

Participant P15

00:04:59 > **Viewing file** (src/index.js)
00:05:09 > **Viewing file** (src/todo/app.css)
00:05:29 > **Viewing file** (src/todo/components/main.jsx)
00:05:47 **Researcher**
You can start, after 30 minutes I'll ask you to stop.
00:05:58 **Participant**
OK. The task is to change the color of a red box.
00:06:08 For now, I'll look for where the message is.
00:06:10 Found the message, the sentence is here.
00:06:27 > **Viewing file** (src/todo/app.css)
00:06:30 > **Viewing file** (src/todo/components/main.jsx)
00:06:35 > **Viewing file** (src/todo/app.css)
00:06:45 **Participant**
Done.
00:06:45 So here is where I have to change.
00:06:55 > **Editing file** (src/todo/app.css)
00:07:02 > **Viewing file** (src/todo/components/main.jsx)
00:07:14 > **Viewing file** (src/todo/app.css)
00:07:28 > **Viewing file** (src/todo/components/main.jsx)
00:07:30 **Participant**
But the case would be to start the application first.
00:07:33 > **Viewing file** (src/todo/app.css)
00:07:34 **Participant**
I think I'll undo the first change and test first.
00:07:40 Just confirm.
00:07:50 > **Viewing file** (src/todo/components/main.jsx)
00:07:58 **Participant**
For now, I think I'll ask how I can run the project?
00:08:05 **Researcher**
OK.
00:08:16 **Chat** (window: 0)
> **Agent**
github.copilot.workspace
> **Context**
workspace, selection (src/todo/components/main.jsx)
> **Message**
@workspace hello, analyze the project and tell me how I can run it
> **Response**
To run the project, follow the steps below:

1. **Install the dependencies:** Make sure you have Node.js ($\geq 18.13.0$) and npm ($\geq 8.19.3$) installed. In the integrated terminal of Visual Studio Code, run the command:

npm install

2. **Start the development server:** After installing the dependencies, start the development server with the command:

npm run dev

3. **Access the application:** The server will automatically open the default browser pointing to the local address (usually `http://localhost:8080`). If it doesn't open, you can access it manually.

These steps use the Webpack Dev Server configured in the [webpack.dev.js](#) file. It serves the files from the `dist` folder and applies hot-reloading to facilitate development.

00:08:56

Participant

It's in blue.

00:09:42

I'll look for where this text is.

00:10:07

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

00:10:11

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

00:10:33

Participant

Now I have to look for where this
div is

00:10:37

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

00:10:39

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

00:11:34

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

00:11:51

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

00:13:21

00:13:53

Researcher

You can use the copilot at will.

00:15:11

Chat (window: 0)

> **Agent**

github.copilot.default

> **Context**

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

> **Message**

In the current file, how is `<ul todo-list/>` organized when it receives user input?

> **Response**

In the current file, the `<ul className="todo-list" />` component is dynamically organized based on the task list ([todos](#) (`src/todo/components/main.jsx`)) filtered and stored in [visibleTodos](#) (`src/todo/components/main.jsx`). When the user adds a new task, it is automatically included in the rendered list because the [todos](#) (`src/todo/components/main.jsx`) state is updated, and React re-renders the component.

Here is the flow of how the list is organized:

1. Task Filtering:

- o The [todos](#) (`src/todo/components/main.jsx`) list is filtered based on the current route ([route](#) (`src/todo/components/main.jsx`)) using the [useMemo](#) (`node_modules/@types/react/index.d.ts`) hook. This creates

the [visibleTodos \(src/todo/components/main.jsx\)](#) list, which contains only the relevant tasks (active, completed, or all).

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

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

      return todo;
    }),
  [todos, route]
);
```

2. List Rendering:

- The `<ul className="todo-list" />` component maps the items from [visibleTodos \(src/todo/components/main.jsx\)](#) and renders an `<Item />` component for each task. Each item receives the properties [todo \(src/todo/components/main.jsx\)](#), [dispatch \(src/todo/components/main.jsx\)](#), and [index \(src/todo/components/main.jsx\)](#).

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

3. Dynamic Update:

- When the user adds a new task, the [todos \(src/todo/components/main.jsx\)](#) state is updated (probably through a [dispatch \(src/todo/components/main.jsx\)](#) in another component). This triggers the reevaluation of [visibleTodos \(src/todo/components/main.jsx\)](#) and consequently the re-rendering of the list.

4. Additional Sorting or Organization:

- Currently, the code does not implement any explicit sorting logic (such as by date or priority). The order of items in the list follows the order in which they appear in the [todos \(src/todo/components/main.jsx\)](#) array.

If you want to implement specific sorting (e.g., by creation date), you can add additional logic to the [useMemo \(node_modules/@types/react/index.d.ts\)](#) that defines [visibleTodos \(src/todo/components/main.jsx\)](#).

00:15:45

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

00:16:06

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

00:16:12

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

00:16:28

Participant

00:16:31

I think this here.

Save.

00:17:23

Chat (window: 0)

> **Agent**

github.copilot.default

> **Context**

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

> **Message**

but useMemo cannot be altered, right?

> **Response**

Correct, the value returned by [useMemo \(node_modules/@types/react/index.d.ts\)](#) cannot be directly altered, as it is just a way to memorize the result of a computed function based on the provided dependencies. In the current code, [useMemo \(node_modules/@types/react/index.d.ts\)](#) is being used to calculate and memorize the list [visibleTodos \(src/todo/components/main.jsx\)](#) based on [todos \(src/todo/components/main.jsx\)](#) and [route \(src/todo/components/main.jsx\)](#).

If you want to change the visible task list, you will need to change the state [todos \(src/todo/components/main.jsx\)](#) (or the route [route \(src/todo/components/main.jsx\)](#)), as [useMemo \(node_modules/@types/react/index.d.ts\)](#) will automatically recalculate the value of [visibleTodos \(src/todo/components/main.jsx\)](#) when any of these dependencies change.

For example: - To add or remove tasks, you must use [dispatch \(src/todo/components/main.jsx\)](#) to update the global state or the state that controls [todos \(src/todo/components/main.jsx\)](#). - To change the filtering (show active, completed, or all tasks), you must change the route ([route \(src/todo/components/main.jsx\)](#)), which will be automatically reflected in [visibleTodos \(src/todo/components/main.jsx\)](#).

The [useMemo \(node_modules/@types/react/index.d.ts\)](#) is just an optimization to avoid recalculating the filtered list on each render, but it does not allow direct modifications to the returned value.

00:18:04

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

00:18:08

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

00:18:11

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

00:18:36

Participant

The other task asks to sort the to-dos.

00:18:48

Everything is fine.

00:18:48

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

00:18:51

Participant

I forgot how to make a comment.

00:19:36

Chat (window: 0)

> **Agent**

github.copilot.default

> **Context**

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

> **Message**

The "todo-list" ul component seems to use a .map() method. Why does it use it?

> **Response**

The `<ul className="todo-list" />` component uses the `.map()` method to iterate over the filtered task list ([visibleTodos \(src/todo/components/main.jsx\)](#)) and dynamically render an `<Item />` component for each task. The `.map()` method is used because it allows transforming each element of an array into a new element, which in this case is a React component.

Why use .map() here?

1. Dynamic Rendering:

- The .map() allows creating a list of <Item /> components based on the data from [visibleTodos \(src/todo/components/main.jsx\)](#). Each task is transformed into an <Item /> component that is rendered inside the .

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

2. Unique Identification with [key \(node_modules/@types/react/index.d.ts\)](#):

- The .map() allows adding a unique [key \(node_modules/@types/react/index.d.ts\)](#) to each <Item /> component (in this case, [todo.id \(src/todo/components/main.jsx\)](#)). This helps React identify which items have changed, been added, or removed, optimizing the re-rendering process.

3. Separation of Responsibilities:

- Each task ([todo \(src/todo/components/main.jsx\)](#)) is passed as a property (prop) to the <Item /> component, which is responsible for rendering the task details. This keeps the code modular and easy to maintain.

4. Scalability:

- Using .map() makes the code scalable. No matter how many tasks exist in [visibleTodos \(src/todo/components/main.jsx\)](#), the method will dynamically create the necessary components.

Example of how it works:

If [visibleTodos \(src/todo/components/main.jsx\)](#) is:

```
[  
  { id: 1, text: "Buy milk", completed: false },  
  { id: 2, text: "Study React", completed: true }  
]
```

The .map() will generate:

```
<Item todo={{ id: 1, text: "Buy milk", completed: false }} key={1}  
dispatch={dispatch} index={0} />  
<Item todo={{ id: 2, text: "Study React", completed: true }} key={2}  
dispatch={dispatch} index={1} />
```

These components will be rendered inside the as list items.

00:20:18

Participant

So here I really can't change,

00:20:22

I'll have to change here.

00:21:33

Chat (window: 0)

> Agent

github.copilot.default

> Context

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

> Message

In the current context, can useMemo receive more arguments? For example, to sort the entries.

> Response

Yes, the [useMemo \(node_modules/@types/react/index.d.ts\)](#) can be adjusted to include additional logic, such as sorting the entries. You can pass more arguments to the function within the [useMemo \(node_modules/@types/react/index.d.ts\)](#) and modify the logic to sort the items before returning them.

For example, to sort tasks by a field like text or [id \(node_modules/@types/react/index.d.ts\)](#), you can do the following:

Example with sorting:

```
const visibleTodos = useMemo(
  () =>
    todos
      .filter((todo) => {
        if (route === "/active") return !todo.completed;
        if (route === "/completed") return todo.completed;
        return todo;
      })
      .sort((a, b) => a.text.localeCompare(b.text)), // Sorts
  [todos, route]
);
```

alphabetically by the "text" field

Explanation:

1. Filtering:

- The filtering logic remains the same, based on the route ([route \(src/todo/components/main.jsx\)](#)).

2. Sorting:

- After filtering, the `.sort()` method is used to sort the items. In the example above, the items are sorted alphabetically by the text field using `localeCompare`.

3. Dependencies:

- The [useMemo \(node_modules/@types/react/index.d.ts\)](#) will be recalculated whenever [todos \(src/todo/components/main.jsx\)](#) or [route \(src/todo/components/main.jsx\)](#) change, ensuring that the filtered and sorted list is always up to date.

Sorting by another criterion:

If you want to sort by another criterion, such as [id \(node_modules/@types/react/index.d.ts\)](#) or `creationDate`, just adjust the comparison function in `.sort()`:

```
.sort((a, b) => a.id - b.id) // Sorts by ID in ascending order
```

Consideration:

Adding sorting directly in the [useMemo \(node_modules/@types/react/index.d.ts\)](#) is efficient because it avoids recalculating the filtered and sorted list on each render, unless the dependencies ([todos \(src/todo/components/main.jsx\)](#) or [route \(src/todo/components/main.jsx\)](#)) change.

00:22:08 > **Implementing Copilot code** by copying part of snippet
(src/todo/components/main.jsx)

00:22:37 **Participant**
But if I put it after filtering, it does it again.

00:22:43 Ah, okay, it calculates.

00:23:20 > **Viewing file** (src/todo/components/main.jsx)

00:23:33 > **Editing file** (src/todo/components/main.jsx)

00:23:59 **Participant**
I think the sort doesn't work.

00:24:04 Because it doesn't work, I can't see if it's adding.

00:24:15 Got it, there's a class.

00:24:49 > **Viewing file** (src/todo/components/main.jsx)

00:24:57 **Participant**
It works when there's more than one item.

00:25:04 It doesn't work here.

00:25:21 **Chat** (window: 0)

> **Agent**
github.copilot.default

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

> **Message**
the sort method seems not to work as expected

> **Response**
If the `.sort()` method is not working as expected, there may be several reasons for this. Here are some possible causes and how to fix them:

1. Field [text \(src/todo/components/main.jsx\)](#) may be undefined or not exist

If any object in the list [todos \(src/todo/components/main.jsx\)](#) does not have the field [text \(src/todo/components/main.jsx\)](#), the `.localeCompare` method may fail or not sort correctly.

Solution:

Ensure that all objects in [todos \(src/todo/components/main.jsx\)](#) have the field [text \(src/todo/components/main.jsx\)](#). Otherwise, provide a default value to avoid errors:

```
.sort((a, b) => (a.text || '').localeCompare(b.text || ''))
```

2. The list [todos \(src/todo/components/main.jsx\)](#) may be mutated elsewhere

If the list [todos \(src/todo/components/main.jsx\)](#) is being directly altered elsewhere in the code, this may interfere with sorting.

Solution:

Ensure that [todos \(src/todo/components/main.jsx\)](#) is immutable and that you are not directly altering the original array. Use methods like [map \(src/todo/components/main.jsx\)](#), [filter \(src/todo/components/main.jsx\)](#), or [concat](#) to create new arrays instead of modifying the original.

3. Case-sensitive sorting

The `.localeCompare` method is case-sensitive by default, which may cause unexpected sorting.

Solution:

Use the option `{ sensitivity: "base" }` in `.localeCompare` to ignore case differences:

```
.sort((a, b) => a.text.localeCompare(b.text, undefined, { sensitivity: "base" })))
```

4. Dependencies of [useMemo](#)

[\(node_modules/@types/react/index.d.ts\)](#) may be incomplete

If [useMemo](#) ([node_modules/@types/react/index.d.ts](#)) is not being recalculated correctly, it may be because the dependencies are incomplete or incorrect.

Solution:

Ensure that all relevant dependencies are included in the dependency array of [useMemo](#) ([node_modules/@types/react/index.d.ts](#)). In the current case, [todos](#) ([src/todo/components/main.jsx](#)) and [route](#) ([src/todo/components/main.jsx](#)) are already included, so this seems correct.

5. Data type issue

If the values of [text](#) ([src/todo/components/main.jsx](#)) are not strings (e.g., numbers or objects), `.localeCompare` will not work correctly.

Solution:

Ensure that [text](#) ([src/todo/components/main.jsx](#)) is always a string. Otherwise, convert it before sorting:

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

Updated code with improvements:

Here is an updated version of the snippet with the mentioned fixes:

```
const visibleTodos = useMemo(
  () =>
    todos
      .filter((todo) => {
        if (route === "/active") return !todo.completed;
        if (route === "/completed") return todo.completed;
        return todo;
      })
      .sort((a, b) => (a.text || "").localeCompare(b.text || "",
undefined, { sensitivity: "base" })),
  [todos, route]
);
```

Testing and debugging:

1. Check the content of [todos \(src/todo/components/main.jsx\)](#) in the console to ensure the data is correct:

```
console.log(todos);
```

2. Ensure that the field [text \(src/todo/components/main.jsx\)](#) exists and is consistent across all objects.
3. Test the sorting with different values to confirm it is working as expected.

If the problem persists, there may be another factor in the code that is interfering.

00:25:59 > **Implementing Copilot code** by copying part of snippet
(src/todo/components/main.jsx)

00:27:32 **Participant**
It's not sorting.

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

00:28:02 **Participant**
The truth is, yes, now it's right.

00:28:43 > **Implementing Copilot code** by copying part of snippet
(src/todo/components/main.jsx)

00:29:15 > **Viewing file** (src/todo/components/main.jsx)

00:29:18 > **Editing file** (src/todo/components/main.jsx)

00:29:51 > **Viewing file** (src/todo/components/main.jsx)

00:31:05

Chat (window: 0)

> Agent

github.copilot.default

> Context

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

> Message

In the current context, would the entire field in `.map()` be the user's text?

> Response

In the current context, the entire field `todo (src/todo/components/main.jsx)` in `.map()` represents an object from the list `visibleTodos (src/todo/components/main.jsx)`. This object contains information about each task, including the text entered by the user (probably stored in the field `text (src/todo/components/main.jsx)`), as well as other data like `id (node_modules/@types/react/index.d.ts)` and `completed (src/todo/components/main.jsx)`.

In the snippet below:

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

The `todo (src/todo/components/main.jsx)` is an individual object from the list `visibleTodos (src/todo/components/main.jsx)`, which was previously filtered and sorted in `useMemo (node_modules/@types/react/index.d.ts)`. If the user's text is stored in the field `text (src/todo/components/main.jsx)` of each `todo (src/todo/components/main.jsx)` object, then yes, the user's text will be accessible as `todo.text (src/todo/components/main.jsx)`.

For example, if `todos (src/todo/components/main.jsx)` is:

```
[  
  { id: 1, text: "Buy milk", completed: false },  
  { id: 2, text: "Study React", completed: true }  
]
```

After filtering and sorting, `visibleTodos (src/todo/components/main.jsx)` might contain something like:

```
[  
  { id: 1, text: "Buy milk", completed: false },  
  { id: 2, text: "Study React", completed: true }  
]
```

And in the `.map()`, each `todo (src/todo/components/main.jsx)` will be one of these objects, and the user's text will be accessible as `todo.text (src/todo/components/main.jsx)`. The `<Item />` component will receive this complete object as a prop and will be responsible for rendering the text or other information.

00:41:44

Researcher

Let's end the task part now. It's been 40 minutes. Yeah,

00:43:09

Alright, it's ending, I'll just ask a few more questions.

00:43:15

Participant

OK.

00:43:16

Researcher

Were you able to orient yourself with the code and how did the copilot influence you in this

00:43:29

Participant

Well, I managed. Yes, I think I oriented myself a bit in the code, not as much as I wanted, but I managed a bit I think

00:43:33

00:43:37 I think the copilot helped me by showing me the direction
00:43:41 **Researcher**
OK
00:43:44 Cool.
00:43:46 Another question, do you feel you understood the basis of the
00:43:51 code and how did the copilot influence your understanding?
00:44:01 **Participant**
Well, you saw there, right?
00:44:03 I. I think I didn't understand that much of the
00:44:04 **Researcher**
Uhum.
00:44:06 **Participant**
code, but what I managed was a lot because of
00:44:09 the copilot.
00:44:11 but I didn't trust much in what it said.
00:44:15 **Researcher**
OK
00:44:17 **Participant**
At least, the copilot showed me a direction to follow.
00:44:20 **Researcher**
Cool.
00:44:23 How would you have solved the tasks without using the copilot.
00:44:33 **Participant**
Well, basically I would open all the project files.
00:44:45 But I would take more time to solve the tasks.
00:44:49 I would use code debugging to better understand the process.
00:44:54 **Researcher**
OK
00:44:58 Can you identify and explain the aspects that you,
00:45:05 liked and didn't like about the copilot?
00:45:15 **Participant**
I think the AI's response is as good as your input prompt.
00:45:21 **Researcher**
OK
00:45:24 **Participant**
so it will try to take, as it is generic. Now,
00:45:39 **Researcher**
During the session
00:45:42 did you change the way you interact with the copilot
00:45:49 **Participant**
I think I followed the same line.
00:45:51 **Researcher**
OK
00:45:53 Now I'm going to send you a questionnaire
00:45:54 **Participant**
Or any way I do with any other AI in this case.
00:45:57 **Researcher**
OK, I thank you for participating in our study.
01:19:19 > Response rendering
- 01:19:23
01:26:15 > Response rendering
- 01:26:25
01:28:22 > Response rendering
- 01:28:25
01:30:36 > Response rendering
- 01:30:43
01:32:37 > Response rendering
- 01:32:42

01:36:20
- 01:36:39

> Response rendering

01:42:06
- 01:42:21

> Response rendering