

# Participant P24

00:14:09 **Researcher**  
ok you can start.

00:14:35 **Participant**  
first task, change the color.

00:14:44 nothing to do here

00:14:51 **> Viewing file** (src/todo/components/main.jsx)

00:15:15 **Researcher**  
What are you thinking now?

00:15:19 **Participant**  
First, analyzing the files,  
I imagine it's in the 'Main' this text  
Then I try to see first if it's simple or not,  
if I can do it without needing the copilot's help  
If I can't, I'll ask the AI for help  
I'll make an entry for the AI  
I do it for.  
Change the color of the messages none.  
To do here.

00:17:51 **Chat** (window: 0)  
**> Agent**  
github.copilot.workspace  
**> Context**  
workspace, viewport (src/todo/components/main.jsx)  
**> Message**  
@workspace how do I change the color of the message no tasks here!  
**> Response**  
To change the color of the message "No tasks here yet," you can modify the CSS style associated with the class `.todo-list-empty-message` in the file <src/todo/app.css>. For example, to change the text color to blue, you can do the following:

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

After saving the change, the text "No tasks here yet" will be displayed in blue. You can replace blue with any other color or hexadecimal code of your choice.

00:18:01 **Researcher**  
Ok

00:18:18 **> Viewing file** (src/todo/app.css)

00:18:21 **> Implementing Copilot code** using apply button (src/todo/app.css)

00:18:38 **Participant**  
Blue with this Hexadecimal, let me see.  
That's it.  
I'll see if it went.

00:18:55 Bing, what to apply? Do I have to confirm?

00:19:04 **Researcher**  
You have to confirm, it's always when it asks for a change,  
it will ask you to confirm application.

00:19:10

00:19:29

**Participant**

Understood.

00:19:50  
- 00:20:00

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

00:20:07

**Participant**

But it has the context of this.

00:20:11

Workspace actually, wanted to change the background color.

00:20:16

because I lost the font color.

00:20:20

Let's see if it recovers the font color that was before.

00:20:25

**Researcher**

The font color was white.

00:20:30

**Participant**

OK?

00:20:46

**Researcher**

You can, you can go to the second one.

00:21:02

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

00:21:06

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

00:21:10

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

00:21:20

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

00:21:39

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

00:21:42

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

00:21:48

**Participant**

Let's see, try to search to see if I find it.

00:21:52

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

00:21:55

**Participant**

Here, what needs to be done?

00:22:08  
- 00:22:08

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

00:22:35

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

00:22:37

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

00:22:40

**Participant**

to be sorted in alphabetical order.

00:23:29

To be displayed, in alphabetical order.

00:23:51

**Chat** (window: 0)

> **Agent**

github.copilot.workspace

> **Context**

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

> **Message**

@workspace How do I make my to-do items display in alphabetical order?

> **Response**

To display the to-do items in alphabetical order, you can use the sort method on the array of [todos](#) (src/todo/app.jsx) before filtering them. Modify the calculation of [visibleTodos](#) (src/todo/components/main.jsx) in the [main.jsx file](#) (src/todo/components/main.jsx) to include sorting. Here's how to do it:

File: src/todo/components/main.jsx

```
const visibleTodos = useMemo(
```

```
  () =>
```

```
    todos
```

```
      .slice() // Creates a copy of the array to avoid mutations
```

```
      .sort((a, b) => a.title.localeCompare(b.title)) // Sorts
```

alphabetically by title

```
      .filter((todo) => {
```

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

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

```
        return true;
```

```

    }),
    [todos, route]
  );

```

## Explanation:

1. **slice()**: Creates a copy of the original array to avoid mutations.
  2. **sort()**: Sorts the items based on the title ([title](#) ([src/todo/components/header.jsx](#))) in alphabetical order.
  3. **localeCompare**: Ensures that the sorting works correctly for different languages and special characters.
- Now, the items will be displayed in alphabetical order in the to-do list.

00:24:14

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

00:25:03  
- 00:25:06

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

00:25:36

**Researcher**

For you to test via application. Enter adds a task

00:25:57

**Participant**

OK?

00:26:06

One.

00:26:12

1123, brown.

00:26:16

**Researcher**

Fox, fox, cool, cool that's right

00:26:30

I'll send you the new tasks now

00:26:44

No need to do in order, validate all requirements before moving to the next one ok.

00:27:06

**Participant**

OK.

00:27:37

Task, save, persist.

00:27:42

I imagine it can be done saving in cookies. I remember there is,

00:27:48

if I'm not mistaken, a Youse cookies library,

00:27:52

that can be used here in react I think Youse.

00:28:17

ask for help from the copilot.

00:28:18

**Researcher**

Cool.

00:28:23

**Participant**

How can I save?

00:28:32

My tasks.

00:29:03

**Chat** (window: 0)

> **Agent**

github.copilot.workspace

### > Context

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

### > Message

@workspace How can I save my scheduled activities in a cookie, I want to save all the information of each task

### > Response

To save the scheduled activities in a cookie, you can use the API [document.cookie \(node\\_modules/typescript/lib/lib.dom.d.ts\)](#) to store the data. However, since cookies have size limitations and are not ideal for storing large amounts of data, it is recommended to serialize the data in JSON before saving it.

Here's how you can implement this in the file [main.jsx \(src/todo/components/main.jsx\)](#):

#### 1. Add a function to save the data in the cookie:

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

function saveTodosToCookie(todos) {
  const serializedTodos = JSON.stringify(todos);
  document.cookie = `todos=${encodeURIComponent(serializedTodos)};
path=/; max-age=31536000`; // Expires in 1 year
}
```

#### 2. Use [useEffect \(node\\_modules/@types/react/index.d.ts\)](#) to save the data whenever the [todos \(src/todo/components/main.jsx\)](#) are updated:

```
File: src/todo/components/main.jsx
useEffect(() => {
  saveTodosToCookie(todos);
}, [todos]);
```

#### 3. Add a function to load the data from the cookie on initialization:

In the file [reducer.js \(src/todo/reducer.js\)](#), you can create a function to load the data from the cookie:

```
File: src/todo/reducer.js
export const loadTodosFromCookie = () => {
  const cookies = document.cookie.split("; ").reduce((acc, cookie) => {
    const [key, value] = cookie.split("=");
    acc[key] = decodeURIComponent(value);
    return acc;
  }, {});

  return cookies.todos ? JSON.parse(cookies.todos) : [];
};
```

#### 4. Update the initial state in the [App \(src/todo/app.jsx\)](#) to load the data from the cookie:

In the file [app.jsx \(src/todo/app.jsx\)](#), replace the initial state with the cookie loading:

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

```

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

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

```

## Result:

With these changes, the activities will be automatically saved in a cookie whenever they are updated and loaded from the cookie when the application initializes.

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

### Participant

The bug is to save the data whenever the data is updated

I imagine it won't break anything. Where it will add.

There has to be confirmation.

### Researcher

This has to be confirmed.

### Participant

This here is essential. I will add a file.

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

> **Implementing Copilot code** using apply button (src/todo/reducer.js)

### Researcher

OK

### Participant

Let's test

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

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

### Researcher

OK.

### Participant

I think I applied all the changes

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

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

### Researcher

OK.

Ok, you can move on to another task

### Participant

If the user reloads the page while typing the task

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

### Researcher

00:33:41

If you have any questions about any task you can ask

**Participant**

OK.

00:33:55

I asked the AI how to do this.

00:34:01

How can I save the draft.

00:34:49

**Chat** (window: 0)

**> Agent**

github.copilot.workspace

**> Context**

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

**> Message**

@workspace How do I save the draft of what I am typing as a new task in a cookie, I want to do this every 2 seconds

**> Response**

To save the draft of what is being typed as a new task in a cookie every 2 seconds, you can use [useEffect\(node\\_modules/@types/react/index.d.ts\)](#) with a `setInterval` to monitor the value of the input field. Here's how you can implement this in the [Input component \(src/todo/components/input.jsx\)](#):

**Change in the [Input Component \(src/todo/components/input.jsx\)](#)**

Add a local state to store the draft and use [useEffect \(node\\_modules/@types/react/index.d.ts\)](#) to save the draft in the cookie every 2 seconds.

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

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

function saveDraftToCookie(draft) {
  document.cookie = `draft=${encodeURIComponent(draft)}; path=/; max-age=31536000`; // Expires in 1 year
}

export function Input({ onSubmit, placeholder, label, defaultValue,
onBlur }) {
  const [draft, setDraft] = useState(defaultValue || "");

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

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

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

        onSubmit(value);
        setDraft(""); // Clears the draft after submission
        e.target.value = "";
      }
    },
    [onSubmit]
  );

  const handleChange = useCallback((e) => {
```

```

        setDraft(e.target.value);
    }, []);

    useEffect(() => {
        const interval = setInterval(() => {
            saveDraftToCookie(draft);
        }, 2000);

        return () => clearInterval(interval); // Clears the interval
        when the component is unmounted
    }, [draft]);

    return (
        <div className="input-container">
            <input
                className="new-todo"
                id="todo-input"
                type="text"
                data-testid="text-input"
                autoFocus
                placeholder={placeholder}
                defaultValue={defaultValue}
                onBlur={handleBlur}
                onKeyDown={handleKeyDown}
                onChange={handleChange} // Updates the draft while
typing
            />
            <label className="visually-hidden" htmlFor="todo-input">
                {label}
            </label>
        </div>
    );
}

```

## Explanation:

1. [useState \(node\\_modules/@types/react/index.d.ts\)](#): Adds a local state draft to store the current value of the input field.
  2. **handleChange**: Updates the draft state whenever the user types something.
  3. [useEffect \(node\\_modules/@types/react/index.d.ts\)](#): Sets up an interval that saves the draft in the cookie every 2 seconds.
  4. **saveDraftToCookie**: Saves the draft in the cookie using [document.cookie \(node\\_modules/typescript/lib/lib.dom.d.ts\)](#).
  5. **Interval Cleanup**: The interval is cleared when the component is unmounted to prevent memory leaks.
- Now, the draft will be automatically saved in the cookie while the user types.

00:35:25

### Participant

Ok, let's see the input file. It seems there are a lot of new things.

00:35:25  
- 00:35:30

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

00:35:37

### Researcher

OK.

00:35:56

### Participant

This here is new.

00:36:04

Let's see if it will work.

00:36:26

### Researcher

ok.

00:38:03

**Chat** (window: 0)

### > Agent

github.copilot.workspace

### > Context

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

### > Message

@workspace I typed the input "teste" but when I reload the page it disappears, I want it to be saved in cache, every two seconds there should be an update of the saved draft data.

### > Response

To ensure that the draft typed in the input field is saved in cache and restored when reloading the page, you can use cookies to store the draft and load it when initializing the component. Here's how you can implement this in the file [input.jsx](src/todo/components/input.jsx) (src/todo/components/input.jsx):

## Component Update [Input \(src/todo/components/input.jsx\)](src/todo/components/input.jsx)

### 1. Add a function to load the draft from the cookie:

```
function loadDraftFromCookie() {  
  const cookies = document.cookie.split("; ").reduce((acc,  
  cookie) => {  
    const [key, value] = cookie.split("=");
```

```

        acc[key] = decodeURIComponent(value);
        return acc;
    }, {}));
    return cookies.draft || ""; // Returns the saved draft or an
empty string
}

```

2. **Update the initial state to load the saved draft:** Replace the initial state of the [draft\(src/todo/components/input.jsx\)](#) to use the value loaded from the cookie.

```
const [draft, setDraft] = useState(loadDraftFromCookie());
```

3. **Ensure the draft is saved every 2 seconds:** The [useEffect](#) ([node\\_modules/@types/react/index.d.ts](#)) is already set up to save the draft to the cookie every 2 seconds. Make sure it is working correctly.
4. **Updated complete code:** Here is the updated code of the component [Input](#) ([src/todo/components/input.jsx](#)):

```

// filepath: /workspaces/copilot-study-
todo/src/todo/components/input.jsx
import { useCallback, useState, useEffect } from "react";

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

function saveDraftToCookie(draft) {
    document.cookie = `draft=${encodeURIComponent(draft)}; path=;/;
max-age=31536000`; // Expires in 1 year
}

function loadDraftFromCookie() {
    const cookies = document.cookie.split("; ").reduce((acc,
cookie) => {
        const [key, value] = cookie.split("=");
        acc[key] = decodeURIComponent(value);
        return acc;
    }, {});
    return cookies.draft || ""; // Returns the saved draft or an
empty string
}

export function Input({ onSubmit, placeholder, label,
defaultValue, onBlur }) {
    const [draft, setDraft] = useState(loadDraftFromCookie());

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

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

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

                onSubmit(value);
                setDraft(""); // Clears the draft after submission
                e.target.value = "";
            }
        },
        [onSubmit]
    );
}

```

```
);  
  
const handleChange = useCallback((e) => {  
  setDraft(e.target.value);  
}, []);  
  
useEffect(() => {  
  const interval = setInterval(() => {  
    saveDraftToCookie(draft);  
  }, 2000);
```

```

        return () => clearInterval(interval); // Clears the
        interval when the component is unmounted
    }, [draft]);

    return (
        <div className="input-container">
            <input
                className="new-todo"
                id="todo-input"
                type="text"
                data-testid="text-input"
                autoFocus
                placeholder={placeholder}
                defaultValue={defaultValue}
                value={draft} // Sets the input value as the draft
                onBlur={handleBlur}
                onKeyDown={handleKeyDown}
                onChange={handleChange} // Updates the draft when
typing
            />
            <label className="visually-hidden" htmlFor="todo-
input">
                {label}
            </label>
        </div>
    );
}

```

## Explanation:

1. **loadDraftFromCookie**: Loads the draft saved in the cookie when initializing the component.
2. **useState(loadDraftFromCookie())**  
([node\\_modules/@types/react/index.d.ts](#)): Sets the initial state of the draft with the value saved in the cookie.
3. **value={draft}** ([src/todo/components/input.jsx](#)): Ensures that the input field value is synchronized with the state [draft](#) ([src/todo/components/input.jsx](#)).
4. **useEffect** ([node\\_modules/@types/react/index.d.ts](#)): Saves the draft to the cookie every 2 seconds.

Now, the draft will be automatically saved in the cookie and restored when reloading the page.

> **Implementing Copilot code** using apply button ([src/todo/components/input.jsx](#))

## Participant

Let's see if this time it will save.

**Chat** (window: 0)

> **Agent**

github.copilot.default

> **Context**

viewport ([src/todo/components/input.jsx](#))

> **Message**

It still didn't work, the draft data is not being saved.

> **Response**

If the draft data is still not being saved, there may be a problem with how cookies are being handled. Let's review and correct the code to ensure the draft is saved correctly.

## Steps to fix:

00:38:39  
- 00:39:25

00:39:58

00:40:25

1. **Check if cookies are being set correctly:** Make sure the browser allows the use of cookies and that the domain and path are set correctly.
2. **Update the function [saveDraftToCookie \(src/todo/components/input.jsx\)](#):** Add a log to check if the function is being called correctly and if the draft value is being saved.

```
function saveDraftToCookie(draft) {
  console.log("Saving draft to cookie:", draft); // Debug log
  document.cookie = `draft=${encodeURIComponent(draft)}; path=;/;
max-age=31536000`; // Expires in 1 year
}
```

3. **Update the function [loadDraftFromCookie \(src/todo/components/input.jsx\)](#):** Add a log to check if the draft is being loaded correctly from the cookie.

```
function loadDraftFromCookie() {
  const cookies = document.cookie.split("; ").reduce((acc,
cookie) => {
    const [key, value] = cookie.split("=");
    acc[key] = decodeURIComponent(value);
    return acc;
  }, {});
  console.log("Loading draft from cookie:", cookies.draft); //
Debug log
  return cookies.draft || ""; // Returns the saved draft or an
empty string
}
```

4. **Ensure that [useEffect \(node\\_modules/@types/react/index.d.ts\)](#) is working correctly:** Add a log inside [useEffect \(node\\_modules/@types/react/index.d.ts\)](#) to check if it is being executed.

```
useEffect(() => {
  console.log("Updating cookie with draft:", draft); // Debug
log
  const interval = setInterval(() => {
    saveDraftToCookie(draft);
  }, 2000);

  return () => clearInterval(interval); // Clears the interval
when the component unmounts
}, [draft]);
```

5. **Test the behavior:**

- Open the browser console (F12) and check the logs to ensure the functions are being called correctly.
- Check if the cookie [draft \(src/todo/components/input.jsx\)](#) is being created in the browser. You can inspect the cookies in the browser console, under the "Application" tab > "Cookies".

6. **Complete updated code:** Here is the updated code with debug logs:

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

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

function saveDraftToCookie(draft) {
  console.log("Saving draft to cookie:", draft); // Debug log
  document.cookie = `draft=${encodeURIComponent(draft)}; path=;/;
max-age=31536000`; // Expires in 1 year
```



```

}

function loadDraftFromCookie() {
  const cookies = document.cookie.split("; ").reduce((acc,
cookie) => {
    const [key, value] = cookie.split("=");
    acc[key] = decodeURIComponent(value);
    return acc;
  }, {});
  console.log("Loading draft from cookie:", cookies.draft); //
Debug log
  return cookies.draft || ""; // Returns the saved draft or an
empty string
}

export function Input({ onSubmit, placeholder, label,
defaultValue, onBlur }) {
  const [draft, setDraft] = useState(loadDraftFromCookie());

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

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

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

        onSubmit(value);
        setDraft(""); // Clears the draft after submission
        e.target.value = "";
      }
    },
    [onSubmit]
  );

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

  useEffect(() => {
    console.log("Updating cookie with draft:", draft); //
Debug log
    const interval = setInterval(() => {
      saveDraftToCookie(draft);
    }, 2000);

    return () => clearInterval(interval); // Clears the
interval when the component unmounts
  }, [draft]);

  return (
    <div className="input-container">

```

```

        <input
          className="new-todo"
          id="todo-input"
          type="text"
          data-testid="text-input"
          autoFocus
          placeholder={placeholder}
          defaultValue={defaultValue}
          value={draft} // Sets the input value as the draft
          onBlur={handleBlur}
          onKeyDown={handleKeyDown}
          onChange={handleChange} // Updates the draft while
            typing
        />
        <label className="visually-hidden" htmlFor="todo-
input">
          {label}
        </label>
      </div>
    );
  }
}

```

### Next steps:

1. Check the logs in the browser console to identify where the problem occurs.
2. Ensure that the cookie [draft \(src/todo/components/input.jsx\)](#) is being created and updated correctly.
3. If the problem persists, share the logs or error messages for further analysis.

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

### Participant

I will investigate what's happening?

Party f 5 saving.

OK, it worked.

It seems to me to be more complex

### Researcher

yes there are details that need to be done

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

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

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

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

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

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

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

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

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

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

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

### Participant

But I don't. I have no idea how to fix it.

So I'm going to try to see with.

with the copilot

### Researcher

OK.

**Chat** (window: 0)

> **Agent**

github.copilot.workspace

### > Context

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

### > Message

@workspace are the parameters for the paginationControls function being passed correctly?

### > Response

No, the parameters for the [paginationControls \(src/todo/components/main.jsx\)](#) function are not being passed correctly. This is because [paginationControls \(src/todo/components/main.jsx\)](#) is being used as a React component, but it was imported as a function (with a lowercase letter). In React, components should start with an uppercase letter.

### Correction:

Change the import of [paginationControls \(src/todo/components/main.jsx\)](#) to [PaginationControls \(src/todo/components/paginationControls.jsx\)](#) (with an uppercase letter) and use the component correctly in the JSX.

### Code Update:

```
File: src/todo/components/main.jsx
import { PaginationControls } from "../paginationControls";

export function Main({ todos, dispatch }) {
  // ...existing code...

  return (
    <main className="main" data-testid="main">
      {visibleTodos.length > 0 ? (
        <>
          <div className="toggle-all-container">
            <input className="toggle-all" type="checkbox"
              id="toggle-all" data-testid="toggle-all" checked=
                {visibleTodos.every((todo) => todo.completed)} onChange={toggleAll} />
            <label className="toggle-all-label"
              htmlFor="toggle-all">
              Toggle All Input
            </label>
          </div>
          <ul className={classnames("todo-list")} data-
            testid="todo-list">
            {visibleTodos.map((todo, index) => (
              <Item todo={todo} key={todo.id} dispatch=
                {dispatch} index={index} />
            ))}
          </ul>
        </>
      ) : (
        <div className="todo-list-empty-container">
          <span className="todo-list-empty-message">No tasks
            here yet</span>
        </div>
      )}

      { /* Use the PaginationControls component correctly */ }
      <PaginationControls
        currentPage={currentPage}
        itemsPerPage={itemsPerPage}
        setCurrentPage={setCurrentPage}
        totalItems={visibleTodos.length}
      />
    </main>
  );
}
```

```

    );
  }

```

## Explanation:

1. **Corrected Import:** [PaginationControls](#) ([src/todo/components/paginationControls.jsx](#)) was imported with the first letter capitalized, as it is a React component.
2. **Usage in JSX:** The [PaginationControls](#) ([src/todo/components/paginationControls.jsx](#)) component is now used correctly with the parameters [currentPage](#) ([src/todo/components/main.jsx](#)), [itemsPerPage](#) ([src/todo/components/main.jsx](#)), [setCurrentPage](#) ([src/todo/components/main.jsx](#)), and [totalItems](#) ([src/todo/components/main.jsx](#)).

With this correction, the parameters will be passed correctly to the [PaginationControls](#) ([src/todo/components/paginationControls.jsx](#)) component.

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

### Participant

Let's see if just changing this function

Not everything is working yet

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

### Participant

The pagination.

**Chat** (window: 0)

### > Agent

github.copilot.workspace

### > Context

workspace, viewport ([src/todo/components/paginationControls.jsx](#))

### > Message

@workspace the implemented pagination is not working, I want it to work displaying 1, 3, or 5 tasks and the value 5 should be the default

### > Response

To fix the pagination and ensure it works correctly displaying 1, 3, or 5 tasks per page (with 5 as the default), you need to adjust the navigation and item display logic. Here's how you can implement this:

## Update in Component [PaginationControls](#)

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

1. Add logic to handle navigation between pages and the number of items per page.
2. Fix the function [pageChange](#) ([src/todo/components/paginationControls.jsx](#)) to handle "prev" and "next" navigation.

Here is the updated code:

File: [src/todo/components/paginationControls.jsx](#)

```

import React, { useState } from "react";
import classNames from "classnames";

```

```

export function PaginationControls({ currentPage, setCurrentPage,

```

```

itemsPerPage, setItemsPerPage, totalItems }) {
  const [clickedButton, setClickedButton] = useState(null);

  const totalPages = Math.ceil(totalItems / itemsPerPage);

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

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

  return (
    <div className="pagination-controls">
      <button
        className={classnames("pagination-button", { inactive:
currentPage === 1, active: clickedButton === "prev" })}
        onClick={() => pageChange("prev")}
        disabled={currentPage === 1}
      >
        Previous
      </button>
      <span className="pagination-info">
        Page {currentPage} of {totalPages}
      </span>
      <button
        className={classnames("pagination-button", { inactive:
currentPage === totalPages, active: clickedButton === "next" })}
        onClick={() => pageChange("next")}
        disabled={currentPage === totalPages}
      >
        Next
      </button>
      <select className="items-per-page" value={itemsPerPage}
onChange={handleItemsPerPageChange}>
        <option value={1}>1</option>
        <option value={3}>3</option>
        <option value={5}>5</option>
      </select>
    </div>
  );
}

```

---

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

In the component [Main \(src/todo/components/main.jsx\)](#), adjust the logic to display only the items of the current page based on the number of items per page.

```

File: src/todo/components/main.jsx
export function Main({ todos, dispatch }) {
  const { pathname: route } = useLocation();

```



```

const [currentPage, setCurrentPage] = useState(1);
const [itemsPerPage, setItemsPerPage] = useState(5); // Default
value: 5

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

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

return (
  <main className="main" data-testid="main">
    {paginatedTodos.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={(e) => dispatch({ type:
TOGGLE_ALL, payload: { completed: e.target.checked } })}
                                />
                                <label className="toggle-all-label"
htmlFor="toggle-all">
                                Toggle All Input
                                </label>
                                </div>
                                <ul className={classnames("todo-list")} data-
testid="todo-list">
                                {paginatedTodos.map((todo, index) => (
                                <Item todo={todo} key={todo.id} dispatch=
{dispatch} index={index} />
                                ))}
                                </ul>
                                </>
                                ) : (
                                <div className="todo-list-empty-container">
                                <span className="todo-list-empty-message">No tasks
here yet</span>
                                </div>
                                )}

                                <PaginationControls
                                currentPage={currentPage}
                                setCurrentPage={setCurrentPage}
                                itemsPerPage={itemsPerPage}
                                setItemsPerPage={setItemsPerPage}
                                totalItems={visibleTodos.length}
                                />
                                </main>
                                );
}

```

## Explanation of Changes

### 1. [PaginationControls \(src/todo/components/paginationControls.jsx\)](#):

- Added logic to calculate the total number of pages.
- Added a selector to change the number of items per page (1, 3, or 5).
- Reset to the first page when changing the number of items per page.

### 2. [Main \(src/todo/components/main.jsx\)](#):

- Added logic to calculate the items displayed based on the current page and the number of items per page.
- Passed the states [itemsPerPage \(src/todo/components/main.jsx\)](#) and [setItemsPerPage](#) to the component [PaginationControls \(src/todo/components/paginationControls.jsx\)](#).

Now, the pagination will work correctly, displaying 1, 3, or 5 tasks per page, with 5 as the default value.

00:50:03  
- 00:50:14

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

00:50:27

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

00:50:55

**Participant**

I think I changed the file and applied the changes incorrectly.

00:51:00

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

00:51:02

**Researcher**

Got it.

00:51:07  
- 00:51:11

00:51:07

00:51:32  
- 00:51:36

00:51:32

00:51:32

00:51:37

00:51:39

00:52:43

00:53:03

00:53:06

00:54:51

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

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

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

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

**Participant**  
probably that's it, I'm all broken.  
The code, after it changed.

**Researcher**  
Inserted half in one file and the other half stayed in this file

**Participant**  
Let's see now.

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

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

**Chat** (window: 0)

**> Agent**  
github.copilot.workspace

**> Context**  
workspace, selection (src/todo/app.css)

**> Message**  
@workspace I want the next and previous buttons, when they are at the pagination limits, without next or previous, to be displayed in gray.

**> Response**  
To ensure that the "Next" and "Previous" buttons are displayed in gray when they are at the pagination limits (i.e., when there is no next or previous), you already have the .pagination-button.inactive class in your CSS. This class applies the gray style and disables the cursor.

Now, just ensure that the logic in the [PaginationControls component \(src/todo/components/paginationControls.jsx\)](#) correctly applies the [inactive \(src/todo/components/paginationControls.jsx\)](#) class when the buttons are disabled.

## CSS Update (if necessary)

Make sure the .pagination-button.inactive class is correctly configured in the [app.css file \(src/todo/app.css\)](#):

```
File: src/todo/app.css
.pagination-button.inactive {
  background-color: #ccc; /* Gray */
  color: #666; /* Darker gray text */
  cursor: not-allowed; /* Disabled cursor */
  opacity: 0.6; /* Transparency to indicate disabled */
}
```

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

In the [paginationControls.jsx file \(src/todo/components/paginationControls.jsx\)](#), the logic already checks if the buttons are disabled based on the [currentPage state \(src/todo/components/main.jsx\)](#). Ensure that the [inactive \(src/todo/components/paginationControls.jsx\)](#) class is being applied correctly:

```

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

export function PaginationControls({ currentPage, setCurrentPage,
itemsPerPage, setItemsPerPage, totalItems }) {
  const [clickedButton, setClickedButton] = useState(null);

  const totalPages = Math.ceil(totalItems / itemsPerPage);

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

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

  return (
    <div className="pagination-controls">
      <button
        className={classNames("pagination-button", { inactive:
currentPage === 1 })}
        onClick={() => pageChange("prev")}
        disabled={currentPage === 1} // Disables the button if
on the first page
      >
        Previous
      </button>
      <span className="pagination-info">
        Page {currentPage} of {totalPages}
      </span>
      <button
        className={classNames("pagination-button", { inactive:
currentPage === totalPages })}
        onClick={() => pageChange("next")}
        disabled={currentPage === totalPages} // Disables the
button if on the last page
      >
        Next
      </button>
      <select className="items-per-page" value={itemsPerPage}
onChange={handleItemsPerPageChange}>
        <option value={1}>1</option>
        <option value={3}>3</option>
        <option value={5}>5</option>
      </select>
    </div>
  );
}

```

**Explanation:**

1. [inactive class \(src/todo/components/paginationControls.jsx\)](#):

- The [inactive class \(src/todo/components/paginationControls.jsx\)](#) is applied to the "Next" and "Previous" buttons when they are disabled ([currentPage === 1 \(src/todo/components/main.jsx\)](#) or [currentPage === totalPages \(src/todo/components/main.jsx\)](#)).
- The gray style is defined in the CSS to indicate that the button is disabled.

2. [disabled property \(node\\_modules/@types/react/index.d.ts\)](#):

- The [disabled property \(node\\_modules/@types/react/index.d.ts\)](#) is used to functionally disable the button, preventing clicks.

3. Visual Style:

- The `.pagination-button.inactive` class applies the gray style and disables the cursor.

With these changes, the "Next" and "Previous" buttons will be displayed in gray when they are at the pagination limits.

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

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

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

**Researcher**

00:55:16  
- 00:55:20

00:55:24

00:55:25  
- 00:55:28

00:55:54

00:56:11 Alright.  
00:58:03 I'm going to ask you to stop now. Our time is up.  
00:58:09 So you can close it.  
00:58:16 Access the link for the post-study questionnaire please  
**Participant**  
ok  
00:58:28 I just the copilot to manage  
00:58:34 to do the code. It was smooth.  
00:58:41 way to is occupy specifically with a Visual Studio Gold.  
00:59:02 Right. the fulfillment.  
00:59:07 I found it excellent  
00:59:21 Details?  
00:59:21 It's excellent. It specifies well where the error is for  
00:59:25 you to understand  
00:59:27 **Researcher**  
ok.  
00:59:28 **Participant**  
Technical terms.  
00:59:31 It is also excellent the corrections.  
01:00:02 It also uses excellent formatting.  
01:00:32 It would also improve my performance. I could do it much faster.  
01:00:46 Yes.  
01:01:14 Is it?  
01:01:14 I find it very intuitive, so it would be easy.  
01:01:18 **Researcher**  
Cool.  
01:01:44 **Participant**  
Play, I won't put it here.  
01:01:46 I completely agree that there might be some detail or another here.  
01:02:29 **Researcher**  
Alright.  
01:02:37 **Participant**  
It was rushed, accelerated.  
01:03:01 medium, I think medium would be zero.  
01:03:04 **Researcher**  
Yes,  
01:03:22 it was indifferent to you.  
01:03:24 **Participant**  
Ah, I see, I see.  
01:03:28 It doesn't reach.  
01:03:31 I almost didn't make much mental effort to achieve it.  
01:03:39 **Researcher**  
Alright.  
01:04:32 **Participant**  
pagination, it's just knowing the concept of how to use the  
01:04:36 cookies, that you can already do it, but still it's easier than the  
01:04:41 pagination.  
01:04:43 **Researcher**  
Cool.  
01:04:45 **Participant**  
Persist.  
01:04:53 Oh, quite a lot.  
01:05:09 **Researcher**  
OK, thanks for answering the survey. Now I'll ask you a few more questions.  
01:05:21 Do you have experience with ReactJS apps?  
01:05:25 **Participant**

01:05:28 **Researcher**  
ok

01:05:29 **Participant**  
Um.

01:05:40 **Researcher**  
Were you able to navigate through the code?

01:05:48 **Participant**  
I was. There's a lot of stuff I've already encountered,  
01:05:52 I just didn't remember how to do it.  
01:05:55 But by looking at the code, there was a good orientation of what  
01:06:02 the functionality was more or less.

01:06:08 **Researcher**  
And how did the copilot influence your code navigation?

01:06:18 **Participant**  
It guided me in the context  
01:06:23 where I didn't know.  
01:06:25 I knew where I was, but didn't remember the details  
01:06:34 In this part, it helped me a lot.

01:06:38 **Researcher**  
Nice  
01:06:42 And how would you have solved these tasks without the help of the copilot?

01:06:50 **Participant**  
I would have searched.  
01:06:53 Searched the problem on Google.  
01:07:05 In the persistence task, I would use cookies,  
01:07:08 but the copilot suggested another way.  
01:07:13 and would try to look for the documentation.

01:07:14 **Researcher**  
ok  
01:07:45 what aspect did you like and what didn't you like about the copilot

01:07:55 **Participant**  
What I liked most about the responses was that you can apply it to the code,  
01:08:00 you can see what was changed.  
01:08:05 The only part I didn't like was the  
01:08:11 bug that happened when applying the code I changed the file that was selected  
01:08:17 the code of the 2 files was broken.

01:08:26 **Researcher**  
ok, I understand.  
01:08:32 Did you change the way you interact with the copilot?

01:08:53 **Participant**  
I only changed the way it was written. I think I wrote it in a more  
01:08:58 clear way. Maybe that's why it worked.

01:09:01 **Researcher**  
ok.  
01:09:04 Thank you for participating in our research.

03:18:00 **> Response rendering**  
- 03:18:03

03:23:55 **> Response rendering**  
- 03:24:00

03:29:06 **> Response rendering**  
- 03:29:12

03:34:56 **> Response rendering**  
- 03:35:11

03:38:06 **> Response rendering**  
- 03:38:21

03:40:24 **> Response rendering**  
- 03:40:38

**> Response rendering**

03:46:49  
- 03:46:56

03:49:37  
- 03:49:51

**> Response rendering**

03:54:54  
- 03:55:05

**> Response rendering**