**I.**HTML Structure  
  
**Index.html**

This file is the primary interface for the app. It includes sections such as:

* A navigation bar with a search feature.
* A list of grocery items displayed in a grid format.
* An editable form to update grocery item details.
* A cart section where users can view items added for checkout.

**Login.html**

A simple login page where users provide credentials to log into the grocery dashboard

**II.**CSS Stylesheet.  
  
The app uses a stylesheet to ensure a responsive and visually appealing design. Key aspects:

* The phone container is designed to resemble a mobile interface.
* The login form and item list have been styled for clarity and user-friendliness.
* Buttons and interactions have hover effects to enhance the user experience.

**III.** **app.js Functionality**

### **Overview**

This script manages a grocery store app where users can view, sort, search, edit, and manage items in their cart. It also includes functionality for checking out.

### **Detailed Line-by-Line Explanation**

// Wait for the entire DOM to load before executing the script

document.addEventListener('DOMContentLoaded', () => {

* **Purpose**: Ensures that the script runs only after the HTML document has been fully loaded and parsed.

// Grab the element that will display the list of items

const itemsContainer = document.getElementById('items-container');

const sortOptions = document.getElementById('sort-options');

* **Purpose**: Gets references to the DOM elements where items will be displayed and where sorting options are selected.

// Grab the form container for editing items

const editFormContainer = document.getElementById('edit-form-container');

// Grab the actual form used for editing an item

const editForm = document.getElementById('editForm');

* **Purpose**: Gets references to the form container and form itself used for editing item details.

// Grab various input fields for editing an item's properties

const editName = document.getElementById('editName');

const editBrand = document.getElementById('editBrand');

const editPrice = document.getElementById('editPrice');

const editWeight = document.getElementById('editWeight');

const editQuantity = document.getElementById('editQuantity');

const editStore = document.getElementById('editStore');

const editCategory = document.getElementById('editCategory');

const editImage = document.getElementById('editImage');

* **Purpose**: Gets references to the input fields within the edit form for updating item details.

// Grab the cart items list and total price display elements

const cartItems = document.getElementById('cart-items');

const cartTotal = document.getElementById('cart-total');

* **Purpose**: Gets references to the elements displaying cart items and the total price.

// Grab the search input and button elements

const searchInput = document.getElementById('searchInput');

const searchBtn = document.getElementById('searchBtn');

* **Purpose**: Gets references to the search input field and button for searching items.

// Grab the checkout button and message container

const checkoutBtn = document.getElementById('checkoutBtn');

const checkoutMessage = document.getElementById('checkout-message');

* **Purpose**: Gets references to the checkout button and the message container for displaying checkout status.

// Array to store the items available in the grocery store

let items = [

// Example item objects with properties like name, brand, price, etc.

{ name: 'Apples', brand: 'FreshFarm', price: 1.99, weight: '1kg', quantity: '1 units', store: 'Ambatogrocery', category: 'Produce', image: 'image/apple.png' },

{ name: 'Milk', brand: 'Organic Valley', price: 3.99, weight: '1L', quantity: '1 bottle', store: 'Ambatogrocery', category: 'Dairy', image: 'image/milk.png' },

{ name: 'Bananas', brand: 'FreshLand', price: 2.49, weight: '1kg', quantity: '3 bunches', store: 'Ambatogrocery', category: 'Produce', image: 'image/banana.png' },

{ name: 'Eggs', brand: 'HappyHens', price: 4.29, weight: '12 pcs', quantity: '1 dozen', store: 'Ambatogrocery', category: 'Dairy', image: 'image/egg.png' },

{ name: 'Orange', brand: 'FreshFarm', price: 1.79, weight: '1kg', quantity: '1 units', store: 'Ambatogrocery', category: 'Produce', image: 'image/orange.png' },

{ name: 'Chocolate', brand: 'FreshFarm', price: 12.79, weight: '2kg', quantity: '1 units', store: 'Ambatogrocery', category: 'Diary', image: 'image/choco.png' },

];

* **Purpose**: Defines an array of items with various properties like name, brand, price, etc.

// Array to store items added to the cart

let cart = [];

* **Purpose**: Initializes an empty array to store items that users add to their cart.

// Variable to keep track of the item being edited (by index)

let editingIndex = null;

* **Purpose**: Holds the index of the item currently being edited, if any.

// Function to display items in the itemsContainer

function displayItems(filteredItems = items) {

itemsContainer.innerHTML = ''; // Clear current items

filteredItems.forEach((item, index) => { // Iterate over each item

const itemDiv = document.createElement('div'); // Create a div for each item

itemDiv.classList.add('item'); // Add a class for styling

// Populate the item div with item details and buttons

itemDiv.innerHTML = `

<img src="${item.image || '/image/banana.png'}" alt="${item.name}" class="product-image">

<div class="product-item">

<p><strong>Product Name:</strong> ${item.name}</p>

<p><strong>Brand:</strong> ${item.brand}</p>

<p><strong>Price:</strong> $${item.price.toFixed(2)}</p>

<p><strong>Weight/Volume:</strong> ${item.weight}</p>

<p><strong>Quantity:</strong> ${item.quantity}</p>

<p><strong>Store:</strong> ${item.store}</p>

<p><strong>Category:</strong> ${item.category}</p>

</div>

<button class="edit-btn" data-index="${index}">Edit</button>

<button class="add-to-cart-btn" data-index="${index}">Add to Cart</button>

`;

itemsContainer.appendChild(itemDiv); // Append the item div to the container

});

}

* **Purpose**: Displays items by creating HTML elements for each item and appending them to the container. Allows for editing and adding items to the cart.

// Function to sort items based on selected criteria

function sortItems(criteria) {

let sortedItems = [...items]; // Create a copy of the items array

switch (criteria) {

case 'name':

sortedItems.sort((a, b) => a.name.localeCompare(b.name)); // Sort alphabetically by name

break;

case 'price-asc':

sortedItems.sort((a, b) => a.price - b.price); // Sort by price (low to high)

break;

case 'price-desc':

sortedItems.sort((a, b) => b.price - a.price); // Sort by price (high to low)

break;

case 'category':

sortedItems.sort((a, b) => a.category.localeCompare(b.category)); // Sort alphabetically by category

break;

default:

break;

}

displayItems(sortedItems); // Display the sorted items

}

* **Purpose**: Sorts items based on the selected criteria (e.g., name, price) and then displays them.

// Function to open the edit form for a specific item

function openEditForm(index) {

const item = items[index]; // Get the item to be edited

editingIndex = index; // Save the index of the item being edited

// Populate the form inputs with the item's details

editName.value = item.name;

editBrand.value = item.brand;

editPrice.value = item.price;

editWeight.value = item.weight;

editQuantity.value = item.quantity;

editStore.value = item.store;

editCategory.value = item.category;

editImage.value = ''; // Clear the image input field

// Show the edit form

editFormContainer.classList.remove('hidden');

}

* **Purpose**: Opens the edit form with pre-filled values for the item being edited.

// Function to save the edits made to the item

function saveEdit(e) {

e.preventDefault(); // Prevent the form from submitting

if (editingIndex !== null) { // Check if an item is being edited

const item = items[editingIndex]; // Get the item being edited

// Update the item's details with the form inputs

item.name = editName.value;

item.brand = editBrand.value;

item.price = parseFloat(editPrice.value);

item.weight = editWeight.value;

item.quantity = editQuantity.value;

item.store = editStore.value;

item.category = editCategory.value;

// Handle updating the image if a new file is uploaded

const file = editImage.files[0];

if (file) {

const reader = new FileReader(); // Create a FileReader to read the image

reader.onload = function(event) {

item.image = event.target.result; // Set the item's image to the uploaded image

displayItems(); // Redisplay the updated items

};

reader.readAsDataURL(file); // Read the image file as a data URL

} else {

displayItems(); // Redisplay items without changing the image

}

// Hide the edit form after saving

editFormContainer.classList.add('hidden');

editingIndex = null; // Reset the editing index

}

}

* **Purpose**: Saves edits to the item, updates the list, and hides the edit form. Handles image updates if a new file is provided.

// Function to add an item to the cart

function addToCart(index) {

const item = items[index]; // Get the selected item

const existingCartItem = cart.find(cartItem => cartItem.name === item.name); // Check if item is already in the cart

// If the item is already in the cart, update the quantity

if (existingCartItem) {

existingCartItem.quantity = `${parseInt(existingCartItem.quantity) + parseInt(item.quantity)} units`;

} else {

cart.push({ ...item }); // If not, add the item to the cart

}

updateCart(); // Update the cart display

}

* **Purpose**: Adds an item to the cart. If the item is already in the cart, it updates the quantity; otherwise, it adds a new entry.

// Function to remove an item from the cart

function removeFromCart(index) {

cart.splice(index, 1); // Remove the item from the cart array

updateCart(); // Update the cart display

}

* **Purpose**: Removes an item from the cart and updates the display.

// Function to remove an item from the list of available items

function removeItem(index) {

items.splice(index, 1); // Remove the item from the items array

displayItems(); // Redisplay the remaining items

}

* **Purpose**: Removes an item from the available items list and updates the display.

// Function to update the cart display

function updateCart() {

cartItems.innerHTML = ''; // Clear the current cart display

let total = 0; // Initialize the total price

cart.forEach((cartItem, index) => { // Iterate over each cart item

const itemDiv = document.createElement('li'); // Create a list item for each cart item

// Populate the cart item with details

itemDiv.innerHTML = `

<p><strong>Product Name:</strong> ${cartItem.name}</p>

<p><strong>Quantity:</strong> ${cartItem.quantity}</p>

<p><strong>Price:</strong> $${cartItem.price.toFixed(2)}</p>

<button class="remove-from-cart-btn" data-index="${index}">Delete</button>

`;

cartItems.appendChild(itemDiv); // Append the cart item to the cart display

total += cartItem.price; // Add the price to the total

});

cartTotal.textContent = total.toFixed(2); // Update the total price display

}

* **Purpose**: Updates the cart display, calculating the total price and showing each cart item.

// Function to search items based on the search input

function searchItems() {

const query = searchInput.value.toLowerCase(); // Get the search query and convert to lowercase

// Filter the items based on whether their name, brand, store, or category matches the query

const filteredItems = items.filter(item =>

item.name.toLowerCase().includes(query) ||

item.brand.toLowerCase().includes(query) ||

item.store.toLowerCase().includes(query) ||

item.category.toLowerCase().includes(query)

);

displayItems(filteredItems); // Display the filtered items

}

* **Purpose**: Filters items based on the search query and updates the display.

// Function to handle checkout and clear the cart

function handleCheckout() {

cart = []; // Empty the cart

updateCart(); // Update the cart display

checkoutMessage.classList.remove('hidden'); // Show a checkout success message

setTimeout(() => checkoutMessage.classList.add('hidden'), 3000); // Hide the message after 3 seconds

}

* **Purpose**: Handles the checkout process by clearing the cart and displaying a success message.

// Add event listeners to handle searching and checkout

searchBtn.addEventListener('click', searchItems);

searchInput.addEventListener('keyup', searchItems);

checkoutBtn.addEventListener('click', handleCheckout);

* **Purpose**: Adds event listeners for searching items and checking out.

// Add event listener to handle item clicks for editing, adding to cart, or removing

itemsContainer.addEventListener('click', (e) => {

const index = e.target.getAttribute('data-index'); // Get the index of the clicked item

if (e.target.classList.contains('edit-btn')) {

openEditForm(index); // Open the edit form if the "Edit" button was clicked

} else if (e.target.classList.contains('add-to-cart-btn')) {

addToCart(index); // Add to cart if the "Add to Cart" button was clicked

} else if (e.target.classList.contains('remove-btn')) {

removeItem(index); // Remove the item if the "Remove" button was clicked

}

});

* **Purpose**: Adds event listeners to handle clicks on item-related buttons (edit, add to cart, remove).

// Add event listener to handle removing items from the cart

cartItems.addEventListener('click', (e) => {

if (e.target.classList.contains('remove-from-cart-btn')) {

const index = e.target.getAttribute('data-index'); // Get the index of the cart item

removeFromCart(index); // Remove the item from the cart

}

});

* **Purpose**: Adds an event listener to handle clicks on cart item removal buttons.

// Add event listener to handle saving edits when the form is submitted

editForm.addEventListener('submit', saveEdit);

* **Purpose**: Adds an event listener to handle form submission for saving edits.

// Event listener for sorting

sortOptions.addEventListener('change', (e) => {

const selectedCriteria = e.target.value; // Get the selected sort option

sortItems(selectedCriteria); // Call the sorting function with the selected criteria

});

* **Purpose**: Adds an event listener to handle changes in sorting options.

// Initial display of items

displayItems();

});

* **Purpose**: Displays items for the first time when the DOM is fully loaded.

This code provides a comprehensive management system for a grocery store app, handling item display, sorting, searching, editing, cart management, and checkout functionalities.

**2. login.php (Login Functionality):**

* Verifies the username and password against the database, logs the user in, and starts a session.
* Error Reporting: The ini\_set() and error\_reporting() functions enable error reporting to help with debugging.
* Database Connection: require\_once 'db.php'; includes the db.php file to establish a connection to the database.
* Handling POST Request:
* $\_POST['uname'] and $\_POST['psw']: The username and password entered by the user are fetched from the login form. The htmlspecialchars() function is used to prevent XSS attacks by encoding special characters.

**Fetching User Data**

****

* A prepared statement is used to prevent SQL injection attacks. This query selects the id and password of the user with the provided username.

****

* The username is bound to the query ("s" indicates a string).
* If a user is found ($stmt->num\_rows > 0), the password from the database is fetched and compared to the entered password using password\_verify(), which checks if the provided password matches the hashed password stored in the database.

**Successful Login:**

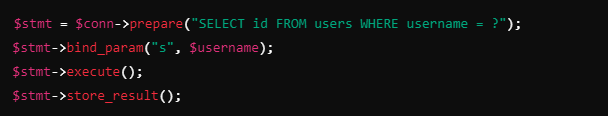
* If the password matches, a session is started with session\_start(), and the user’s ID is stored in the session ($\_SESSION['user\_id']).
* The user is redirected to the dashboard (dashboard.html).

**Failed Login:**

* If the password is incorrect or the user doesn't exist, a JavaScript alert is shown, and the user is redirected back to the login page.

**3. register.php (User Registration Functionality)**

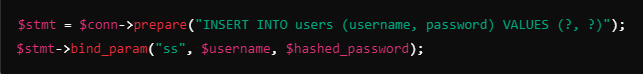
* Handles user registration, checks for duplicate usernames, hashes the password, and inserts the new user into the database.
* Error Reporting: Same as login.php, error reporting is enabled.
* Database Connection: Again, db.php is included to establish the database connection.
* Handling POST Request
* Checking for Duplicate Username

****

* This checks if a username already exists in the database. If it does ($stmt->num\_rows > 0), an error message is shown and the script stops.
* **Hashing the Password:**

****

* The password\_hash() function hashes the password using the BCRYPT algorithm for secure storage.
* **Inserting the New User**:



* The username and hashed password are inserted into the users table. If the insertion is successful, a success message is shown, and the user is redirected to the login page.