**University of Wolverhampton**

**School of Engineering, Computational and Mathematical Sciences**

**5CS020 Human-Computer Interaction**

**Workshop 5 – Dynamic HTML User Interfaces**

In this workshop you will be experimenting with a dynamic user interface based on single web page apps.

**Creating a simple To-Do web application with Local Storage**

**Setting Up the HTML Structure**

First, create a new HTML file ("todo.html") and set up the basic structure:

**<!DOCTYPE html>**

**<html>**

**<head>**

**<title>To-Do List</title>**

**<link rel="stylesheet" href="style.css">**

**</head>**

**<body>**

**<h1>To-Do List</h1>**

**<form>**

**<input type="text" id="new-item" placeholder="Add New Item">**

**<button type="submit" id="add-btn">Add</button>**

**</form>**

**<ul id="items"> </ul>**

**<script src="app.js"></script>**

**</body>**

**</html>**

This creates a basic HTML structure with a heading, a form for adding new items, an unordered list for displaying the to-do items, and a script tag linking to an external JavaScript file called "app.js".

We've also added a link tag to the head section of the HTML file that specifies the path to the CSS file "style.css" using the href attribute. When the HTML file is loaded in a web browser, it will automatically load the CSS file and apply the styles to the elements in the HTML file. Make sure that the href attribute points to the correct file path for your CSS file.

**Adding CSS Styling**

Add some CSS styling to the app to make it look a bit nicer.

**body {**

**font-family: Arial, sans-serif;**

**}**

**h1 {**

**text-align: center;**

**}**

**form {**

**margin-bottom: 20px;**

**}**

**input[type="text"] {**

**padding: 10px;**

**font-size: 16px;**

**border: none;**

**border-bottom: 2px solid #ccc;**

**width: 70%;**

**}**

**button {**

**padding: 10px;**

**font-size: 16px;**

**border: none;**

**background-colour: #4CAF50;**

**color: #fff;**

**cursor: pointer;**

**}**

**button:hover {**

**background-colour: #3e8e41;**

**}**

**ul {**

**list-style-type: none;**

**padding: 0;**

**}**

**li {**

**display: flex;**

**justify-content: space-between;**

**align-items: center;**

**padding: 10px;**

**border-bottom: 1px solid #ccc;**

**}**

**li .delete-btn {**

**color: #f44336;**

**font-size: 16px;**

**cursor: pointer;**

**}**

This adds some basic styling to the app, including fonts, colours, and layout.

**Creating the JavaScript Functionality**

Next, create the JavaScript functionality for the app. This will involve adding event listeners to the form and delete buttons, and using local storage to store the to-do items. Create the following in a file called "app.js"

**// Select the form, input, and list elements from the HTML**

**const form = document.querySelector('form');**

**const input = document.querySelector('#new-item');**

**const itemsList = document.querySelector('#items');**

**// Load items from local storage, or create an empty array**

**// if there are none**

**const items = JSON.parse(localStorage.getItem('items')) || [];**

**// Display the items in the list**

**function displayItems() {**

**// Create an array of list items, one for each item in the items array**

**const itemsHTML = items.map((item, index) => {**

**// Each item in the array will be a list item with a span for**

**// the text and a button for deleting**

**return `**

**<li>**

**<span>${item}</span>**

**<button class="delete-btn" data-index="${index}">Delete</button>**

**</li>**

**`;**

**});**

**// Join the array of list items into a single string and add it to**

**// the list element in the HTML**

**itemsList.innerHTML = itemsHTML.join('');**

**}**

**// Add a new item to the list**

**function addItem(e) {**

**// Prevent the default form submission**

**e.preventDefault();**

**// Get the text value from the input and trim any whitespace**

**const text = input.value.trim();**

**if (text.length) {**

**// Add the text to the items array and store it in local storage**

**items.push(text);**

**localStorage.setItem('items', JSON.stringify(items));**

**// Clear the input field and display the updated list of items**

**input.value = '';**

**displayItems();**

**}**

**}**

**// Delete an item from the list**

**function deleteItem(e) {**

**// Check if the clicked element is a delete button**

**if (e.target.matches('.delete-btn')) {**

**// Get the index of the item from the data-index attribute**

**// of the delete button**

**const index = e.target.dataset.index;**

**// Remove the item from the items array and update local storage**

**items.splice(index, 1);**

**localStorage.setItem('items', JSON.stringify(items));**

**// Display the updated list of items**

**displayItems();**

**}**

**}**

**// Add event listeners to the form and list elements**

**form.addEventListener('submit', addItem);**

**itemsList.addEventListener('click', deleteItem);**

**// Call the displayItems function to initially display the items**

**// in the list**

**displayItems();**

This JavaScript code first selects the necessary elements from the HTML and loads the existing items from local storage, or creates an empty array if there are none.

Then, it defines a function displayItems that generates the HTML for the list items and updates the list element in the HTML.

It also defines two event listener functions, addItem and deleteItem, for adding and deleting items from the list.

Finally, it adds event listeners to the form and list elements and calls the displayItems function to initially display the items in the list.

With this setup, the app should be able to add new to-do items to the list, delete items from the list, and store the items in local storage so that they persist even when the browser is closed or the page is refreshed.

**Adding a priority to the tasks**

We are now going to add a "priority" field to each item in our To-Do app.

**Modify the HTML structure to include a priority field for each item:**

**<!DOCTYPE html>**

**<html>**

**<head>**

**<title>To-Do List</title>**

**<link rel="stylesheet" href="style.css">**

**</head>**

**<body>**

**<h1>To-Do List</h1>**

**<form>**

**<input type="text" id="new-item" placeholder="Add New Item">**

**<input type="number" id="priority" placeholder="Priority (1-5)">**

**<button type="submit" id="add-btn">Add</button>**

**</form>**

**<ul id="items"></ul>**

**<script src="app.js"></script>**

**</body>**

**</html>**

Here we've added an input field for the priority, which will be a number between 1 and 5. Modify the display items function to include the priority field for each item:

**function displayItems() {**

**const itemsHTML = items.map((item, index) => {**

**// Split the item string into the text and priority values**

**const [text, priority] = item.split(':');**

**// Create a list item with a span for the text and**

**// a span for the priority**

**return `**

**<li>**

**<span class="item-text">${text}</span>**

**<span class="item-priority">${priority}</span>**

**<button class="delete-btn" data-index="${index}">Delete</button>**

**</li>**

**`;**

**});**

**itemsList.innerHTML = itemsHTML.join('');**

**}**

Here we've modified the itemsHTML array to split each item string into its text and priority values, and created a new list item with a span for each value.

Create the priorityInput variable and modify the addItem function to include the priority field in the items array:

**const priorityInput = document.querySelector('#priority');**

**function addItem(e) {**

**e.preventDefault();**

**const text = input.value.trim();**

**const priority = priorityInput.value.trim();**

**if (text.length && priority.length && priority >= 1 && priority <= 5) {**

**items.push('${text}:${priority}');**

**localStorage.setItem('items', JSON.stringify(items));**

**input.value = '';**

**priorityInput.value = '';**

**displayItems();**

**}**

**}**

Here we've added a check to make sure that the priority value is between 1 and 5, and we've added the text and priority values to the items array as a single string separated by a colon. Modify the deleteItem function to account for the new format of the items array:

**function deleteItem(e) {**

**if (e.target.matches('.delete-btn')) {**

**const index = e.target.dataset.index;**

**items.splice(index, 1);**

**localStorage.setItem('items', JSON.stringify(items));**

**displayItems();**

**}**

**}**

Here we've removed the extra code that was extracting the text from the item string, as we can now delete the items from the items array directly. With these modifications, the app should now allow users to add to-do items with priority values, and display the items with both text and priority fields.

**Add Priority Sorting**

Add a "Sort" button to the HTML structure:

**<!DOCTYPE html>**

**<html>**

**<head>**

**<title>To-Do List</title>**

**<link rel="stylesheet" href="style.css">**

**</head>**

**<body>**

**<h1>To-Do List</h1>**

**<form>**

**<input type="text" id="new-item" placeholder="Add New Item">**

**<input type="number" id="priority" placeholder="Priority (1-5)">**

**<button type="submit" id="add-btn">Add</button>**

**<button type="button" id="sort-btn">Sort</button>**

**</form>**

**<ul id="items"></ul>**

**<script src="app.js"></script>**

**</body>**

**</html>**

Here we've added a "Sort" button to the form, which the user can click to sort the tasks in priority order. Add a "sortItems" function to the JavaScript code:

**function sortItems() {**

**// Sort the items array by priority, using a custom sort function**

**items.sort((a, b) => {**

**const priorityA = a.split(':')[1];**

**const priorityB = b.split(':')[1];**

**return priorityA - priorityB;**

**});**

**// Update the display with the sorted items**

**displayItems();**

**}**

Here we've added a new function, sortItems, which sorts the items array by priority using a custom sort function that extracts the priority value from each item string.

Add an event listener to the "Sort" button:

**const sortBtn = document.querySelector('#sort-btn');**

**sortBtn.addEventListener('click', sortItems);**

Here we've added an event listener to the "Sort" button that calls the sortItems function when the button is clicked.

With these modifications, the app should now allow users to sort the tasks in priority order by clicking the "Sort" button.

**Adding a "done" checkbox**

Modify the HTML structure to include a checkbox for each task:

**<!DOCTYPE html>**

**<html>**

**<head>**

**<title>To-Do List</title>**

**<link rel="stylesheet" href="style.css">**

**</head>**

**<body>**

**<h1>To-Do List</h1>**

**<form>**

**<input type="text" id="new-item" placeholder="Add New Item">**

**<input type="number" id="priority" placeholder="Priority (1-5)">**

**<button type="submit" id="add-btn">Add</button>**

**<button type="button" id="sort-btn">Sort</button>**

**</form>**

**<ul id="items"></ul>**

**<script src="app.js"></script>**

**</body>**

**</html>**

Here we will add a checkbox to the list item for each task in the displayItems function. Modify the addItem function to include the "done" field in the items array:

**function addItem(e) {**

**e.preventDefault();**

**const text = input.value.trim();**

**const priority = priorityInput.value.trim();**

**// Initialize the "done" field to false**

**const done = false;**

**if (text.length && priority.length && priority >= 1 && priority <= 5) {**

**// Modify the items array to add the text, priority,**

**// and done values as a single object**

**items.push({ text, priority, done });**

**localStorage.setItem('items', JSON.stringify(items));**

**input.value = '';**

**priorityInput.value = '';**

**displayItems();**

**}**

**}**

Here we define the addItem function, which is called when the form is submitted. We prevent the default form submission behavior with e.preventDefault(). We get the text and priority values from the input fields, and initialize the done field to false. If the text and priority values are valid, we add a new object to the items array with the text, priority, and done fields, save the updated items array to local storage, clear the input fields, and update the display.

Modify the displayItems function to include the "done" field for each task:

**function displayItems() {**

**const itemsHTML = items.map((item, index) => {**

**// Create a list item with a checkbox for the "done" field**

**return `**

**<li>**

**<input type="checkbox" class="item-done"**

**data-index="${index}" ${item.done ? 'checked' : ''}>**

**<span class="item-text ${item.done ? 'done' : ''}">${item.text}</span>**

**<span class="item-priority">${item.priority}</span>**

**<button class="delete-btn" data-index="${index}">Delete</button>**

**</li>**

**`;**

**});**

**itemsList.innerHTML = itemsHTML.join('');**

**}**

Here we define the displayItems function, which maps over the items array to create a list item for each item. We add a checkbox for the done field, which is checked if the task is marked as done. We also add a class of "done" to the text span if the task is marked as done. Finally, we join the itemsHTML array into a single string and set it as the innerHTML of the items list.

Modify the toggleDone function to update the "done" field:

**function toggleDone(e) {**

**if (e.target.matches('.item-done')) {**

**const index = e.target.dataset.index;**

**// Update the "done" field of the task in the items array**

**items[index].done = e.target.checked;**

**localStorage.setItem('items', JSON.stringify(items));**

**displayItems();**

**}**

**}**

Here we define the toggleDone function, which is called when a checkbox is checked or unchecked. If the target of the click event matches a checkbox, we get the index of the item to be updated from the data-index attribute of the checkbox, update the done field of the item in the items array with the checked state of the checkbox, save the updated items array to local storage, and update the display.

**Adding colour based on priority**

We are going to modify the displayItems function to add a background colour to each list item based on the priority and done status of the item:

**function displayItems() {**

**const itemsHTML = items.map((item, index) => {**

**const priority = item.priority;**

**const done = item.done;**

**// Calculate the colour for the background based on the priority**

**// and done status of the item**

**const hue = done ? 120 : 60 \* (5 - priority);**

**const background = 'hsl(${hue}, 70%, 85%)';**

**// Add the background colour to the style attribute of the list item**

**return '**

**<li style="background-colour: ${background}">**

**<input type="checkbox" class="item-done"**

**data-index="${index}" ${done ? 'checked' : ''}>**

**<span class="item-text ${done ? 'done' : ''}">${item.text}</span>**

**<span class="item-priority">${priority}</span>**

**<button class="delete-btn" data-index="${index}">Delete</button>**

**</li>**

**';**

**});**

**itemsList.innerHTML = itemsHTML.join('');**

**}**

We get the priority and done fields from the item object, and calculate the hue of the background colour based on the priority and done status. If the item is marked as done, the hue is set to 120 (green), otherwise it is set to a value between 60 (yellow) and 300 (red) based on the priority (with higher priority items being more red).

We add the background colour to the style attribute of the list item, so that each item is displayed with a background colour that corresponds to its priority and done status.

**Re-ordering items via dragging and dropping**

We are going to modify the displayItems function that adds drag and drop functionality to re-order the items:

**function displayItems() {**

**const itemsHTML = items.map((item, index) => {**

**const priority = item.priority;**

**const done = item.done;**

**const hue = done ? 120 : 60 + 48 \* (5 - priority);**

**const background = 'hsl(${hue}, 70%, 85%)';**

**return `**

**<li style="background-colour: ${background}" draggable="true"**

**data-index="${index}">**

**<input type="checkbox" class="item-done" ${done ? 'checked' : ''}>**

**<span class="item-text ${done ? 'done' : ''}">${item.text}</span>**

**<span class="item-priority">${priority}</span>**

**<button class="delete-btn">Delete</button>**

**</li>**

**`;**

**});**

**itemsList.innerHTML = itemsHTML.join('');**

**const listItems = itemsList.querySelectorAll('li');**

**listItems.forEach((item) => {**

**item.addEventListener('dragstart', handleDragStart);**

**item.addEventListener('dragover', handleDragOver);**

**item.addEventListener('drop', handleDrop);**

**});**

**}**

**function handleDragStart(e) {**

**e.dataTransfer.effectAllowed = 'move';**

**e.dataTransfer.setData('text/plain', e.target.dataset.index);**

**e.target.classList.add('dragging');**

**}**

**function handleDragOver(e) {**

**e.preventDefault();**

**e.dataTransfer.dropEffect = 'move';**

**const target = e.target.closest('li');**

**const dragging = itemsList.querySelector('.dragging');**

**if (target !== dragging && target) {**

**const targetRect = target.getBoundingClientRect();**

**const draggingRect = dragging.getBoundingClientRect();**

**if (e.clientY - targetRect.top < targetRect.height / 2) {**

**itemsList.insertBefore(dragging, target);**

**} else {**

**itemsList.insertBefore(dragging, target.nextSibling);**

**}**

**const items = Array.from(itemsList.querySelectorAll('li')).map((item, index) =>**

**{**

**item.dataset.index = index;**

**return {**

**text: item.querySelector('.item-text').textContent,**

**priority: item.querySelector('.item-priority').textContent,**

**done: item.querySelector('.item-done').checked**

**};**

**});**

**localStorage.setItem('items', JSON.stringify(items));**

**}**

**}**

**function handleDrop(e) {**

**e.preventDefault();**

**}**

**// Add event listener to the items list for the delete button click event**

**itemsList.addEventListener('click', (e) => {**

**if (e.target.matches('.delete-btn')) {**

**const index = e.target.parentElement.dataset.index;**

**items.splice(index, 1);**

**localStorage.setItem('items', JSON.stringify(items));**

**displayItems();**

**}**

**});**

**// Add event listener to the items list for the checkbox change event**

**itemsList.addEventListener('change', (e) => {**

**if (e.target.matches('.item-done')) {**

**const index = e.target.parentElement.dataset.index;**

**items[index].done = e.target.checked;**

**localStorage.setItem('items', JSON.stringify(items));**

**displayItems();**

**}**

**});**

**// Display the items on page load**

**displayItems();**

We have added the draggable attribute and data-index attribute to each list item. The draggable attribute makes the list item draggable, and the data-index attribute stores the index of the item in the items array.

**Challenge Task**

See if you can add a "complete by" date field to the items with a date picker (<https://www.w3schools.com/tags/att_input_type_date.asp>)