How to Build a Subscription System with Stripe and Laravel?

Subscriptions are a common requirement for SaaS and membership-based applications. While Laravel Cashier provides a great abstraction for subscription billing, sometimes you want finer control over the subscription flow, especially when working directly with Stripe APIs. In this guide, we'll build a subscription system in Laravel with Stripe, covering plans, webhooks, billing UI, and subscription lifecycle management (start, cancel, resume). We'll also integrate Blade forms with Stripe Elements for secure payments.

Stripe Setup

Create products and prices in your Stripe Dashboard. Each price ID (e.g., `price_basic`, `price_pro`) will represent a subscription plan. Then add your keys to `.env`:

```
STRIPE_KEY=pk_test_123
STRIPE_SECRET=sk_test_123Code language: Bash (bash)
```

Install Stripe's PHP SDK:

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

Database & User Model

We'll store subscription details in the database so the app can track user subscriptions

locally.

```
php artisan make:migration create_subscriptions_tableCode language: Bash (bash)
```

```
// database/migrations/xxxx_xx_xx_create_subscriptions_table.php
Schema::create('subscriptions', function (Blueprint $table) {
    $table->id();
    $table->foreignId('user_id')->constrained()->onDelete('cascade');
    $table->string('stripe_id');
    $table->string('stripe_status');
    $table->string('stripe_price');
    $table->timestamp('ends_at')->nullable();
    $table->timestamps();
});Code language: PHP (php)
```

In your `User` model, define a relation:

```
public function subscription()
{
    return $this->hasOne(\App\Models\Subscription::class);
}Code language: PHP (php)
```

Subscription Controller

```
php artisan make:controller StripeSubscriptionControllerCode language: Bash (bash)
```

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

use App\Models\Subscription;
use Illuminate\Http\Request;
use Stripe\StripeClient;
```

```
class StripeSubscriptionController extends Controller
{
    protected StripeClient $stripe;

    public function __construct()
    {
        $this->stripe = new
StripeClient(config('services.stripe.secret'));
    }

    public function index(Request $request)
    {
        return view('subscriptions.index', [
            'intent' => $request->user()->createSetupIntent(),
            'plans' => [
                'Basic' => 'price_basic',
                'Pro' => 'price_pro'
            ]
        ]);
    }

    public function subscribe(Request $request)
    {
        $request->validate([
            'paymentMethod' => 'required',
            'price_id' => 'required|string'
        ]);

        $user = $request->user();
        $customer = $this->stripe->customers->create([
            'email' => $user->email,
            'payment_method' => $request->paymentMethod,
            'invoice_settings' => ['default_payment_method' =>
$request->paymentMethod],
        ]);

        $subscription = $this->stripe->subscriptions->create([
            'customer' => $customer->id,
```

```
            'items' => [['price' => $request->price_id]],
        ]);

        Subscription::create([
            'user_id' => $user->id,
            'stripe_id' => $subscription->id,
            'stripe_status' => $subscription->status,
            'stripe_price' => $request->price_id,
        ]);

        return redirect()->route('subscriptions.index')
            ->with('success','Subscription created!');
    }

    public function cancel(Request $request)
    {
        $subscription = $request->user()->subscription;

        if ($subscription) {
$this->stripe->subscriptions->cancel($subscription->stripe_id);
            $subscription->update(['stripe_status' => 'canceled']);
        }

        return back()->with('success','Subscription canceled.');
    }
}Code language: PHP (php)
```

This controller handles creating customers, subscribing them to a plan, and canceling subscriptions. Webhooks should also be configured for status updates.

Routes

```
// routes/web.php
use App\Http\Controllers\StripeSubscriptionController;

Route::middleware(['auth'])->group(function () {
    Route::get('/subscriptions',
[StripeSubscriptionController::class,'index'])->name('subscriptions.in
dex');
    Route::post('/subscriptions/subscribe',
[StripeSubscriptionController::class,'subscribe'])->name('subscription
s.subscribe');
    Route::post('/subscriptions/cancel',
[StripeSubscriptionController::class,'cancel'])->name('subscriptions.c
ancel');
});Code language: PHP (php)
```

Blade Subscription Form

```
<!-- resources/views/subscriptions/index.blade.php -->
<h2>Choose a Plan</h2>

@if(session('success'))
    <p style="color:green">{{ session('success') }}</p>
@endif

<form id="subscription-form" method="POST" action="{{
route('subscriptions.subscribe') }}">
    @csrf
    <select name="price_id">
        @foreach($plans as $name => $id)
            <option value="{{ $id }}">{{ $name }}</option>
```

```
        @endforeach
    </select>

    <label>Card</label>
    <div id="card-element"></div>

    <input type="hidden" name="paymentMethod" id="payment-method" />
    <button type="submit">Subscribe</button>
</form>

<form method="POST" action="{{ route('subscriptions.cancel') }}">
    @csrf
    <button>Cancel Subscription</button>
</form>

<script src="https://js.stripe.com/v3/"></script>
<script>
const stripe = Stripe('{{ config('services.stripe.key') }}');
const elements = stripe.elements();
const card = elements.create('card');
card.mount('#card-element');

const form = document.getElementById('subscription-form');
form.addEventListener('submit', async (e) => {
    e.preventDefault();
    const {paymentMethod, error} = await
stripe.createPaymentMethod('card', card);
    if (error) {
        alert(error.message);
    } else {
        document.getElementById('payment-method').value =
paymentMethod.id;
        form.submit();
    }
});
</script>Code language: PHP (php)
```

This form allows users to pick a plan, enter card details via Stripe Elements, and create a

subscription securely.

Handling Webhooks

Stripe will send events (invoice paid, subscription canceled, etc.) to your webhook endpoint. Configure in Stripe Dashboard and add a route in Laravel:

```
// routes/web.php
Route::post('/stripe/webhook',
\Laravel\Cashier\Http\Controllers\WebhookController::class);Code language: PHP (php)
```

Alternatively, create a custom controller to update your `subscriptions` table based on events like `customer.subscription.deleted`.

Wrapping Up

We've built a subscription system in Laravel with Stripe, covering plans, controllers, routes, Blade UI, and webhook handling. While Cashier provides a higher-level abstraction, rolling your own integration gives you more control over lifecycle events and custom billing logic. With this setup, you can extend functionality with plan upgrades, discounts, and multi-tenant billing features.

What’s Next

Expand your subscription system with these guides:

- Building a Simple SaaS Billing System with Laravel and Cashier
- How to Integrate Stripe Payments in Laravel
- PayPal Integration in Laravel (Step by Step)