DOCUMENTATION

BULK TEMPLATE.DOCX EDITOR

Abstract

The "Bulk Template.docx Editor" is an automation system that modifies Word templates based on received data. It streamlines document generation by populating placeholders with JSON data and saving the filled-in documents. The system includes error logging, template selection, and a user interface for data editing and download.

Developer Rigoberto Moreira

rigomoreirar@gmail.com

System Overview

The "**Bulk Template.docx Editor**" is a robust automation system, specifically designed to streamline the process of modifying Microsoft Word (.docx) templates based on received data. Built primarily using PHP, it's hosted on a DigitalOcean Droplet running an Apache server, leveraging the **PhpOffice\PhpWord** library to manage Microsoft Word document operations.

The system operates by receiving a data string, which it subsequently transforms into a JSON object. This JSON object is then used to populate placeholders within a specified Word template. After filling the template, the system saves the populated document in a designated directory, from where users can download it.

The system's functionality is distributed across several PHP scripts, each handling distinct parts of the operation. Below is a brief description of each:

SSDocxReplaceplaceholders.php: This is the core script of the system. It receives incoming JSON data, processes it to extract the necessary fields, selects an appropriate template based on the data, fills in the template's placeholders using the received data, and saves the final document to a specific directory.

SSprocessTemplateData.php: This script is responsible for handling static data required by the templates, such as merchant company names, phone numbers, and other service-related information. It fetches this information from a JSON file, making it accessible for **SSDocxReplaceplaceholders.php**.

SSTrialSetter.php: This script manages values related to trial status and trial days, which are then displayed in the **UITableModifyVariables.php**'s table.

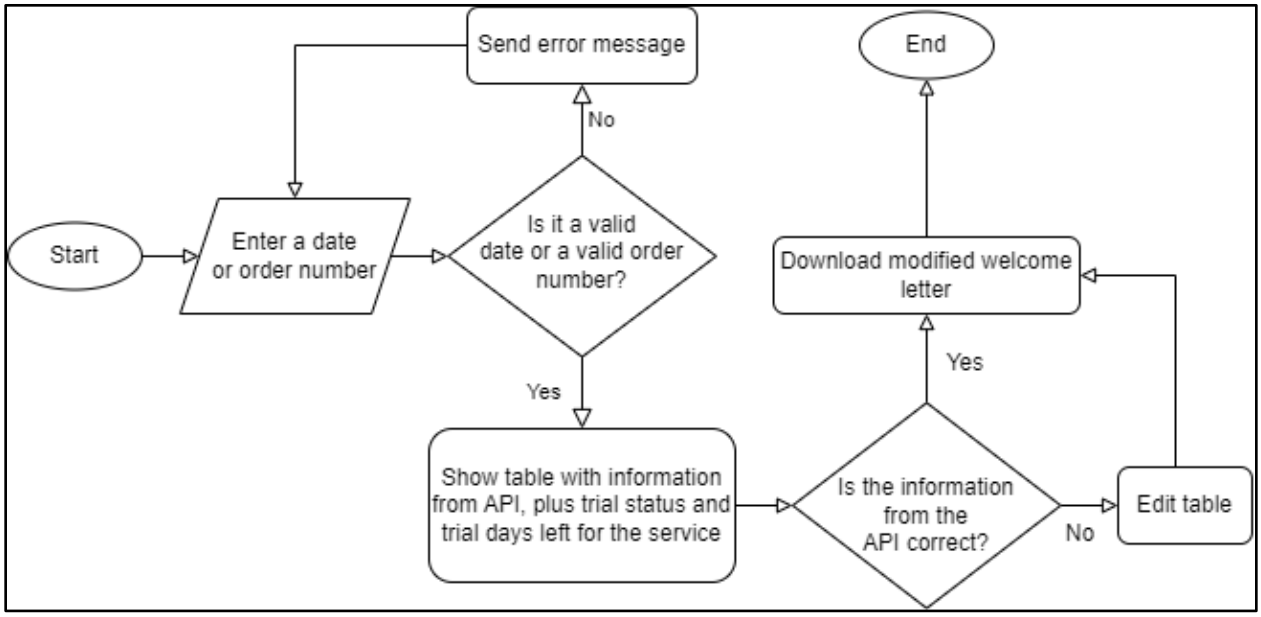
UICallAPI.php: This script is in charge of calling an external API, thereby receiving the data necessary for the system.

UITableModifyVariables.php: This script presents the embedded JSON data as a table to the user, allows the user to edit the data, and triggers **SSDocxReplaceplaceholders.php** to modify the templates and download the filled-in documents.

The system is equipped with an error logging mechanism that records any missing data from the JSON file. It generates a filename using a specific format that enables effective file tracking and records their creation time. Templates used by the system are stored in a structured

directory based on the type of order and form of payment, allowing the system to select the right template based on several criteria.

In summary, the "Bulk Template.docx Editor" is an automated solution for mass editing and generation of .docx files, depending on varied input data. Users can use a date or an order number to search for customers' information to print a customized welcome letter.



Flowchart - Figure 1

Requirements

The "Bulk Template.docx Editor" system has both technical and functional requirements to ensure efficient operation. These requirements are critical for the setup, maintenance, and seamless running of the system.

Technical Requirements

Web Server: A server to host the PHP scripts. The current deployment uses a DigitalOcean Droplet running Apache server version: Apache/2.4.54 (Ubuntu). However, any server supporting PHP should be compatible.

PHP: The system is developed on PHP, specifically version PHP 8.1.7-1ubuntu3.5, and uses the PhpOffice\PhpWord library.

Server Storage: The server should have sufficient storage capacity to store generated .docx files and original template files.

Network Access: Network access is required to receive JSON data and call external APIs.

Error Logging: A mechanism for error logging is needed for effective troubleshooting and system monitoring.

The recommended server specification for a single-user constant use is 1 GB Memory / 1 Intel vCPU / 25 GB Disk / SFO3 - Ubuntu 22.10 x64. Increased usage demands more memory and CPU resources.

Functional Requirements

Template Selection: The system should determine the appropriate template to use based on the received data.

Placeholder Replacement: The system should replace the placeholders in the selected template with the appropriate data from the received API string.

File Generation: SSDocxReplaceplaceholders.php should have read and write permissions to create the .docx files.

Static Data Handling: SSprocessTemplateData.php requires read access to a JSON file to fetch static data for the templates (e.g., merchant company name, phone number).

API Call: UICallAPI.php should successfully call an external API to receive the necessary data.

Error Handling and Logging: In the event of missing data from the API string file, the system should log this in an error log directory. This is critical for troubleshooting and ensuring data completeness.

Installation & Setup

Installing and setting up the "Bulk Template.docx Editor " system involves multiple steps and requirements. Here's a detailed guide to get the system up and running:

3.1 Server Preparation

Select a Server: Choose a server to host the system. The current deployment uses a DigitalOcean Droplet with Apache server version: **Apache/2.4.54 (Ubuntu)**. However, any server that supports PHP will do, no matter the OS.

Set Up the Environment: Install the necessary server software. For an Apache server, use the command sudo apt install apache2. Also, install PHP version: **PHP 8.1.7-1ubuntu3.5** with the command sudo apt install php. Please ensure that the installed PHP version is compatible with the **PhpOffice\PhpWord** library used in the system.

Permissions and Directories: Set up the correct permissions and directories for the PHP scripts to run correctly. Ensure that your server allows PHP scripts to create and write to files, as the system generates .docx files based on incoming data. Specifically, **SSDocxReplaceplaceholders.php** should have both read and write permissions, and **SSprocessTemplateData.php** only needs read permissions.

3.2. System Installation

Download the Scripts: Download the PHP scripts of the system: **SSDocxReplaceplaceholders.php, SSprocessTemplateData.php, SSTrialSetter.php, UICallAPI.php, UITableModifyVariables.php**. You can do this by either directly downloading them to your server or uploading them using a method like FTP.

Place the Scripts: Place the downloaded PHP scripts in a directory that's accessible by your web server. For example, in an Apache server, you might place them in the /var/www/html/ directory.

Permissions: Make sure that the scripts have the correct permissions. They need to be executable by the server. You might need to use a command like chmod to set these permissions.

3.3. System Setup

API Data: The system operates by calling an external API via **UICallAPI.php** and receiving necessary data. Ensure the details of the API call are correctly configured.

Template Files: The system uses .docx template files, with placeholders with this format **${placeholder}** to be replaced by data. Place these templates in a directory that **SSDocxReplaceplaceholders.php** can access. Update the script with the correct directory path.

File Output: Configure the location where the system will save the generated .docx files. The files are saved by **SSDocxReplaceplaceholders.php** and the script should be updated with the correct directory path.

Static Data JSON: The system uses a JSON file for static data that's used in the templates, which is handled by **SSprocessTemplateData.php**. Make sure this file is available and the path is correctly set in the script.

Trial Data: The system uses **SSTrialSetter.php** to manage values related to trial status and trial days. Ensure it's correctly configured according to service and sales policies.

User Interface: UITableModifyVariables.php allows users to view and modify embedded JSON data. Check that it can interact correctly with **SSDocxReplaceplaceholders.php**.

Error Logging: Ensure that PHP's error logging is enabled and configured correctly. This is important for troubleshooting and system monitoring.

3.4. Testing

Finally, make sure to thoroughly test the system before relying on it for critical tasks. Verify that data is correctly received, processed, and output as .docx files. Check that error logging works, and that the system behaves correctly when data is missing or incorrect. Check the **UITableModifyVariables.php** can correctly display and allow user modifications of the JSON data, and that the trial status and days are correctly managed by **SSTrialSetter.php**.

Code Explanation

This section provides an overview of the key PHP files in the "Bulk Template.docx Editor" system, describing their functionality, their role within the overall system, and some particulars about their code structure.

4.1. SSDocxReplaceplaceholders.php

SSDocxReplaceplaceholders.php is the main operational script of the system. It accepts JSON data, typically via a POST request, and executes the necessary functions to process this data. The JSON data holds keys and values used to substitute placeholders within the .docx templates. **SSDocxReplaceplaceholders.php** functions as the liaison, transmitting this data to **SSprocessTemplateData.php**. This script also handles errors and exceptions, ideally logging them for future analysis. Regular monitoring of these logs ensures the smooth operation of the system.

4.2. SSprocessTemplateData.php

SSprocessTemplateData.php plays a crucial role in the system. It processes the static JSON data containing the service information, using it to replace the placeholders within the .docx templates.

4.3. SSTrialSetter.php

SSTrialSetter.php handles the decision-making process for selecting the appropriate .docx template based on the received data. The selection mechanism can range from simple to complex, depending on the requirements, but the end goal is to find the suitable template for the incoming data.

4.4. UICallAPI.php

UICallAPI.php ensures successful interaction with external APIs. It's either used to fetch additional data for use within the .docx templates or to send notifications, logs, or other data to external systems. The specifics of this API call, such as the URL, request type, and data, can be configured in this script.

4.5. UITableModifyVariables.php

UITableModifyVariables.php provides a user interface for users to view and modify embedded JSON data. It's configured to interact correctly with **SSDocxReplaceplaceholders.php**, facilitating user modifications to the JSON data.

Core PHP code explanation

In this section, the core php code will be explained. Therefore, you will be able to modify the system as change in policies or API modifications occur over time.

UICallAPI.php

Variable Initialization:

```
$error_message = '';
$APIjson = "";
$tokenPath = "../../staticJsonInfo/APItoken.json";
$tokenJson = json_decode(file_get_contents($tokenPath), true);
$Token = $tokenJson['Token'];
```

- The $error_message variable is initialized to an empty string and will be used to store any error messages that occur during the process.
- The $APIjson variable is initialized to an empty string and will be used to store the final JSON response from the API.
- The $tokenPath variable specifies the path to a JSON file that contains the API token.
- The contents of the JSON file are loaded and decoded into an associative array using json_decode().
- The API token is extracted from the decoded JSON array and assigned to the $Token variable.

Handling HTTP POST Requests:

```
if ($_SERVER["REQUEST_METHOD"] == "POST") {
    if (isset($_POST['date'])) {
        $selectedDate = $_POST['date'];
        $url_one =
"https://app.periodicalservices.com/PSOnlineAGM/api/agm_data_query.asp?DateAdded=".$selectedDate."&Tok
en=".$Token;
    } else if (isset($_POST['orderNumber'])) {
        $orderNumber = $_POST['orderNumber'];
        $url_one =
"https://app.periodicalservices.com/PSOnlineAGM/api/agm_data_query.asp?OrderNumber=".$orderNumber."&T
oken=".$Token;
    } else {
        die('Invalid form submission.');
    }
}$Token = $tokenJson['Token'];
```

- The code checks if the current request method is POST using $_SERVER["REQUEST_METHOD"].
- If it is a POST request, it proceeds to handle the form submission.
- It checks if the 'date' parameter is set in the POST data. If so, it assigns the value to the $selectedDate variable and constructs the API URL accordingly.
- If the 'date' parameter is not set, it checks if the 'orderNumber' parameter is set. If so, it assigns the value to the $orderNumber variable and constructs the API URL accordingly.
- If neither the 'date' nor the 'orderNumber' parameter is set, it terminates the script with an error message indicating an invalid form submission.

Performing a cURL Request:

```
$ch = curl_init();
curl_setopt($ch, CURLOPT_URL, $url_one);
curl_setopt($ch, CURLOPT_RETURNTRANSFER, true);
curl_setopt($ch, CURLOPT_FOLLOWLOCATION, 1);
curl_setopt($ch, CURLOPT_HEADER, 0);
curl_setopt($ch, CURLOPT_NOBODY, 0);
curl_setopt($ch, CURLOPT_TIMEOUT, 30);
$res = curl_exec($ch);
$httpcode = curl_getinfo($ch, CURLINFO_HTTP_CODE);
curl_close($ch);
```

- A cURL session is initialized using curl_init().
- Various options are set using curl_setopt() to configure the session, such as the URL to fetch, returning the transfer as a string, following location headers, excluding the header from the output, including the body in the output, and setting a timeout.
- The cURL request is executed using curl_exec(), and the response is stored in the variable $res.
- The HTTP response code is obtained using curl_getinfo() and stored in $httpcode.
- The cURL session is closed using curl_close().

Handling the API Response:

```
if ($httpcode != 200 || $res == " PSOnline has encountered an error while generating this page. (Error ID 1083)
") {
    $error_message = 'Invalid Order Number. Please try again';
} else {
    $splitByOrderType = explode('"OrderType"', $res);
    $res = "";
    foreach ($splitByOrderType as $index => $segment) {
        if ($index !== 0) {
            $res .= '"OrderType",';
        }
        $res .= $segment;
    }
    $APIfetchArray = str_getcsv($res);
    $APIfetchArray = array_slice($APIfetchArray, 21);
    $individualItem = [
        "OrderNumber", "CustomerFirstName", "CustomerLastName", "Address", "City",
"State", "ZipCode", "HomePhone", "CellPhone", "Email", "DateAdded", "IPBillDate",
"TotalPaid", "IPAmount", "MonthlyPayments", "CostCenter", "OrderStatus",
"FormOfPayment", "MerchantAccount", "NextBillDate", "OrderType"
    ];
    $APIjson = [];
    for ($i = 0; $i < count($APIfetchArray); $i += count($individualItem)) {
        $temp = array_slice($APIfetchArray, $i, count($individualItem));
        if(count($temp) == count($individualItem)) {
            $temp[0]= trim($temp[0], "\r\n");
            $APIjson[] = array_combine($individualItem, $temp);
        }
    }
    $APIjson = json_encode($APIjson);
}
```

- If the HTTP response code is not 200 or the response contains a specific error message, it sets the $error_message variable to indicate an invalid order number.
- Otherwise, it proceeds to process the API response data:
- The response is split into segments based on the occurrence of the string "OrderType".
- The segments are then reconstructed by adding a missing comma after each occurrence of "OrderType".

- The reconstructed response is converted into an array using str_getcsv().
- The first 20 values (keys) in the array are omitted using array_slice().
- The structure of an individual array item is defined as an array containing field names.
- A final JSON array, $APIjson, is defined to hold the mapped data.
- The fetched array is iterated over in chunks of the size equal to the individual item structure.
- Each chunk is sliced from the fetched array, and if the chunk's length matches the individual item structure, it is trimmed and combined with the field names using array_combine().
- The resulting mapped arrays are appended to the $APIjson array.
- Finally, the $APIjson array is converted to JSON format using json_encode().

After executing the code, the resulting $error_message variable will contain an error message if an error occurs during the process. Otherwise, the $APIjson variable will hold the JSON response from the API if the request is successful and the response is processed correctly.

UITableModifyVariables.php

Server-Side Code:

```
<?php
include '../SSphpFiles/SSTrialSetter.php';
if ($_SERVER["REQUEST_METHOD"] == "POST") {
    $jsonData = json_decode($_POST['jsonData'], true);
    echo '<table id="dataTable">';
    echo '<tr>';
    echo '<th>Action</th><th>Submit</th><th>Order Number</th><th>Customer First Name</th>
    echo '</tr>';
    foreach ($jsonData as $key => $value) {
        list($trialStatus, $trialDays) = getTrialStatusAndDays($value);
        echo '<tr contenteditable="false">';
        echo '<td><button class="editBtn">Edit</button></td>';
        echo '<td><button class="downloadBtn">Download</button></td>';
        echo '<td>' . htmlspecialchars($value["OrderNumber"]) . '</td>';
        // ...
        echo '</tr>';
    }
    echo '</table>';
} else {
    echo 'Invalid or missing JSON data';
}
?>
```

- The code sets error reporting settings to display all errors and includes a file (SSTrialSetter.php) containing additional helper functions.
- It checks if the current request method is POST.
- If it is a POST request, it retrieves the JSON data from the jsonData parameter and decodes it into an associative array using json_decode().
- It displays the JSON data in an HTML table.
- Each row of the JSON data is iterated over, and the values are displayed in the table cells.
- The getTrialStatusAndDays() function is used to retrieve the trial status and trial days for each row.
- The table is enclosed within an if-else block to handle cases where the JSON data is invalid or missing.

JavaScript Code:

```
<script>
    document.querySelectorAll('.downloadBtn').forEach(function(btn) {
        btn.addEventListener('click', function(e) {
            e.preventDefault();
            const row = this.parentNode.parentNode;
            const data = {
                "OrderNumber": row.children[2].textContent,
                // ...
            };
            var xhttp = new XMLHttpRequest();
            xhttp.onreadystatechange = function() {
                if (this.readyState == 4 && this.status == 200) {
                    var response = JSON.parse(this.responseText);
                    var fileURL = response.filePath;
                    var fileName = response.fileName;
                    var link = document.createElement('a');
                    link.href = fileURL;
                    link.style.display = 'none';
                    link.setAttribute('download', fileName);
                    document.body.appendChild(link);
                    link.click();
                    document.body.removeChild(link);
                }
            };
```

```
            xhttp.open("POST", "../SSphpFiles/SSDocxReplaceplaceholders.php", true);
            xhttp.setRequestHeader("Content-type", "application/x-www-form-urlencoded");
            xhttp.send("process=download&data=" + JSON.stringify(data));
        });
    });
    document.getElementById('filter').addEventListener('change', function() {
        const filter = this.value;
        const table = document.getElementById('dataTable');
        const rows = Array.from(table.rows).slice(1); // Slice to skip header row
        rows.forEach(row => {
            const email = row.children[11].textContent; // Column index for "Email"
            if (filter === 'noEmails' && email.trim() !== '') {
                row.style.display = 'none';
            } else {
                row.style.display = '';
            }
        });
    });
    document.querySelectorAll('.editBtn').forEach(function(btn) {
        btn.addEventListener('click', function() {
            const row = this.parentNode.parentNode;
            const isEditable = row.isContentEditable;
            row.contentEditable = !isEditable;
            this.textContent = isEditable ? 'Edit' : 'Save';
        });
    });
</script>
```

- The code defines JavaScript functionality to handle user interactions and AJAX requests.
- It adds event listeners to all "Download" buttons for individual rows.
- When a "Download" button is clicked, it prevents the default form submission behavior.
- It extracts the data from the respective row and creates a JavaScript object (data) with the extracted values.
- It creates a new AJAX request using XMLHttpRequest().
- When the AJAX request is complete (readyState == 4 and status == 200), it parses the JSON response and retrieves the file URL and file name.

- It creates a new anchor element, sets the URL and download attribute, and triggers a click event on the anchor element to initiate the file download.
- The code adds an event listener to the "filter" dropdown, which filters the rows of the table based on the selected option.
- It adds event listeners to all "Edit" buttons, which toggle the content editable state of the row and change the button text between "Edit" and "Save".

SSTrialSetter.php

```
<?php
function getTrialStatusAndDays($data)
{
    $dateAdded = new DateTime($data["DateAdded"]);
    $IPBillDate = new DateTime($data["IPBillDate"]);
    $now = new DateTime();
    $interval = $dateAdded->diff($IPBillDate);
    $intervalToNow = $dateAdded->diff($now);
    $buyersTrialDays = 21;
    $IDTheftTrialDays = 24;
    if ($data["OrderType"] == "Buyers") {
        if ($interval->days > 3) {
            $trialStatus = "trial";
            $trialDays = $buyersTrialDays - $intervalToNow->days; // Subtract the days already passed
            if ($trialDays <= 0) {
                $trialDays = 0;
            }
        } else {
            $trialStatus = "noTrial";
            $trialDays = 0;  // No trial days as it's not a trial
        }
    } elseif ($data["OrderType"] == "IDTheft") {
        if ($data["IPAmount"] <= 3.49) {
            $trialStatus = "trial";
            // Calculate remaining trial days for IDTheft
            $trialDays = $IDTheftTrialDays - $intervalToNow->days; // Subtract the days already passed
            if ($trialDays <= 0) {
                $trialDays = 0;
            }
```

```
        } else {
            $trialStatus = "noTrial";
            $trialDays = 0;  // No trial days as it's not a trial
        }
    } elseif ($data["OrderType"] == "Telemed") {
        $trialStatus = "trial";
        $trialDays = 30;
    } else {
        $trialStatus = "missingValue";
        $trialDays = 0;  // No trial days as it's not a trial
    }
    return array($trialStatus, $trialDays);
}
?>            xhttp.open("POST", "../SSphpFiles/SSDocxReplaceplaceholders.php", true);
            xhttp.setRequestHeader("Content-type", "application/x-www-form-urlencoded");
            xhttp.send("process=download&data=" + JSON.stringify(data));
        });
    });
    document.getElementById('filter').addEventListener('change', function() {
        const filter = this.value;
        const table = document.getElementById('dataTable');
        const rows = Array.from(table.rows).slice(1); // Slice to skip header row
        rows.forEach(row => {
            const email = row.children[11].textContent; // Column index for "Email"
            if (filter === 'noEmails' && email.trim() !== '') {
                row.style.display = 'none';
            } else {
                row.style.display = '';
            }
        });
    });
    document.querySelectorAll('.editBtn').forEach(function(btn) {
        btn.addEventListener('click', function() {
            const row = this.parentNode.parentNode;
            const isEditable = row.isContentEditable;
```

```
            row.contentEditable = !isEditable;
            this.textContent = isEditable ? 'Edit' : 'Save';
        });
    });
</script>
```

- The code defines a function getTrialStatusAndDays() that takes a data array as input.
- It calculates the difference between the DateAdded and IPBillDate values using the DateTime class.
- The function initializes variables to store the trial days for different order types ($buyersTrialDays and $IDTheftTrialDays).
- Based on the OrderType value in the data array, the function determines the trial status and calculates the remaining trial days.
- If the OrderType is "Buyers", it checks if the interval between DateAdded and IPBillDate is greater than 3 days. If so, it sets the trial status to "trial" and calculates the remaining trial days by subtracting the interval from the $buyersTrialDays. Otherwise, it sets the trial status to "noTrial" and the trial days to 0.
- If the OrderType is "IDTheft", it checks if the IPAmount is less than or equal to 3.49. If so, it sets the trial status to "trial" and calculates the remaining trial days by subtracting the interval from the $IDTheftTrialDays. Otherwise, it sets the trial status to "noTrial" and the trial days to 0.
- If the OrderType is "Telemed", it sets the trial status to "trial" and the trial days to a fixed value of 30.
- If the OrderType does not match any of the specified cases, it sets the trial status to "missingValue" and the trial days to 0.
- The function returns an array containing the trial status and trial days.

SSDocxReplaceplaceholders.php

```
<?php
ini_set('display_errors', 1);
ini_set('display_startup_errors', 1);
error_reporting(E_ALL);
ini_set('memory_limit', '256M');
require_once 'SSprocessTemplateData.php';
require_once '../Dependancies/vendor/autoload.php';
use PhpOffice\PhpWord\IOFactory;
use PhpOffice\PhpWord\TemplateProcessor;
if (isset($_POST['process']) && $_POST['process'] == 'download') {
    $data = json_decode($_POST['data'], true);
    $RegBillDay = date('d', strtotime($data['IPBillDate']));
    $FormOfPayment = "";
```

```
    $legacyNew = "";
    $trialStatus = $data["TrialStatus"];
    $trialDays = $data["TrialDays"];
    if ($data["FormOfPayment"] == "Credit Card" || $data["FormOfPayment"] == "Debit Card") {
        $FormOfPayment = "CC";
    } else {
        $FormOfPayment = "PAC";
    }
    $dateAdded = $data["DateAdded"];
    $targetDate = "07/06/2023";
    $compareDate = DateTime::createFromFormat('m/d/Y', $dateAdded);
    $targetDateObj = DateTime::createFromFormat('m/d/Y', $targetDate);
    if ($compareDate >= $targetDateObj) {
        $legacyNew = "new";
        $LegacyNewJson = "New";
    } else {
        $legacyNew = "legacy";
        $LegacyNewJson = "Legacy";
    }
    $dir = "../../WLtemplates/" . $legacyNew . "/" . $data["OrderType"] . "/" . $FormOfPayment . "/" . $trialStatus .
"/template.docx";
    $templateProcessor = new TemplateProcessor($dir);
    $jsonFile = file_get_contents('../../staticJsonInfo/staticPACandCCinfo.json');
    $jsonData = json_decode($jsonFile, true);
    $paymentKey = $FormOfPayment;
    $innerKey = $FormOfPayment === 'CC' ? $data["MerchantAccount"] : $LegacyNewJson;
    $orderType = $data["OrderType"];
    $datetime = date('YmdHis');
    $result = processTemplateData($paymentKey, $innerKey, $orderType, $data, $datetime);
    $templateProcessor->setValue('MerchantCompanyName', $result['MerchantCompanyName']);
    $templateProcessor->setValue('MerchantPhone', $result['MerchantPhone']);
    $templateProcessor->setValue('Fulfillment', $result['Fulfillment']);
    $templateProcessor->setValue('TOS', $result['TOS']);
    $templateProcessor->setValue('Descriptor', $result['Descriptor']);
    $templateProcessor->setValue('trialDays', $trialDays);
    $templateProcessor->setValue('RegBillDay', $RegBillDay);
    foreach ($data as $key => $value) {
```

```
        $templateProcessor->setValue($key, $value);

    }

    $newFile = "../../modified_files/" . $data["OrderNumber"] . '_' . $data["OrderType"] . '_' . $FormOfPayment . '_' . $trialStatus . '_' . $datetime . '.docx';

    $templateProcessor->saveAs($newFile);

    $responseData = array(

        'filePath' => $newFile,

        'fileName' => $data["OrderNumber"] . '_' . $data["OrderType"] . '_' . $FormOfPayment . '_' . $trialStatus . '_' . $datetime . '.docx'

    );

    $responseJson = json_encode($responseData);

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

    echo $responseJson;

    exit;

}

?>
```

- Error Reporting and Memory Limit Settings: Set the error reporting level to display all errors, set the memory limit to 256MB.
- Include Required Files and Namespaces: Import the necessary files and namespaces for working with PHPWord.
- Check Process and Handle File Generation: Check if the process is 'download' and proceed with file generation.
- Prepare Variables: Initialize variables for storing data needed for file generation.
- Determine Form of Payment: Check the FormOfPayment value and assign the appropriate code.
- Determine Legacy or New: Compare the DateAdded with a target date to determine if the customer is legacy or new.
- Determine Template File Directory: Construct the directory path for the template file based on the form of payment, trial status, and legacy/new status.
- Create TemplateProcessor Instance: Create a TemplateProcessor instance using the template file directory.
- Get Static Data from JSON: Read the static data from the JSON file into an array.
- Determine Keys for Static Data: Determine the keys to access the static data in the JSON based on the form of payment and legacy/new status.
- Determine OrderType: Get the order type from the data.
- Process Template Data: Call the processTemplateData function to process the template data based on the keys, order type, and input data.
- Set Values in Template: Set the values from the processed template data to the template placeholders.
- Set Remaining Static Values: Set the remaining static values in the template.

- Replace Placeholders: Iterate through the input data and replace the placeholders in the template with the corresponding values.
- Generate New Filename: Generate a new filename for the modified document based on order number, order type, form of payment, trial status, and timestamp.
- Save Modified Document: Save the modified document to the modified_files directory.
- Prepare Response Data: Prepare the response data with the file path and filename of the modified document.
- Convert Response Data to JSON: Convert the response data into JSON format.
- Send JSON Response: Send the JSON response with the appropriate headers.
- Exit Script: Stop the execution of the script after sending the response.

SSprocessTemplateData.php

```
<?php

function processTemplateData($paymentKey, $innerKey, $orderType, $data, $datetime){

    $MerchantCompanyName = "";

    $MerchantPhone = "";

    $Fulfillment = "";

    $TOS = "";

    $descriptor = "";

    $jsonFile = file_get_contents('../../staticJsonInfo/staticPACandCCinfo.json');

    $jsonData = json_decode($jsonFile, true);

    if ($paymentKey == "CC") {

        if (empty(trim($MerchantCompanyName)) || empty(trim($MerchantPhone)) || empty(trim($Fulfillment)) || empty(trim($TOS)) || empty(trim($descriptor))) {

            file_put_contents('../Errorlogs/ERROR_' . $data["OrderNumber"] . $datetime . '.log', "MerchantAccount IS MISSING IN JSON FILE\nMerchantCompanyName:" . $MerchantCompanyName . "\nMerchantPhone:" . $MerchantPhone . "\nFulfillment:" . $Fulfillment . "\nTOS:" . $TOS . "\ndescriptor:" . $descriptor . "\nSET DATA TO MISSING_DATA", FILE_APPEND);

            $MerchantCompanyName = "MISSING_DATA";

            $MerchantPhone = "MISSING_DATA";

            $Fulfillment = "MISSING_DATA";

            $TOS = "MISSING_DATA";

            $descriptor = "MISSING_DATA";

            file_put_contents('../Errorlogs/ERROR_' . $data["OrderNumber"] . $datetime . '.log', "\n\nMerchantAccount IS MISSING IN JSON FILE\nMerchantCompanyName:" . $MerchantCompanyName . "\nMerchantPhone:" . $MerchantPhone . "\nFulfillment:" . $Fulfillment . "\nTOS:" . $TOS . "\ndescriptor:" . $descriptor . "\nSET DATA TO MISSING_DATA", FILE_APPEND);

        } else {

            $MerchantCompanyName = $jsonData[$paymentKey][$innerKey]["CompanyName"];

            $MerchantPhone = $jsonData[$paymentKey][$innerKey]["PhoneNumber"];

            $Fulfillment = $jsonData[$paymentKey][$innerKey]["Fulfillment"];

            $TOS = $jsonData[$paymentKey][$innerKey]["TOS"];

            $descriptor = $jsonData[$paymentKey][$innerKey]["CompanyName"] . " " . $jsonData[$paymentKey][$innerKey]["PhoneNumber"];
```

```
        }
    } else {
        $MerchantCompanyName = $jsonData[$paymentKey][$innerKey][$orderType]["CompanyName"];
        $MerchantPhone = $jsonData[$paymentKey][$innerKey][$orderType]["PhoneNumber"];
        $Fulfillment = $jsonData[$paymentKey][$innerKey][$orderType]["Fulfillment"];
        $TOS = $jsonData[$paymentKey][$innerKey][$orderType]["TOS"];
        $descriptor = $jsonData[$paymentKey][$innerKey][$orderType]["CompanyName"] . " " .
$jsonData[$paymentKey][$innerKey][$orderType]["PhoneNumber"];
    }
    return [
        'MerchantCompanyName' => $MerchantCompanyName,
        'MerchantPhone' => $MerchantPhone,
        'Fulfillment' => $Fulfillment,
        'TOS' => $TOS,
        'Descriptor' => $descriptor,
    ];
}
?>
```

- Process Template Data Function: Define a function processTemplateData to handle the processing of template data.
- Initialize Variables: Initialize variables for storing template data (MerchantCompanyName, MerchantPhone, Fulfillment, TOS, descriptor).
- Read Static Data from JSON: Read the static data from the JSON file into an array.
- Process Template Data: Process the template data based on the paymentKey, innerKey, and orderType.
- Handle Credit Card Payment: If the paymentKey is "CC", check if any of the required data fields are empty. If any field is empty, log an error in the error log file and set the corresponding variables to "MISSING_DATA". Otherwise, retrieve the data from the JSON based on the paymentKey and innerKey.
- Handle Other Payment Types: If the paymentKey is not "CC", retrieve the data from the JSON based on the paymentKey, innerKey, and orderType.
- Return Processed Template Data: Return an associative array containing the processed template data (MerchantCompanyName, MerchantPhone, Fulfillment, TOS, descriptor).