Sales Program Documentation

Author: John Mazarakis – 05/04/2025

Sales Management Page.html

1. Form Structure

The form (<form id="brand-form">) allows users to input details about a promotional offer, including:

- **Offer Name (textField)**: A text input for the name of the offer (e.g., "Spring Discounts"). Limited to 150 characters.
- **Brands Selection (#brands-container)**: A placeholder for dynamically loaded brand options, populated via AJAX.
- **Discount Percentage (percentage)**: A numeric input (0-100) to define the discount percentage.
- **Start Date (startdate)**: A date input specifying when the offer begins.
- **End Date (enddate)**: A date input specifying when the offer ends.
- **Submit Button (Καταχώρηση)**: Submits the form with the selected values.

Below the form, there are sections to display:

- **Selected brands (#selected-brands)** after submission.
- **Active sales (#active-sales)**, which are dynamically loaded.

2. JavaScript and jQuery Functionalities

The JavaScript section ensures smooth user interactions using AJAX and jQuery.

a. Loading Brands Dynamically

The loadBrands() function:

- Sends a GET request to /Sales_Programm/Brands.php to fetch available brands.
- Inserts the received brand options into #brands-container.
- Uses **Select2** for enhanced brand selection (dropdown with search and clear options).
- Logs an error message if the dropdown isn't found.

b. Loading Active Sales

The loadActiveSales() function:

- Sends a GET request to /Sales_Programm/get_active_sales.php to fetch active sales data.
- Updates #active-sales with the retrieved information.
- Displays an error message if loading fails.

c. Deleting a Sale

The deleteSale(saleId) function:

- Prompts the user for confirmation before deleting a sale.
- Sends a POST request to /Sales_Programm/delete_sale.php with the sale_id.
- Reloads the active sales list upon success.
- Alerts the user if an error occurs.

d. Form Submission via AJAX

When the user submits the form:

1. The script prevents the default submission (e.preventDefault();).
2. It gathers user inputs:
   - Selected brands, percentage, start date, end date, and offer name.
3. It validates:
   - **Percentage** must be between 0 and 100.
   - **At least one brand** must be selected.
   - **Start and End dates** must be chosen and properly ordered.
4. It sends the data to /Sales_Programm/submit_form.php via an AJAX POST request.
5. On success:
   - Updates #selected-brands with response data.
   - Reloads the active sales list.

e. Setting Date Restrictions

- Automatically sets the **minimum start date** to today's date to prevent past entries.

3. Dependencies

- **jQuery**: Handles AJAX and DOM manipulation (https://code.jquery.com/jquery-3.6.0.min.js).
- **Select2**: Enhances the brand dropdown (https://cdnjs.cloudflare.com/ajax/libs/select2/4.0.13/js/select2.min.js and CSS).

4. Functional Flow

1. **Page Loads**:
   - Calls loadBrands() to populate the brands list.
   - Calls loadActiveSales() to display active promotions.
   - Restricts date input to future dates.
2. **User Fills the Form**:
   - Enters an offer name, selects brands, sets a percentage, and chooses dates.
3. **Validation Occurs**:
   - If any field is invalid, an alert notifies the user.
4. **Form Submits via AJAX**:
   - Sends data to the server.
   - Updates the UI with the selected brands and active sales.
5. **User Can Delete Offers**:
   - Clicking delete prompts a confirmation.
   - On confirmation, AJAX removes the sale and refreshes the list.

5. Key Features

✅ **AJAX for dynamic content**
✅ **Select2 for brand selection**
✅ **Real-time validation**
✅ **Date restrictions to prevent errors**
✅ **Error handling and alerts**

This setup creates an **efficient, user-friendly system** for managing brand promotions.

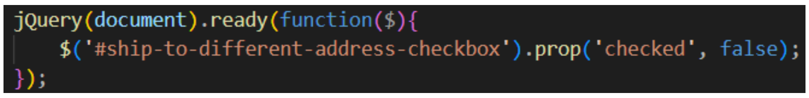
Cart_behaviour.js

This JavaScript code is a **WooCommerce checkout customization script** that:

1. **Disables the "Ship to a different address" checkbox** by default.
2. **Automatically updates the shipping method** based on cart total and selected payment method.
3. **Shows or hides additional billing fields** based on the selected invoice type.
4. **Tracks cart item quantities & SKUs**, then sends them to a PHP script for custom pricing calculations.
5. **Updates the WooCommerce cart total dynamically** based on server-calculated pricing.

Code Breakdown

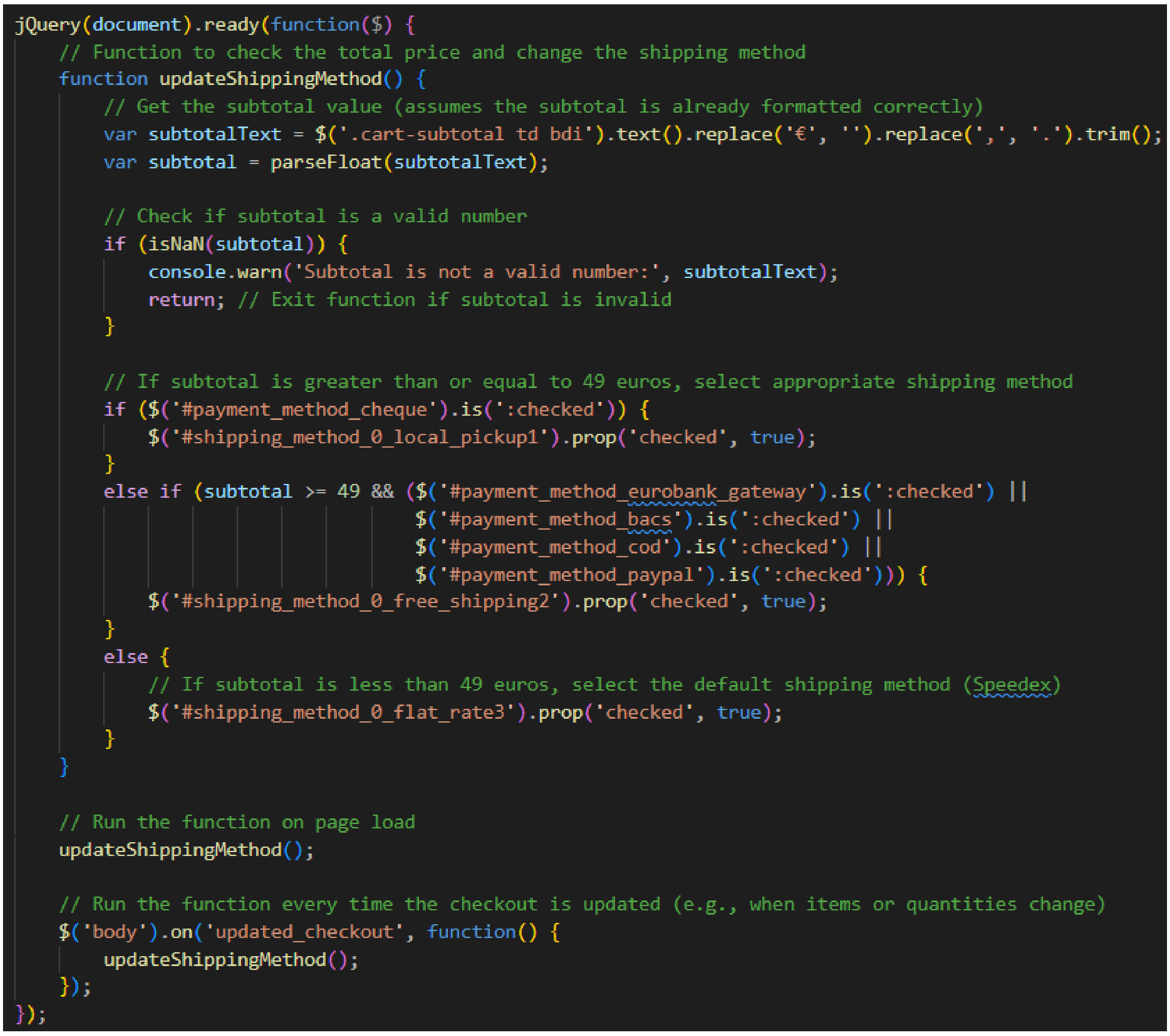
1. Disable "Ship to Different Address" Checkbox



```
jQuery(document).ready(function($){
    $('#ship-to-different-address-checkbox').prop('checked', false);
});
```

- Ensures that the checkbox is unchecked when the page loads.

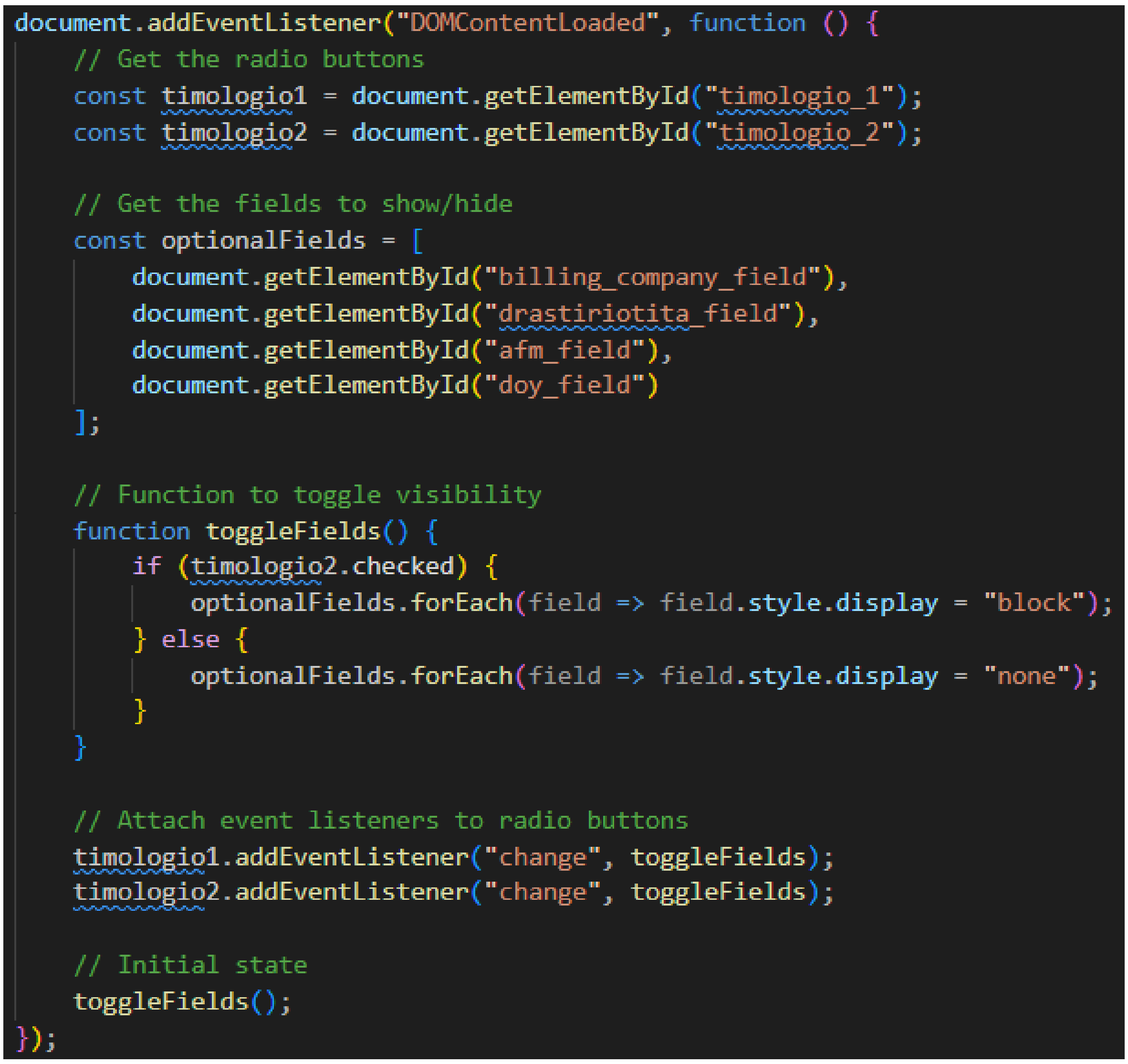
2. Update Shipping Method Based on Cart Total & Payment Method



```
jQuery(document).ready(function($) {
    // Function to check the total price and change the shipping method
    function updateShippingMethod() {
        // Get the subtotal value (assumes the subtotal is already formatted correctly)
        var subtotalText = $('.cart-subtotal td bdi').text().replace('€', '').replace(',', '.').trim();
        var subtotal = parseFloat(subtotalText);

        // Check if subtotal is a valid number
        if (isNaN(subtotal)) {
            console.warn('Subtotal is not a valid number:', subtotalText);
            return; // Exit function if subtotal is invalid
        }

        // If subtotal is greater than or equal to 49 euros, select appropriate shipping method
        if ($('#payment_method_cheque').is(':checked')) {
            $('#shipping_method_0_local_pickup1').prop('checked', true);
        }
        else if (subtotal >= 49 && ($('#payment_method_eurobank_gateway').is(':checked') ||
                                    $('#payment_method_bacs').is(':checked') ||
                                    $('#payment_method_cod').is(':checked') ||
                                    $('#payment_method_paypal').is(':checked'))) {
            $('#shipping_method_0_free_shipping2').prop('checked', true);
        }
        else {
            // If subtotal is less than 49 euros, select the default shipping method (Speedex)
            $('#shipping_method_0_flat_rate3').prop('checked', true);
        }
    }

    // Run the function on page load
    updateShippingMethod();

    // Run the function every time the checkout is updated (e.g., when items or quantities change)
    $('body').on('updated_checkout', function() {
        updateShippingMethod();
    });
});
```

- **Fetches the cart subtotal.**
- **Checks selected payment method.**
- **Applies free shipping if subtotal is €49+** (except for local pickup).
- **Sets Speedex (flat-rate) shipping if subtotal < €49.**
- **Triggers on checkout updates** to recalculate shipping dynamically.

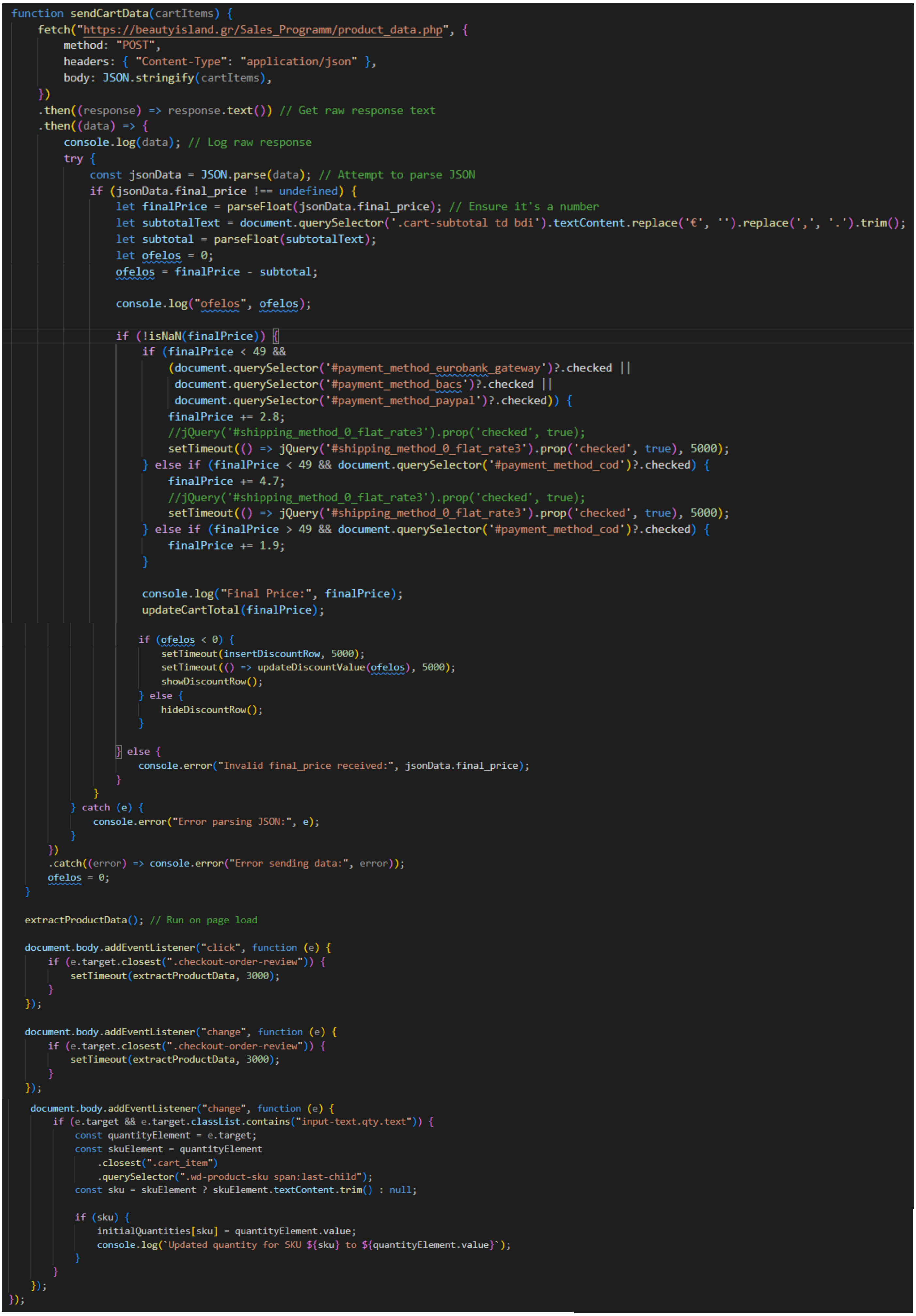
3. Show/Hide Invoice Fields Based on Selection



```javascript
document.addEventListener("DOMContentLoaded", function () {
    // Get the radio buttons
    const timologio1 = document.getElementById("timologio_1");
    const timologio2 = document.getElementById("timologio_2");

    // Get the fields to show/hide
    const optionalFields = [
        document.getElementById("billing_company_field"),
        document.getElementById("drastiriotita_field"),
        document.getElementById("afm_field"),
        document.getElementById("doy_field")
    ];

    // Function to toggle visibility
    function toggleFields() {
        if (timologio2.checked) {
            optionalFields.forEach(field => field.style.display = "block");
        } else {
            optionalFields.forEach(field => field.style.display = "none");
        }
    }

    // Attach event listeners to radio buttons
    timologio1.addEventListener("change", toggleFields);
    timologio2.addEventListener("change", toggleFields);

    // Initial state
    toggleFields();
});
```

- **If "Timologio" is selected**, it shows extra billing fields (company, tax ID, etc.).
- **If not selected, those fields remain hidden.**
- **Auto-applies the correct display settings on page load.**

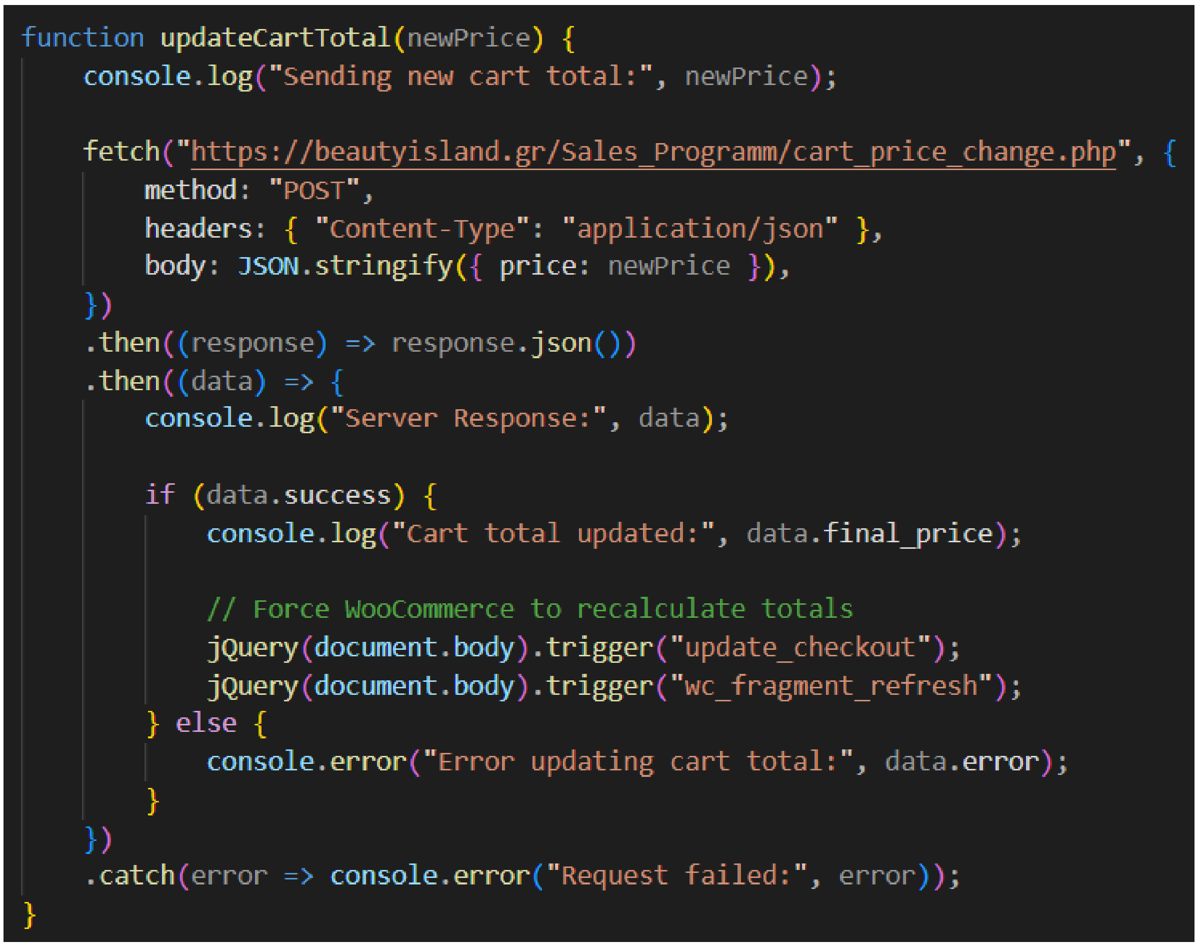
4. Track Product SKUs & Quantities, Send to PHP for Discounts



```javascript
function sendCartData(cartItems) {
    fetch("https://beautyisland.gr/Sales_Programm/product_data.php", {
        method: "POST",
        headers: { "Content-Type": "application/json" },
        body: JSON.stringify(cartItems),
    })
    .then((response) => response.text()) // Get raw response text
    .then((data) => {
        console.log(data); // Log raw response
        try {
            const jsonData = JSON.parse(data); // Attempt to parse JSON
            if (jsonData.final_price !== undefined) {
                let finalPrice = parseFloat(jsonData.final_price); // Ensure it's a number
                let subtotalText = document.querySelector('.cart-subtotal td bdi').textContent.replace('€', '').replace(',', '.').trim();
                let subtotal = parseFloat(subtotalText);
                let ofelos = 0;
                ofelos = finalPrice - subtotal;

                console.log("ofelos", ofelos);

                if (!isNaN(finalPrice)) {
                    if (finalPrice < 49 &&
                        (document.querySelector('#payment_method_eurobank_gateway')?.checked ||
                         document.querySelector('#payment_method_bacs')?.checked ||
                         document.querySelector('#payment_method_paypal')?.checked)) {
                        finalPrice += 2.8;
                        //jQuery('#shipping_method_0_flat_rate3').prop('checked', true);
                        setTimeout(() => jQuery('#shipping_method_0_flat_rate3').prop('checked', true), 5000);
                    } else if (finalPrice < 49 && document.querySelector('#payment_method_cod')?.checked) {
                        finalPrice += 4.7;
                        //jQuery('#shipping_method_0_flat_rate3').prop('checked', true);
                        setTimeout(() => jQuery('#shipping_method_0_flat_rate3').prop('checked', true), 5000);
                    } else if (finalPrice > 49 && document.querySelector('#payment_method_cod')?.checked) {
                        finalPrice += 1.9;
                    }

                    console.log("Final Price:", finalPrice);
                    updateCartTotal(finalPrice);

                    if (ofelos < 0) {
                        setTimeout(insertDiscountRow, 5000);
                        setTimeout(() => updateDiscountValue(ofelos), 5000);
                        showDiscountRow();
                    } else {
                        hideDiscountRow();
                    }

                } else {
                    console.error("Invalid final_price received:", jsonData.final_price);
                }
            }
        } catch (e) {
            console.error("Error parsing JSON:", e);
        }
    })
    .catch((error) => console.error("Error sending data:", error));
    ofelos = 0;
}

extractProductData(); // Run on page load

document.body.addEventListener("click", function (e) {
    if (e.target.closest(".checkout-order-review")) {
        setTimeout(extractProductData, 3000);
    }
});

document.body.addEventListener("change", function (e) {
    if (e.target.closest(".checkout-order-review")) {
        setTimeout(extractProductData, 3000);
    }
});
 document.body.addEventListener("change", function (e) {
     if (e.target && e.target.classList.contains("input-text.qty.text")) {
         const quantityElement = e.target;
         const skuElement = quantityElement
             .closest(".cart_item")
             .querySelector(".wd-product-sku span:last-child");
         const sku = skuElement ? skuElement.textContent.trim() : null;

         if (sku) {
             initialQuantities[sku] = quantityElement.value;
             console.log(`Updated quantity for SKU ${sku} to ${quantityElement.value}`);
         }
     }
 });
});
```

- **Extracts product SKUs & quantities.**
- **Detects changes in cart item quantities.**
- **Sends updated cart data to product_data.php for custom pricing.**
- **Triggers recalculation when checkout page updates.**

5. Update WooCommerce Cart Total Dynamically



```
function updateCartTotal(newPrice) {
    console.log("Sending new cart total:", newPrice);

    fetch("https://beautyisland.gr/Sales_Programm/cart_price_change.php", {
        method: "POST",
        headers: { "Content-Type": "application/json" },
        body: JSON.stringify({ price: newPrice }),
    })
    .then((response) => response.json())
    .then((data) => {
        console.log("Server Response:", data);

        if (data.success) {
            console.log("Cart total updated:", data.final_price);

            // Force WooCommerce to recalculate totals
            jQuery(document.body).trigger("update_checkout");
            jQuery(document.body).trigger("wc_fragment_refresh");
        } else {
            console.error("Error updating cart total:", data.error);
        }
    })
    .catch(error => console.error("Request failed:", error));
}
```

- **Sends the new calculated cart total to WooCommerce.**
- **Forces WooCommerce to refresh checkout totals.**
- **Ensures real-time price updates based on applied discounts.**

Key Features & Best Practices

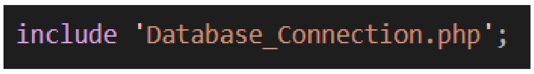
✅ **Uses event listeners efficiently** to track changes dynamically.
✅ **Prevents unnecessary API calls** when cart quantities remain unchanged.
✅ **Ensures compatibility with WooCommerce checkout updates.**
✅ **Applies correct shipping & discount rules automatically.**
✅ **Prevents pricing errors with proper validation.**

Brands.php

This PHP script retrieves a list of product brands from a **WordPress database** (WooCommerce with **YITH WooCommerce Brands Add-On**) and generates a <select> dropdown with multiple selection enabled. It uses **AJAX** to dynamically populate the brands field in the form.

Code Breakdown

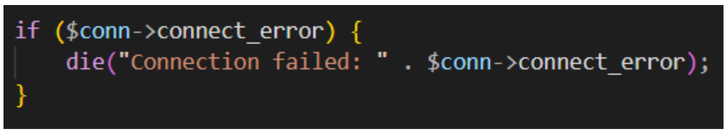
1. Include Database Connection



```
include 'Database_Connection.php';
```

- This line includes the Database_Connection.php file, which is assumed to contain the database credentials and the $conn variable for the **MySQL connection**.
- Ensures that the script has access to the database.

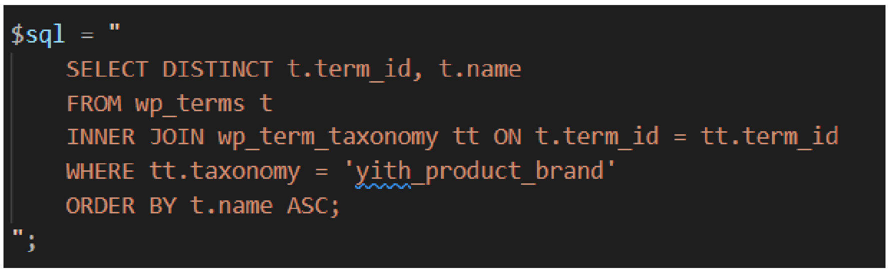
2. Database Connection Error Handling



```
if ($conn->connect_error) {
    die("Connection failed: " . $conn->connect_error);
}
```

- Checks if there was an error while connecting to the database.
- If the connection fails, the script stops execution and outputs an **error message**.

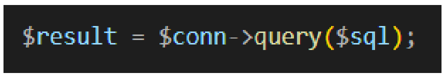
3. Query to Fetch Brands



```
$sql = "
    SELECT DISTINCT t.term_id, t.name
    FROM wp_terms t
    INNER JOIN wp_term_taxonomy tt ON t.term_id = tt.term_id
    WHERE tt.taxonomy = 'yith_product_brand'
    ORDER BY t.name ASC;
";
```

- Retrieves **unique brands** from the WordPress **wp_terms** table.
- Joins with wp_term_taxonomy to filter only brands (taxonomy = 'yith_product_brand').
- Sorts the brands **alphabetically (ORDER BY t.name ASC)** for better usability.

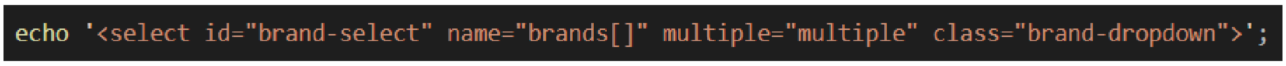
4. Execute the Query



```
$result = $conn->query($sql);
```

- Runs the SQL query against the database.
- The result is stored in $result, which is used to populate the dropdown.

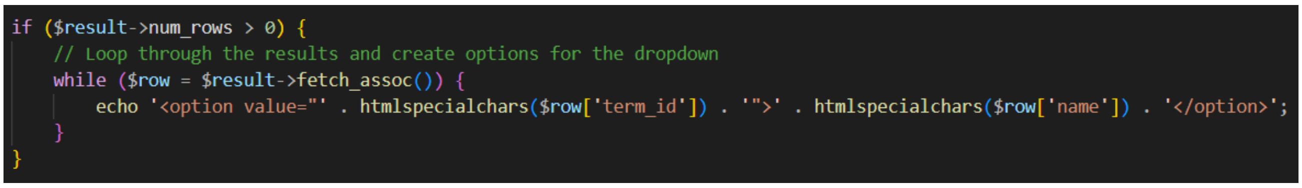
5. Generate <select> Dropdown



```
echo '<select id="brand-select" name="brands[]" multiple="multiple" class="brand-dropdown">';
```

- Creates a <select> element with:
  - id="brand-select" → Used by **JavaScript** to enhance the dropdown.
  - name="brands[]" → Allows selecting **multiple brands**.
  - multiple="multiple" → Enables **multi-selection**.
  - class="brand-dropdown" → Custom class (possibly for CSS styling).

6. Populate Dropdown with Options



```php
if ($result->num_rows > 0) {
    // Loop through the results and create options for the dropdown
    while ($row = $result->fetch_assoc()) {
        echo '<option value="' . htmlspecialchars($row['term_id']) . '">' . htmlspecialchars($row['name']) . '</option>';
    }
}
```

- **Checks if results exist** ($result->num_rows > 0).
- Loops through **each brand** in the result set.
- Outputs an <option> element for each brand:
  - value="term_id" → Uses the **brand ID**.
  - htmlspecialchars($row['name']) → Prevents **XSS attacks** by escaping special characters.

Functional Flow

1. **AJAX Request (loadBrands())** calls this PHP script.
2. **PHP fetches brands** from the database.
3. **Dynamically generates <select> dropdown** with brand names.
4. **Response is inserted** into the <div id="brands-container"> of the form.
5. **Select2 JavaScript Plugin Enhances Dropdown** for a better user experience.

Key Features & Best Practices

✅ **Secure Database Handling**
✅ **Prevents XSS with htmlspecialchars()**
✅ **Uses ORDER BY for alphabetical sorting**
✅ **Optimized Query with DISTINCT for unique brands**
✅ **Enhances UX with multiple selection and Select**

Database_connection.php

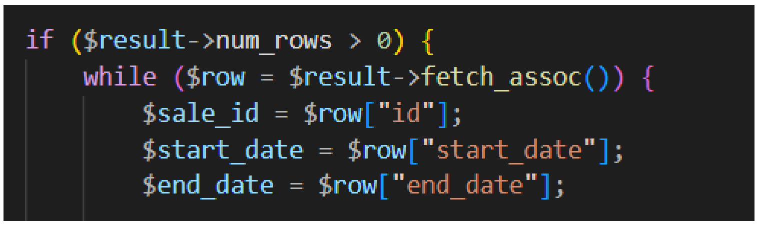
- **Establishes a connection** to a MySQL database.
- It **checks the connection** and outputs an error message if the connection fails.

Dates_check_and_delete.php

This script:

1. **Connects to the database**.
2. **Fetches all sales records** (id, start_date, end_date).
3. **Compares today's date ($today) with the sale period**.
   - **If today is within the sale period → Updates the sale's status to 1** (active).
   - **If today is past the sale period → Deletes the sale from the database**.
4. **Logs errors if updates or deletions fail**.
5. **Closes the database connection**.

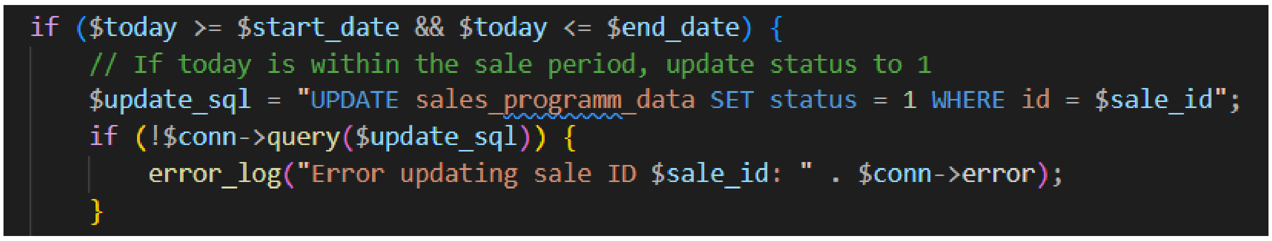
1. Loop Through Sales and Check Dates



```php
if ($result->num_rows > 0) {
    while ($row = $result->fetch_assoc()) {
        $sale_id = $row["id"];
        $start_date = $row["start_date"];
        $end_date = $row["end_date"];
```

- Checks if there are **any sales in the database**.
- Iterates through each sale using **while ($row = $result->fetch_assoc())**.

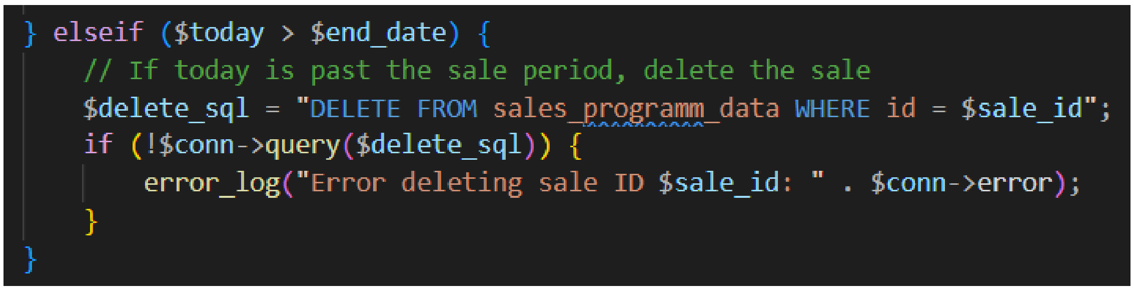
2. If the Sale is Active, Update Status



```php
if ($today >= $start_date && $today <= $end_date) {
    // If today is within the sale period, update status to 1
    $update_sql = "UPDATE sales_programm_data SET status = 1 WHERE id = $sale_id";
    if (!$conn->query($update_sql)) {
        error_log("Error updating sale ID $sale_id: " . $conn->error);
    }
```

- If today is **between** start_date and end_date, the sale is **active**.
- **Updates status to 1** in sales_programm_data:
    - **Active sales → status = 1**.
- If the query fails, **logs an error** in the server logs.

3. If the Sale is Expired, Delete It



```php
} elseif ($today > $end_date) {
    // If today is past the sale period, delete the sale
    $delete_sql = "DELETE FROM sales_programm_data WHERE id = $sale_id";
    if (!$conn->query($delete_sql)) {
        error_log("Error deleting sale ID $sale_id: " . $conn->error);
    }
}
```

- If **today is later than end_date**, the sale has **expired**.
- **Deletes the sale** from sales_programm_data.
- If deletion **fails**, logs an error.

Functional Flow

1. **Fetch all sales records** from the database.
2. **Loop through each sale**:
    - If **sale is active** (today within start & end date) → **Update status = 1**.
    - If **sale is expired** (today > end_date) → **Delete it**.
3. **Log errors if database operations fail**.
4. **Close database connection**.

Key Features & Best Practices

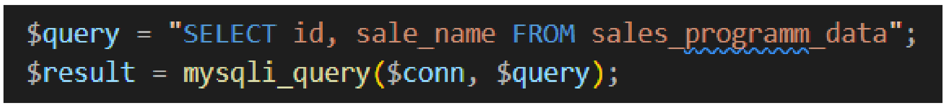
✅ **Uses date("Y-m-d") for correct date formatting**
✅ **Efficient sales status update (UPDATE ... WHERE id = ?)**
✅ **Removes expired sales (DELETE FROM ... WHERE id = ?)**
✅ **Uses error_log() for debugging instead of die()**
✅ **Minimizes unnecessary database operations**

Get_active_sales.php

This script:

1. **Connects to the database** (Database_Connection.php).
2. **Fetches sales records** (id, sale_name) from sales_programm_data.
3. **Displays each sale with a delete button** (deleteSale(id) JavaScript function).
4. **Handles cases where no active sales exist**.
5. **Closes the database connection**.

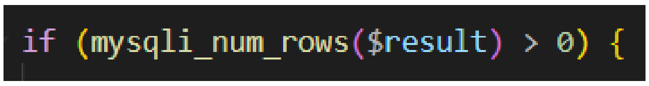
1. Fetch Sales from sales_programm_data



```php
$query = "SELECT id, sale_name FROM sales_programm_data";
$result = mysqli_query($conn, $query);
```

- Retrieves **id** (sale identifier) and **sale_name** (sale description).
- mysqli_query($conn, $query) executes the query.

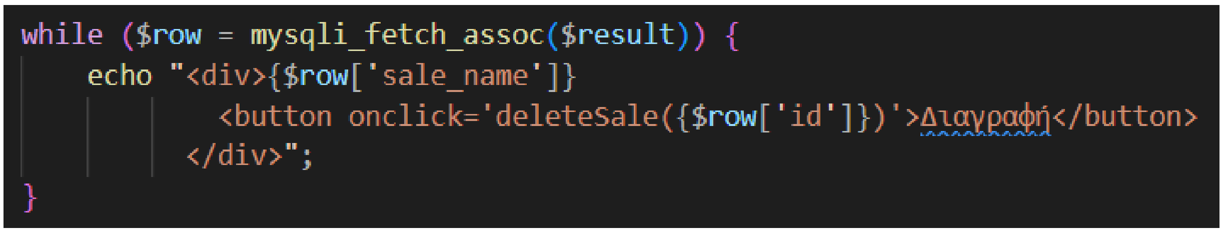
2. Check if There Are Any Sales



```php
if (mysqli_num_rows($result) > 0) {
```

- mysqli_num_rows($result) > 0 checks if the query **returned any results**.
- If there are sales, **loops through them**.

3. Loop Through Sales and Display Them



```php
while ($row = mysqli_fetch_assoc($result)) {
    echo "<div>{$row['sale_name']}
            <button onclick='deleteSale({$row['id']})'>Διαγραφή</button>
          </div>";
}
```

- **Loops through each sale** (while ($row = mysqli_fetch_assoc($result))).
- **Displays the sale name** inside a <div>.
- **Adds a "Delete" button** (<button>):
  - Calls deleteSale({$row['id']}) **when clicked**.
  - **Passes the sale id** to the function.

Functional Flow

1. **Query sales_programm_data** for sales.
2. **If sales exist**, display each sale with a delete button.
3. **If no sales exist**, show "No active sales found."
4. **Close the database connection**.

Key Features & Best Practices

✅ **Uses mysqli_fetch_assoc($result) for efficient fetching**
✅ **Uses mysqli_num_rows() to check for existing sales**
✅ **Dynamically generates a delete button for each sale**
✅ **Closes database connection properly**

Get_active_sales_for_cart.php

This script:

1. **Includes necessary files**:
    - dates_check_and_delete.php: Likely updates or deletes expired sales.
    - Database_Connection.php: Establishes a database connection.
2. **Queries active sales (status = 1)** from sales_programm_data.
3. **Fetches sales details** (id, brands_in_sale, sale_percentage).
4. **Stores the results in an array ($salesData)** for structured use.
5. **Closes the database connection**.

Code Breakdown

1. Include External Scripts

```
include 'dates_check_and_delete.php';
include 'Database_Connection.php';
```

- **dates_check_and_delete.php**:
    - Likely **updates sale statuses** or **deletes expired sales**.
    - Ensures only active sales (status = 1) are retrieved.
- **Database_Connection.php**:
    - Establishes a **database connection ($conn)**.

2. Query Active Sales (status = 1)

```
$query = "SELECT id, brands_in_sale, sale_percentage FROM sales_programm_data WHERE status = '1'";
$result = mysqli_query($conn, $query);
```

- **Retrieves active sales (status = '1')**.
- **Fields selected**:
    - id → Unique identifier of the sale.
    - brands_in_sale → List of brands in the sale.
    - sale_percentage → Discount percentage.

3. Initialize an Empty Sales Data Array

```php
$salesData = []; // Initialize an empty array
```

- **Prepares an empty array ($salesData)** to store sale details.

4. Process Query Results

```php
if (mysqli_num_rows($result) > 0) {
    while ($row = mysqli_fetch_assoc($result)) {
        $salesData[] = [
            'id' => $row['id'],
            'brands_in_sale' => $row['brands_in_sale'],
            'sale_percentage' => $row['sale_percentage']
        ];
    }
} else {
    $salesData = null; // If no data found
}
```

- **Checks if sales exist (mysqli_num_rows($result) > 0)**:
  - **If sales exist**, loops through each row:
    - Extracts id, brands_in_sale, and sale_percentage.
    - Appends them as an **associative array** inside $salesData[].
  - **If no active sales exist**, sets $salesData = null.

Functional Flow

1. **Include scripts (dates_check_and_delete.php & Database_Connection.php)**.
2. **Query sales_programm_data for active sales (status = 1)**.
3. **Check if sales exist**:
   - If **sales exist**, store them in an array ($salesData).
   - If **no sales exist**, set $salesData = null.
4. **Close the database connection**.

Key Features & Best Practices

✅ **Uses mysqli_fetch_assoc($result) for efficient row fetching**
✅ **Filters only active sales (status = 1)**
✅ **Stores sales in a structured array for further processing**
✅ **Handles cases where no sales exist ($salesData = null)**
✅ **Properly closes database connection (mysqli_close($conn))**

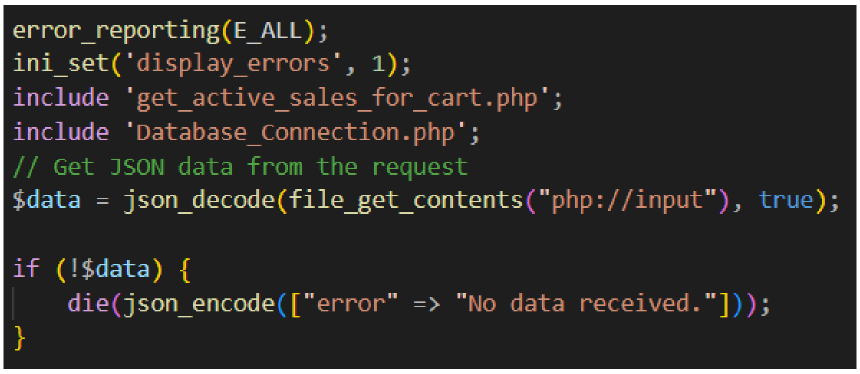
<u>Product_data.php</u>

This script:

1. **Handles a POST request** with product data (including SKU and quantity).
2. **Fetches product details** from the WordPress database based on SKU.
3. **Determines applicable sales** based on the product's associated brands.
4. **Calculates the final price** after applying discounts for relevant sales.
5. **Returns the final price** as a JSON response.

Code Breakdown

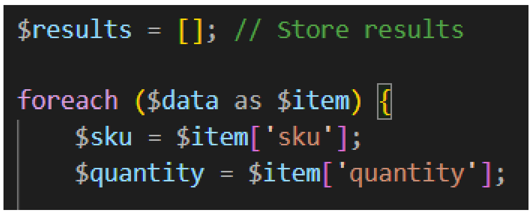
1. Error Handling and Data Fetching



```php
error_reporting(E_ALL);
ini_set('display_errors', 1);
include 'get_active_sales_for_cart.php';
include 'Database_Connection.php';
// Get JSON data from the request
$data = json_decode(file_get_contents("php://input"), true);

if (!$data) {
    die(json_encode(["error" => "No data received."]));
}
```

- **Enables error reporting** for debugging.
- **Includes necessary files**:
    - get_active_sales_for_cart.php: Likely contains logic for fetching active sales.
    - Database_Connection.php: Establishes the database connection.
- **Gets the raw JSON input** from the request body and decodes it. If no data is provided, it **returns an error**.

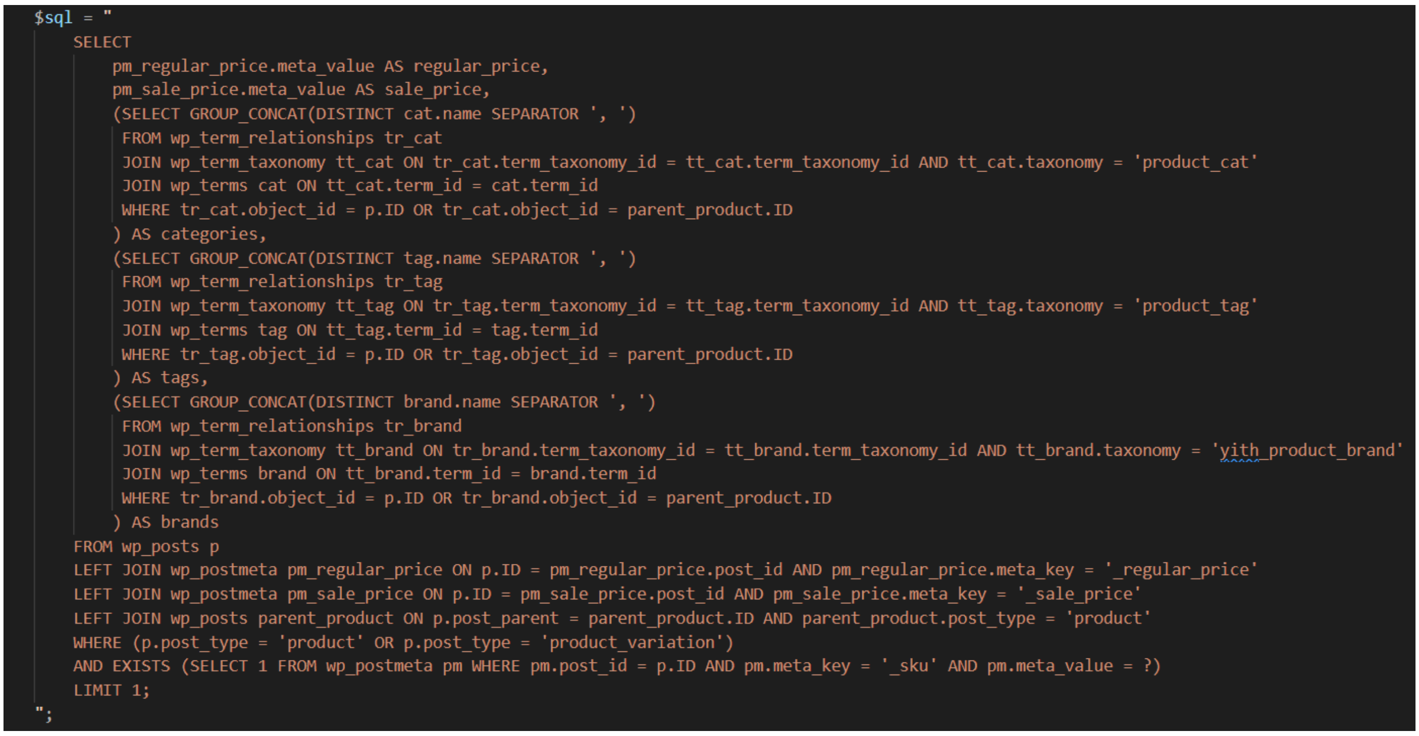
2. Loop Through Products in Cart



```php
$results = []; // Store results

foreach ($data as $item) {
    $sku = $item['sku'];
    $quantity = $item['quantity'];
```

- Initializes an empty $results array to store the processed products.
- **Loops through each item** in the cart ($data), which contains sku and quantity.

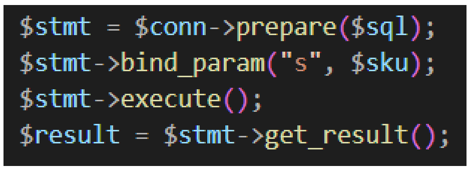
3. Query Product Details Based on SKU



```php
$sql = "
    SELECT
        pm_regular_price.meta_value AS regular_price,
        pm_sale_price.meta_value AS sale_price,
        (SELECT GROUP_CONCAT(DISTINCT cat.name SEPARATOR ', ')
         FROM wp_term_relationships tr_cat
         JOIN wp_term_taxonomy tt_cat ON tr_cat.term_taxonomy_id = tt_cat.term_taxonomy_id AND tt_cat.taxonomy = 'product_cat'
         JOIN wp_terms cat ON tt_cat.term_id = cat.term_id
         WHERE tr_cat.object_id = p.ID OR tr_cat.object_id = parent_product.ID
        ) AS categories,
        (SELECT GROUP_CONCAT(DISTINCT tag.name SEPARATOR ', ')
         FROM wp_term_relationships tr_tag
         JOIN wp_term_taxonomy tt_tag ON tr_tag.term_taxonomy_id = tt_tag.term_taxonomy_id AND tt_tag.taxonomy = 'product_tag'
         JOIN wp_terms tag ON tt_tag.term_id = tag.term_id
         WHERE tr_tag.object_id = p.ID OR tr_tag.object_id = parent_product.ID
        ) AS tags,
        (SELECT GROUP_CONCAT(DISTINCT brand.name SEPARATOR ', ')
         FROM wp_term_relationships tr_brand
         JOIN wp_term_taxonomy tt_brand ON tr_brand.term_taxonomy_id = tt_brand.term_taxonomy_id AND tt_brand.taxonomy = 'yith_product_brand'
         JOIN wp_terms brand ON tt_brand.term_id = brand.term_id
         WHERE tr_brand.object_id = p.ID OR tr_brand.object_id = parent_product.ID
        ) AS brands
    FROM wp_posts p
    LEFT JOIN wp_postmeta pm_regular_price ON p.ID = pm_regular_price.post_id AND pm_regular_price.meta_key = '_regular_price'
    LEFT JOIN wp_postmeta pm_sale_price ON p.ID = pm_sale_price.post_id AND pm_sale_price.meta_key = '_sale_price'
    LEFT JOIN wp_posts parent_product ON p.post_parent = parent_product.ID AND parent_product.post_type = 'product'
    WHERE (p.post_type = 'product' OR p.post_type = 'product_variation')
    AND EXISTS (SELECT 1 FROM wp_postmeta pm WHERE pm.post_id = p.ID AND pm.meta_key = '_sku' AND pm.meta_value = ?)
    LIMIT 1;
";
```

- **Prepares an SQL query** to fetch the **regular price**, **sale price**, **categories**, **tags**, and **brands** associated with the product using the SKU.
- It uses **LEFT JOIN** to join multiple tables to get all necessary product metadata.

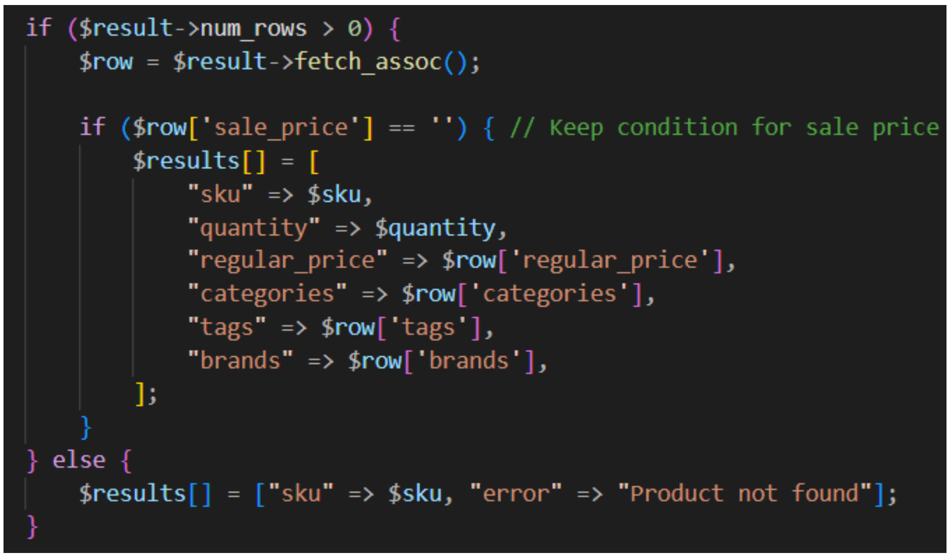
4. Fetch Product Data



```
$stmt = $conn->prepare($sql);
$stmt->bind_param("s", $sku);
$stmt->execute();
$result = $stmt->get_result();
```

- Prepares and executes the query using the provided sku.
- **Fetches the product details** (like price, categories, tags, brands) from the database.

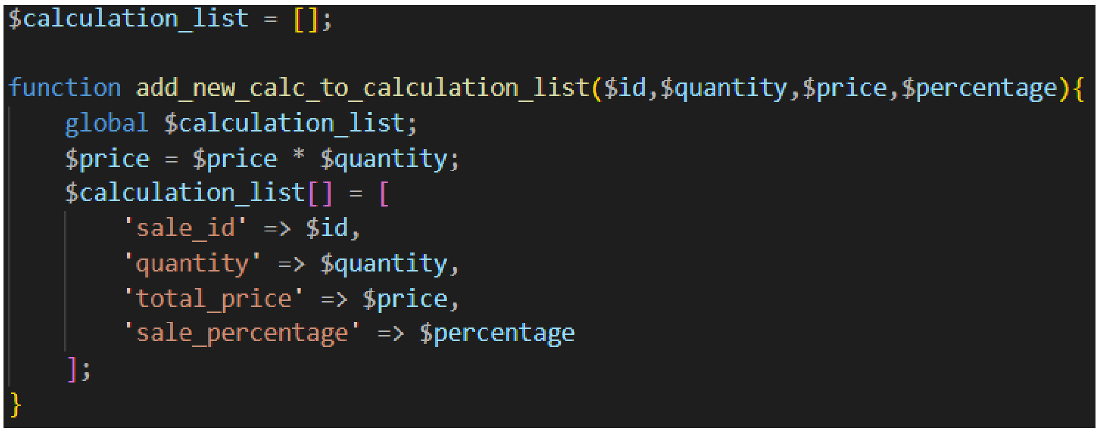
5. Process Query Results



```
if ($result->num_rows > 0) {
    $row = $result->fetch_assoc();

    if ($row['sale_price'] == '') { // Keep condition for sale price
        $results[] = [
            "sku" => $sku,
            "quantity" => $quantity,
            "regular_price" => $row['regular_price'],
            "categories" => $row['categories'],
            "tags" => $row['tags'],
            "brands" => $row['brands'],
        ];
    }
} else {
    $results[] = ["sku" => $sku, "error" => "Product not found"];
}
```

- **If a product is found**, it checks if there is no **sale price** and adds the product details to the $results array.
- **If the product is not found**, an error message is added to the $results.

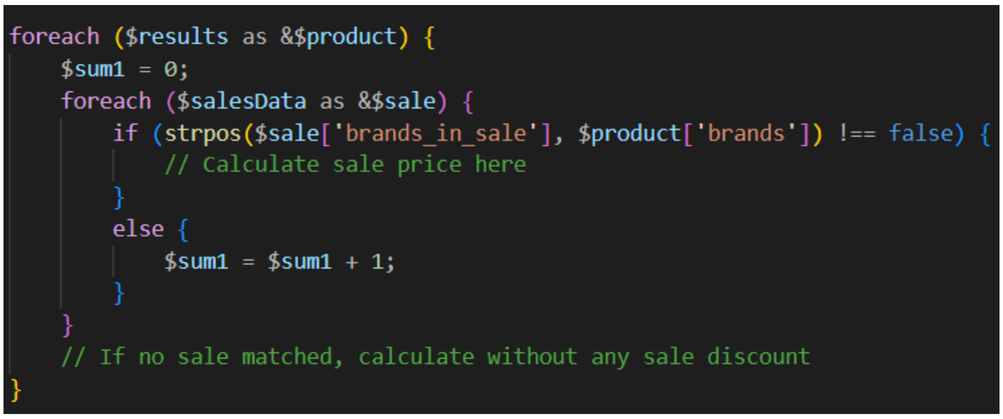
6. Define and Add Calculations to **calculation_list**



```
$calculation_list = [];

function add_new_calc_to_calculation_list($id,$quantity,$price,$percentage){
    global $calculation_list;
    $price = $price * $quantity;
    $calculation_list[] = [
        'sale_id' => $id,
        'quantity' => $quantity,
        'total_price' => $price,
        'sale_percentage' => $percentage
    ];
}
```

- Initializes an empty array ($calculation_list) to store calculations.
- **add_new_calc_to_calculation_list** adds a sale calculation to this list, including the **sale ID**, **quantity**, **total price**, and **sale percentage**.

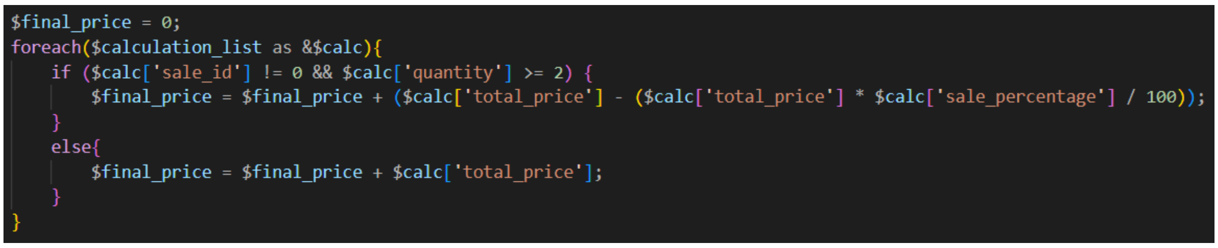
7. Check for Applicable Sales and Calculate Price



```php
foreach ($results as &$product) {
    $sum1 = 0;
    foreach ($salesData as &$sale) {
        if (strpos($sale['brands_in_sale'], $product['brands']) !== false) {
            // Calculate sale price here
        }
        else {
            $sum1 = $sum1 + 1;
        }
    }
    // If no sale matched, calculate without any sale discount
}
```

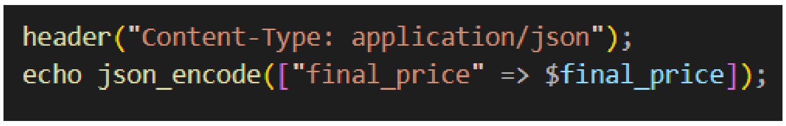
- **Checks if the product's brand matches any sales**.
- If a sale matches, it updates the calculation list with the **discounted price**.
- If no sale matches, it adds the product’s regular price without any discount.

8. Final Price Calculation



```php
$final_price = 0;
foreach($calculation_list as &$calc){
    if ($calc['sale_id'] != 0 && $calc['quantity'] >= 2) {
        $final_price = $final_price + ($calc['total_price'] - ($calc['total_price'] * $calc['sale_percentage'] / 100));
    }
    else{
        $final_price = $final_price + $calc['total_price'];
    }
}
```

9. Return Final Price as JSON



```php
header("Content-Type: application/json");
echo json_encode(["final_price" => $final_price]);
```

- **Returns the final price** as a **JSON response** to the client.

Key Features & Best Practices

✅ **Uses prepared statements** for secure SQL queries to prevent SQL injection.
✅ **Calculates prices dynamically** based on product details and applicable sales.
✅ **Handles cart data as JSON** for easy integration with JavaScript.
✅ **Applies sale discounts** only when conditions are met (e.g., minimum quantity).
✅ **Returns a structured JSON response** for use in JavaScript.

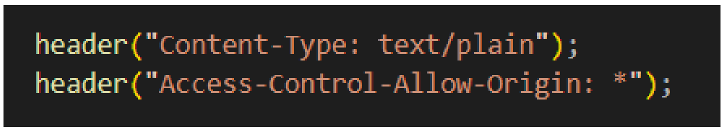
Submit_form.php

This PHP script processes **AJAX POST requests** from the form. It:

1. **Validates received data** (brands, percentage, dates, offer name).
2. **Fetches brand names** based on selected IDs.
3. **Inserts data into the sales_programm_data table**.
4. **Prevents SQL injection** using **prepared statements**.
5. **Includes dates_check_and_delete.php** to handle expired sales.

Code Breakdown

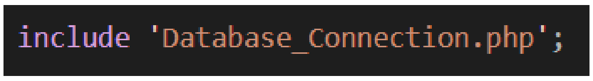
1. Enable Cross-Origin Requests (CORS)



```php
header("Content-Type: text/plain");
header("Access-Control-Allow-Origin: *");
```

- Content-Type: text/plain → Ensures the response is **plain text**.
- Access-Control-Allow-Origin: * → Allows **AJAX requests from any domain** (may need to be restricted for security).

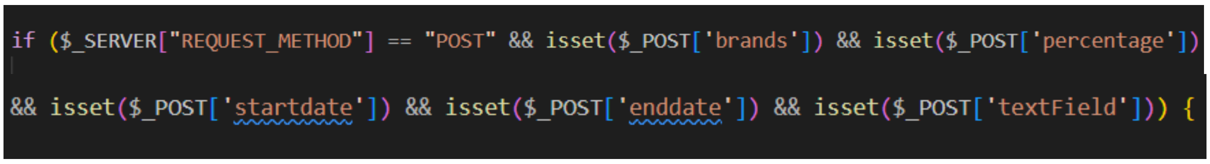
2. Include Database Connection



```php
include 'Database_Connection.php';
```

- Loads database connection settings ($conn).

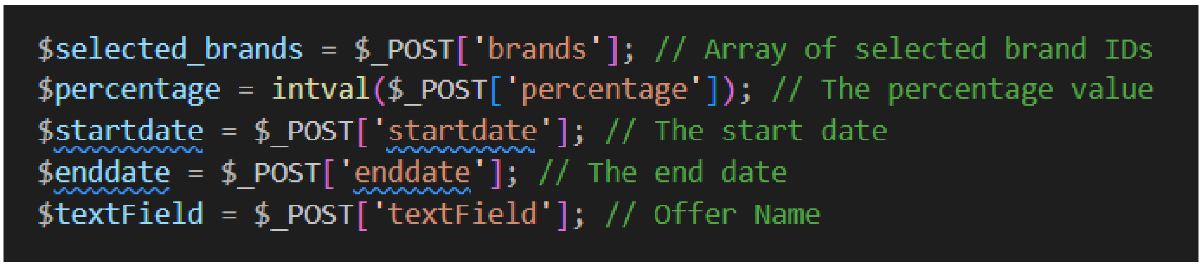
3. Validate the Incoming POST Request



```php
if ($_SERVER["REQUEST_METHOD"] == "POST" && isset($_POST['brands']) && isset($_POST['percentage'])
&& isset($_POST['startdate']) && isset($_POST['enddate']) && isset($_POST['textField'])) {
```

- Checks that all required fields (brands, percentage, startdate, enddate, textField) are provided.
- Prevents errors due to missing data.

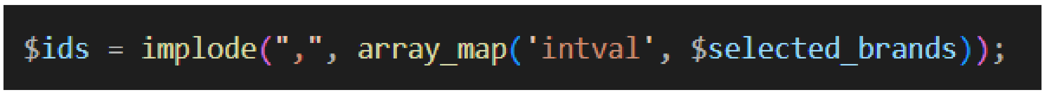
4. Retrieve and Sanitize User Inputs



```php
$selected_brands = $_POST['brands']; // Array of selected brand IDs
$percentage = intval($_POST['percentage']); // The percentage value
$startdate = $_POST['startdate']; // The start date
$enddate = $_POST['enddate']; // The end date
$textField = $_POST['textField']; // Offer Name
```

- **intval() ensures $percentage is a valid integer.**
- **Avoids SQL injection** by treating $selected_brands as an array.

5. Convert Brand IDs into a String

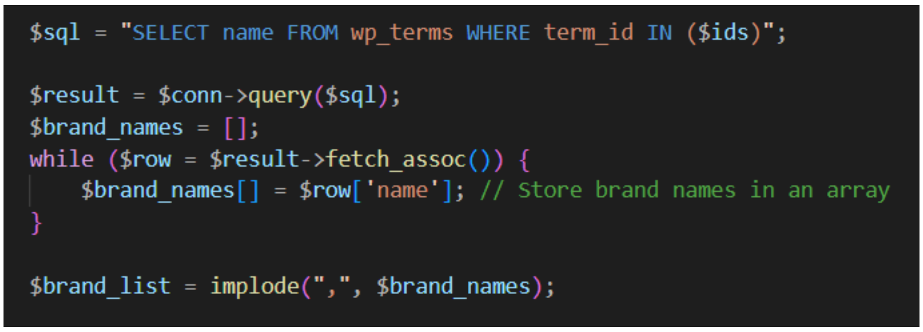


```php
$ids = implode(",", array_map('intval', $selected_brands));
```

- **array_map('intval', $selected_brands')** → Ensures all IDs are numeric.
- **implode(",", $array)** → Converts the array into a comma-separated string

.

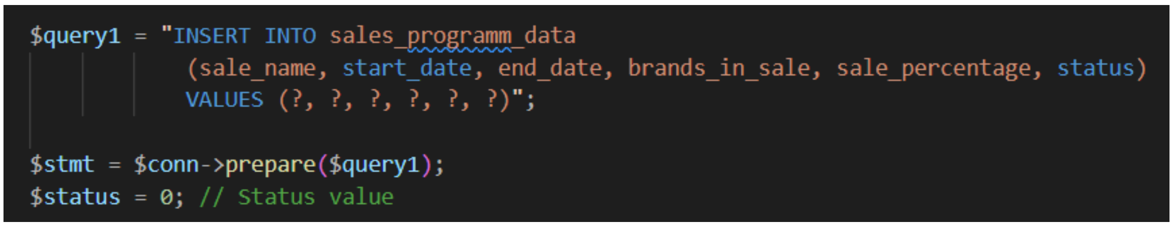
6. Fetch Brand Names Based on IDs



```php
$sql = "SELECT name FROM wp_terms WHERE term_id IN ($ids)";

$result = $conn->query($sql);
$brand_names = [];
while ($row = $result->fetch_assoc()) {
    $brand_names[] = $row['name']; // Store brand names in an array
}

$brand_list = implode(",", $brand_names);
```

- Retrieves brand **names** corresponding to selected **brand IDs**.
- Stores brand names in $brand_names and converts them into a **comma-separated string** ($brand_list).

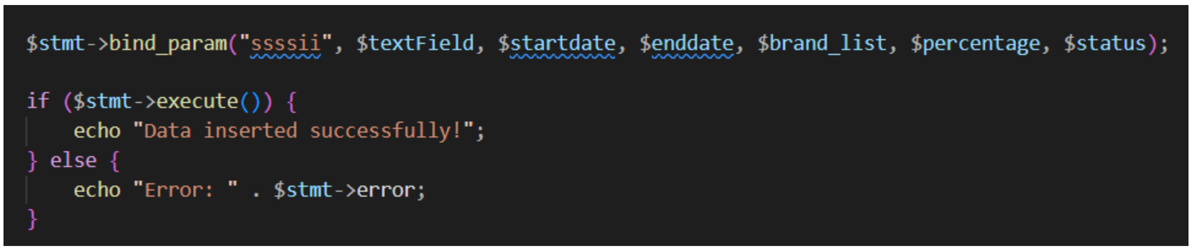
7. Insert Data into sales_programm_data Table



```php
$query1 = "INSERT INTO sales_programm_data
            (sale_name, start_date, end_date, brands_in_sale, sale_percentage, status)
            VALUES (?, ?, ?, ?, ?, ?)";

$stmt = $conn->prepare($query1);
$status = 0; // Status value
```

- Uses **prepared statements** to prevent **SQL injection**.
- Inserts:
  - sale_name: Offer name ($textField).
  - start_date: Start date.
  - end_date: End date.
  - brands_in_sale: Selected brands.
  - sale_percentage: Discount percentage.
  - status: Default **status = 0** (inactive?).

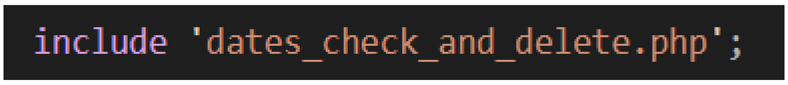
8. Bind Parameters and Execute the Query



```php
$stmt->bind_param("ssssii", $textField, $startdate, $enddate, $brand_list, $percentage, $status);

if ($stmt->execute()) {
    echo "Data inserted successfully!";
} else {
    echo "Error: " . $stmt->error;
}
```

- **bind_param("ssssii", ...)**:
  - "ssssii" → Defines **data types**:
    - s (string) → Offer Name
    - s (string) → Start Date
    - s (string) → End Date
    - s (string) → Brands in Sale
    - i (integer) → Sale Percentage
    - i (integer) → Status
- If insertion **succeeds**, displays "Data inserted successfully!".
- If **fails**, displays "Error: " followed by the SQL error.

9. Include dates_check_and_delete.php



```php
include 'dates_check_and_delete.php';
```

- Likely **removes expired sales** after inserting new data.

Functional Flow

1. **AJAX Sends Form Data** → PHP receives brand IDs, percentage, dates, and offer name.
2. **PHP Validates Inputs** → Ensures all fields exist.
3. **Fetches Brand Names** → Converts IDs into readable names.
4. **Inserts Data into MySQL** (sales_programm_data table).
5. **Executes dates_check_and_delete.php** → Cleans up expired offers.
6. **Returns a Success or Error Message**.

Key Features & Best Practices

✅ **Cross-Origin Requests Allowed (CORS)**
✅ **SQL Injection Prevention (Prepared Statements)**
✅ **Error Handling for Database Queries**
✅ **Efficient Data Processing with implode() & array_map()**
✅ **Security: Uses intval() to Validate Numeric Inputs**

Cart_price_change.php

This PHP script is a **WooCommerce integration** that handles **updating a custom cart total** based on an external price input. Below is a detailed breakdown:

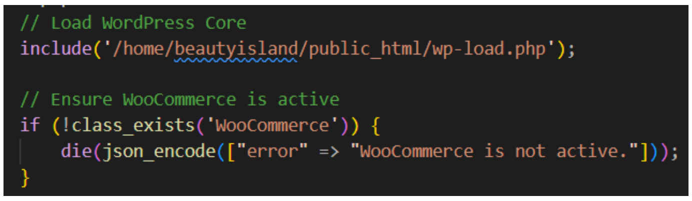
Overview

This script:

1. **Loads WordPress and WooCommerce.**
2. **Validates JSON input** to ensure a valid price is received.
3. **Updates the WooCommerce session** with the new custom cart total.
4. **Returns a JSON response** indicating success or failure.

Code Breakdown

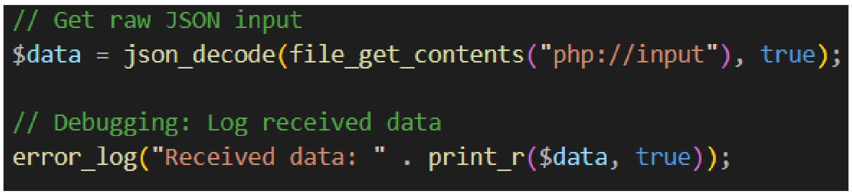
1. Load WordPress & WooCommerce



```php
// Load WordPress Core
include('/home/beautyisland/public_html/wp-load.php');

// Ensure WooCommerce is active
if (!class_exists('WooCommerce')) {
    die(json_encode(["error" => "WooCommerce is not active."]));
}
```

- **Includes wp-load.php** to access WordPress functions.
- **Checks if WooCommerce is active** using class_exists('WooCommerce').
- If WooCommerce is **not active**, it returns a JSON error message.

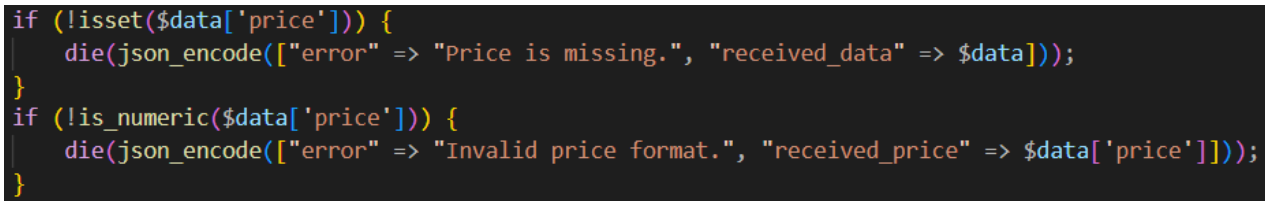
2. Get & Decode JSON Input



```php
// Get raw JSON input
$data = json_decode(file_get_contents("php://input"), true);

// Debugging: Log received data
error_log("Received data: " . print_r($data, true));
```

- **Reads JSON input** from the request body.
- **Logs the received data** for debugging (useful for identifying incorrect API calls).

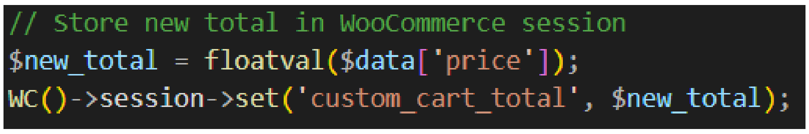
3. Validate Input



```php
if (!isset($data['price'])) {
    die(json_encode(["error" => "Price is missing.", "received_data" => $data]));
}
if (!is_numeric($data['price'])) {
    die(json_encode(["error" => "Invalid price format.", "received_price" => $data['price']]));
}
```

- **Ensures price is present** in the JSON input.
- **Ensures price is numeric**, preventing potential errors or security risks.
- **If validation fails**, it returns an error response.

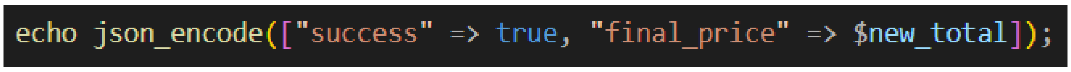
4. Update WooCommerce Session



```php
// Store new total in WooCommerce session
$new_total = floatval($data['price']);
WC()->session->set('custom_cart_total', $new_total);
```

- **Converts price to a float** (floatval() ensures numerical precision).
- **Stores the value in the WooCommerce session** using WC()->session->set('custom_cart_total', $new_total).
- The custom_cart_total session variable can be used later in the cart or checkout process.

5. Return JSON Response



```php
echo json_encode(["success" => true, "final_price" => $new_total]);
```

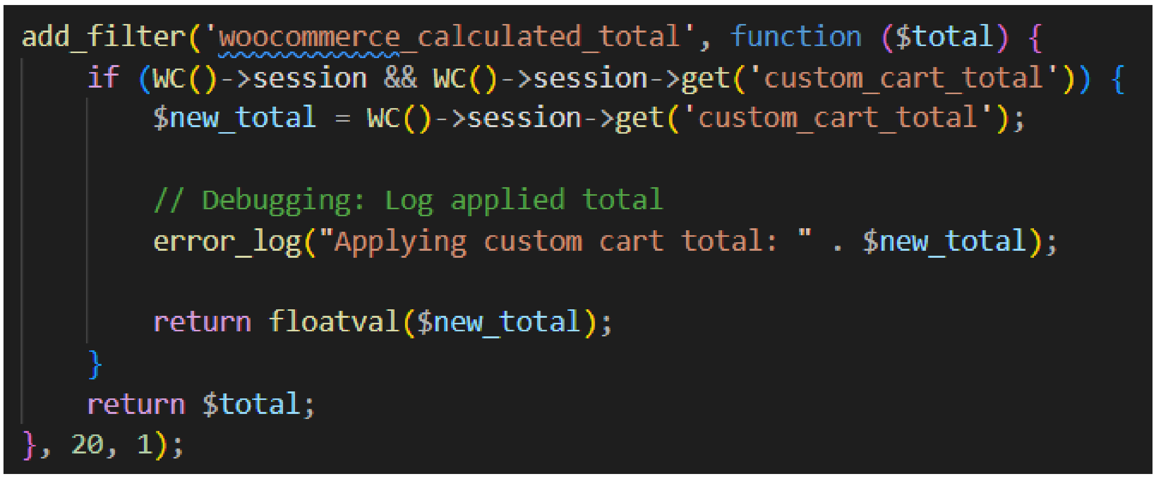
- **Returns a JSON response** with the updated price.

Key Features & Best Practices

✅ **Uses wp-load.php** to access WordPress functions outside the core system.
✅ **Checks WooCommerce availability** to prevent errors if the plugin is deactivated.
✅ **Validates JSON input** to avoid security risks and unexpected failures.
✅ **Stores the total in WooCommerce's session** for later use.
✅ **Returns structured JSON responses**, making it easy to debug and integrate.

Functions.php – (located in home/public_html/wp_content/themes/theme_child)

This PHP snippet, which modifies **WooCommerce cart total dynamically** based on a custom session value should be added in this file. It ensures that any external price adjustments (e.g., from JavaScript calculations) are correctly applied at checkout.



```php
add_filter('woocommerce_calculated_total', function ($total) {
    if (WC()->session && WC()->session->get('custom_cart_total')) {
        $new_total = WC()->session->get('custom_cart_total');

        // Debugging: Log applied total
        error_log("Applying custom cart total: " . $new_total);

        return floatval($new_total);
    }
    return $total;
}, 20, 1);
```

What This Code Does

✅ **Hooks into WooCommerce’s woocommerce_calculated_total filter** to modify the cart total.
✅ **Checks if a custom total is stored in the WooCommerce session.**
✅ **If a custom price exists, it overrides the default total with the new value.**
✅ **Ensures that the cart displays the correct adjusted total in checkout.**
✅ **Logs the applied total for debugging in WooCommerce’s error log.**