let tasks = JSON.parse(localStorage.getItem('tasks')) || [];

let lastRandomTaskIndex = null; // To store the index of the last picked random task

let currentlySelectedTask = null; // Add this at the beginning of your todo.js

**function pickRandomTask() {**

**if (tasks.length === 0) {**

**alert('No tasks available to pick.');**

**return;**

**}**

**clearTaskHighlight();** // Clear highlight from any previously picked task

// Generate a random index for main tasks only

**const randomTaskIndex = Math.floor(Math.random() \* tasks.length);**

// Highlight the picked main task and scroll into view

**let taskListItems = document.querySelectorAll('#task-list li');**

**let highlightedItem = null;**

**taskListItems.forEach(item => {**

// Check if the item is a main task and not a subtask

**if (item.dataset.taskIndex && parseInt(item.dataset.taskIndex) === randomTaskIndex) {**

**item.classList.add('highlighted-task');**

**highlightedItem = item;**

**}**

**});**

// Scroll the highlighted item into view

**if (highlightedItem) {**

**highlightedItem.scrollIntoView({ behavior: 'smooth', block: 'center' });**

**}**

**}**

**function clearTaskHighlight() {**

**let taskListItems = document.querySelectorAll('#task-list li');**

**taskListItems.forEach(item => {**

**item.classList.remove('highlighted-task');**

**});**

**}**

function addTask(isSubtask = false, parentIndex = null) {

let taskText;

if (isSubtask) {

taskText = document.getElementById(`new-subtask-${parentIndex}`).value;

} else {

taskText = document.getElementById('new-task').value;

}

let taskLines = taskText.split('\n').filter(line => line.trim() !== '');

if (isSubtask) {

// Add each line as a new subtask

taskLines.forEach(line => {

const newSubtask = { text: line.trim(), completed: false, subtasks: [], note: "" };

// Find the index of the first completed subtask

const firstCompletedIndex = tasks[parentIndex].subtasks.findIndex(subtask => subtask.completed);

if (firstCompletedIndex === -1) {

// If there are no completed subtasks, push to the end

tasks[parentIndex].subtasks.push(newSubtask);

} else {

// Insert new subtask before the first completed subtask

tasks[parentIndex].subtasks.splice(firstCompletedIndex, 0, newSubtask);

}

});

} else {

if (taskLines.length > 1) {

// Adding multiple tasks - prepend to the front in the correct order

taskLines.reverse().forEach(line => {

const newTask = { text: line.trim(), completed: false, subtasks: [], note: "" };

tasks.unshift(newTask);

});

} else if (taskLines.length === 1) {

// Adding a single task - add to the beginning

const newTask = { text: taskLines[0].trim(), completed: false, subtasks: [], note: "" };

tasks.unshift(newTask);

}

}

if (taskLines.length > 0) {

updateLocalStorage();

renderTasks();

if (isSubtask) {

document.getElementById(`new-subtask-${parentIndex}`).value = '';

document.getElementById(`new-subtask-${parentIndex}`).focus(); // Set focus back to the subtask input

} else {

document.getElementById('new-task').value = '';

// Optionally, set focus back to the main task input if needed

// document.getElementById('new-task').focus();

}

}

}

document.getElementById('new-task').addEventListener('keypress', function(event) {

if (event.key === 'Enter') {

addTask();

}

});

function deleteTask(index, parentIndex = null) {

if (parentIndex !== null) {

tasks[parentIndex].subtasks.splice(index, 1);

} else {

tasks.splice(index, 1);

}

updateLocalStorage();

renderTasks();

}

function toggleTaskCompletion(index, parentIndex = null) {

if (parentIndex !== null) {

tasks[parentIndex].subtasks[index].completed = !tasks[parentIndex].subtasks[index].completed;

if (tasks[parentIndex].subtasks[index].completed) {

// Move the completed subtask to the bottom of its parent's subtask list

let completedSubtask = tasks[parentIndex].subtasks.splice(index, 1)[0];

tasks[parentIndex].subtasks.push(completedSubtask);

}

} else {

tasks[index].completed = !tasks[index].completed;

if (tasks[index].completed) {

// Move the completed task to the bottom of the main task list

let completedTask = tasks.splice(index, 1)[0];

tasks.push(completedTask);

}

}

updateLocalStorage();

renderTasks();

}

function renderTasks() {

let taskList = document.getElementById('task-list');

taskList.innerHTML = '';

tasks.forEach((task, index) => {

taskList.appendChild(createTaskElement(task, index));

});

}

function createTaskElement(task, index, parentIndex = null) {

let taskItem = document.createElement('li');

taskItem.className = 'task-item';

// Use a different data attribute for tasks and subtasks

if (parentIndex === null) {

taskItem.dataset.taskIndex = index; // For main tasks

} else {

taskItem.dataset.subtaskIndex = index; // For subtasks

taskItem.dataset.parentIndex = parentIndex; // Include parent index for subtasks

}

// Task main content (checkbox, text, and controls)

let mainContent = document.createElement('div');

mainContent.className = 'main-content';

taskItem.appendChild(mainContent);

// Checkbox for completion

let checkbox = document.createElement('input');

checkbox.type = 'checkbox';

checkbox.checked = task.completed;

checkbox.onchange = () => toggleTaskCompletion(index, parentIndex);

mainContent.appendChild(checkbox);

// Task number label

let taskNumber = document.createElement('span');

taskNumber.textContent = `${index + 1}. `;

mainContent.appendChild(taskNumber);

// Task text

let taskText = document.createElement('span');

taskText.textContent = task.text;

taskText.className = 'task-name'; // Assigning a class

if (task.completed) {

taskText.classList.add('completed-task');

}

taskText.onclick = function() { // Make the task text clickable for editing

editTaskName(index, parentIndex);

};

mainContent.appendChild(taskText);

// Controls container

let controls = document.createElement('div');

mainContent.appendChild(controls);

// Handle subtasks

if (parentIndex === null) {

let subtaskInput = document.createElement('textarea'); // Change this line

subtaskInput.id = `new-subtask-${index}`;

subtaskInput.rows = 1; // Set the number of rows for the textarea

subtaskInput.placeholder = "Enter subtasks here..."; // Optional placeholder

subtaskInput.addEventListener('keypress', function(event) {

if (event.key === 'Enter' && !event.shiftKey) { // Allow shift+enter for new lines

event.preventDefault(); // Prevent default to avoid form submission

addTask(true, index);

}

});

controls.appendChild(subtaskInput);

let addSubtaskBtn = document.createElement('button');

addSubtaskBtn.textContent = 'Add Subtask';

addSubtaskBtn.onclick = () => addTask(true, index);

controls.appendChild(addSubtaskBtn);

}

// Delete button

let deleteBtn = document.createElement('button');

deleteBtn.textContent = 'Delete';

deleteBtn.onclick = () => deleteTask(index, parentIndex);

controls.appendChild(deleteBtn);

if (parentIndex !== null) {

let promoteBtn = document.createElement('button');

promoteBtn.textContent = 'Promote to Task';

promoteBtn.onclick = function(event) {

event.stopPropagation(); // Prevent the event from bubbling up

promoteSubtaskToTask(parentIndex, index);

};

controls.appendChild(promoteBtn);

}

// Subtask list

if (parentIndex === null && task.subtasks && task.subtasks.length > 0) {

let subtaskList = document.createElement('ul');

task.subtasks.forEach((subtask, subIndex) => {

subtaskList.appendChild(createTaskElement(subtask, subIndex, index));

});

taskItem.appendChild(subtaskList);

}

// Add onclick event for task selection

taskItem.oncontextmenu = function(event) {

event.preventDefault(); // Prevent the default right-click menu

selectTask(index, parentIndex);

return false; // Prevent the default right-click menu

};

// Add the random subtask button if the task has subtasks

if (parentIndex === null && task.subtasks && task.subtasks.length > 0) {

let pickRandomSubtaskBtn = document.createElement('button');

pickRandomSubtaskBtn.textContent = 'Pick Random Subtask';

pickRandomSubtaskBtn.onclick = function(event) {

event.stopPropagation(); // Prevent click event from bubbling up to the task item

pickRandomSubtask(index);

};

taskItem.appendChild(pickRandomSubtaskBtn);

}

// Move Custom Positions button

let moveCustomBtn = document.createElement('button');

moveCustomBtn.textContent = 'Move Custom';

moveCustomBtn.onclick = function(event) {

event.stopPropagation(); // Prevent the event from bubbling up

let positions = prompt('Enter the number of positions to move (negative for up, positive for down):');

// Check if the user entered a number and it's an integer

if (positions !== null && !isNaN(positions) && positions.trim() !== '' && Number.isInteger(Number(positions))) {

moveTaskByPositions(index, parseInt(positions), parentIndex);

} else {

alert('Please enter a valid integer number.');

}

};

controls.appendChild(moveCustomBtn);

// Move to Top button

let moveToTopBtn = document.createElement('button');

moveToTopBtn.textContent = 'Top';

moveToTopBtn.onclick = function(event) {

event.stopPropagation();

moveTaskToTop(index, parentIndex);

};

controls.appendChild(moveToTopBtn);

// Move to Bottom button

let moveToBottomBtn = document.createElement('button');

moveToBottomBtn.textContent = 'Bottom';

moveToBottomBtn.onclick = function(event) {

event.stopPropagation();

moveTaskToBottom(index, parentIndex);

};

controls.appendChild(moveToBottomBtn);

// Move 5 Rows Up button

let moveUpFiveBtn = document.createElement('button');

moveUpFiveBtn.innerHTML = '&uarr;5'; // Using HTML entity for 5-up arrow

moveUpFiveBtn.onclick = function(event) {

event.stopPropagation();

moveTaskFiveRowsUp(index, parentIndex);

};

controls.appendChild(moveUpFiveBtn);

// Move 5 Rows Down button

let moveDownFiveBtn = document.createElement('button');

moveDownFiveBtn.innerHTML = '&darr;5';

moveDownFiveBtn.onclick = function(event) {

event.stopPropagation();

moveTaskFiveRowsDown(index, parentIndex);

};

controls.appendChild(moveDownFiveBtn);

// Move Up button

let moveUpBtn = document.createElement('button');

moveUpBtn.innerHTML = '&uarr;'; // Using HTML entity for up arrow

moveUpBtn.onclick = function(event) {

event.stopPropagation();

moveTask(index, -1, parentIndex);

};

controls.appendChild(moveUpBtn);

// Move Down button

let moveDownBtn = document.createElement('button');

moveDownBtn.innerHTML = '&darr;'; // Using HTML entity for down arrow

moveDownBtn.onclick = function(event) {

event.stopPropagation();

moveTask(index, 1, parentIndex);

};

controls.appendChild(moveDownBtn);

// Text area for notes

**let noteTextArea = document.createElement('textarea');**

**noteTextArea.className = 'note-textarea';**

**noteTextArea.value = task.note || '';**

**noteTextArea.rows = 2;** // Set initial number of rows

**noteTextArea.oninput = function() {**

// Automatically adjust height to fit content

**this.style.height = 'auto';**

**this.style.height = (this.scrollHeight) + 'px';**

**};**

**noteTextArea.onchange = function(event) {**

**saveTaskNote(index, parentIndex, this.value);**

**};**

**mainContent.appendChild(noteTextArea);**

**return taskItem;**

**}**

function promoteSubtaskToTask(parentIndex, subtaskIndex) {

let subtask = tasks[parentIndex].subtasks.splice(subtaskIndex, 1)[0]; // Remove subtask from its parent

tasks.unshift(subtask); // Add subtask as a new task at the beginning of the tasks array

updateLocalStorage();

renderTasks();

}

function moveTaskFiveRowsDown(index, parentIndex = null) {

if (parentIndex === null) {

// Main task movement

if (index + 5 < tasks.length) {

const taskToMove = tasks.splice(index, 1)[0];

tasks.splice(index + 5, 0, taskToMove);

} else {

// If index + 5 exceeds array length, move to the end

const taskToMove = tasks.splice(index, 1)[0];

tasks.push(taskToMove);

}

} else {

// Subtask movement within a parent task

if (index + 5 < tasks[parentIndex].subtasks.length) {

const taskToMove = tasks[parentIndex].subtasks.splice(index, 1)[0];

tasks[parentIndex].subtasks.splice(index + 5, 0, taskToMove);

} else {

// If index + 5 exceeds array length, move to the end

const taskToMove = tasks[parentIndex].subtasks.splice(index, 1)[0];

tasks[parentIndex].subtasks.push(taskToMove);

}

}

updateLocalStorage();

renderTasks();

}

function moveTaskFiveRowsUp(index, parentIndex = null) {

if (parentIndex === null) {

// Main task movement

if (index - 5 >= 0) {

const taskToMove = tasks.splice(index, 1)[0];

tasks.splice(index - 5, 0, taskToMove);

} else {

// If index - 5 is less than 0, move to the start

const taskToMove = tasks.splice(index, 1)[0];

tasks.unshift(taskToMove);

}

} else {

// Subtask movement within a parent task

if (index - 5 >= 0) {

const taskToMove = tasks[parentIndex].subtasks.splice(index, 1)[0];

tasks[parentIndex].subtasks.splice(index - 5, 0, taskToMove);

} else {

// If index - 5 is less than 0, move to the start

const taskToMove = tasks[parentIndex].subtasks.splice(index, 1)[0];

tasks[parentIndex].subtasks.unshift(taskToMove);

}

}

updateLocalStorage();

renderTasks();

}

function saveTaskNote(index, parentIndex, noteContent) {

let taskToUpdate = parentIndex === null ? tasks[index] : tasks[parentIndex].subtasks[index];

taskToUpdate.note = noteContent;

updateLocalStorage();

}

function editTaskName(index, parentIndex = null) {

let taskToEdit = parentIndex === null ? tasks[index] : tasks[parentIndex].subtasks[index];

let newTaskName = prompt("Edit Task Name:", taskToEdit.text);

if (newTaskName !== null && newTaskName.trim() !== "") {

taskToEdit.text = newTaskName.trim();

updateLocalStorage();

renderTasks();

}

}

function moveTask(index, direction, parentIndex = null) {

let taskList;

// Check if it's a subtask

if (parentIndex !== null) {

// Handle subtask reordering

if ((direction === -1 && index === 0) || (direction === 1 && index === tasks[parentIndex].subtasks.length - 1)) {

// Cannot move beyond array bounds

return;

}

taskList = tasks[parentIndex].subtasks;

} else {

// Handle main task reordering

if ((direction === -1 && index === 0) || (direction === 1 && index === tasks.length - 1)) {

// Cannot move beyond array bounds

return;

}

taskList = tasks;

}

const taskToMove = taskList.splice(index, 1)[0];

taskList.splice(index + direction, 0, taskToMove);

updateLocalStorage();

renderTasks();

}

function moveTaskToTop(index, parentIndex = null) {

if (parentIndex === null) {

// Move main task to top

const taskToMove = tasks.splice(index, 1)[0];

tasks.unshift(taskToMove);

} else {

// Move subtask to top within its parent

const taskToMove = tasks[parentIndex].subtasks.splice(index, 1)[0];

tasks[parentIndex].subtasks.unshift(taskToMove);

}

updateLocalStorage();

renderTasks();

}

function moveTaskToBottom(index, parentIndex = null) {

let taskList, taskToMove;

if (parentIndex === null) {

// Handle main tasks

taskToMove = tasks.splice(index, 1)[0];

taskList = tasks;

} else {

// Handle subtasks

taskToMove = tasks[parentIndex].subtasks.splice(index, 1)[0];

taskList = tasks[parentIndex].subtasks;

}

// Find the first completed task's index

let firstCompletedIndex = taskList.findIndex(task => task.completed);

if (firstCompletedIndex === -1) {

// If no completed tasks, push to the end

taskList.push(taskToMove);

} else {

// Insert before the first completed task

taskList.splice(firstCompletedIndex, 0, taskToMove);

}

updateLocalStorage();

renderTasks();

}

**function pickRandomSubtask(taskIndex) {**

**clearTaskHighlight();** // Clear any previous highlights

**const task = tasks[taskIndex];**

**if (!task.subtasks || task.subtasks.length === 0) {**

**alert('This task has no subtasks.');**

**return;**

**}**

**const randomSubtaskIndex = Math.floor(Math.random() \* task.subtasks.length);**

// Highlight the randomly picked subtask

**let subtaskItems = document.querySelectorAll(`[data-parent-index="${taskIndex}"]`);**

**subtaskItems.forEach(item => {**

**if (parseInt(item.dataset.subtaskIndex) === randomSubtaskIndex) {**

**item.classList.add('highlighted-task');**

**}**

**});**

**}**

function selectTask(index, parentIndex = null) {

let taskItems = document.querySelectorAll('#task-list li');

let alreadySelected = false;

taskItems.forEach((item, idx) => {

if (parentIndex === null && parseInt(item.dataset.taskIndex) === index) {

if (item.classList.contains('selected-task')) {

alreadySelected = true;

}

item.classList.toggle('selected-task');

} else if (parentIndex !== null && parseInt(item.dataset.subtaskIndex) === index && parseInt(item.dataset.parentIndex) === parentIndex) {

if (item.classList.contains('selected-task')) {

alreadySelected = true;

}

item.classList.toggle('selected-task');

}

});

// If the task was already selected, clear all selections

if (alreadySelected) {

clearTaskSelection();

}

// Store the selected task index and parent index

currentlySelectedTask = { index, parentIndex };

}

function clearTaskSelection() {

let taskItems = document.querySelectorAll('#task-list li');

taskItems.forEach(item => {

item.classList.remove('selected-task');

});

currentlySelectedTask = null; // Reset the currently selected task

}

function updateLocalStorage() {

localStorage.setItem('tasks', JSON.stringify(tasks));

}

function exportTasksAndSubtasks() {

let exportText = 'Tasks:\n';

// Adding tasks-only list

tasks.forEach(task => {

exportText += task.text + '\n';

});

exportText += '\nSubtasks:\n';

// Iterate over tasks for subtasks

tasks.forEach(task => {

if (task.subtasks && task.subtasks.length > 0) {

exportText += '\n' + task.text + '\n'; // Task name as a header for subtasks

task.subtasks.forEach(subtask => {

exportText += subtask.text + '\n';

});

}

});

// Adding notes

exportText += '\nNotes:\n';

tasks.forEach(task => {

if (task.note && task.note.trim() !== '') {

exportText += '\nTask: ' + task.text + '\nNote:\n' + task.note + '\n';

}

if (task.subtasks) {

task.subtasks.forEach(subtask => {

if (subtask.note && subtask.note.trim() !== '') {

exportText += '\nSubtask: ' + subtask.text + '\nNote:\n' + subtask.note + '\n';

}

});

}

});

// Generate a filename with the current date

const currentDate = new Date().toISOString().slice(0, 10); // Format: YYYY-MM-DD

const filename = `tasks\_and\_subtasks\_${currentDate}.txt`;

// Trigger download

const dataStr = "data:text/plain;charset=utf-8," + encodeURIComponent(exportText);

const downloadAnchorNode = document.createElement('a');

downloadAnchorNode.setAttribute("href", dataStr);

downloadAnchorNode.setAttribute("download", filename);

document.body.appendChild(downloadAnchorNode);

downloadAnchorNode.click();

downloadAnchorNode.remove();

}

function moveTaskByPositions(index, positions, parentIndex = null) {

let taskList = parentIndex === null ? tasks : tasks[parentIndex].subtasks;

let newIndex = index + positions;

// Ensure the new index is within the bounds of the task list

newIndex = Math.max(0, Math.min(newIndex, taskList.length - 1));

// Move the task

const [taskToMove] = taskList.splice(index, 1);

taskList.splice(newIndex, 0, taskToMove);

updateLocalStorage();

renderTasks();

}

**function highlightNextTask() {**

**clearTaskHighlight(); // Clear highlight from any previously picked task**

**let nextTaskIndex = null;**

**// If a task is currently selected, find the next task index**

**if (currentlySelectedTask) {**

**nextTaskIndex = currentlySelectedTask.index + 1;**

**} else {**

**// If no task is selected, start from the beginning**

**nextTaskIndex = 0;**

**}**

**// Ensure the next task index is within the bounds of the task array**

**if (nextTaskIndex >= tasks.length) {**

**alert('No more tasks in the list.');**

**return;**

**}**

**// Highlight the next task and scroll into view**

**let taskListItems = document.querySelectorAll('#task-list li');**

**let highlightedItem = null;**

**taskListItems.forEach(item => {**

**if (item.dataset.taskIndex && parseInt(item.dataset.taskIndex) === nextTaskIndex) {**

**item.classList.add('highlighted-task');**

**highlightedItem = item;**

**}**

**});**

**// Scroll the highlighted item into view**

**if (highlightedItem) {**

**highlightedItem.scrollIntoView({ behavior: 'smooth', block: 'center' });**

**}**

**// Update the currently selected task**

**currentlySelectedTask = { index: nextTaskIndex, parentIndex: null };**

**}**

renderTasks(); // Initial render

window.addEventListener('keydown', function(event) {

// Check if Ctrl and Shift are held down and if the 'R' key is pressed

if (event.ctrlKey && event.shiftKey && event.key === 'R') {

event.preventDefault(); // Prevent the default action to avoid conflicts

pickRandomTask(); // Call your function to pick a random task

}

});

window.addEventListener('keydown', function(event) {

// Check if Ctrl and Shift are held down and if the 'F' key is pressed

if (event.ctrlKey && event.shiftKey && event.key === 'F') {

event.preventDefault(); // Prevent the default action to avoid conflicts

window.scrollTo(0, 0); // Scroll to the top of the page

}

});

**window.addEventListener('keydown', function(event) {**

**// Check if Ctrl and Shift are held down and if the 'E' key is pressed**

**if (event.ctrlKey && event.shiftKey && event.key === 'E') {**

**event.preventDefault(); // Prevent the default action to avoid conflicts**

**highlightNextTask(); // Call the function to highlight the next task**

**}**

**});**