

Participant P04

00:04:26

Chat (window: 0)

> Agent

none

> Context

> Message

/help

> Response

You can ask me general programming questions, or chat with the following participants which have specialized expertise and can perform actions:

- [workspace](#) - Ask about your workspace
 - [/explain](#) - Explain how the code in your active editor works
 - [/tests](#) - Generate unit tests for the selected code
 - [/fix](#) - Propose a fix for the problems in the selected code
 - [/new](#) - Scaffold code for a new file or project in a workspace
 - [/newNotebook](#) - Create a new Jupyter Notebook
- [vscode](#) - Ask questions about VS Code
 - [/search](#) - Generate query parameters for workspace search
- [terminal](#) - Ask how to do something in the terminal
 - [/explain](#) - Explain something in the terminal
- [github](#) - Get answers grounded in web search, code search, and your enterprise's knowledge bases

You can also help me understand your question by using the following variables to give me extra context: * #file - Choose a file in the workspace

To have a great conversation, ask me questions as if I was a real programmer:

- **Show me the code** you want to talk about by having the files open and selecting the most important lines.
- **Make refinements** by asking me follow-up questions, adding clarifications, providing errors, etc.
- **Review my suggested code** and tell me about issues or improvements, so I can iterate on it.

You can also ask me questions about your editor selection by [starting an inline chat session](#).

Learn more about [GitHub Copilot](#) in [Visual Studio Code](#). Or explore the [Copilot walkthrough](#).

00:04:26

- 00:04:26

00:04:26

> Response rendering

Participant

Can I?

00:04:29

Researcher

You can.

00:04:29

You can do the the help thing.

00:04:31

You can experiment a bit with how these commands work.

00:04:36

Whatever works for you.

00:04:44

Participant

OK, cool.

00:04:51

Pretty straightforward, yeah.

00:04:51

Researcher

Before.

00:04:53

Hopefully, yeah.

00:04:54

Participant

00:04:56 **Researcher**
Hopefully.
And if not, then that's also fine, right?

00:04:58 **Participant**
Yeah.

00:05:02 **Researcher**
Right. So and before you start the first
00:05:05 practice task, could you please try and figure out how
00:05:06 **Participant**
Mm hmm.

00:05:08 **Researcher**
to start the application?

00:05:41 **Participant**
I don't think I missed anything in the getting started. I hope I didn't.
00:05:47 I don't see anything here on how to?
00:05:52 Actually start?
00:05:55 Do I have to go to a file specifically and then just?
00:06:03 **Researcher**
So we're letting you figure this out yourself.
00:06:07 If you could figure out how to run the web application that we're that we have
00:06:13 here.
00:06:18 So what's your? What's your first idea on how to approach
00:06:21 this?
00:06:21 Would you have any idea?
00:06:35 **Participant**
No, not really.

00:06:40 **Researcher**
OK.
00:06:41 Feel free to look around in the in the code that we have here.
00:06:47 > **Viewing file** (src/index.js)

00:06:48 **Researcher**
Or.
00:06:49 To see if this is familiar or you could also ask Copilot if you want.

00:07:03 **Participant**
Yeah, I see.
00:07:06 Yeah, it's React I think.
00:07:09 I could ask about this.
00:07:40 So what I have to do now is like.
00:07:44 Start Copilot on this file?

00:07:49 **Researcher**
IYou don't start Copilot or anything.

00:07:51 **Participant**
Yeah.

00:07:52 **Researcher**
So we have a. This is the code for a web application.

00:07:56 **Participant**
Yeah.

00:07:58 **Researcher**
And we just ask you to to run this web application so you can see the the
00:08:03 website that we're that we're working on here.
00:08:13 Does that make sense or is it still vague?

00:08:21 **Participant**
So Copilot can start the web application for me, but I have to.

00:08:27 **Researcher**
I don't think.
00:08:28 I don't think Copilot can do it for you.

00:08:33 Have you worked with a web application like this before?

00:08:36 **Participant**
Yeah, I'm working on React on another course
at the moment, yeah.

00:08:41 **Researcher**
OK.

00:08:44 **Participant**
But not using Copilot I I can start it on the terminal myself.
Is that what is needed?

00:08:51 **Researcher**
OK.

00:08:54 Well, yeah, please do that.

00:09:05 **Participant**
Is not with npm oh.

00:09:09 **Researcher**
I can't give you any hints.
Any hints anymore.
So what are you thinking right now?

00:09:11 **Participant**
I could ask Copilot.
About.

00:09:23 **Researcher**
OK.

00:09:31 **Participant**
Starting.

00:09:34 **Participant**
OK.

00:09:37 **Chat** (window: 0)

00:09:45 **> Agent**
github.copilot.workspace

> Context
workspace, viewport (src/index.js)

> Message
how do i start my app from index.js

> Response
To start your app, you can use the dev script defined in your [package.json](#). This script uses webpack-dev-server to serve your app in development mode.

Run the following command in the integrated terminal of Visual Studio Code:

```
npm run dev
```

This will start the development server, and your app will be accessible in your default web browser. The [render\(src/index.js\)](#) function in your [src/index.js](#) file will mount the React app to the #root element in your [public/index.html](#) file.

00:09:56 **> Response rendering**
- 00:09:57

00:10:04 **Participant**
Oh, OK.
[unintelligible] I think.
And now I have to investigate the error.
Cool.

00:10:04 **Chat** (window: 0)

00:10:19 **> Agent**
github.copilot.default

00:10:25 **> Context**
viewport (src/index.js)

00:10:50 **> Message**
when i run the scrips, it does not find a webpack webpack.dev.js

> Response

It seems like your project might be missing the webpack configuration file (`webpack.dev.js`) required for development. If you're using webpack and it's not configured, you'll need to create the necessary configuration files.

Here's how you can set up a basic [webpack.dev.js](#) file:

1. Install Required Dependencies

Run the following command to install webpack, webpack-cli, and webpack-dev-server:

```
npm install --save-dev webpack webpack-cli webpack-dev-server
```

2. Create [webpack.dev.js](#)

Create a new file named [webpack.dev.js](#) in the root of your project and add the following configuration:

```
File: webpack.dev.js
const path = require("path");

module.exports = {
  mode: "development",
  entry: "./src/index.js",
  output: {
    path: path.resolve(__dirname, "dist"),
    filename: "bundle.js",
    publicPath: "/",
  },
  devServer: {
    static: {
      directory: path.join(__dirname, "dist"),
    },
    compress: true,
    port: 3000,
    historyApiFallback: true,
  },
  module: {
    rules: [
      {
        test: /\.js$/,
        exclude: /node_modules/,
        use: {
          loader: "babel-loader",
        },
      },
      {
        test: /\.css$/,
        use: ["style-loader", "css-loader"],
      },
    ],
  },
  resolve: {
    extensions: [".js", ".jsx"],
  },
};
```

3. Update [package.json](#)

Add or update the dev script in your [package.json](#) file to use webpack-dev-server:

```
"scripts": {  
  "dev": "webpack serve --config webpack.dev.js"  
}
```

4. Run the Development Server

Now, you can start your app by running:

```
npm run dev
```

This will start the development server, and your app should be accessible at `http://localhost:3000`.

Let me know if you encounter any issues!

> Response rendering

00:10:51
- 00:11:02

00:11:09

Participant

Oh, that's a lot.

00:11:23

Researcher

So what do you think when you see this response?

00:11:27

Participant

Sure. It's a step by step, but.

00:11:34

I feel like I have to go through everything before.

00:11:39

I actually want to do it because I don't trust this to give a perfect file from

00:11:44

the start like this. I would have to go through it to see if

00:11:49

it's actually written correctly, and it's a lot.

00:11:55

Yeah, I expected an install.

00:11:59

Then create a file.

00:12:03

Update the JSON.

00:12:06

Sure.

00:12:07

And then we can start.

00:12:11

Sure.

00:12:14

I can copy sure.

00:12:20

Researcher

Oh, sorry. Yeah, to copy and paste you can allow this.

00:12:24

This pop up that comes up.

00:12:31

Participant

I need to look.

00:12:32

Yeah, that's how it looks when it installs.

00:12:38

Let's prepare for the next step, and hopefully the Copilot is right.

00:12:44

And I don't have to create the file for nothing.

00:12:56

OK.

00:12:57
00:13:01

00:13:03
00:13:07
00:13:07
00:13:22
00:13:57

> **Viewing file** (webpack.dev.js)

Participant

We already have something here.

Cool. Well, is it the same thing?

It is not.

So let's see what is missing.

OK.

Chat (window: 0)

> **Agent**

github.copilot.default

> **Context**

selection (webpack.dev.js)

> **Message**

i already have a file webpack.dev.js that has some code. do i incorporate your code or can i take your directly?

> **Response**

Since you already have a [webpack.dev.js](#) file, you should **merge the necessary parts** of my suggested configuration into your existing file rather than replacing it entirely. Here's how you can incorporate the missing pieces:

Updated [webpack.dev.js](#)

```
File: webpack.dev.js
const { merge } = require("webpack-merge");
const common = require("../webpack.common.js");

module.exports = merge(common, {
  mode: "development",
  devtool: "inline-source-map",
  devServer: {
    static: "./dist",
    compress: true, // Enables gzip compression
    port: 3000, // Sets the port to 3000
    historyApiFallback: true, // Ensures routing works with React
  },
  Router: {
    module: {
      rules: [
        {
          test: /\.css$/i,
          use: ["style-loader", "css-loader"],
        },
      ],
    },
  },
});
```

Explanation of Changes

1. **compress: true:** Enables gzip compression for better performance.
2. **port: 3000:** Specifies the port for the development server (you can change it if needed).
3. **historyApiFallback: true:** Ensures React Router works properly by redirecting all requests to index.html.

Next Steps

1. Save the updated [webpack.dev.js](#).
2. Run the development server using:

npm run dev

This should work without replacing your existing configuration. Let me know if you encounter any issues!

00:13:58
- 00:14:01

> Response rendering

00:14:16

Participant

OK.

00:14:42

Let's see if it does it for me.

00:14:58

Chat (window: 0)

> Agent

github.copilot.default

> Context

selection (webpack.dev.js)

> Message

i have this. what is to be incorporated? : const { merge } = require("webpack-merge");
const common = require("./webpack.common.js"); module.exports = merge(common,
{ mode: "development", devtool: "inline-source-map", devServer: { static: "./dist", },
module: { rules: [{ test: /\.css\$/i, use: ["style-loader", "css-loader"], },], }, });

> Response

Your existing [webpack.dev.js](#) file is mostly complete. You only need to incorporate a few additional configurations to ensure your React app works properly with React Router and the development server. Here's the updated version with the necessary additions:

Updated [webpack.dev.js](#)

```
File: webpack.dev.js
const { merge } = require("webpack-merge");
const common = require("./webpack.common.js");

module.exports = merge(common, {
  mode: "development",
  devtool: "inline-source-map",
  devServer: {
    static: "./dist",
    compress: true, // Enables gzip compression for better
performance
    port: 3000, // Sets the port for the dev server
    historyApiFallback: true, // Ensures React Router works
properly
  },
  module: {
    rules: [
      {
        test: /\.css$/i,
        use: ["style-loader", "css-loader"],
      },
    ],
  },
});
```

Changes Incorporated:

1. **compress: true:** Enables gzip compression for better performance.
2. **port: 3000:** Specifies the port for the development server (you can change it if needed).
3. **historyApiFallback: true:** Ensures React Router works properly by redirecting all requests to index.html. This is critical for apps using React Router's HashRouter or BrowserRouter.

Next Steps:

1. Save the updated [webpack.dev.js](#).
2. Run the development server using:

```
npm run dev
```

Your app should now work correctly with React Router and the development server.
Let me know if you encounter any issues!

00:14:59
- 00:15:08

> Response rendering

00:15:18

Participant

OK, I will trust him.

00:15:19

> **Implementing Copilot code** by copying full snippet (`webpack.dev.js`)

00:15:21

Participant

And this should be enough.

00:15:29 Oh, he was right.
00:15:32 Thankfully.
00:15:33 **Researcher**
Great. There you go.
00:15:34 **Participant**
It should be done. Yay. Yep. OK.
00:15:37 **Researcher**
Yeah. Thanks for thinking out loud the way you're doing by the way.
00:15:40 That's really helping.
00:15:40 **Participant**
Yeah, yeah, I'm trying.
00:15:42 It's a little weird at the beginning, yeah.
00:15:44 And yeah, I'm a little skeptical with Copilot
00:15:44 **Researcher**
Uh.
00:15:46 **Participant**
because I told you I'm not necessarily using it.
00:15:49 We are not allowed in our course to use Copilot.
00:15:51 So it's yeah, I. So I use ChatGPT and need to also
00:15:56 think before I do something.
00:15:59 But it's nice that it's reliable when incorporating code. I think it's OK.
00:16:03 **Researcher**
Yeah.
00:16:04 **Participant**
Yeah.
00:16:06 **Researcher**
And I by the way, I saw you copied and pasted code from
00:16:10 your file.
00:16:13 But just to just to confirm.
00:16:13 **Participant**
Oh, true.
00:16:17 **Researcher**
It seems to file you're currently working in.
00:16:22 And if you want it to see all of the code in your project,
00:16:24 there's also some commands you can use for that.
00:16:28 Still feel free to use in any way you want, but just so you know, right?
00:16:28 **Participant**
Yeah.
00:16:31 OK.
00:16:31 Maybe that's a toxic behavior because I copy paste into ChatGPT every time.
00:16:35 So that was.
00:16:37 I didn't realize doing that, but I'm trying to remember. Yeah.
00:16:41 **Researcher**
Yeah, that makes sense.
00:16:46 **Participant**
Should we get started with the todos?
00:16:48 **Researcher**
Yeah.
00:16:49 **Participant**
Yeah.
00:17:05 OK.
00:17:10 I wonder if it can find.
00:17:13 This button specifically.
00:17:41 **Chat** (window: 0)
> Agent
github.copilot.default

> Context

viewport (webpack.dev.js)

> Message

find in my files where a button named "No TODOs here yet" is

> Response

To locate the button with the text "No TODOs here yet" in your project, you can search for the text in your workspace. Use the following steps:

In Visual Studio Code:

1. Press Ctrl+Shift+F (or Cmd+Shift+F on macOS) to open the global search.
2. Enter the text "No TODOs here yet" (including the quotes) in the search bar.
3. Review the search results to find the file and line where this text is defined.

Alternatively, you can search for it in the terminal using the grep command:

In the Terminal:

Run the following command in your project directory:

```
grep -rnw './' -e "No TODOs here yet"
```

This will search recursively in all files and display the file path and line number where the text is found. Let me know if you need further assistance!

00:17:42
- 00:17:44

> Response rendering

00:17:55

Participant

OK.

00:17:57

> Viewing file (src/index.js)

00:17:57

Participant

Not necessarily what I was expecting.

00:18:05

> Viewing file (package.json)

00:18:05

Researcher

Just to just to say again, if you look at the grey box in the

00:18:09

> Viewing file (package-lock.json)

00:18:11

Researcher

instructions.

00:18:15

There it has some some recommendations about the commands.

00:18:26

Participant

OK.

00:18:28

Researcher

Just so you know.

00:18:29

Participant

Oh.

00:18:31

Then if I.

00:18:35

Have the same question but.

00:18:39

At workspace.

00:18:46

Chat (window: 0)

> Agent

github.copilot.workspace

> Context

workspace, viewport (package-lock.json)

> Message

@workspace find in my files where a button named "No TODOs here yet" is

> Response

The "No TODOs here yet" message is not a button but a span element. It is located in the [src/todo/components/main.jsx](#) file within the [Main \(src/todo/components/main.jsx\)](#) component:

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

00:18:47

Participant

Then it should look through all my files.

00:18:51

And hopefully it can find them where the button is located.

00:19:00

- 00:19:02

> Response rendering

00:19:11

Participant

OK, nice.

00:19:13

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

00:19:25

Participant

OK, hopefully this doesn't break the app.

00:19:29

Should be around here.

00:19:33

There we go.

00:19:37

And we need to change it to.

00:19:41

What color?

00:19:52

Chat (window: 0)

> Agent

github.copilot.default

> Context

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

> Message

change the color of this button to #007bff

> Response

To change the color of the "No TODOs here yet" message, you can add a CSS rule targeting the [todo-list-empty-message \(src/todo/components/main.jsx\)](#) class. Here's how you can do it:

Update the CSS File

Locate the CSS file where styles for your components are defined (e.g., [main.css \(node_modules/@types/react/index.d.ts\)](#) or a global CSS file). Add the following rule:

```
File: src/todo/components/main.css
.todo-list-empty-message {
  color: #007bff;
}
```

Ensure the CSS is Imported

If the CSS file is not already imported in your [src/todo/components/main.jsx](#) file, add the import statement at the top:

```
File: src/todo/components/main.jsx
import "../main.css";
```

Save and Test

Save the changes and restart your development server if necessary. The "No TODOs here yet" message should now appear in the color #007bff. Let me know if you need further assistance!

00:19:53
- 00:20:01

> Response rendering

00:20:12

Participant

In CSS here.

00:20:35

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

00:20:39

Participant

Oh yeah that that's the CSS OK.

00:20:43

I guess it did not directly reference to my main CSS.

00:20:44

> **Viewing file** (node_modules/%40types/react/index.d.ts) through Copilot link

00:20:47

Participant

OK for some reason it says here it's main but it's called.

00:20:53

Oh, OK.

00:20:58

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

00:20:58

Participant

Sure.

00:21:01

Researcher

So what are you thinking?

00:21:02

Participant

It kind of confuses me with the naming.

00:21:17

OK, I found it here that I need to change it.

00:21:20

I, it threw me off with this.

00:21:25

Sure.

00:21:27

> **Implementing Copilot code** by copying part of snippet (src/todo/app.css)

00:21:38

Participant

Cool.

00:21:41

I mean, maybe it's just I'm, I'm still a noob with development.

00:21:44 Maybe that's an issue with me, but.

00:21:50 > **Viewing file** (node_modules/%40types/react/index.d.ts) through Copilot link

00:21:50 **Participant**

Yeah, I was thrown a bit off with this.

00:21:53 > **Viewing file** (src/todo/components/input.jsx)

00:21:54 **Researcher**

That's good to hear.

00:21:58 **Participant**

OK.

00:22:00 > **Viewing file** (src/todo/app.css)

00:22:02 **Participant**

I guess that's the first task.

00:22:07 OK.

00:22:12 And now I have to change what needs to be done this part in the text box.

00:22:20 To another text.

00:22:22 This should be fine.

00:22:23 Let's see if we can find it again.

00:22:28 What was it? Workspace.

00:22:30 I think this one is quite useful.

00:22:32 If only I had more experience.

00:22:59 I'm also trying to be nice to him.

00:23:03 Just because it's recorded, you know it sounds weird, but.

00:23:09 **Researcher**

To be nice in what way?

00:23:11 **Participant**

Like.

00:23:13 Try to be as specific as possible and not gives too many queries or sound

00:23:18 mean.

00:23:19 Or maybe it's just me.

00:23:20 I don't know.

00:23:22 **Researcher**

Yep, please feel free to interact with

00:23:24 Copilot however you want.

00:23:27 It'll be fully anonymous.

00:23:29 **Participant**

I mean, maybe if I get used to Copilot, I won't be as nice, but for now,

00:23:33 I'm skeptical. 'cause I don't know how it will interact

00:23:36 with my messages.

00:23:37 Like if I'm not specific enough, it will not understand what I'm telling

00:23:41 him to do.

00:23:43 **Researcher**

OK.

00:23:43 That's fair enough.

00:23:44 **Chat** (window: 0)

> **Agent**

github.copilot.workspace

> **Context**

workspace, viewport (src/todo/app.css)

> **Message**

@workspace find a text container that says What needs to be done

> **Response**

The text container with the placeholder "What needs to be done?" is located in the [Input \(src/todo/components/input.jsx\)](#) component in [src/todo/components/header.jsx](#). It is rendered as part of the [Header \(src/todo/components/header.jsx\)](#) component:

```
File: src/todo/components/header.jsx
<Input onSubmit={addItem} label="New Todo Input" placeholder="What
needs to be done?" />
```

00:23:48

Participant

OK.

00:23:49

I think I hope this is enough information for him.

00:23:57
- 00:23:57

> Response rendering

00:24:06

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

00:24:11

Participant

OK.

00:24:12

Let's see if it's this one.

00:24:14

What needs to be done.?

00:24:29

And we need to change it to enter a task here.

00:24:39
- 00:24:47

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

00:24:39

Participant

Do that myself, if that's OK.

00:24:56

Cool, cool.

00:24:56

I mean, it's also the simplest things that you

00:25:01

can do yourself and not consume.

00:25:06

His resources. That's also, I find it unsustainable.

00:25:13

So.

00:25:14

Researcher

Unsustainable in what way?

00:25:17

Participant

I know how.

00:25:20

Environmentally unfriendly.

00:25:23

AI's can be so if I know how to do a task, why not just do it myself?

00:25:32

Because of course, I know he would do it as well.

00:25:36

That's also possible, but in this case I can also be faster

00:25:40

than him.

00:25:42

Than.

00:25:44

Copilot.

00:25:45

So.

00:25:48

Task is done.

00:26:01

Task C.

00:26:20

OK.

00:26:43

I'm trying to understand what is being asked.

00:26:51

Where can I find the?

00:26:56

Containers where these are stored on submit an item.

00:27:16

Let's see if I can get some help.

00:27:19

I'm gonna ask for.

00:27:25

Uh, where the items from the?

00:27:30

Uh.

00:27:33

This part specifically maybe.

00:27:36

Can I actually like?

00:27:40

Select code and he will know what I'm referring to.

00:27:46

Researcher

I'm. I'm not sure, you can try this out if you want to.

00:27:49

00:27:57

Participant

Oh, actually. I think he can see the lines, OK.

00:27:59

I didn't realize this before.

00:28:04

Umm.

00:28:26

Chat (window: 0)

> Agent

github.copilot.default

> Context

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

> Message

where does this onSubmit addItem action store the input

> Response

The [onSubmit \(src/todo/components/header.jsx\)](#) action ([addItem \(src/todo/components/header.jsx\)](#)) in the [Header \(src/todo/components/header.jsx\)](#) component dispatches an action to a reducer, which likely updates the state of the todos list in your application. Here's how it works:

Breakdown:

1. [addItem \(src/todo/components/header.jsx\)](#) Function:

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

- This function creates an action with the type [ADD_ITEM \(src/todo/constants.js\)](#) and a payload containing the [title \(src/todo/components/header.jsx\)](#) (the input value).
- It then dispatches this action to the reducer.

2. Reducer Handling [ADDITEM \(src/todo/constants.js\)](#): The [dispatch \(src/todo/components/header.jsx\)](#) function is connected to a reducer (likely in a *useReducer* or *Redux* setup). The reducer processes the [ADDITEM \(src/todo/constants.js\)](#) action and updates the state by adding the new todo item to the todos list.

Example of how the reducer might handle this:

```
function todoReducer(state, action) {
  switch (action.type) {
    case ADD_ITEM:
      return {
        ...state,
        todos: [...state.todos, { id: Date.now(), title:
action.payload.title, completed: false }],
      };
      // Other cases...
    default:
      return state;
  }
}
```

3. State Storage: The updated todos list is stored in the state managed by the reducer. This state is then passed down as a prop to components like *Main*, which renders the list of todos.

Summary:

The input value is stored in the todos array in the application's state, managed by the reducer. The [ADD_ITEM\(src/todo/constants.js\)](#) action adds a new todo item to this array. Let me know if you'd like to see the reducer code or further details!

00:28:28
- 00:28:35

> Response rendering

00:28:31

Participant

I'm not sure if this is enough for him to understand.

00:28:40

OK.

00:28:58

I I just need to understand this.

00:29:40

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

00:29:40

Participant

OK.

00:29:43

The functions.

00:29:46

But.

00:29:55

But where is it stored?

00:29:57

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

00:30:00

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

00:30:15

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

00:30:18

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

00:30:28

Participant

OK.

00:30:29

I think I have a question.

00:30:36

Chat (window: 0)

> Agent

github.copilot.default

> Context

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

> Message

what is a reducer

> Response

A **reducer** is a pure function in JavaScript that determines how the state of an application should change in response to an action. Reducers are commonly used in state management libraries like Redux or useReducer in React.

Key Concepts of a Reducer:

1. Pure Function:

- A reducer is a function that takes the current state and an action as arguments and returns a new state.
- It does not modify the existing state directly (immutability is key).

2. Signature:

```
function reducer(state, action) {  
  // Logic to handle the action and return the new state  
  return newState;  
}
```

3. Arguments:

- state: The current state of the application.
- action: An object that describes what happened. It typically has:
 - [type\(src/todo/components/header.jsx\)](#): A string that specifies the type of action (e.g., "ADD_ITEM", "REMOVE_ITEM").
 - [payload\(src/todo/components/header.jsx\)](#): Additional data needed to perform the action (e.g., the new todo item).

4. Return Value:

- The reducer returns a new state object based on the action.

Example Reducer:

Here's an example of a reducer for managing a todo list:

```
function todoReducer(state, action) {
  switch (action.type) {
    case "ADD_ITEM":
      return {
        ...state,
        todos: [...state.todos, { id: Date.now(), title:
action.payload.title, completed: false }],
      };
    case "REMOVE_ITEM":
      return {
        ...state,
        todos: state.todos.filter((todo) => todo.id !==
action.payload.id),
      };
    default:
      return state; // Return the current state if the action
type is not recognized
  }
}
```

How It Works:

1. Initial State:

```
const initialState = { todos: [] };
```

2. Dispatching Actions: Actions are dispatched to the reducer to update the state:

```
dispatch({ type: "ADD_ITEM", payload: { title: "Learn Reducers" }
});
```

3. Reducer Updates State: The reducer processes the action and returns a new state:

```
const newState = todoReducer(currentState, action);
```

In React:

Reducers are often used with the `useReducer` hook:

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

Summary:

A reducer is a function that takes the current state and an action, processes the action, and returns a new state. It's a core concept in managing predictable state transitions in applications. Let me know if you'd like to see how this applies to your project!

00:30:39
- 00:30:49

00:31:10

00:31:12

00:31:20

00:31:25

00:31:32

00:31:35

00:31:44

00:32:03

> Response rendering

Researcher

So what are you thinking now?

Participant

I need to study some more on JavaScript.

Thinking of questions to ask because I need to find.

How it's this thing stores values so that I can modify that specific array to be to store them alphabetically.

Uh.

OK.

Chat (window: 0)

> Agent

github.copilot.default

> Context

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

> Message

how can i modify this to have the input stored alphabetically?

> Response

To store the input alphabetically, you can modify the reducer that handles the [ADD_ITEM\(src/todo/constants.js\)](#) action to sort the todos array after adding a new item. Here's how you can do it:

Update the Reducer

Modify the reducer to sort the todos array alphabetically by the [title \(src/todo/components/header.jsx\)](#) whenever a new item is added.

```
File: src/todo/reducer.js
function todoReducer(state, action) {
  switch (action.type) {
    case "ADD_ITEM":
      const updatedTodos = [
        ...state.todos,
        { id: Date.now(), title: action.payload.title,
completed: false },
      ];
      // Sort todos alphabetically by title
      updatedTodos.sort((a, b) =>
a.title.localeCompare(b.title));
      return {
```

```

        ...state,
        todos: updatedTodos,
    };
    // Other cases...
    default:
        return state;
    }
}

```

Explanation:

1. Add the New Item:

- The new item is added to the todos array using the spread operator (`...state.todos`).

2. Sort the Array:

- The sort method is used to sort the array alphabetically by the [title](#) ([src/todo/components/header.jsx](#)) property.
- `localeCompare` ensures proper alphabetical sorting, including handling case sensitivity.

3. Return the Updated State:

- The updated and sorted todos array is returned as part of the new state.

No Changes Needed in [Header.jsx](#) ([src/todo/components/header.jsx](#))

The [Header](#) ([src/todo/components/header.jsx](#)) component remains the same because the sorting logic is handled in the reducer. The [addItem](#) ([src/todo/components/header.jsx](#)) function will dispatch the [ADD_ITEM](#) ([src/todo/constants.js](#)) action as usual, and the reducer will ensure the todos list is sorted.

Let me know if you need further clarification or assistance!

00:32:06
– 00:32:14

> Response rendering

00:32:24

Participant

Expected. So where are we going?

00:32:33

Researcher

00:32:35 What are you looking for right now?

00:32:40 **Participant**

00:32:40 Update the reducer. What reducer?

00:32:40 Is there?

00:32:40 Do I have a reducer? I don't think.

00:32:48 OK.

00:33:00 **Chat** (window: 0)

> **Agent**

github.copilot.workspace

> **Context**

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

> **Message**

@workspace do i have a reducer that can be modified for this task

> **Response**

Yes, your reducer in [src/todo/reducer.js](#) can be modified to handle the addition of new tasks. It already has a case for the [ADD_ITEM \(src/todo/constants.js\)](#) action:

```
File: src/todo/reducer.js
case ADD_ITEM:
  newState = state.concat({ id: generateUniqueID(), title:
action.payload.title, completed: false });
  break;
```

This logic appends a new task with a unique ID, the title provided in the action payload, and a default [completed: false \(src/todo/components/item.jsx\)](#) status. If you need to modify or extend this behavior, you can update this case in the reducer.

00:33:16
- 00:33:19

00:33:24 > **Response rendering**

00:33:24 > **Viewing file** (src/todo/reducer.js)

00:33:24 **Participant**

00:33:27 Oh.

00:33:27 OK, nice.

00:33:39 Nice.

00:33:41 Add, new state.

00:33:53 Is that one? Now I have to.

00:34:00 Modify it such that it has alphabetical input.

00:34:06 I also expected it to also implement this part.

00:34:13 Can I quote him on this?

00:34:14 No, I don't think so.

00:34:19 Then we have this.

00:34:22 Let's ask him.

00:34:24 Umm.

00:34:28 To make it such that the input.

00:34:32 Is stored.

00:34:34 Alphabetically.

00:34:35 **Chat** (window: 0)

> **Agent**

github.copilot.default

> **Context**

selection (src/todo/reducer.js)

> **Message**

make it that input is stored alphabetically

> **Response**

To store the input alphabetically, you can modify the [ADD_ITEM \(src/todo/constants.js\)](#) case in your [todoReducer \(src/todo/reducer.js\)](#) to sort the [state \(src/todo/reducer.js\)](#) array after adding the new item. Here's the updated code:

Updated Reducer

```
File: src/todo/reducer.js
case ADD_ITEM:
  newState = state
    .concat({ id: generateUniqueID(), title: action.payload.title,
completed: false })
    .sort((a, b) => a.title.localeCompare(b.title)); // Sort
alphabetically by title
  break;
```

Explanation:

1. Add the New Item:

- The `state.concat()` ([src/todo/reducer.js](#)) method adds the new todo item to the existing state array.

2. Sort Alphabetically:

- The `sort()` method is used to sort the array alphabetically by the `title` ([src/todo/reducer.js](#)) property.
- `localeCompare()` ensures proper alphabetical sorting, including handling case sensitivity.

3. Save the Updated State:

- The `saveState(newState)` ([src/todo/reducer.js](#)) function will save the sorted state to `localStorage` ([node_modules/@types/node/globals.d.ts](#)).

Full Updated Reducer Code:

```
File: src/todo/reducer.js
case ADD_ITEM:
  newState = state
    .concat({ id: generateUniqueID(), title: action.payload.title,
completed: false })
    .sort((a, b) => a.title.localeCompare(b.title)); // Sort
alphabetically by title
  break;
```

Result:

Now, whenever a new todo item is added, the todos list will automatically be sorted alphabetically by the `title` ([src/todo/reducer.js](#)). Let me know if you need further assistance!

00:34:37
- 00:34:43

> Response rendering

00:34:37

Participant

Uh.

00:34:41

Hopefully.

00:34:59

Why, this is.

00:35:03

The same.

00:35:04

OK.

00:35:09

I won't judge it.

00:35:10

I'll just take it.

00:35:12

> **Implementing Copilot code** by copying full snippet (`src/todo/reducer.js`)

00:35:13
- 00:35:15

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

00:35:30 **Participant**
OK.

00:35:34 I'm not sure about this why it's giving a problem but.

00:35:45 OK.

00:35:48 I can't write in it.

00:35:52 Let's see.

00:36:02 And.

00:36:05 Let's.

00:36:06 OK.

00:36:10 Can I just?

00:36:12 The.

00:36:22 OK, nice.

00:36:30 **Researcher**
Perfect. Thanks.

00:36:30 **Participant**
And it's working. Yeah, that was it.

00:36:34 **Researcher**
Great. Yes, those were the practice tasks.

00:36:38 So.

00:36:38 **Participant**
Cool.

00:36:40 **Researcher**
I'll send you the other tasks right now.

00:36:44 We have 5 minutes left, but no worries at all about that.

00:36:48 Just get started and see how far you can get.

00:36:53 **Participant**
OK.

00:36:55 **Researcher**
Thanks for thinking out loud, by the way. It's really great.

00:37:00 **Participant**
I'm glad I can I can help.

00:37:38 **Researcher**
So what are you thinking right now?

00:37:40 **Participant**
Asking him how to make the input.

00:37:47 Stay there without even though the pages is refreshed.

00:38:12 **Chat** (window: 0)
> Agent
github.copilot.default
> Context
selection (src/todo/reducer.js)
> Message
how do i change this so the input is persistent even after a refresh
> Response
To make the input persistent even after a page refresh, you can leverage [localStorage](#) ([node_modules/@types/node/globals.d.ts](#)) to save and load the state. Your reducer already has a [saveState \(src/todo/reducer.js\)](#) function and an [initialState \(src/todo/reducer.js\)](#) that loads from [localStorage](#) ([node_modules/@types/node/globals.d.ts](#)). You just need to ensure that the [ADD_ITEM \(src/todo/constants.js\)](#) action properly updates [localStorage](#) ([node_modules/@types/node/globals.d.ts](#)) when a new item is added.

Steps to Ensure Persistence:

1. **Save State After Adding an Item:** The [saveState \(src/todo/reducer.js\)](#) function is already being called after every action in your reducer. This ensures

that the updated state is saved to [localStorage](#) ([node_modules/@types/node/globals.d.ts](#)).

2. **Load State on Initialization:** The [loadState](#) ([src/todo/reducer.js](#)) function is already being used to initialize the state from [localStorage](#) ([node_modules/@types/node/globals.d.ts](#)). This ensures that the state is restored when the app is refreshed.

Updated Reducer Code:

Your current reducer already handles persistence correctly. Here's how the [ADD_ITEM](#) ([src/todo/constants.js](#)) case works with persistence:

```
File: src/todo/reducer.js
case ADD_ITEM:
  newState = state
    .concat({ id: generateUniqueID(), title: action.payload.title,
completed: false })
    .sort((a, b) => a.title.localeCompare(b.title)); // Sort
alphabetically by title
  saveState(newState); // Save the updated state to localStorage
  break;
```

How It Works:

1. Adding a New Item:

- When a new item is added, the [ADD_ITEM](#) ([src/todo/constants.js](#)) case updates the [state](#) ([src/todo/reducer.js](#)) and sorts it alphabetically.
- The [saveState\(newState\)](#) ([src/todo/reducer.js](#)) function saves the updated state to [localStorage](#) ([node_modules/@types/node/globals.d.ts](#)).

2. Loading State on Refresh:

- When the app is loaded, the [initialState](#) ([src/todo/reducer.js](#)) is populated using the [loadState](#) ([src/todo/reducer.js](#)) function, which retrieves the state from [localStorage](#) ([node_modules/@types/node/globals.d.ts](#)).

Full Reducer Code (for Reference):

```
export const todoReducer = (state, action) => {
  let newState;
  switch (action.type) {
    case ADD_ITEM:
      newState = state
        .concat({ id: generateUniqueID(), title:
action.payload.title, completed: false })
        .sort((a, b) => a.title.localeCompare(b.title)); //
Sort alphabetically by title
      saveState(newState); // Save the updated state to
localStorage
      break;
    case UPDATE_ITEM:
      newState = state.map((todo) => (todo.id ===
action.payload.id ? { ...todo, title: action.payload.title } : todo));
      saveState(newState);
      break;
    case REMOVE_ITEM:
      newState = state.filter((todo) => todo.id !==
action.payload.id);
```

```

        saveState(newState);
        break;
    case TOGGLE_ITEM:
        newState = state.map((todo) => (todo.id ===
action.payload.id ? { ...todo, completed: !todo.completed } : todo));
        saveState(newState);
        break;
    case REMOVE_ALL_ITEMS:
        newState = [];
        saveState(newState);
        break;
    case TOGGLE_ALL:
        newState = state.map((todo) => (todo.completed !==
action.payload.completed ? { ...todo, completed:
action.payload.completed } : todo));
        saveState(newState);
        break;
    case REMOVE_COMPLETED_ITEMS:
        newState = state.filter((todo) => !todo.completed);
        saveState(newState);
        break;
    case LOOD_STATE:
        newState = action.payload;
        saveState(newState);
        break;
    default:
        throw Error(`Unknown action: ${action.type}`);
}

```

```
    return newState;
};
```

Summary:

Your current setup already ensures persistence using [localStorage](#) ([node_modules/@types/node/globals.d.ts](#)). The [saveState\(src/todo/reducer.js\)](#) function saves the state after every action, and the [loadState\(src/todo/reducer.js\)](#) function initializes the state from [localStorage](#) ([node_modules/@types/node/globals.d.ts](#)) when the app loads. This means the input will remain persistent even after a refresh. Let me know if you encounter any issues!

> Response rendering

Researcher

So what do you think of this?

Participant

I didn't need this.

I didn't.

Because I think I specifically selected this.

So I think I didn't ask for all the file.

Let's see.

That's it.

Try.

So.

Now it doesn't work.

That's just.

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

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

Participant

It doesn't work.

That's all that needs to be done so.

Chat (window: 0)

> Agent

github.copilot.default

> Context

selection (src/todo/reducer.js)

> Message

that did not work

> Response

If the persistence is not working, the issue might be with how the state is being saved or loaded from [localStorage](#) ([node_modules/@types/node/globals.d.ts](#)). Let's debug and ensure everything is set up correctly.

Steps to Fix Persistence:

1. **Check [saveState\(src/todo/reducer.js\)](#) Function:** Ensure that the [saveState\(src/todo/reducer.js\)](#) function is saving the state correctly to [localStorage](#) ([node_modules/@types/node/globals.d.ts](#)). The key used in [localStorage](#) ([node_modules/@types/node/globals.d.ts](#)) should match the one used in [loadState\(src/todo/reducer.js\)](#).

Update the [saveState\(src/todo/reducer.js\)](#) function to use the correct key:

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

```
Ensure the key matches "todoS"
};
```

2. **Check [loadState \(src/todo/reducer.js\)](#) Function:** Ensure the [loadState \(src/todo/reducer.js\)](#) function is retrieving the state correctly from [localStorage \(node_modules/@types/node/globals.d.ts\)](#). The key used here should also match.

Update the [loadState \(src/todo/reducer.js\)](#) function:

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

3. **Ensure [initialState \(src/todo/reducer.js\)](#) is Used:** Verify that the [initialState \(src/todo/reducer.js\)](#) is being passed to the useReducer hook or wherever the reducer is initialized. For example:

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

4. **Debugging Steps:** Add console.log statements to verify that the state is being saved and loaded correctly:

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

export const loadState = () => {
  const state = localStorage.getItem("todoS");
  console.log("Loading state:", state);
  return state ? JSON.parse(state) : [];
};
```

5. **Verify the Reducer:** Ensure the [ADD_ITEM \(src/todo/constants.js\)](#) case is saving the updated state after sorting:

```
case ADD_ITEM:
  newState = state
    .concat({ id: generateUniqueID(), title:
      action.payload.title, completed: false })
    .sort((a, b) => a.title.localeCompare(b.title)); // Sort
  alphabetically by title
  saveState(newState); // Save the updated state
  break;
```

6. **Clear [localStorage \(src/todo/reducer.js\)](#) for Testing:** If [localStorage \(src/todo/reducer.js\)](#) contains invalid or outdated data, clear it and test again:

```
localStorage.clear();
```

Final Code for [saveState \(src/todo/reducer.js\)](#) and [loadState \(src/todo/reducer.js\)](#):

```
export const loadState = () => {
  const state = localStorage.getItem("todoS");
  console.log("Loading state:", state);
  return state ? JSON.parse(state) : [];
};

const saveState = (state) => {
```

```
    console.log("Saving state:", state);  
    localStorage.setItem("todoS", JSON.stringify(state));  
};
```

Testing:

1. Add a new todo item.
2. Refresh the page.
3. Verify that the todo item persists.

If this still doesn't work, let me know, and we can debug further!

> Response rendering

Researcher

What are you thinking now?

Participant

I mean it seems this I already had this thing before and it didn't work.

And.

Is this this is the same thing?

I'm so confused right now.

Researcher

What are you confused about?

Participant

So it's telling me to check that.

The.

Items are sent correctly.

These two are called differently.

For some reason.

Anything else?

Let me just check something else if it's.

OK.

Let's try.

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

Participant

This first.

OK.

I don't understand why this was a bigger s

I hope this isn't a problem later.

I don't think it will be.

00:40:58
- 00:41:12

00:41:45

00:41:47

00:41:58

00:42:00

00:42:20

00:42:26

00:42:30

00:42:34

00:42:36

00:42:37

00:42:41

00:42:46

00:42:47

00:43:02

00:43:02

00:43:03

00:43:07

00:43:13

00:43:15

00:43:18

00:43:19

00:43:22 Hopefully.
00:43:25 Is this also different? Oh my God.
00:43:56 > **Viewing file** (src/todo/constants.js) through Copilot link
00:43:56 **Researcher**
Do you thinking now?
00:43:59 > **Viewing file** (src/todo/components/input.jsx)
00:44:00 **Participant**
I am going back and forth with what it says and.
00:44:03 > **Viewing file** (src/todo/components/item.jsx)
00:44:07 **Participant**
Trying to.
00:44:09 Pinpoint if there's any bugs.
00:44:11 > **Viewing file** (src/todo/reducer.js)
00:44:24 **Participant**
I.
00:44:25 I don't want to give it to give the task.
00:44:29 The full task for him, for it to do alone.
00:44:37 **Researcher**
And why do you not want this?
00:44:43 **Participant**
It seems like too much.
00:44:57 This s is, it's annoying me so much.
00:45:07 > **Editing file** (src/todo/reducer.js)
- 00:45:11
00:45:28 **Participant**
OK.
00:45:30 Umm.
00:45:39 > **Viewing file** (src/todo/constants.js) through Copilot link
00:46:18 > **Viewing file** (src/todo/components/header.jsx)
00:46:21 > **Viewing file** (src/todo/reducer.js)
00:46:21 **Participant**
Umm.
00:46:22 **Chat** (window: 0)
> **Agent**
github.copilot.workspace
> **Context**
workspace, viewport (src/todo/reducer.js)
> **Message**
@workspace what and where do i have to modify my code so that the input is persistent
> **Response**
To make the input persistent, you need to ensure that the state of your todos is saved to and loaded from [localStorage \(node_modules/@types/node/globals.d.ts\)](#). Your current implementation already includes functions for saving ([saveState \(src/todo/reducer.js\)](#)) and loading ([loadState \(src/todo/reducer.js\)](#)) the state. However, you need to ensure that the [initialState \(src/todo/reducer.js\)](#) is used when initializing the reducer and that the state is saved whenever it changes.

Here are the steps to modify your code:

1. Ensure [initialState \(src/todo/reducer.js\)](#) is used in the reducer initialization

In [src/todo/app.jsx](#), pass [initialState \(src/todo/reducer.js\)](#) to the [useReducer \(node_modules/@types/react/index.d.ts\)](#) hook:

```

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

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

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

```

2. Save state after every action

Your reducer in [src/todo/reducer.js](#) already calls [saveState\(newState\)](#) ([src/todo/reducer.js](#)) after every action. This ensures that any changes to the todos are saved to [localStorage\(node_modules/@types/node/globals.d.ts\)](#).

3. Verify [loadState\(src/todo/reducer.js\)](#) works correctly

The [loadState\(src/todo/reducer.js\)](#) function in [src/todo/reducer.js](#) already retrieves the todos from [localStorage\(node_modules/@types/node/globals.d.ts\)](#). Ensure that the key "todoS" matches the one used in [saveState\(src/todo/reducer.js\)](#).

4. Test the persistence

Run your app, add some todos, and refresh the page. The todos should persist because they are saved in [localStorage\(node_modules/@types/node/globals.d.ts\)](#) and loaded when the app initializes.

00:46:33

Participant

Yeah, I'm, I know. I'm just.

00:46:34
- 00:46:43

> Response rendering

00:46:36

Participant

Confused how to follow his instructions.

00:46:41

So hopefully now I will have better steps.

00:46:58

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

00:47:13

Participant

I guess there's an import.

00:47:14

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

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

00:47:18
- 00:47:24

00:47:19

Participant

Oh, it's missing from him.

00:47:28

And.

00:47:30
- 00:47:34

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

00:47:49

Participant

Let's try now.

00:47:52

See.

00:47:54

OK.

00:47:54

Now it's working.

00:47:56

I just now I want to check if it said anything about.

00:48:03

The app.

00:48:08

The app file and I don't see.

00:48:13

Any anything in the last three response so.

00:48:19

OK.

00:48:24

A bit ambiguous.

00:48:27

That I didn't get something back there.

00:48:38

Did he work?

00:48:42

Yeah, I think.

00:48:44

I think the task this one is good, yeah.

00:48:48

Researcher

Great. Thanks a lot.

00:51:55

Participant

I think he was OK for a first try, for a first try.

00:51:59

I think I did OK I think.

00:53:05

Researcher

Great. Thanks. Yeah.

00:53:06

Participant

Yeah.

00:53:08

Yeah, no problem.

00:53:10

Researcher

So to start with with the questions.

00:53:12

Participant

Mm hmm.

00:53:14

Researcher

So in the survey you talked about or you you pressed some ratings about aspects of Copilot responses that you liked or disliked.

00:53:19

00:53:23

Participant

Yeah.

00:53:23

Researcher

Did any of these particularly stand out to you?

00:53:28

Participant

Umm.

00:53:31

Researcher

If you want, you can refresh this window to see them again.

00:53:33

00:53:36

Participant

I saw something with the length and detail. Yeah, yeah.

00:53:42

I mean, they were correct, which also surprised me a bit.

00:53:47

Because yeah, I'm only using ChatGPT mostly.

00:53:52

So that was a nice.

00:53:54

Nice detail.

00:53:57

I think it was a bit too detailed.

00:54:00

The tone is nice.

00:54:01

Formatting it was good.

00:54:04 Technical terms were not that difficult to understand.
00:54:09 The length was unnecessary a couple of times.
00:54:15 Like repeat.
00:54:15 **Researcher**
Too long for you?
00:54:18 **Participant**
Yeah.
00:54:20 Like, I wouldn't say that necessarily was when
00:54:24 it was repeating the same code that was.
00:54:29 Managed to be modified like it was announced with this is how you modify and
00:54:35 then two lines later it was final code and it was the same.
00:54:42 **Researcher**
Yeah.
00:54:42 **Participant**
Yeah.
00:54:44 **Researcher**
And and apart from that from that instance was the length OK or also
00:54:50 sometimes too long or too short?
00:54:55 **Participant**
I think it works to make it a long answer, when sometimes you don't need a long
00:55:02 answer.
00:55:03 **Researcher**
OK.
00:55:08 OK, great.
00:55:12 Also, I saw you ranked following your
00:55:14 instructions a bit lower than the rest.
00:55:17 Do you have a specific reason for this?
00:55:21 **Participant**
Following instructions from Copilot.
00:55:24 **Researcher**
Yes.
00:55:25 **Participant**
Yeah.
00:55:28 At the last task I was a bit disappointed because it did not mention anything about
00:55:37 modifying the main app file.
00:55:41 Which I think was important to my instructions so.
00:55:48 Yeah.
00:55:48 **Researcher**
Yeah. I I I think when you asked another question.
00:55:52 It did talk about the app, but you would have wanted it to talk
00:55:53 **Participant**
Yes, yes.
00:55:56 **Researcher**
about this in the first question as well then.
00:55:58 **Participant**
In the previous one, yeah.
00:56:03 **Researcher**
And did this frustrate you?
00:56:07 Or what made make you think about this?
00:56:10 **Participant**
I was disappointed because I was starting to gain a bit of trust in Copilot and so
00:56:17 having.
00:56:20 To persist with that to with the question to see if anything was missing.
00:56:26 That's a bit sad.
00:56:30 Is I trusted that.
00:56:31 There wasn't anything more to do from the previous previous question because it

00:56:36 maybe it was also the tone because it was the tone is meant to be.
00:56:42 Umm.
00:56:43 Credible, that it knows what it's doing.
00:56:47 So having to ask again just to make sure that nothing is missing and only then get
00:56:53 the result and.
00:56:55 Yeah, bit disappointing.
00:56:55 **Researcher**
OK.
00:56:57 And suppose in this first answer, Copilot said, oh,
00:57:02 my answer may not be complete or correct.
00:57:07 Would this make a difference for you or not really?
00:57:18 **Participant**
Maybe it would make me question again his actions.
00:57:24 But if it gives me a good answer from the start,
00:57:26 then there's no need to persist with the question.
00:57:30 **Researcher**
Yeah.
00:57:32 So you think adding a disclaimer like this will make make it worse actually.
00:57:38 **Participant**
Yeah, yeah, actually, yeah. Because you kind of want to have the
00:57:40 **Researcher**
Yeah.
00:57:44 **Participant**
trust, like you have someone.
00:57:47 Well, not someone.
00:57:48 Something to work with that can help you build upon you would like it to be.
00:57:55 Confident in the answer it gives.
00:57:57 **Researcher**
OK.
00:57:58 Yeah, that makes sense actually.
00:58:01 Great.
00:58:04 So could you find your way through the project and the code?
00:58:07 And did Copilot help you find your way through the code,
00:58:11 or did it actually hinder your navigation?
00:58:17 **Participant**
No, I don't think so.
00:58:20 I think it was just something new I was getting used to because yeah,
00:58:25 I'm not allowed to use it for my course.
00:58:28 So I think it's a nice addition next to all that I have already.
00:58:36 So I I wouldn't say hindering.
00:58:38 No, not at all.
00:58:38 **Researcher**
OK.
00:58:40 Would you say it helped you?
00:58:41 **Participant**
Uh.
00:58:43 Yeah, definitely.
00:58:44 Because I've never seen this app before the code,
00:58:50 so getting used to it took a bit. I have to agree.
00:58:55 **Researcher**
Mm hmm.
00:58:57 **Participant**
Because.
00:59:00 **Researcher**
Did you feel like you?
00:59:01 **Participant**

00:59:02 I'm not.
Researcher
Did you feel like you understood the code well?

00:59:05 **Participant**
Yeah, yeah. Even though I'm not that big of an expert,
00:59:10 you know?
00:59:12 I can navigate through through this I can.
00:59:17 Get what the code is meant to do, but it's nice to have this like you can
00:59:22 click here and it sends you directly to the function in this case.
00:59:28 So I think it's easier to look stuff up inside the workspace,
00:59:33 even if especially when you're new.
00:59:38 **Researcher**
So. So you feel like Copilot also helped you
00:59:42 understand the code better then?
00:59:45 **Participant**
Yeah.
00:59:48 Yeah, definitely.
00:59:48 **Researcher**
OK.
00:59:52 And do you have any idea how you would have solved?
00:59:56 These tasks without using Copilots?
01:00:01 **Participant**
And instead using another compiler, another.
01:00:07 Like another AI?
01:00:07 **Researcher**
Yeah, so say you.
01:00:11 You could do.
01:00:12 You could solve this task anyway you liked.
01:00:16 How would you have done it?
01:00:19 **Participant**
Well, it's.
01:00:22 Either ChatGPT, where I have to.
01:00:27 Copy code, paste code and trial and error until I
01:00:30 find a good answer or search Google on Stack Overflow or something else.
01:00:38 In the end, it's online resources.
01:00:38 **Researcher**
Yeah.
01:00:40 **Participant**
Just get the job done.
01:00:43 **Researcher**
It makes sense.
01:00:45 And in that case.
01:00:49 How would you have navigated through the code?
01:00:52 For example when you need to change?
01:00:55 The color for this text box I saw now you used Copilot to to find the the right
01:00:57 **Participant**
Oh.
01:01:00 **Researcher**
place in the code.
01:01:01 How would you ever approach this without Copilot?
01:01:04 **Participant**
Yeah, that would have taken a while.
01:01:11 Like I didn't even know there was.
01:01:12 We were working with the CSS.
01:01:17 I use tailwind for example.
01:01:23 So.

01:01:26 It would have taken a bit just to see all the components and their tags and then
01:01:34 see the tags in CSS specifically to change the the color here.
01:01:42 **Researcher**
So which you have gone through the the files manually or how do you think you
01:01:47 might have possibly approached it then?
01:01:47 **Participant**
Yeah.
01:01:50 I.
01:01:50 I think so.
01:01:50 I mean, there's also the inspect here to see each
01:01:55 **Researcher**
Mm hmm.
01:01:56 **Participant**
thing and then look for it here but.
01:02:02 Yeah, it would have been.
01:02:05 Umm.
01:02:06 Back and forth to see how the web page looks and then see the code and why it
01:02:11 looks like this.
01:02:14 **Researcher**
Yeah, that makes sense.
01:02:17 OK.
01:02:20 So.
01:02:23 Did you?
01:02:24 Sort of change the way you asked your questions after seeing things that
01:02:28 Copilot was good at or not so good at?
01:02:32 **Participant**
I think I changed just a bit 'cause I saw it was reacting well.
01:02:39 Like giving good answers. So I start giving so many details.
01:02:42 I can also select the code and just.
01:02:48 Talk about the code and what I have. I want to do with the code so.
01:02:53 It was mostly help me do this task and not necessarily help me do this task
01:02:59 given this stuff and stuff.
01:03:02 And other code.
01:03:05 **Researcher**
So if I understand correctly, you should have put less details into
01:03:10 your questions.
01:03:14 **Participant**
You can do that.
01:03:15 I started not to give as much details because I saw it was not needed.
01:03:22 **Researcher**
OK.
01:03:22 **Participant**
He was answering quite well without this many details.
01:03:28 **Researcher**
Yeah, that makes sense.
01:03:30 Also at the start you mentioned that you think you might be more mean to Copilot
01:03:36 after a while.
01:03:39 Do you still have this impression now after using it for a bit?
01:03:43 **Participant**
Umm.
01:03:48 Like I was trying to be as nice as possible with giving as many details as
01:03:54 possible.
01:03:55 So if me going from nice with details and mean without details.
01:04:02 I wouldn't say I was mean.
01:04:04 Maybe a bit more.
01:04:10 I say that did not work.

01:04:14 I didn't say what went wrong in his.
01:04:18 Response I just mentioned, it did not work.
01:04:22 Maybe that's why I didn't give.
01:04:26 My answer I was looking for here.
01:04:29 Perhaps I don't know.
01:04:30 **Researcher**
Yeah, who knows?
01:04:33 All right.