API Endpoint Setup for PHP Laravel

1. Endpoint Details

URL: https://api.yourdomain.com/api/push-data

HTTP Method: POST

Headers:

Content-Type: application/json

2. Expected JSON Payload Structure

Format for incoming data. This matches the latest database schema.

```
{
retailer_id: "amazon_uk",
retailer_name: "Amazon UK",
retailer_country: "UK",
retailer_website: "https://amazon.co.uk",
product_id: "B08KFZLX7R",
product_title: "Lenovo Yoga Slim 7",
product_description: "Slim laptop with 16GB RAM.",
promotion_type: "Loyalty Price",
promotion_description: "Save £100 on select models.",
promotion_price: 499.99,
promotion_discount: 10.0,
promotion_conditions: "Valid until December 2024.",
promotion_start_date: "2024-10-01",
promotion_expiry: "2024-12-31",
promotion_badge_type: "Clubcard Price",
rich_content_displayed: true,
rich_content_images: [
https://image.url/rich1.jpg,
https://image.url/rich2.jpg
],
timestamp: "2024-11-18T12:00:00Z"
}
```

3. Validation Rules:

The back-end should validate the incoming JSON payload against these rules. This ensures only valid data is stored in the database.

Field	Validation Rules	Example
retailer_id	`required	string`
retailer_name	`required	string`
retailer_country	`required	string`
retailer_website	`required	url`
product_id	`required	string`
product_title	`required	string`
product_description	`nullable	string`
promotion_type	`nullable	string`
promotion_description	`nullable	string`
promotion_price	`nullable	numeric`
promotion_discount	`nullable	numeric`
promotion_conditions	`nullable	string`
promotion_start_date	`nullable	date`
promotion_expiry	`nullable	date`
promotion_badge_type	`nullable	string`
rich_content_displayed	`nullable	boolean`
rich_content_images	`nullable	array`
timestamp	`required	date`

4. Implementation Steps

Implement the following steps in PHP Laravel:

1.) Create the Route Add the following route in the routes/api.php file:

```
Route::post('/push-data', [WebScrapingDataController::class, 'store']);
```

2.) Create the Controller Generate a controller using Laravel's artisan command:

```
php artisan make:controller WebScrapingDataController
```

Inside the controller, handle data validation and storage:

```
namespace App\Http\Controllers;

namespace App\Http\Controllers;

use Illuminate\Http\Request;
use Illuminate\Support\Facades\DB;
```

```
class WebScrapingDataController extends Controller
{
public function store(Request $request)
{
// Validate incoming data
$validated = $request->validate([
retailer_id' => 'required|string',
retailer_name' => 'required|string',
retailer_country' => 'required|string',
retailer_website' => 'required|url',
product_id' => 'required|string',
product_title' => 'required|string',
promotion_type' => 'nullable|string',
promotion_description' => 'nullable|string',
promotion_price' => 'nullable|numeric',
promotion_discount' => 'nullable|numeric',
promotion_conditions' => 'nullable|string',
promotion_start_date' => 'nullable|date',
promotion_expiry' => 'nullable|date',
promotion_badge_type' => 'nullable|string',
rich_content_displayed' => 'nullable|boolean',
rich_content_images' => 'nullable|array',
timestamp' => 'required|date',
]);

// Insert validated data into the database
DB::table('scraped_data')->insert($validated);

return response()->json(['message' => 'Data successfully stored'], 201);
}
}
```

3.) Set Up Middleware for Security Ensure security by:

- Using API tokens or OAuth to authenticate the web scraper.
- Adding rate limiting to the endpoint to prevent abuse.

4.) Test the API Use tools like Postman or cURL to test the endpoint with the sample JSON payload.

5.) Testing Workflow

Web Scraper: Send a POST request with the sample JSON to the endpoint.

Back-End: Validate and store the data. If validation fails, return an error message with details.

Example Response from API

If successful:

```
{
message: "Data successfully stored"
}
```

If validation fails:

```
{
errors: {
retailer_id: ["The retailer_id field is required."],
product_id: ["The product_id field is required."]
}
}
```

Deliverables for Both Teams

Web Scraping Team:

Provide a sample JSON file to ensure proper testing.

Push test data to the API endpoint once live.

Back-End Team:

Set up and share the API endpoint URL.

Validate the endpoint functionality with the provided sample JSON.

Integration Guide for Web Scraping Team

1. API Endpoint Details

Endpoint URL: https://api.yourdomain.com/api/push-data

HTTP Method: POST

Headers:

Content-Type: application/json

Authorization: Include API token if authentication is required (e.g., Bearer <API_TOKEN>).

2. Prepare the Data for Submission

Ensure the scraped data aligns with the JSON payload structure expected by the backend.

Example JSON Payload:

```
{
retailer_id: "amazon_uk",
retailer_name: "Amazon UK",
retailer_country: "UK",
retailer_website: "https://amazon.co.uk",
product_id: "B08KFZLX7R",
product_title: "Lenovo Yoga Slim 7",
product_description: "Slim laptop with 16GB RAM.",
promotion_type: "Loyalty Price",
promotion_description: "Save £100 on select models.",
promotion_price: 499.99,
promotion_discount: 10.0,
promotion_conditions: "Valid until December 2024.",
promotion_start_date: "2024-10-01",
promotion_expiry: "2024-12-31",
promotion_badge_type: "Clubcard Price",
rich_content_displayed: true,
rich_content_images: [
https://image.url/rich1.jpg,
https://image.url/rich2.jpg
],
timestamp: "2024-11-18T12:00:00Z"
}
```

3. Process for Sending Data

The scraping script should include logic to:

Scrape data from target retailer websites.

Clean and validate the scraped data (e.g., ensure no missing required fields).

Format the data into the JSON structure shown above.

Send the data to the API endpoint using an HTTP POST request.

4. Code Example for Sending Data

Here's a sample Python script using the requests library:

```
import requests
import json

# API endpoint and headers
url = "https://api.yourdomain.com/api/push-data"
headers = {
Content-Type: "application/json",
Authorization: "Bearer <API_TOKEN>"  # Add token if authentication is required
}

# Example scraped data
data = {
retailer_id: "amazon_uk",
retailer_name: "Amazon UK",
retailer_country: "UK",
retailer_website: "https://amazon.co.uk",
product_id: "B08KFZLX7R",
product_title: "Lenovo Yoga Slim 7",
product_description: "Slim laptop with 16GB RAM.",
promotion_type: "Loyalty Price",
promotion_description: "Save £100 on select models.",
promotion_price: 499.99,
promotion_discount: 10.0,
promotion_conditions: "Valid until December 2024.",
promotion_start_date: "2024-10-01",
promotion_expiry: "2024-12-31",
promotion_badge_type: "Clubcard Price",
rich_content_displayed: True,
rich_content_images: [
https://image.url/rich1.jpg,
https://image.url/rich2.jpg
],
timestamp: "2024-11-18T12:00:00Z"
}

# Send POST request
response = requests.post(url, headers=headers, data=json.dumps(data))

# Check response
if response.status_code == 201:
print("Data successfully pushed.")
else:
```

print(f"Failed to push data: {response.status_code}, {response.text}")
5. Error Handling
Ensure robust error handling to handle issues like:
Invalid data: If the API rejects the data due to validation errors, log the response and reprocess the data.
Connection issues: Retry sending data if the server is temporarily unavailable.
API limits: Implement throttling if rate limits are imposed on the API.
Example error handling in Python:
try:
response = requests.post(url, headers=headers, data=json.dumps(data))
response.raise_for_status() # Raises HTTPError for bad responses (4xx and 5xx)
print("Data successfully pushed.")
except requests.exceptions.RequestException as e:
print(f"An error occurred: {e}")
# Log the error and retry if necessary
6. Test the Integration
Before pushing live data:
Use sample JSON to test the API endpoint.
Validate that the data is successfully stored in the database.
7. Key Considerations
Authentication: Ensure you're using the correct API token if the endpoint requires authentication.
Data Validation: The web scraper should handle basic data validation to minimize rejected requests.
Retries and Failures: Implement retry logic for transient errors and log failures for debugging.
Secure Connection: Use HTTPS to encrypt the data during transmission.
8. Deliverables
Scraping team provides:
Sample JSON feed matching the structure above.
Python scripts or other programming language scripts capable of sending the data to the API.
Logs or error reports for debugging.
This detailed workflow ensures smooth integration between the web scraping data and the PHP Laravel backend. It also avoids mismatched fields or data loss.