Trogon PHP Interview Task

Total Questions: **6**

Time: **48 Hours**

QUESTION 1:

Create a simple PHP backend that serves JSON data and then write a jQuery script to retrieve this data and display it on a webpage. This task evaluates the candidate's proficiency in both PHP and jQuery, focusing on API development, handling asynchronous requests, and parsing JSON data.

Requirements:

- **Back-end (PHP or any PHP frameworks)**
  - Develop a PHP script that acts as an API endpoint.
  - The API should handle a GET request and return a JSON object containing a list of items (e.g., products, users, etc.), with each item having at least two fields (like id, name).
  - Implement basic error handling to manage cases where the request fails or incorrect data is provided.
- **Front-end (jQuery)**
  - Write jQuery code to make an AJAX call to the PHP API to retrieve the data.
  - Parse the returned JSON data and dynamically generate HTML content to display the items on a webpage.
  - Include error handling in the jQuery script to manage situations where the AJAX call fails (e.g., network issues, server error).

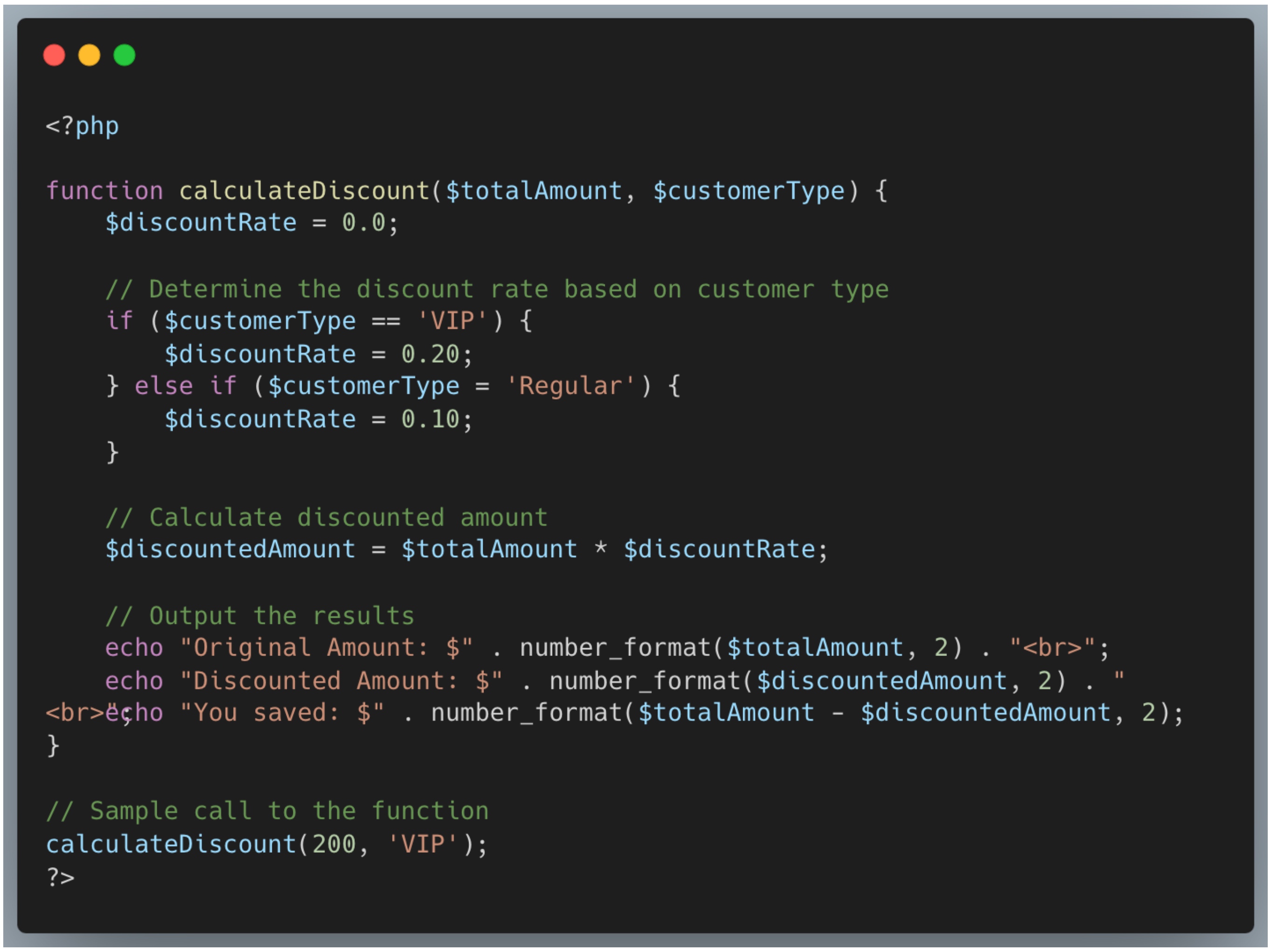
Evaluation Criteria:

- **API Design:** The clarity and efficiency of the PHP script, including how well it handles different request scenarios.
- **AJAX Implementation:** Proficiency in using jQuery for AJAX calls, including setup, error handling, and response management.
- **Data Handling and UI Update:** Ability to parse JSON data and manipulate the DOM to update the UI dynamically.
- **Error Handling:** Robustness of error handling on both the client and server sides.

QUESTION 2:

PHP Error Handling Task: Script with a Logical Error

Background: Below is a PHP script intended to calculate a discount based on the customer type and display the original, discounted amount, and savings. However, the script contains a logical error and may also lack sufficient error handling mechanisms.



```php
<?php

function calculateDiscount($totalAmount, $customerType) {
    $discountRate = 0.0;

    // Determine the discount rate based on customer type
    if ($customerType == 'VIP') {
        $discountRate = 0.20;
    } else if ($customerType = 'Regular') {
        $discountRate = 0.10;
    }

    // Calculate discounted amount
    $discountedAmount = $totalAmount * $discountRate;

    // Output the results
    echo "Original Amount: $" . number_format($totalAmount, 2) . "<br>";
    echo "Discounted Amount: $" . number_format($discountedAmount, 2) . "
<br>";echo "You saved: $" . number_format($totalAmount - $discountedAmount, 2);
}

// Sample call to the function
calculateDiscount(200, 'VIP');
?>
```

Task:

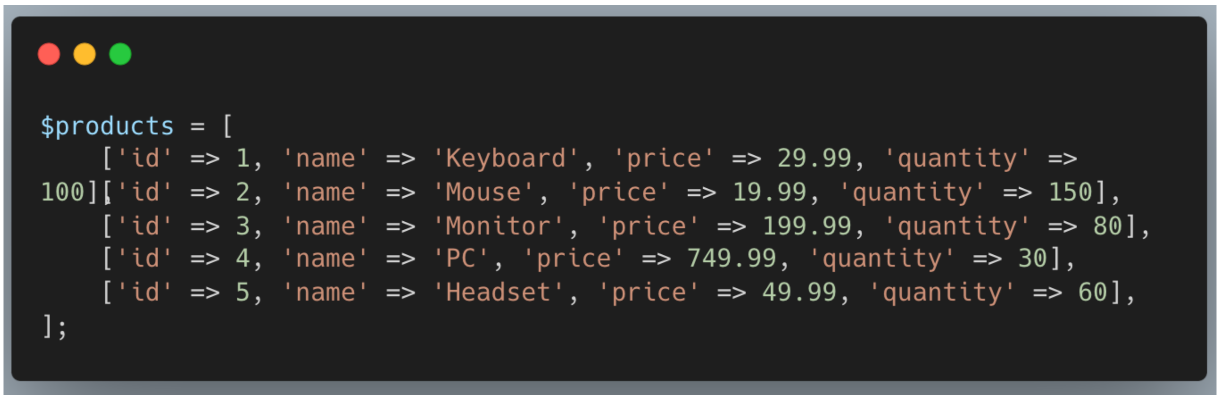
- **Identify and Correct the Error:** Review the script and identify the logical error present. Correct this error and explain why it was incorrect.
- **Enhance Error Handling:** The script currently does minimal error checking. Propose and implement improvements to make the script more robust and secure. Include your code modifications and describe how each change contributes to the script's reliability and security.
- **Explanation:** Provide a brief explanation of how your improvements enhance the functionality and error handling of the script in real-world scenarios.

QUESTION 3:

Manual Data Manipulation in PHP

In PHP, developers often rely on built-in functions to handle common data manipulation tasks such as sorting, filtering, and aggregating data. For this exercise, you will demonstrate your understanding of data structures and algorithms by performing these operations manually.

Provided Dataset:



```php
$products = [
    ['id' => 1, 'name' => 'Keyboard', 'price' => 29.99, 'quantity' =>
100]['id' => 2, 'name' => 'Mouse', 'price' => 19.99, 'quantity' => 150],
    ['id' => 3, 'name' => 'Monitor', 'price' => 199.99, 'quantity' => 80],
    ['id' => 4, 'name' => 'PC', 'price' => 749.99, 'quantity' => 30],
    ['id' => 5, 'name' => 'Headset', 'price' => 49.99, 'quantity' => 60],
];
```

Task:

- **Sorting:** Write a function in PHP to sort the array ***$products*** by price in ascending order. You should not use any built-in sorting functions. Explain the sorting algorithm you chose and why.
- **Filtering:** Implement a function to filter out products that have a quantity of less than 50 units. Again, do not use built-in array filtering functions. Describe how your function works.
- **Aggregation:** Manually calculate the total value of all products in the inventory (i.e., the sum of price * quantity for each product). Provide a function that performs this calculation without using built-in aggregation functions like ***array_sum***.

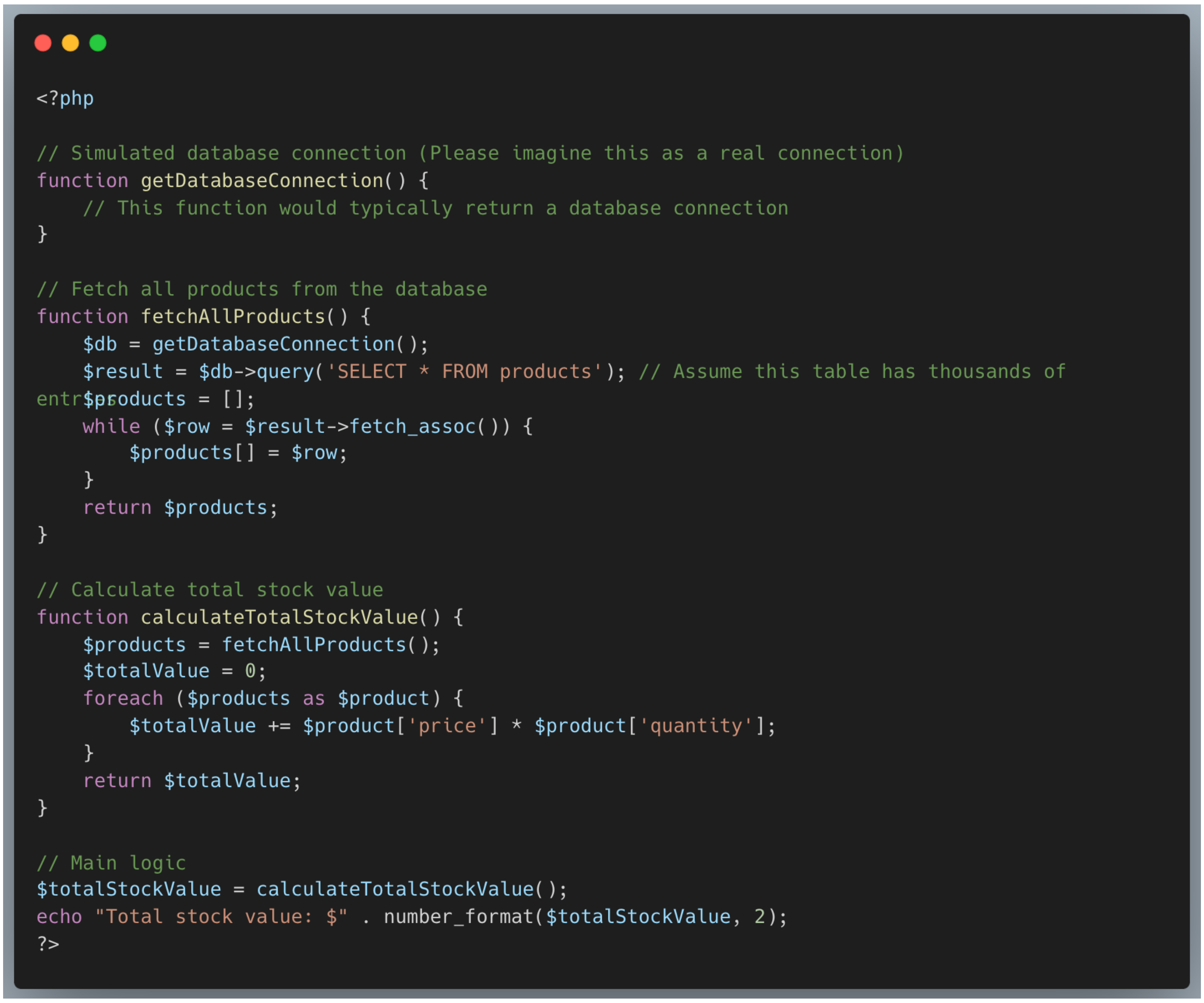
Requirements:

- Your functions should be efficient and clearly commented to explain your logic.
- Discuss the computational complexity (big O notation) of each operation you perform, and explain the potential impact on performance with larger datasets.

QUESTION 4:

PHP Performance Optimization Challenge

Performance optimization is crucial in developing efficient and scalable applications. Poorly optimized code can lead to slow response times and increased server load, affecting user experience and operational costs.



```php
<?php

// Simulated database connection (Please imagine this as a real connection)
function getDatabaseConnection() {
    // This function would typically return a database connection
}

// Fetch all products from the database
function fetchAllProducts() {
    $db = getDatabaseConnection();
    $result = $db->query('SELECT * FROM products'); // Assume this table has thousands of
entr$products = [];
    while ($row = $result->fetch_assoc()) {
        $products[] = $row;
    }
    return $products;
}

// Calculate total stock value
function calculateTotalStockValue() {
    $products = fetchAllProducts();
    $totalValue = 0;
    foreach ($products as $product) {
        $totalValue += $product['price'] * $product['quantity'];
    }
    return $totalValue;
}

// Main logic
$totalStockValue = calculateTotalStockValue();
echo "Total stock value: $" . number_format($totalStockValue, 2);
?>
```

Task:

- **Identify Bottlenecks:** Review the script and identify potential performance bottlenecks & opmtmize the code.
- **Implement Caching:** Suggest and simulate an implementation of caching to reduce the frequency of database queries. You can use a pseudo-code or describe your caching logic if actual caching setup is not feasible.
- **Evaluate Changes:** Discuss how each of your optimizations would improve the performance of the script. Include considerations such as reduced server load, faster response times, and lower resource consumption.

QUESTION 5:

Write a MySQL Query with Conditional Logic

Create a MySQL query that summarizes order statuses from an orders table. The summary should categorize the orders into different status groups based on the number of days since the order was placed.

Tables:

orders(order_id, order_date, status)

- Write a query that selects all orders and categorizes them into different groups based on the status and the number of days since the order was placed (order_date) using the **CASE** statement. The groups are as follows:
    - **"Pending Review":** *Orders with status 'Pending' and placed within the last 7 days.*
    - **"Urgent Review":** *Orders with status 'Pending' and placed more than 7 days ago.*
    - **"Processing":** *Orders with status 'Processing'.*
    - **"Shipped":** *Orders with status 'Shipped'.*
    - **"Delayed":** *Orders with status 'Processing' and placed more than 10 days ago.*
    - **"Cancelled":** *Orders with any status marked as 'Cancelled'.*
- Explain the logic behind each condition in the CASE statement.
- Discuss potential performance impacts of this query on a large dataset and suggest any indexes that might optimize its execution.

QUESTION 6:

Implement a Custom String Formatting Function

Objective: Write a PHP function that formats a given string according to user-defined rules, handling various string manipulation operations dynamically.

Background: String manipulation is a common requirement in many web applications, from formatting user input to generating text-based reports. Mastering string operations is crucial for efficient PHP development.

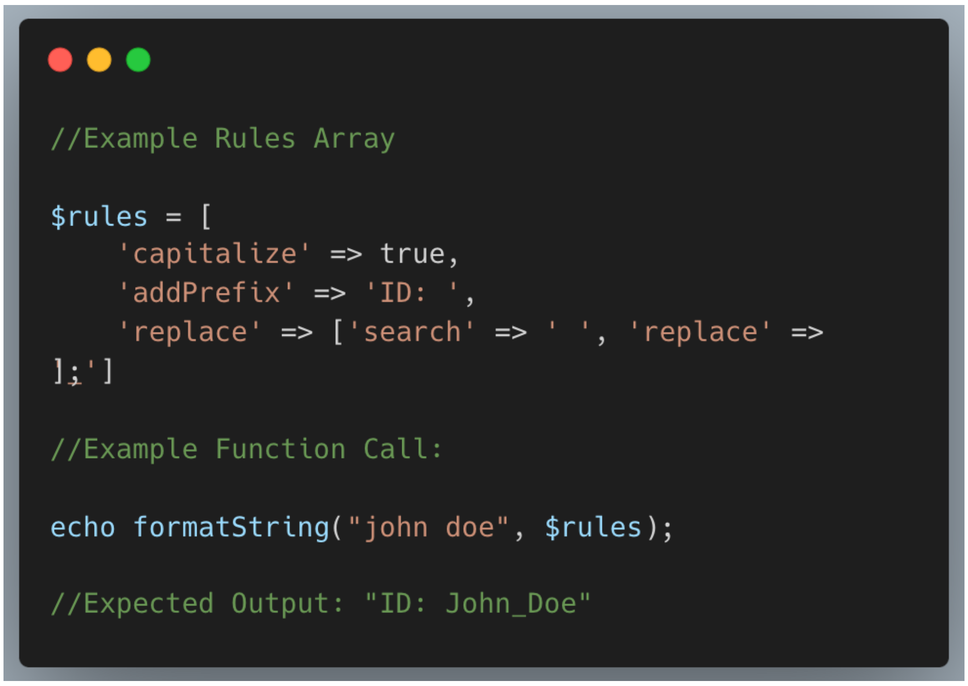
Provided Scenario:

Imagine a scenario where you need to prepare text data for a report. The data comes from various sources and may require different formatting rules to standardize its appearance and readability.

Task:

- Write a PHP function ***formatString($string, $rules)*** that:
  - Takes a string ***$string*** and an associative array ***$rules*** that defines how the string should be manipulated.
  - The ***$rules*** array can include operations like uppercase, lowercase, capitalize, addPrefix, addSuffix, and replace (for simple find and replace operations).
  - Applies the rules in the order they are defined in the array.

Example Code:



```
//Example Rules Array

$rules = [
    'capitalize' => true,
    'addPrefix' => 'ID: ',
    'replace' => ['search' => ' ', 'replace' =>
];'_']

//Example Function Call:

echo formatString("john doe", $rules);

//Expected Output: "ID: John_Doe"
```

Evaluation Criteria:

- **Functionality:** The function must apply all specified transformations correctly.
- **Flexibility:** It should handle any combination of rules defined in the ***$rules*** array.
- **Efficiency:** The function should perform operations with minimal overhead, especially important for large strings or high-volume requests.
- **Edge Cases:** Consider how your function handles cases where rules contradict or overlap (e.g., both uppercase and capitalize).

Please complete the assigned tasks within 48 hours and upload your solutions along with the documentation to a private GitHub repository.

Once completed, kindly share access to the repository by adding **[php.trogon@gmail.com]** as a collaborator. This will enable us to review your submissions promptly.