Ah, ich sehe genau das gleiche Muster wie bei deinem letzten Problem mit dem ersten Klick 👍  
Der Knackpunkt ist wieder: **Du fügst den EventListener erst in dem Moment hinzu, wenn du schon geklickt hast**.

Schauen wir uns deinen Code hier an:

workingFoldersContainer.addEventListener("click", function(e){

let folderId = e.target.dataset.id

let folderAddTaskBtn = document.querySelector(`.folder-add-task-btn[data-id=${folderId}]`)

if(folderAddTaskBtn){

folderAddTaskBtn.addEventListener("click", function(){

console.log("btn!")

})

}

})

🔎 Ablauf:

1. Du klickst auf den Button.
2. Dein „großer“ Listener (workingFoldersContainer) läuft los.
3. Dort **fügst du erst den EventListener hinzu**.
4. Deswegen passiert beim **ersten Klick nichts**, weil der neue Listener erst *nach* dem Klick existiert.
5. Beim **zweiten Klick** wird der neue Listener ausgeführt → „btn!“ erscheint.

**Lösung: Event Delegation richtig nutzen**

Du brauchst **keinen zweiten EventListener** im Inneren. Stattdessen prüfst du einfach, ob dein Klick von einem .folder-add-task-btn kam:

workingFoldersContainer.addEventListener("click", function(e){

// prüfe, ob der geklickte Button die Klasse hat

if (e.target.classList.contains("folder-add-task-btn")) {

let folderId = e.target.dataset.id;

console.log("btn für Folder:", folderId);

// Hier kannst du jetzt alles machen:

// - task-container für diesen Folder finden

// - Input einblenden

// - Aufgabe zum Array hinzufügen usw.

}

});

Damit:

* funktioniert es direkt beim ersten Klick ✅
* du brauchst keine addEventListener-Verschachtelung ✅
* der Code bleibt einfacher ✅

**Bonus: Zugriff auf das passende Task-Container-Div**

Wenn du gleich das richtige <div class="task-container" data-id="..."> brauchst:

const taskContainer = document.querySelector(`.task-container[data-id="${folderId}"]`);

taskContainer.classList.toggle("hide"); // Beispiel: sichtbar/unsichtbar

✨ Empfehlung:  
Verwende bei dynamisch erzeugten Elementen immer **Event Delegation** (so wie oben).  
Dann hast du keine „beim zweiten Klick“-Probleme mehr, weil die Events direkt im ersten Schritt erkannt werden.

Umschreiben:

/\* Aufgabe erteilen und mit Folder verknüpfen \*/

if (workingFoldersContainer) {

workingFoldersContainer.addEventListener("click", function (e) {

// Prüfen, ob auf einen "Create task"-Button geklickt wurde

if (e.target.classList.contains("folder-add-task-btn")) {

const folderId = e.target.dataset.id;

console.log("Create task Button für Folder:", folderId);

// Das passende Task-Container-Div finden

const taskContainer = document.querySelector(`.task-container[data-id="${folderId}"]`);

// Beispiel: sichtbar machen (oder später Input einfügen)

taskContainer.classList.toggle("hide");

// Zugriff auf das Folder-Objekt aus deinem Array

const targetFolder = myFolders.find(item => item.id === folderId);

if (targetFolder) {

console.log("Gefundener Folder:", targetFolder);

// hier kannst du dann z. B. targetFolder.tasks.push(...) machen

}

}

// Beispiel: wenn später noch andere Buttons (z. B. Delete Folder) dazu kommen:

if (e.target.textContent === "Delete Folder") {

const folderDiv = e.target.closest(".folder-div");

folderDiv.remove(); // löscht das ganze Folder-Div aus dem DOM

console.log("Folder gelöscht");

}

});

}

**Warum ist das besser?**

* **Kein zweites addEventListener** mehr → kein „zweiter Klick“-Problem.
* Du prüfst nur, ob das Event von einem bestimmten Button kam (classList.contains).
* Du kannst mehrere unterschiedliche Buttons (Create Task, Delete Folder, etc.) alle im selben Listener abfangen.
* Funktioniert sofort, auch wenn du später neue Folder ins DOM renderst.

Allgemeine Verbesserung

**Beispiel: Datenmodell in JavaScript**

// Alle Folder liegen in einem Array

let myFolders = [

{

id: "urgent-0", // eindeutige ID für den Folder

name: "Urgent", // Folder-Name

onTasks: 1, // noch offene Aufgaben

completedTasks: 1, // erledigte Aufgaben

cancelledTasks: 0, // abgebrochene Aufgaben

// Array mit Aufgaben (Tasks) in diesem Folder

tasks: [

{

id: "task-0", // eindeutige ID für die Aufgabe

text: "E-Mail an Chef schreiben",

status: "open" // mögliche Werte: "open", "completed", "cancelled"

},

{

id: "task-1",

text: "Präsentation vorbereiten",

status: "completed"

}

]

},

{

id: "shopping-1",

name: "Shopping",

onTasks: 2,

completedTasks: 0,

cancelledTasks: 0,

tasks: [

{

id: "task-0",

text: "Milch kaufen",

status: "open"

},

{

id: "task-1",

text: "Geschenk für Mama",

status: "open"

}

]

}

];

**🔹 Warum diese Struktur gut ist:**

* Jeder **Folder** hat seine eigenen Eigenschaften (id, name, Statistik).
* Jeder **Task** gehört zu genau einem Folder (über das tasks-Array).
* Der **Status** ist ein String → später kannst du leicht filtern oder zählen.
* IDs sind eindeutig (z. B. "urgent-0", "task-1") → macht es einfacher, gezielt Elemente zu finden.

**🔹 Beispiel: neue Aufgabe hinzufügen**

function addTaskToFolder(folderId, taskText) {

const folder = myFolders.find(f => f.id === folderId);

if (!folder) return;

const newTask = {

id: "task-" + folder.tasks.length,

text: taskText,

status: "open"

};

folder.tasks.push(newTask);

folder.onTasks++;

}

**🔹 Beispiel: Aufgabe erledigt markieren**

function completeTask(folderId, taskId) {

const folder = myFolders.find(f => f.id === folderId);

if (!folder) return;

const task = folder.tasks.find(t => t.id === taskId);

if (!task) return;

if (task.status === "open") {

task.status = "completed";

folder.onTasks--;

folder.completedTasks++;

}

}

✨ Mit dieser Struktur kannst du sehr leicht:

* neue Folder anlegen,
* Tasks hinzufügen,
* Tasks nach Status zählen,
* oder das Ganze in HTML rendern.

**Schritt 1: HTML-Grundgerüst**

In deiner Seite gibt es ja schon:

<div id="working-folders-container"></div>

Dort sollen alle Folder + Tasks rein.

Schritt 2: Render-Funktion

function renderFolders() {

// Container leeren

workingFoldersContainer.innerHTML = "";

// Alle Folder durchgehen

myFolders.forEach(folder => {

// Folder-Div erstellen

const folderDiv = document.createElement("div");

folderDiv.classList.add("folder-div");

folderDiv.dataset.id = folder.id;

// Name + Statistik

folderDiv.innerHTML = `

<h2>${folder.name}</h2>

<button type="button" class="delete-folder-btn" data-id="${folder.id}">Delete Folder</button>

<button type="button" class="folder-add-task-btn" data-id="${folder.id}">Create Task</button>

<div class="folders-statistik">

<h3>Completed: ${folder.completedTasks}</h3>

</div>

<div class="folders-statistik">

<h3>To do: ${folder.onTasks}</h3>

</div>

<div class="folders-statistik">

<h3>Cancelled: ${folder.cancelledTasks || 0}</h3>

</div>

`;

// Task-Container erstellen

const taskContainer = document.createElement("div");

taskContainer.classList.add("task-container");

taskContainer.dataset.id = folder.id;

// Tasks rendern

folder.tasks.forEach(task => {

const taskDiv = document.createElement("div");

taskDiv.classList.add("task");

taskDiv.dataset.id = task.id;

taskDiv.innerHTML = `

<span class="task-text ${task.status}">${task.text}</span>

<button class="complete-task-btn" data-folder-id="${folder.id}" data-task-id="${task.id}">✔</button>

<button class="cancel-task-btn" data-folder-id="${folder.id}" data-task-id="${task.id}">✘</button>

`;

taskContainer.appendChild(taskDiv);

});

folderDiv.appendChild(taskContainer);

workingFoldersContainer.appendChild(folderDiv);

});

}

**Schritt 3: Events mit Delegation**

Da die Folder/Tasks dynamisch entstehen, verwenden wir **einen Listener** am Container:

workingFoldersContainer.addEventListener("click", function(e) {

// Task hinzufügen

if (e.target.classList.contains("folder-add-task-btn")) {

const folderId = e.target.dataset.id;

const taskText = prompt("Neue Aufgabe:");

if (taskText) {

addTaskToFolder(folderId, taskText);

renderFolders();

}

}

// Task erledigen

if (e.target.classList.contains("complete-task-btn")) {

const folderId = e.target.dataset.folderId;

const taskId = e.target.dataset.taskId;

completeTask(folderId, taskId);

renderFolders();

}

// Task abbrechen

if (e.target.classList.contains("cancel-task-btn")) {

const folderId = e.target.dataset.folderId;

const taskId = e.target.dataset.taskId;

cancelTask(folderId, taskId);

renderFolders();

}

// Folder löschen

if (e.target.classList.contains("delete-folder-btn")) {

const folderId = e.target.dataset.id;

myFolders = myFolders.filter(f => f.id !== folderId);

renderFolders();

}

});

Schritt 4: Beispiel-Logik für Cancel Task

function cancelTask(folderId, taskId) {

const folder = myFolders.find(f => f.id === folderId);

if (!folder) return;

const task = folder.tasks.find(t => t.id === taskId);

if (!task) return;

if (task.status === "open") {

task.status = "cancelled";

folder.onTasks--;

folder.cancelledTasks = (folder.cancelledTasks || 0) + 1;

}

}

**Schritt 5: Start**

Zum Schluss einfach renderFolders() aufrufen, wenn du startest:

renderFolders();

Schritt 1: Statistik-Funktion schreiben

function getFolderStats(folder) {

const stats = {

completed: 0,

open: 0,

cancelled: 0

};

folder.tasks.forEach(task => {

if (task.status === "completed") stats.completed++;

else if (task.status === "cancelled") stats.cancelled++;

else stats.open++;

});

return stats;

}

**Schritt 2: Render anpassen**

Jetzt holen wir die Statistik pro Folder **live**, statt sie aus folder.completedTasks usw. zu nehmen:

function renderFolders() {

workingFoldersContainer.innerHTML = "";

myFolders.forEach(folder => {

const stats = getFolderStats(folder); // ✅ automatisch berechnen

const folderDiv = document.createElement("div");

folderDiv.classList.add("folder-div");

folderDiv.dataset.id = folder.id;

folderDiv.innerHTML = `

<h2>${folder.name}</h2>

<button type="button" class="delete-folder-btn" data-id="${folder.id}">Delete Folder</button>

<button type="button" class="folder-add-task-btn" data-id="${folder.id}">Create Task</button>

<div class="folders-statistik">

<h3>Completed: ${stats.completed}</h3>

</div>

<div class="folders-statistik">

<h3>To do: ${stats.open}</h3>

</div>

<div class="folders-statistik">

<h3>Cancelled: ${stats.cancelled}</h3>

</div>

`;

const taskContainer = document.createElement("div");

taskContainer.classList.add("task-container");

taskContainer.dataset.id = folder.id;

folder.tasks.forEach(task => {

const taskDiv = document.createElement("div");

taskDiv.classList.add("task");

taskDiv.dataset.id = task.id;

taskDiv.innerHTML = `

<span class="task-text ${task.status}">${task.text}</span>

<button class="complete-task-btn" data-folder-id="${folder.id}" data-task-id="${task.id}">✔</button>

<button class="cancel-task-btn" data-folder-id="${folder.id}" data-task-id="${task.id}">✘</button>

`;

taskContainer.appendChild(taskDiv);

});

folderDiv.appendChild(taskContainer);

workingFoldersContainer.appendChild(folderDiv);

});

}

**Schritt 3: Task-Update-Logik vereinfachen**

Weil die Statistik automatisch berechnet wird, brauchen wir keine Zähler im Folder mehr hoch/runterzählen.  
Beispiel:

function completeTask(folderId, taskId) {

const folder = myFolders.find(f => f.id === folderId);

if (!folder) return;

const task = folder.tasks.find(t => t.id === taskId);

if (!task) return;

task.status = "completed"; // ✅ nur Status setzen

}

function cancelTask(folderId, taskId) {

const folder = myFolders.find(f => f.id === folderId);

if (!folder) return;

const task = folder.tasks.find(t => t.id === taskId);

if (!task) return;

task.status = "cancelled"; // ✅ nur Status setzen

}