```php
<?php

namespace App\Services;

use Illuminate\Support\Facades\Http;
use Illuminate\Support\Facades\Log;
use Illuminate\Support\Facades\Cache;

class HotmartService
{
    // Production API endpoints
    private $prodAuthUrl = 'https://api-sec-vlc.hotmart.com';
    private $prodApiUrl = 'https://api-hot-connect.hotmart.com';
    private $prodMarketplaceUrl = 'https://api.hotmart.com';
    private $prodPaymentsUrl = 'https://api-sec-vlc.hotmart.com';

    // Sandbox API endpoints
    private $sandboxBaseUrl;
    private $sandboxAuthUrl;
    private $sandboxApiUrl;
    private $sandboxMarketplaceUrl;
    private $sandboxPaymentsUrl;

    protected $accessToken;
    protected $basicAuth;
```

```
public function __construct()
{
    // Initialize sandbox URLs - using sandbox.hotmart.com as the base domain
    $this->sandboxBaseUrl = 'https://sandbox.hotmart.com';
    $this->sandboxAuthUrl = $this->sandboxBaseUrl;
    $this->sandboxApiUrl = $this->sandboxBaseUrl;
    $this->sandboxMarketplaceUrl = $this->sandboxBaseUrl;
    $this->sandboxPaymentsUrl = $this->sandboxBaseUrl;

    // Initialize basic auth string (used as an alternative to OAuth)
    $clientId = config('hotmart.client_id');
    $clientSecret = config('hotmart.client_secret');
    $this->basicAuth = base64_encode("$clientId:$clientSecret");

    Log::info('Hotmart info', [
        'client_id' => $clientId,
        'client_secret' => $clientSecret,
        'environment' => $this->isSandbox() ? 'sandbox' : 'production'
    ]);

    // Initialize access token
    $this->accessToken = $this->getAccessToken();

    // Log the environment we're using
    Log::info('Hotmart Service initialized', [
        'environment' => $this->isSandbox() ? 'sandbox' : 'production'
```

```
    ]);
}

/**
 * Determine if we're in sandbox mode
 */
public function isSandbox(): bool
{
    return !empty(config('hotmart.sandbox'));

}



/**
 * Get the appropriate auth URL based on environment
 */
public function getAuthUrl(): string
{
    return $this->isSandbox() ? $this->sandboxAuthUrl : $this->prodAuthUrl;
}

/**
 * Get the appropriate API URL based on environment
 */
public function getApiUrl(): string
{
```

```
        return $this->isSandbox() ? $this->sandboxApiUrl : $this->prodApiUrl;
    }

    /**
     * Get the appropriate marketplace URL based on environment
     */
    public function getMarketplaceUrl(): string
    {
        return $this->isSandbox() ? $this->sandboxMarketplaceUrl : $this->prodMarketplaceUrl;
    }

    /**
     * Get the appropriate payments URL based on environment
     */
    public function getPaymentsUrl(): string
    {
        return $this->isSandbox() ? $this->sandboxPaymentsUrl : $this->prodPaymentsUrl;
    }

    /**
     * Get the appropriate subscriptions URL based on environment
     */
    public function getSubscriptionsUrl(): string
    {
        return $this->isSandbox()
```

```
        ? 'https://sandbox.hotmart.com/payments/api/v1/subscriptions'
        : 'https://developers.hotmart.com/payments/api/v1/subscriptions';
}




/**
 * Get alternative subscription URLs to try in sandbox mode
 */
private function getAlternativeSubscriptionUrls(): array
{
    if (!$this->isSandbox()) {
        return [];
    }

    return [
        $this->sandboxBaseUrl . '/api/v1/subscriptions',
        $this->sandboxBaseUrl . '/v1/subscriptions',
        $this->sandboxBaseUrl . '/subscriptions',
        $this->sandboxBaseUrl . '/hot-connect/subscriptions',
        $this->sandboxBaseUrl . '/hot-connect/api/v1/subscriptions'
    ];
}

/**
 * Get access token using multiple methods
```

```
     */
    public function getAccessToken(): ?string
    {
        // Try to get from cache first
        $cachedToken = Cache::get('HOTMART_ACCESS_TOKEN');
        if ($cachedToken) {
            Log::info('Using cached Hotmart token');
            return $cachedToken;
        }

        // Try multiple authentication methods
        return $this->getTokenWithClientCredentials();
    }

    /**
     * Get token using client credentials grant
     */
    private function getTokenWithClientCredentials(): ?string
    {
        try {
            $clientId = config('hotmart.client_id');
            $clientSecret = config('hotmart.client_secret');

            if (!$clientId || !$clientSecret) {
                Log::error('Hotmart credentials not configured');
                return null;
```

```
        }

        if (!$this->basicAuth) {
            Log::error('Basic Auth not initialized');
            return null;
        }

        Log::info('Attempting to get Hotmart token with client credentials', [
            'environment' => $this->isSandbox() ? 'sandbox' : 'production'
        ]);

        $response = Http::asForm()
        ->withBasicAuth($clientId, $clientSecret) // equivalente ao -u client:secret
        ->post('https://api-sec-vlc.hotmart.com/security/oauth/token', [
            'grant_type' => 'client_credentials'
        ]);



        if ($response->successful()) {
            $data = $response->json();
            if (isset($data['access_token'])) {
                $token = $data['access_token'];
                $expiresIn = $data['expires_in'] ?? 3600;

                Log::info('Successfully obtained Hotmart token with client credentials', [
                    'expires_in' => $expiresIn,
```

```
                'token' => $token,
                'environment' => $this->isSandbox() ? 'sandbox' : 'production'
            ]);

            // Cache token for 5 minutes less than its expiration time to ensure freshness
            Cache::put('HOTMART_ACCESS_TOKEN', $token, now()->addSeconds($expiresIn - 300));
            return $token;
        }

        Log::warning('Token not found in Hotmart response', [
            'response' => $data,
            'environment' => $this->isSandbox() ? 'sandbox' : 'production'
        ]);
    } else {
        Log::error('Failed to get Hotmart token with client credentials', [
            'status' => $response->status(),
            'response' => $response->body(),
            'environment' => $this->isSandbox() ? 'sandbox' : 'production'
        ]);
    }

    return null;
} catch (\Exception $e) {
    Log::error('Exception getting Hotmart token with client credentials', [
        'error' => $e->getMessage(),
```

```
            'trace' => $e->getTraceAsString(),
            'environment' => $this->isSandbox() ? 'sandbox' : 'production'
        ]);
        return null;
    }
  }

  /**
   * Make an authenticated request to Hotmart API
   */
  protected function makeRequest(string $method, string $endpoint, array $data = [],
?string $baseUrl = null): ?array
  {
    $token = $this->getAccessToken();
    if (!$token) {
      Log::error('Cannot make Hotmart API request: No access token available', [
        'environment' => $this->isSandbox() ? 'sandbox' : 'production'
      ]);
      return null;
    }

    try {
      // Determine the URL to use
      $url = '';

      // If endpoint is already a full URL, use it directly
```

```
if (filter_var($endpoint, FILTER_VALIDATE_URL)) {
    $url = $endpoint;
} else {
    // Otherwise, combine baseUrl with endpoint
    // Use provided baseUrl or default to apiUrl
    $baseUrl = $baseUrl ?: $this->getApiUrl();

    // Ensure baseUrl doesn't end with / and endpoint doesn't start with /
    $baseUrl = rtrim($baseUrl, '/');
    $endpoint = ltrim($endpoint, '/');

    $url = $baseUrl . '/' . $endpoint;
}

//$response = Http::withToken($token)
//    ->withHeaders([
//        'Accept' => 'application/json',
//        'Content-Type' => 'application/json',
//        'User-Agent' => 'HotmartIntegration/1.0'
//    ])
//    ->$method($url, $data);

$response = Http::withHeaders([
        'Authorization' => 'Bearer ' . $token,
        'Accept' => 'application/json',
        'Content-Type' => 'application/json',
```

```
            'User-Agent' => 'HotmartIntegration/1.0' // <- Adicionado
        ])->post($url, $data);

        Log::info('📦 Payload enviado para Hotmart', ['payload' => json_encode($data)]);

        Log::info("🔄 Requisição para Hotmart", [
            'method' => $method,
            'url' => $url,
            'data' => $data,
            'token_preview' => app()->environment('local') ? substr($token, 0, 10) . '...' :
'[hidden]',
            'environment' => $this->isSandbox() ? 'sandbox' : 'production'
        ]);

        Log::info('🔍 Resposta bruta Hotmart', [
            'status' => $response->status(),
            'headers' => $response->headers(),
            'body' => $response->body(),
            'url' => $url
        ]);

        if ($response->successful()) {
            Log::info('✅ Resposta da API Hotmart com sucesso', [
                'status' => $response->status(),
                'content_type' => $response->header('Content-Type'),
                'is_json' => $response->header('Content-Type') === 'application/json',
```

```
            'body' => $response->body(),
        ]);

        return $response->json();
    }

    Log::error('❌ Erro na chamada da API Hotmart', [
        'status' => $response->status(),
        'headers' => $response->headers(),
        'raw_body' => $response->body(),
        'url' => $url,
        'method' => $method,
        'data_sent' => $data
    ]);

    return null;
  } catch (\Exception $e) {
    Log::error("Exception in Hotmart API request: $method $url", [
        'error' => $e->getMessage(),
        'environment' => $this->isSandbox() ? 'sandbox' : 'production'
    ]);
    return null;
  }
}

/**
```

```
     * Create an order in Hotmart
     */
    public function createOrder(array $orderData): ?array
    {
        try {
            // In production, just use the standard URL
            if (!$this->isSandbox()) {

                $subscriptionsUrl = $this->getSubscriptionsUrl();

                Log::info('🚀 Criando nova assinatura no Hotmart', [
                    'orderData' => $orderData,
                    'url' => $this->getSubscriptionsUrl(),
                    'sandbox' => $this->isSandbox()
                ]);

                return $this->makeRequest('post', $subscriptionsUrl, $orderData);
            }

            // In sandbox mode, try multiple endpoints until one works
            $urls = array_merge([$this->getSubscriptionsUrl()], $this->getAlternativeSubscriptionUrls());

            Log::info("Attempting to create order in sandbox mode with multiple endpoints", [
                'urls_to_try' => $urls,
                'data_sample' => [
```

```
            'buyer' => $orderData['buyer'] ?? null,
            'product' => $orderData['product'] ?? null,
            'payment_type' => $orderData['payment']['type'] ?? null
        ]
    ]);

    foreach ($urls as $url) {
        Log::info("Trying to create order with URL: $url", ['environment' => 'sandbox']);

        $token = $this->getAccessToken();
        if (!$token) {
            Log::error("No access token available for Hotmart API", ['environment' =>
'sandbox']);
            continue;
        }

        $response = Http::withToken($token)
            ->withHeaders([
                'Accept' => 'application/json',
                'Content-Type' => 'application/json',
                'User-Agent' => 'HotmartIntegration/1.0'
            ])
            ->post($url, $orderData);

        if ($response->successful()) {
            Log::info("Successfully created order with URL: $url", [
```

```
                'environment' => 'sandbox',
                'response' => $response->json()
            ]);
            return $response->json();
        }

        Log::warning("Failed to create order with URL: $url", [
            'status' => $response->status(),
            'response' => $response->body(),
            'environment' => 'sandbox'
        ]);
    }

    Log::error("All attempts to create order in sandbox mode failed", [
        'environment' => 'sandbox',
        'tried_urls' => $urls
    ]);

    return null;
} catch (\Exception $e) {
    Log::error("Exception in createOrder", [
        'error' => $e->getMessage(),
        'trace' => $e->getTraceAsString(),
        'environment' => $this->isSandbox() ? 'sandbox' : 'production'
    ]);
    return null;
```

```
    }
  }

  /**
   * Get product information - try multiple endpoints
   */
  public function getProduct(string $productId): ?array
  {
    // Try multiple endpoints
    $endpoints = [
      ['url' => "/payments/api/v1/products/$productId", 'base' => $this->getPaymentsUrl()],
      ['url' => "/products/api/v1/products/$productId", 'base' => $this->getMarketplaceUrl()],
      ['url' => "/v2/product/$productId", 'base' => $this->getApiUrl()]
    ];

    foreach ($endpoints as $endpoint) {
      Log::info("Trying to get product from endpoint: {$endpoint['base']}{$endpoint['url']}", [
        'environment' => $this->isSandbox() ? 'sandbox' : 'production'
      ]);
      $result = $this->makeRequest('get', $endpoint['url'], [], $endpoint['base']);
      if ($result) {
        return $result;
      }
    }
```

```
    return null;
}

/**
 * Update product price
 */
public function updateProductPrice(string $productId, float $price): bool
{
    $result = $this->makeRequest('put', "/payments/api/v1/products/$productId/price", [
        'price' => $price
    ], $this->getPaymentsUrl());

    return $result !== null;
}

/**
 * Pause subscription
 */
public function pauseSubscription(string $subscriptionCode): bool
{
    $endpoint = "/payments/api/v1/subscriptions/$subscriptionCode/actions/suspend";
    $result = $this->makeRequest('post', $endpoint, [], $this->getPaymentsUrl());
    return $result !== null;
}

/**
```

```
     * Resume subscription
     */
    public function resumeSubscription(string $subscriptionCode): bool
    {
        $endpoint = "/payments/api/v1/subscriptions/$subscriptionCode/actions/reactivate";
        $result = $this->makeRequest('post', $endpoint, [], $this->getPaymentsUrl());
        return $result !== null;
    }

    /**
     * Get product price - try multiple approaches
     */
    public function getProductPrice(string $productId): ?float
    {
        // Try getting from product info first
        $product = $this->getProduct($productId);
        if ($product) {
            // Check different possible price fields
            $priceFields = ['price', 'Price', 'value', 'amount', 'basePrice'];
            foreach ($priceFields as $field) {
                if (isset($product[$field])) {
                    return floatval($product[$field]);
                }
            }

            // Check for nested price objects
```

```
        if (isset($product['price']) && is_array($product['price']) &&
isset($product['price']['value'])) {
            return floatval($product['price']['value']);
        }

        // Log the product structure to help debug
        Log::info("Product found but price field not identified. Product structure:", [
            'product' => $product,
            'environment' => $this->isSandbox() ? 'sandbox' : 'production'
        ]);
    }

    // Try direct price endpoint as fallback
    $endpoints = [
        ['url' => "/payments/api/v1/products/$productId/price", 'base' => $this-
>getPaymentsUrl()],
        ['url' => "/products/api/v1/products/$productId/price", 'base' => $this-
>getMarketplaceUrl()]
    ];

    foreach ($endpoints as $endpoint) {
        Log::info("Trying to get price from endpoint: {$endpoint['base']}{$endpoint['url']}", [
            'environment' => $this->isSandbox() ? 'sandbox' : 'production'
        ]);
        $result = $this->makeRequest('get', $endpoint['url'], [], $endpoint['base']);
        if ($result && isset($result['price'])) {
            return floatval($result['price']);
```

```
        }
    }

    return null;
}

/**
 * Sync prices from Hotmart to local system
 */
public function syncPricesFromHotmart(): array
{
    $results = [];
    $agents = \App\Models\Agent::whereNotNull('hotmart_product_id')->get();

    foreach ($agents as $agent) {
        if ($agent->hotmart_product_id) {
            $productData = $this->getProduct($agent->hotmart_product_id);

            if ($productData && isset($productData['price'])) {
                $hotmartPrice = $productData['price'];

                if ($agent->price != $hotmartPrice) {
                    $agent->update(['price' => $hotmartPrice]);

                    $results[] = [
                        'agent_id' => $agent->id,
```

```
                    'agent_name' => $agent->name,
                    'old_price' => $agent->price,
                    'new_price' => $hotmartPrice,
                    'updated' => true,
                    'environment' => $this->isSandbox() ? 'sandbox' : 'production'
                ];
            }
        }
    }
}

    return $results;
}

/**
 * Sync all product prices to Hotmart
 */
public function syncAllProductPrices(): array
{
    $results = [];
    $agents = \App\Models\Agent::whereNotNull('hotmart_product_id')->get();

    foreach ($agents as $agent) {
        if ($agent->hotmart_product_id) {
            $success = $this->updateProductPrice($agent->hotmart_product_id, $agent->price);
```

```
            $results[] = [
                'agent_id' => $agent->id,
                'agent_name' => $agent->name,
                'product_id' => $agent->hotmart_product_id,
                'price' => $agent->price,
                'success' => $success,
                'environment' => $this->isSandbox() ? 'sandbox' : 'production'
            ];
        }
    }

    return $results;
  }
}
```