happen.

00:47:36

That quickly.

00:47:38

Researcher

OK.

00:47:38

That makes sense.

00:47:39