**University of Wolverhampton**

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

**5CS020 Human-Computer Interaction**

**Workshop 6 – Dynamic HTML User Interfaces with server database storage**

In this workshop you will be connecting a dynamic web app with a server-based database system.

**Converting the previous To-Do web application to work with the Mi-Linux server and MySQL database system**

**Modifying the App.js**

We won't need to modify the HTML file from the previous week.

However, we will need to change the JavaScript file and structure to convert the app from using localStorage to using MySQL on a server via PHP, we'll need to make some changes to the JavaScript code to use AJAX requests to send and receive data from the server:

**// Select the form, input fields, and items list from the HTML**

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

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

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

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

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

**// Function to display the items in the items list**

**function displayItems(items) {**

**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-color: ${background}" draggable="true"**

**data-id="${item.id}">**

**<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.id);**

**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.id = index + 1;**

**return {**

**id: item.dataset.id,**

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

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

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

**};**

**});**

**updateItems(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 id = e.target.parentElement.dataset.id;**

**deleteItem(id);**

**}**

**});**

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

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

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

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

**const done = e.target.checked;**

**updateItem(id, { done });**

**}**

**});**

**// Function to add a new item to the items list**

**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) {**

**// Send a POST request to the server to add the new item to the database**

**const data = {**

**text,**

**priority,**

**done**

**};**

**fetch('add\_item.php', {**

**method: 'POST',**

**body: JSON.stringify(data)**

**})**

**.then(response => response.json())**

**.then(item => {**

**// Add the item to the items array and display the updated list**

**items.push(item);**

**displayItems(items);**

**});**

**input.value = '';**

**priorityInput.value = '';**

**}**

**}**

**// Function to delete an item from the items list**

**function deleteItem(id) {**

**// Send a POST request to the server to delete the item from the database**

**fetch('delete\_item.php', {**

**method: 'POST',**

**body: JSON.stringify({**

**id**

**})**

**})**

**.then(() => {**

**// Remove the item from the items array and display the updated list**

**const index = items.findIndex(item => item.id === id);**

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

**displayItems(items);**

**});**

**}**

**// Function to update an item in the items list**

**function updateItem(id, updates) {**

**// Send a POST request to the server to update the item in the database**

**fetch('update\_item.php', {**

**method: 'POST',**

**body: JSON.stringify({**

**id,**

**updates**

**})**

**})**

**.then(() => {**

**// Update the item in the items array and display the updated list**

**const index = items.findIndex(item => item.id === id);**

**Object.assign(items[index], updates);**

**displayItems(items);**

**});**

**}**

**// Function to update all items in the items list**

**function updateItems(items) {**

**// Send a POST request to the server to update all items in the database**

**fetch('update\_items.php', {**

**method: 'POST',**

**body: JSON.stringify(items)**

**});**

**}**

**// Function to sort the items in the items list by priority**

**function sortItems() {**

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

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

**const priorityA = a.priority;**

**const priorityB = b.priority;**

**return priorityB - priorityA;**

**});**

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

**displayItems(items);**

**}**

**// Add event listeners to the form, items list, and sort button**

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

**document.querySelector('#sort-btn').addEventListener('click', sortItems);**

**// Display the items on page load by sending a GET request to the server to retrieve the items from the database**

**fetch('get\_items.php')**

**.then(response => response.json())**

**.then(data => {**

**items.push(...data);**

**displayItems(items);**

**});**

As you can see, the vast majority of the original JavaScript code doesn't change. It's only the parts that originally stored data into the localStorage that changes.

**Server-side PHP files**

For the web app to work with the MySQL database system on the server, we will need some PHP files to be set up on the server to communicate with the database as an intermediary. You have done very similar things in 4CS017 Internet Software Architecture in Level 4.

**"config.php" – Setting the connection between PHP and MySQL:**

**<?php**

**// Define the database connection parameters**

**define('DB\_HOST', 'localhost');**

**define('DB\_USER', 'username'); //student ID e.g. 1234567**

**define('DB\_PASS', 'password'); // your MySQL password**

**define('DB\_NAME', 'database\_name'); // Your MySQL database, normally 'db1234567'**

**// Create a new MySQLi object and connect to the database**

**$mysqli = new mysqli(DB\_HOST, DB\_USER, DB\_PASS, DB\_NAME);**

**if ($mysqli->connect\_error) {**

**die('Connect Error (' . $mysqli->connect\_errno . ') '**

**. $mysqli->connect\_error);**

**}**

**"add\_item.php" – Adding an item to the MySQL database**

**<?php**

**// Include the database connection**

**require\_once 'config.php';**

**// Get the JSON request body**

**$request\_body = file\_get\_contents('php://input');**

**$data = json\_decode($request\_body, true);**

**// Insert the new item into the database**

**$stmt = $mysqli->prepare('INSERT INTO items (text, priority, done) VALUES (?, ?, ?)');**

**$stmt->bind\_param('sii', $data['text'], $data['priority'], $data['done']);**

**$stmt->execute();**

**// Get the ID of the new item**

**$id = $mysqli->insert\_id;**

**// Select the new item from the database**

**$result = $mysqli->query("SELECT \* FROM items WHERE id = $id");**

**$item = $result->fetch\_assoc();**

**// Return the new item as JSON**

**header('Content-Type: application/json');**

**echo json\_encode($item);**

**"delete\_item.php" – Deleting an item from the MySQL database:**

**<?php**

**// Include the database connection**

**require\_once 'config.php';**

**// Get the JSON request body**

**$request\_body = file\_get\_contents('php://input');**

**$data = json\_decode($request\_body, true);**

**// Delete the item from the database**

**$stmt = $mysqli->prepare('DELETE FROM items WHERE id = ?');**

**$stmt->bind\_param('i', $data['id']);**

**$stmt->execute();**

**get\_items.php - What it says on the tin**

**<?php**

**// Include the database connection**

**require\_once 'config.php';**

**// Select all items from the database**

**$result = $mysqli->query('SELECT \* FROM items');**

**$items = [];**

**while ($row = $result->fetch\_assoc()) {**

**$items[] = $row;**

**}**

**// Return the items as JSON**

**header('Content-Type: application/json');**

**echo json\_encode($items);**

**update\_item.php - updates the item in the database**

**<?php**

**// Include the database connection**

**require\_once 'config.php';**

**// Get the JSON request body**

**$request\_body = file\_get\_contents('php://input');**

**$data = json\_decode($request\_body, true);**

**// Update the item in the database**

**$updates = [];**

**foreach ($data['updates'] as $key => $value) {**

**$updates[] = "$key = $value";**

**}**

**$stmt = $mysqli->prepare('UPDATE items SET ' . implode(', ', $updates) . ' WHERE id = ?');**

**$stmt->bind\_param('i', $data['id']);**

**$stmt->execute();**

**update\_items.php - updates all the items in the database in one go**

**<?php**

**// Include the database connection**

**require\_once 'config.php';**

**// Get the JSON request body**

**$request\_body = file\_get\_contents('php://input');**

**$items = json\_decode($request\_body, true);**

**// Update all items in the database**

**foreach ($items as $item) {**

**$id = $item['id'];**

**$text = $item['text'];**

**$priority = $item['priority'];**

**$done = $item['done'];**

**$stmt = $mysqli->prepare('UPDATE items SET text=?, priority=?, done=? WHERE id=?');**

**$stmt->bind\_param('siii', $text, $priority, $done, $id);**

**$stmt->execute();**

**}**

**"create\_items.php" – Creating the items table on MySQL**

Before the database can be used, the table "items" must be created:

**<?php**

**// Define the database connection parameters**

**define('DB\_HOST', 'localhost');**

**define('DB\_USER', '1234567');**

**define('DB\_PASS', 'password');**

**define('DB\_NAME', 'db1234567');**

**// Create a new MySQLi object and connect to the database**

**$mysqli = new mysqli(DB\_HOST, DB\_USER, DB\_PASS, DB\_NAME);**

**if ($mysqli->connect\_error) {**

**die('Connect Error (' . $mysqli->connect\_errno . ') ' . $mysqli->connect\_error);**

**}**

**// Define the SQL query to create the items table**

**$sql = "DROP TABLE items";**

**// Execute the SQL query**

**if ($mysqli->query($sql) === TRUE) {**

**echo "Table items dropped\n";**

**} else {**

**echo "Error dropping table: " . $mysqli->error;**

**}**

**// Define the SQL query to create the items table**

**$sql = "CREATE TABLE items (**

**id INT(11) UNSIGNED AUTO\_INCREMENT PRIMARY KEY,**

**text VARCHAR(255) NOT NULL,**

**priority INT(11) NOT NULL,**

**done INT(1) NOT NULL DEFAULT 0**

**)";**

**// Execute the SQL query**

**if ($mysqli->query($sql) === TRUE) {**

**echo "Table items created successfully";**

**} else {**

**echo "Error creating table: " . $mysqli->error;**

**}**

**Explanation of the PHP files**

1. config.php: This file defines the database connection parameters and creates a new MySQLi object to connect to the database.
2. add\_item.php: This file receives a POST request containing the details of a new item to be added to the database. It inserts the new item into the database, retrieves the ID of the new item, selects the new item from the database, and returns the new item as JSON.
3. delete\_item.php: This file receives a POST request containing the ID of an item to be deleted from the database. It deletes the item from the database.
4. get\_items.php: This file receives a GET request to retrieve all items from the database. It selects all items from the database, encodes them as JSON, and returns them.
5. update\_item.php: This file receives a POST request containing the ID and updates for an item to be updated in the database. It updates the item in the database with the new values.
6. update\_items.php: This file receives a POST request containing updates for multiple items to be updated in the database. It updates all items in the database with the new values.
7. Create\_itmes.php: This file creates the "items" table in the database with the appropriate columns (id, text, priority, done) to store the items, when it is run.

These PHP files should be placed on mi-linux in the "public\_html" folder, and the app's fetch requests should be updated with the correct URLs to access these files.

If you are not sure how to do this, refer back to your workshops in 4CS017 Internet Software Architecture in Level 4.

**Challenge Task**

As it was before, 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>)