

Participant P20

00:02:19 **Researcher**
ok, you can start.

00:02:19 **Participant**
OK

00:02:29 Do you prefer that I talk about what I'm doing now.

00:02:32 **Researcher**
Yes.

00:03:38 **Participant**
They are first doing the task of changing the color of the box

00:03:42 **Researcher**
ok.

00:03:46 **> Viewing file** (src/todo/app.css)

00:03:51 **Participant**
I'll say that I'm not very familiar with Javascript.

00:03:57 **> Viewing file** (src/todo/app.jsx)

00:03:58 **Researcher**
No problem.

00:04:02 **> Viewing file** (src/todo/components/header.jsx)

00:04:06 **> Viewing file** (src/todo/components/main.jsx)

00:04:06 **> Viewing file** (src/todo/components/item.jsx)

00:04:27 **Participant**
I don't remember how to run the project.

00:04:31 **Researcher**
Ok, feel free to
ask any kind of question to the copilot. If you feel it didn't help you,
you can also search the web.

00:04:41 **Participant**
OK

00:04:51 **Chat** (window: 0)
> Agent
github.copilot.default
> Context
codebase
> Message
How to run the JavaScript project #codebase
> Response
To run the project, follow the steps below:

- 1. Install the dependencies:**
In the terminal, run the command below to install the project dependencies listed in the [package.json](#):

npm install
- 2. Start the development server:**
After installing the dependencies, run the command below to start the development server:

npm run dev
- 3. Access the project in the browser:**
The development server will automatically open the project in the default

browser. If this does not happen, manually access the default address <http://localhost:8080>.

Make sure you have Node.js ($\geq 18.13.0$) and npm ($\geq 8.19.3$) installed, as specified in the [package.json](#).

00:04:57
- 00:05:01

> Response rendering

00:06:50

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

00:06:54

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

00:07:05

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

00:07:10

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

00:07:17

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

00:07:22

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

00:07:33

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

00:07:38

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

00:07:39

Researcher

What are you thinking now?

00:07:47

Participant

I'm looking for where that

00:07:47

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

00:07:52

Participant

label is, 'do this here'

00:07:55

Researcher

OK.

00:07:57

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

00:08:02

Participant

Can I search the entire base?

00:08:04

Researcher

Yes

00:08:43

Chat (window: 0)

> Agent

github.copilot.default

> Context

codebase

> Message

#codebase I would like to change the color of the "Nothing to do yet" field from red to blue

> Response

To change the color of the "Nothing to do here yet" field from red to blue, you can modify the style in the corresponding CSS file. In this case, the style is defined in the file /workspaces/copilot-study-todo/src/todo/app.css.

Change the background property of the class .todo-list-empty-message to the color blue. Here is the modification:

```
File: src/todo/app.css
.todo-list-empty-message {
  padding: 10px;
  background: #007bff; /* Changed from red (#b83f45) to blue
(#007bff) */
  color: white;
  border-radius: 5px;
  font-weight: bolder;
}
```

After saving the file, the style will be automatically updated if you are running the development server (npm run dev). Otherwise, restart the server to apply the changes.

00:08:49
- 00:08:53

> Response rendering

00:08:58
00:09:28
00:09:34

00:09:42
- 00:09:44

00:10:10

00:10:31
00:10:46

00:11:01
- 00:11:05

00:11:29

00:11:35
00:11:38
00:11:41

00:12:06

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

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

Participant

Ah, ok, I got it. It's through the page's CSS

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

Participant

Change to blue.

Next task, is to change the 'placeholder'.

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

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

Participant

Now, sorting.

But I'm not really sure what I'm doing here.

I'm going to ask the copilot for help.

Researcher

OK

Chat (window: 0)

> **Agent**

github.copilot.default

> **Context**

codebase

> **Message**

#codebase how would I sort multiple tasks alphabetically?

> **Response**

To sort the tasks alphabetically, you can modify the display logic in the component [Main\(src/todo/components/main.jsx\)](#) where the tasks are filtered and rendered. Add a step to sort the list of tasks before displaying them.

Here is the modification in the file /workspaces/copilot-study-todo/src/todo/components/main.jsx:

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

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

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

Explanation:

1. **Filtering:** The list of tasks is filtered based on the route (/active, /completed, or all).
 2. **Sorting:** The `.sort()` function is used to sort the tasks alphabetically by the [title\(src/todo/components/header.jsx\)](#) field. The `localeCompare` function ensures that the sorting respects language rules.
- After saving the change, the tasks will be displayed in alphabetical order.

00:12:11

Participant

I believe it will get the context of the application

00:12:12
- 00:12:16

> Response rendering

00:12:14

Participant

maybe it will understand what to do.

00:12:36

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

00:12:51
- 00:13:54

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

00:13:02

Participant

I'm going to test first, see how it's being done here.

00:14:02

Ah, then you have to call everything again,

00:14:04

that's right.

00:14:22

apparently, everything right gave 123, brown

00:14:26

Researcher

OK.

00:14:30

Participant

quick, Fox.

00:14:32

Researcher

OK, I'll send you the tasks now

00:14:40

Participant

OK?

00:14:47

Researcher

Same scheme, keep working on this same project,

00:14:54

You can do the tasks in the order you want.

00:14:58

make sure to check all the requirements of each task before moving on to the next

00:15:19

Participant

OK.

00:15:51

Is it mandatory to use the copilot?

00:15:57

Researcher

No.

00:15:58

That's more up to your feeling.

00:16:11

Participant

OK.

00:16:18

Chat (window: 0)

> Agent

github.copilot.default

> Context

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

> Message

Currently, all tasks disappear when the page is reloaded. Modify the application so that the tasks remain even after a reload. Ensure that the state of each task (whether it is completed or active) is also preserved. An attempt to implement this functionality was made, but it is currently not working correctly. Fix or reimplement the functionality.

> Response

To ensure that the tasks remain even after the page is reloaded, you can use `localStorage` to save and load the tasks. Here is how you can fix or implement this functionality:

Modification in the component [Main](#)
[\(src/todo/components/main.jsx\)](#)

Add logic to save the tasks in `localStorage` whenever they are updated and load them when initializing the component.

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

```

import classNames from "classnames";
import { TOGGLE_ALL } from "../constants";
import { useLocation } from "react-router-dom";
import { useMemo, useCallback, useState, useEffect } from "react";
import { paginationControls } from "../paginationControls";

export function Main({ todos, dispatch }) {
  const { pathname: route } = useLocation();
  const [currentPage, setCurrentPage] = useState(1);
  const [lastAddedItem, setLastAddedItem] = useState(null);

  const itemsPerPage = 5;

  // Load tasks from localStorage when initializing
  useEffect(() => {
    const savedTodos = JSON.parse(localStorage.getItem("todos")) ||
    [];
    if (savedTodos.length > 0) {
      dispatch({ type: "LOAD_TODOS", payload: savedTodos });
    }
  }, [dispatch]);

  // Save tasks in localStorage whenever they are updated
  useEffect(() => {
    localStorage.setItem("todos", JSON.stringify(todos));
  }, [todos]);

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

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

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

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

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

  return (
    <main className="main" data-testid="main">
      {visibleTodos.length > 0 ? (
        <>
          <div className="toggle-all-container">
            <input className="toggle-all" type="checkbox"
            id="toggle-all" data-testid="toggle-all" checked=
            {visibleTodos.every((todo) => todo.completed)} onChange={toggleAll} />
            <label className="toggle-all-label"
            htmlFor="toggle-all">

```



```

        Toggle All Input
      </label>
    </div>
    <ul className={classnames("todo-list")} data-
testid="todo-list">
      {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>
)
}

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

```

Modification in the reducer (if applicable)

Add a new case in the reducer to handle the LOAD_TODOS action:

```

File: src/todo/reducer.js
function todoReducer(state, action) {
  switch (action.type) {
    case "LOAD_TODOS":
      return action.payload;
    // ...other existing cases...
    default:
      return state;
  }
}

```

Explanation:

1. **Load tasks from localStorage:** In [useEffect \(node_modules/@types/react/index.d.ts\)](#), when initializing the component, the tasks are loaded from localStorage and dispatched to the global state.
2. **Save tasks in localStorage:** Whenever the task list ([todos \(src/todo/components/main.jsx\)](#)) is updated, it will be saved in localStorage.
3. **New case in the reducer:** The LOAD_TODOS case allows loading the saved tasks into the global state.

With these changes, the tasks and their states will be preserved even after the page is reloaded.

00:16:20
- 00:16:30

> Response rendering

Chat (window: 0)

> Agent

github.copilot.default

00:16:48

> Context

codebase

> Message

#codebase Currently, all tasks disappear when the page is reloaded. Modify the application so that the tasks remain even after a reload. Ensure that the state of each task (whether it is completed or active) is also preserved. An attempt to implement this functionality was made, but it is currently not working correctly. Fix or reimplement the functionality.

> Response

To fix the task persistence functionality in [localStorage](#) ([node_modules/@types/node/globals.d.ts](#)), you can adjust the existing logic in the reducer and the main component of the application. Here is the complete solution:

1. Fix the reducer to correctly load and save the tasks

In the file `/workspaces/copilot-study-todo/src/todo/reducer.js`, adjust the [loadState](#) ([src/todo/reducer.js](#)) and the [saveState](#) ([src/todo/reducer.js](#)) to ensure that the data is handled correctly.

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

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

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

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

export const initialState = loadState();

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

```
    case TOGGLE_ALL:
      newState = state.map((todo) => (todo.completed !==
action.payload.completed ? { ...todo, completed:
action.payload.completed } : todo));
      break;
    case REMOVE_COMPLETED_ITEMS:
      newState = state.filter((todo) => !todo.completed);
      break;
    case LOAD_STATE:
      newState = action.payload;
      break;
    default:
```

```

        throw Error(`Unknown action: ${action.type}`);
    }

    // Save the updated state in localStorage
    saveState(newState);
    return newState;
};

```

2. Adjust the [App component \(src/todo/app.jsx\)](#) to initialize the state with data from [localStorage \(node_modules/@types/node/globals.d.ts\)](#)

In the file `/workspaces/copilot-study-todo/src/todo/app.jsx`, use the [initialState \(src/todo/reducer.js\)](#) from the reducer to load the saved tasks.

```

File: src/todo/app.jsx
import { useReducer } from "react";
import { Header } from "../components/header";
import { Main } from "../components/main";
import { Footer } from "../components/footer";

import { todoReducer, initialState } from "../reducer";

import "../app.css";

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

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

```

3. Ensure that the state is saved and loaded correctly

- The [loadState \(src/todo/reducer.js\)](#) loads the tasks from [localStorage \(node_modules/@types/node/globals.d.ts\)](#) upon initialization.
- The [saveState \(src/todo/reducer.js\)](#) saves the tasks in [localStorage \(node_modules/@types/node/globals.d.ts\)](#) whenever the state is updated.

4. Test the functionality

1. Add some tasks in the application.
2. Refresh the page.
3. Check if the tasks remain in the list, with the correct state (completed or active).

With these changes, the tasks will be persisted in [localStorage \(node_modules/@types/node/globals.d.ts\)](#) and correctly restored when the page is reloaded.

00:16:55
- 00:17:17

> Response rendering

00:17:28

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

00:18:08
- 00:18:34

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

00:18:32

Participant

This task says there's already a part of the code implemented, right?

00:18:33

Researcher

00:18:39 Yes, that's right.
00:18:45 > **Viewing file** (src/todo/app.jsx) through Copilot link
00:18:53 **Participant**
- 00:19:07 ok.
00:19:05 > **Implementing Copilot code** using apply button (src/todo/app.jsx)
00:19:05 **Participant**
Import error
00:19:18 I'll test now.
00:19:19 **Researcher**
OK.
00:19:34 **Participant**
You load the page and confirm that the tasks are still there.
00:19:39 mark some tasks as completed, reload the page and confirm that the
00:19:44 tasks remained.
00:19:51 OK, edit some tasks.
00:19:54 You load the page and confirm that the changes were kept.
00:20:21 Alright.
00:20:25 With the tasks you load the page and I confirm that the tasks remain
00:20:29 absent.
00:20:36 Correct?
00:20:37 **Researcher**
Alright.
00:21:19 **Participant**
draft task.
00:21:45 **Chat** (window: 0)
> **Agent**
github.copilot.default
> **Context**
codebase
> **Message**
#codebase If a user reloads the page while typing a new task, the text in the input field disappears. Modify the application so that the input field automatically saves drafts every few seconds and restores them when the page is reloaded.
> **Response**
To implement the functionality of automatically saving the text in the input field as a draft and restoring it when the page is reloaded, you can use [localStorage](#) ([node_modules/@types/node/globals.d.ts](#)). Here's how to do it:

1. Modify the component [Input](#) (src/todo/components/input.jsx).

Update the component [Input](#) (src/todo/components/input.jsx) to save the typed text in [localStorage](#) ([node_modules/@types/node/globals.d.ts](#)) and restore it on load.

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

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

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

  // Restore draft from localStorage on load
  useEffect(() => {
```

```

        const savedDraft = localStorage.getItem("todo-draft");
        if (savedDraft) {
            setValue(savedDraft);
        }
    }, []);

    // Save draft to localStorage whenever the value changes
    useEffect(() => {
        localStorage.setItem("todo-draft", value);
    }, [value]);

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

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

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

                onSubmit(trimmedValue);
                setValue(""); // Clear the field after submission
                localStorage.removeItem("todo-draft"); // Remove the
draft after submission
            }
        },
        [value, onSubmit]
    );

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

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

```

2. Explanation of changes

1. **Restore draft:** In the initial [useEffect](#) ([node_modules/@types/react/index.d.ts](#)), the saved draft value in [localStorage](#) ([node_modules/@types/node/globals.d.ts](#)) is loaded and set in the input field state.

2. **Save draft:** Whenever the input field value changes, it is saved in [localStorage](#) ([node_modules/@types/node/globals.d.ts](#)).
3. **Remove draft after submission:** When the user presses "Enter" to submit the task, the draft is removed from [localStorage](#) ([node_modules/@types/node/globals.d.ts](#)).

3. Test the functionality

1. Type something in the input field without pressing "Enter".
2. Reload the page.
3. Check if the typed text was restored in the input field.
4. Press "Enter" to add the task and confirm that the draft was removed.

With these changes, the input field will automatically save drafts and restore them when the page is reloaded.

> Response rendering

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

Participant

It's a bit strange that it gives the entire code block here
It doesn't just give the part that should be changed.

Researcher

nice observation about the copilot

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

00:21:53
- 00:22:02

00:22:19

00:22:39

00:22:44

00:22:49

00:23:01
- 00:23:24

00:23:38

Participant

Confirm that the text in the input field was not changed.

00:24:11

It didn't work. Hold on.

00:24:40

I think it's right now.

00:24:58

Researcher

Alright, you can move on to the last task

00:26:04

Chat (window: 0)

> Agent

github.copilot.default

> Context

codebase

> Message

#codebase Currently, the To-Do List application displays all tasks in a single list. In this task, you will implement pagination to limit the number of to-dos displayed per page, allowing users to easily navigate through their to-dos. Modify the application to display a fixed number of to-dos per page, with options to show 1, 3, or 5 to-dos at a time (the default value should be 5). Implement navigation controls to allow users to move between to-do pages. The application should have "Next" and "Previous" buttons to navigate between pages. The "Next" button should be disabled and displayed in gray when the user is on the last page, and the "Previous" button should be disabled and displayed in gray when the user is on the first page. The page the user is currently on should also be indicated. An attempt to implement the navigation functionality was made, but it is currently not working correctly. Fix or reimplement the navigation and implement the other requirements.

> Response

To implement pagination with options to display 1, 3, or 5 to-dos per page, follow the steps below. We will fix and reimplement the navigation functionality and add the necessary controls.

1. Modify the [Main \(src/todo/components/main.jsx\)](#) component

Update the [Main \(src/todo/components/main.jsx\)](#) component to handle pagination and allow the user to choose the number of items per page.

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

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

  // Filter and sort the to-dos based on the route
  const filteredTodos = useMemo(() => {
    return todos
      .filter((todo) => {
        if (route === "/active") return !todo.completed;
        if (route === "/completed") return todo.completed;
        return true;
      })
      .sort((a, b) => a.title.localeCompare(b.title)); // Sort
    alphabetically
  }, [todos, route]);
```

```

    // Calculate the visible to-dos based on the current page and
    number of items per page
    const visibleTodos = useMemo(() => {
      const startIndex = (currentPage - 1) * itemsPerPage;
      const endIndex = startIndex + itemsPerPage;
      return filteredTodos.slice(startIndex, endIndex);
    }, [filteredTodos, currentPage, itemsPerPage]);

    // Calculate the total number of pages
    const totalPages = useMemo(() => Math.ceil(filteredTodos.length /
itemsPerPage), [filteredTodos, itemsPerPage]);

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

    // Update the current page if the number of items per page changes
    useEffect(() => {
      setCurrentPage(1); // Go back to the first page
    }, [itemsPerPage]);

    return (
      <main className="main" data-testid="main">
        {filteredTodos.length > 0 ? (
          <>
            <div className="toggle-all-container">
              <input
                className="toggle-all"
                type="checkbox"
                id="toggle-all"
                data-testid="toggle-all"
                checked={filteredTodos.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) => (
                <Item todo={todo} key={todo.id} dispatch=
{dispatch} />
              ))}
            </ul>
            <div className="pagination-controls">
              <button
                className="pagination-button"
                disabled={currentPage === 1}
                onClick={() => setCurrentPage((prev) =>
Math.max(prev - 1, 1))}
              >
                Previous
              </button>
              <span className="pagination-info">
                Page {currentPage} of {totalPages}
              </span>
              <button
                className="pagination-button"
                disabled={currentPage === totalPages}

```



```
                                onClick={() => setCurrentPage((prev) =>
Math.min(prev + 1, totalPages))}
                                >
                                Next
                                </button>
</div>
<div className="items-per-page-controls">
  <label htmlFor="items-per-page">Items per page:
```

```

</label>
                <select
                    id="items-per-page"
                    value={itemsPerPage}
                    onChange={(e) =>
setItemsPerPage(Number(e.target.value))}
                >
                    <option value={1}>1</option>
                    <option value={3}>3</option>
                    <option value={5}>5</option>
                </select>
            </div>
        </>
    ) : (
        <div className="todo-list-empty-container">
            <span className="todo-list-empty-message">No to-dos
here yet</span>
        </div>
    )}
</main>
);
}

```

2. Add styles for pagination controls

Update the CSS file to style the pagination controls and the items per page selector.

```

File: src/todo/app.css
.pagination-controls {
    display: flex;
    justify-content: center;
    align-items: center;
    gap: 10px;
    margin-top: 20px;
}

.pagination-button {
    padding: 10px;
    border: none;
    cursor: pointer;
    background-color: #007bff;
    color: white;
    border-radius: 5px;
}

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

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

.items-per-page-controls {
    display: flex;
    justify-content: center;
    align-items: center;
    margin-top: 10px;
    gap: 5px;
}

.items-per-page-controls select {

```

```
padding: 5px;
border-radius: 5px;
border: 1px solid #ccc;
}
```

3. Test the functionality

1. **Initial display:** Ensure that 5 to-dos are displayed by default.
 2. **Navigation:** Use the "Next" and "Previous" buttons to navigate between pages.
 3. **Change items per page:** Change the number of items per page using the selector and verify that the display updates correctly.
 4. **Button disabling:** Check that the "Next" and "Previous" buttons are correctly disabled on the first and last pages.
-

With these changes, the application now supports pagination with options to display 1, 3, or 5 to-dos per page, along with functional navigation controls.

> Response rendering

Participant

I think Copilot is better GPT 4.0.

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

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

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

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

Researcher

135 is, it's not 13 and 5, that's right, that's it. It's 3 and 5. Is that what's being asked in the task?

If you add more items, the app has to build the pagination according to the number of items per page you selected.

Participant

I confirm that users can select options to display 3 or 5 tasks. only the chosen amount of tasks received at a time.

confirm that the last page contains at

one item, at most the selected number of tasks.

Alright.

I confirm that all selected tasks are present.

Researcher

ok.

OK, everything is fine. Cool that it finishes well before the 40-minute mark

I'll send you the post-study survey now. Can you please open the link?

if you want to comment on any question or anything, feel free.

Participant

OK.

How would you evaluate the response in terms of compliance?

I think it was a good compliance.

Could improve the part where the code is changing. But I think it was good details.

I didn't have much trouble understanding, so I believe it's correct

Researcher

Right.

Participant

technical terms.

He took the code that was already there. The copilot saw what was happening and adapted to work, that was excellent.

following your instructions, it didn't cause any bottleneck.

excellent as well.

00:26:10
- 00:26:31

00:26:12

00:26:45

00:26:46
- 00:27:13

00:27:21
- 00:27:30

00:27:21

00:28:21

00:28:32

00:28:37

00:28:54

00:29:07

00:30:14

00:30:16

00:30:21

00:30:30

00:30:38

00:30:57

00:32:45

00:32:50

00:32:57

00:33:06

00:33:13

00:33:19

00:33:30

00:33:35

00:33:36

00:33:54

00:34:00

00:34:04

00:34:08

00:34:10 **Researcher**
Cool.

00:34:13 **Participant**
humorous.

00:34:17 Ah, it was neutral. I think there wasn't any. It was straight to the point.

00:35:02 To what extent do you agree with this information? About the interaction with Copilot?

00:35:06 Using copilot in my work or study would allow me to perform tasks more

00:35:10 quickly.

00:35:12 Well, in my work.

00:35:16 Specifically in my company, I think it wouldn't be that useful.

00:35:24 I don't disagree a bit, in the company where I work we use a different language, [omitted]

00:35:40 Vscode doesn't support it.

00:36:38 study, I'll put, I agree, because it's more of a study.

00:36:42 Learning to operate copilot would be easy for me

00:36:43 **Researcher**
OK.

00:37:08 **Participant**
Action was clear and understandable.

00:37:17 I agree. It looks for the steps I needed

00:37:47 **Researcher**
Perfect.

00:37:54 **Participant**
It wasn't too rushed, it was normal

00:38:00 I think I was successful.

00:38:25 I felt a little insecure for not knowing how to use JavaScript very well.

00:38:31 **Researcher**
Do you work with a team that has worked with JavaScript or just in college

00:38:36 **Participant**
Only from graduation

00:38:48 **Researcher**
But it's normal anyway.

00:39:03 **Participant**
about the tasks. Task of persisting.

00:39:10 very easy, I think because it was basically all ready

00:39:15 task to save drafts.

00:39:17 **Researcher**
Hum, Hum.

00:39:22 **Participant**
It's easy, there was a little more to

00:39:27 change, so.

00:39:33 Yeah, they are all real tasks that would happen

00:39:36 In real life.

00:39:39 **Researcher**
OK.

00:39:42 Thank you for answering. Now I'll ask you a few more questions

00:39:46 **Participant**
Alright.

00:39:50 **Researcher**
Were you able to orient yourself in the code and how

00:39:57 did the copilot influence your understanding?

00:40:05 **Participant**
Well, yeah, I was a little lost in the code,

00:40:08 but then later, with the help of the copilot,

00:40:12 those support links to access the code.

00:40:18 I was able to locate myself better, where I was

00:40:22 Where to make the changes

00:40:27

Researcher

Cool. And do you feel you understood the code?
how did the copilot influence your understanding?

00:40:39

00:40:46

Participant

Yes.

00:40:50

Actually, I think it was almost entirely the copilot
that taught me to navigate the code, locate where things were.

00:40:55

00:41:01

Researcher

OK, so it had a, it had a strong influence on your
understanding of the code?

00:41:07

00:41:09

Participant

Yes, except in the second activity where I had already looked at the
code more or less. I knew where the components that needed to be changed were

00:41:16

00:41:32

Researcher

Alright and how would you have solved these tasks without the help of the copilot?

00:41:42

Participant

it would take a little longer, because.

00:41:46

First I would test the code.

00:41:51

Then I would have to search the internet on how to run the project. Something I didn't
remember

00:42:00

Researcher

OK

00:42:04

Participant

Then I.

00:42:07

Probably would have to go to the website, stackoverflow, for example
to find possible questions/answers for my doubts.

00:42:17

00:42:26

Researcher

Cool.

00:42:26

Participant

Or look in the language documentation

00:42:30

It's interesting that in my work I don't use much artificial
intelligence

00:42:36

00:42:38

Researcher

ok

00:42:39

Participant

When I have a doubt, I first see if I find something
in previous codes that I have already done, right?

00:42:45

00:42:49

sometimes it's just something I forgot, how to do it

00:43:00

Then as a second alternative, I go straight to the documentation and search

00:43:17

As a last resort, I ask a colleague from the team

00:43:52

Researcher

ok

00:44:01

What aspects of the copilot did you like and which did you not like?

00:44:16

Participant

Hmm. Ah, I liked it a lot.

00:44:23

the code generated by the copilot is shown above the current code
the red and what is being removed

00:44:34

00:44:40

Researcher

Hmm, Hmm.

00:44:41

Participant

You can easily visualize the changes that will be made. You can choose which to apply

00:44:57

Researcher

Cool. Was there anything you didn't like?

00:45:08

Participant

I didn't have any.

00:45:11

Researcher

ok.

00:45:12

Participant

I liked it a lot.

00:45:13

Researcher

And during your interaction with the copilot
did you change your interaction approach?

00:45:18

00:45:29

Participant

I think. I think not.

00:45:51

Researcher

At the beginning you started asking key things
I noticed that you started using the task statements.

00:45:56

00:45:57

Participant

Ah, yes. Using the task descriptions the results were better.

00:46:13

Researcher

Ok, thank you for participating in our study, I will end the recording.