Integrating Laravel with Third-Party APIs (Mail, SMS, Payment)

Integrating Laravel with Third-Party APIs (Mail, SMS, Payment)

Most apps rely on third-party APIs for critical features: sending emails, delivering SMS, and processing payments. Laravel makes it easy with built-in Mail and Notification facades, and clean HTTP client integration for external services. In this guide, you'll integrate Mailgun for email, Twilio for SMS, and Stripe for payments, complete with secure validation and example UI.

1 – Mail Integration with Mailgun

Laravel's `Mail` facade supports many drivers, including Mailgun. Configure credentials and send a test message.

# .env MAIL_MAILER=mailgun MAILGUN_DOMAIN=your-domain.mailgun.org MAILGUN_SECRET=your-mailgun-api-key MAIL_FROM_ADDRESS=no-reply@example.com MAIL_FROM_NAME="MyApp"

These credentials authenticate Laravel against Mailgun's API. `MAIL_MAILER` selects the driver, and `MAIL_FROM_ADDRESS` defines the sender identity visible to users.

```
// app/Mail/WelcomeMail.php
namespace App\Mail;

use Illuminate\Bus\Queueable;
use Illuminate\Mail\Mailable;
use Illuminate\Queue\SerializesModels;
```

```php
class WelcomeMail extends Mailable
{
    use Queueable, SerializesModels;

    public function __construct(public string $name) {}

    public function build()
    {
        return $this->subject('Welcome!')
            ->view('emails.welcome');
    }
}
```
Code language: PHP (php)

This mailable builds a message using a Blade view. You can pass variables like `$name` into the template for personalization.

```php
// resources/views/emails/welcome.blade.php
<h1>Welcome, {{ $name }}!</h1>
<p>Thanks for joining our platform.</p>
```
Code language: PHP (php)

The Blade template defines the HTML body. Laravel will send it via Mailgun with proper headers.

2 - SMS Integration with Twilio

SMS notifications keep users engaged. Use Twilio's REST API to send messages.

# .env TWILIO_SID=your-twilio-sid TWILIO_TOKEN=your-twilio-auth-token TWILIO_FROM=+1234567890

The SID and Token authenticate your API requests. `TWILIO_FROM` is your Twilio number that messages are sent from.

```
// app/Services/TwilioService.php
namespace App\Services;

use Illuminate\Support\Facades\Http;

class TwilioService
{
    public function sendSms(string $to, string $message): bool
    {
        $response = Http::withBasicAuth(
            config('services.twilio.sid'),
            config('services.twilio.token')
        )->asForm()->post(
'https://api.twilio.com/2010-04-01/Accounts/'.config('services.twilio.
sid').'/Messages.json',
            [
                'From' => config('services.twilio.from'),
                'To'   => $to,
                'Body' => $message,
            ]
        );

        return $response->successful();
    }
}
```
Code language: PHP (php)

This service wraps Twilio's API. It sends form-encoded requests with credentials and message details. The `Http` client simplifies authentication and error handling.

```
// config/services.php
'twilio' => [
    'sid'   => env('TWILIO_SID'),
    'token' => env('TWILIO_TOKEN'),
    'from'  => env('TWILIO_FROM'),
],
```
Code language: PHP (php)

Centralize Twilio config in `services.php` to keep code clean and manageable.

3 – Payment Integration with Stripe

Stripe provides a secure API for handling payments. Install the PHP SDK and connect via Laravel's service container.

```
composer require stripe/stripe-phpCode language: Bash (bash)
```

# .env STRIPE_SECRET=sk_test_your_secret STRIPE_KEY=pk_test_your_public_key

The `STRIPE_SECRET` is used on the server to create charges. The `STRIPE_KEY` is exposed on the frontend to generate payment methods securely.

```
// app/Http/Controllers/PaymentController.php
namespace App\Http\Controllers;

use Illuminate\Http\Request;
use Stripe\Stripe;
use Stripe\PaymentIntent;

class PaymentController extends Controller
{
    public function createIntent(Request $request)
    {
        $request->validate([
            'amount' => 'required|integer|min:100', // cents
            'currency' => 'required|string',
        ]);

        Stripe::setApiKey(config('services.stripe.secret'));

        $intent = PaymentIntent::create([
            'amount'   => $request->amount,
            'currency' => $request->currency,
            'metadata' => ['user_id' => $request->user()->id],
        ]);

        return response()->json(['client_secret' =>
$intent->client_secret]);
    }
}
```
Code language: PHP (php)

This controller endpoint creates a Payment Intent with Stripe. The mobile app uses the returned `client_secret` to complete payment using Stripe's mobile SDKs (iOS/Android). Metadata helps link the payment to your DB records.

Related Reading: For a full step-by-step tutorial on payments, see How to Integrate Stripe Payments in Laravel, where we cover card forms, error handling, and secure webhook validation.

```
// config/services.php
'stripe' => [
    'secret' => env('STRIPE_SECRET'),
    'key'    => env('STRIPE_KEY'),
],
```
Code language: PHP (php)

Centralizing Stripe config makes swapping keys easy between dev, staging, and prod environments.

4 - Minimal UI Tester

Here's a basic Blade UI to test sending email, SMS, and creating a payment intent from your browser (use your test credentials).

```
<!-- resources/views/api/tester.blade.php -->
@extends('layouts.app')
```

```
@section('content')
<div class="container">
  <h1>API Integration Tester</h1>

  <form method="POST" action="/api/test/mail" class="mb-4">
    @csrf
    <input type="text" name="email" placeholder="Email" class="form-
control mb-2">
    <button class="btn btn-theme">Send Test Mail</button>
  </form>

  <form method="POST" action="/api/test/sms" class="mb-4">
    @csrf
    <input type="text" name="to" placeholder="Phone Number"
class="form-control mb-2">
    <input type="text" name="message" placeholder="Message"
class="form-control mb-2">
    <button class="btn btn-secondary">Send SMS</button>
  </form>

  <form method="POST" action="/api/test/payment">
    @csrf
    <input type="number" name="amount" placeholder="Amount in cents"
class="form-control mb-2">
    <input type="text" name="currency" placeholder="Currency"
class="form-control mb-2" value="usd">
    <button class="btn btn-success">Create Payment Intent</button>
  </form>
</div>
@endsection
```
Code language: HTML, XML (xml)

This UI lets you quickly verify Mail, SMS, and Stripe integrations with test credentials. It posts to your test routes which internally use Mailgun, Twilio, and Stripe APIs.

Wrapping Up

You integrated three common APIs into a Laravel app: Mailgun for emails, Twilio for SMS, and Stripe for payments. You learned how to configure credentials, call APIs with Laravel's Http client or SDKs, and secure sensitive keys. With consistent services and a tester UI, you can extend this approach to any third-party API.

What's Next

- PayPal Integration in Laravel (Step by Step) — if you prefer PayPal, this article shows the full flow of creating and capturing payments.
- How to Build a Secure File Upload API in Laravel — essential if your app also needs to handle documents or images alongside payments.
- Building a Mobile App Backend with Laravel 12 API — learn how to expose these integrations cleanly to iOS and Android apps.