Building a Simple SaaS Billing System with Laravel and Cashier

Billing is at the heart of every SaaS application. Laravel provides a powerful integration with Stripe and Paddle via **Laravel Cashier**, making it easy to implement subscriptions, handle invoices, manage trials, and process payments. In this article, you'll learn how to build a simple SaaS billing system with Cashier, starting from installation and environment setup to building UI components for subscription management. We'll also cover migrations, controllers, Blade templates, and testing to ensure a production-ready billing system.

Installing Laravel Cashier

First, install Cashier for Stripe (most common) or Paddle. We'll use Stripe in this example.

```
composer require laravel/cashierCode language: Bash (bash)
```

Next, add Stripe environment keys to your `.env` file:

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

Ensure you've configured webhooks in your Stripe dashboard to point to `/stripe/webhook`. Cashier will handle these automatically.

User Model & Database Setup

Add the `Billable` trait to your `User` model so it can manage subscriptions and payment methods.

```
// app/Models/User.php
use Laravel\Cashier\Billable;

class User extends Authenticatable
{
    use Billable;
}Code language: PHP (php)
```

Publish and run Cashier's migrations:

```
php artisan vendor:publish --tag="cashier-migrations"
php artisan migrateCode language: Bash (bash)
```

This creates the necessary tables (`subscriptions`, `subscription_items`, etc.) to track active subscriptions and payments.

Controller for Subscription Management

Create a controller to handle showing the billing page, starting a subscription, and canceling subscriptions.

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

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

use Illuminate\Http\Request;
```

```
class SubscriptionController extends Controller
{
    public function index(Request $request)
    {
        return view('billing.index', [
            'intent' => $request->user()->createSetupIntent()
        ]);
    }

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

        $request->user()->newSubscription('default', 'price_basic')
            ->create($request->paymentMethod);

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

    public function cancel(Request $request)
    {
        $request->user()->subscription('default')?->cancel();
        return back()->with('success','Subscription canceled.');
    }
}Code language: PHP (php)
```

This controller allows users to subscribe to a Stripe price ID (`price_basic`) and cancel their subscriptions. Replace with your Stripe product price IDs.

Routes & Middleware

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

Route::middleware(['auth'])->group(function () {
    Route::get('/billing',
[SubscriptionController::class,'index'])->name('billing.index');
    Route::post('/billing/subscribe',
[SubscriptionController::class,'subscribe'])->name('billing.subscribe'
);
    Route::post('/billing/cancel',
[SubscriptionController::class,'cancel'])->name('billing.cancel');
});Code language: PHP (php)
```

Restrict billing routes to authenticated users so each subscription ties to a user account.

Blade Billing UI

```
<!-- resources/views/billing/index.blade.php -->
<h2>Billing</h2>

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

@if(! auth()->user()->subscribed('default'))
    <form id="subscribe-form" method="POST" action="{{
route('billing.subscribe') }}">
        @csrf
        <label>Card</label>
        <div id="card-element"></div>
```

```
        <input type="hidden" name="paymentMethod" id="payment-method"
/>
        <button type="submit">Subscribe</button>
    </form>
@else
    <p>You are subscribed.</p>
    <form method="POST" action="{{ route('billing.cancel') }}">
        @csrf
        <button>Cancel Subscription</button>
    </form>
@endif

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

const form = document.getElementById('subscribe-form');
if (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 UI integrates Stripe Elements to securely capture card details. On subscribe, it creates a payment method and posts it to Laravel.

Invoices and Trials

Cashier includes helpers for invoices and trials:

```
// Display invoices
$invoices = auth()->user()->invoices();

// Add free trial on subscription creation
$request->user()->newSubscription('default', 'price_basic')
    ->trialDays(14)
    ->create($request->paymentMethod);Code language: PHP (php)
```

You can display invoices in a Blade template and allow users to download PDF receipts easily with `$invoice->asPdf()`.

Testing the Billing Flow

You can fake Stripe in tests using Laravel's HTTP client mocking or by using Stripe test cards in a sandbox environment.

```
// tests/Feature/SubscriptionTest.php
use Tests\TestCase;

class SubscriptionTest extends TestCase
{
    public function test_user_sees_billing_page()
    {
```

```
        $user = User::factory()->create();
        $this->actingAs($user)
            ->get('/billing')
            ->assertStatus(200);
    }
}Code language: PHP (php)
```

Feature tests confirm authenticated users can access billing pages. For integration tests, use Stripe's test keys and cards to simulate subscriptions.

Wrapping Up

With Laravel Cashier, you can implement SaaS billing quickly and reliably. You've seen how to set up Cashier, manage subscriptions, integrate Stripe Elements, handle trials, and provide invoices. Extend this foundation with plan upgrades/downgrades, team billing, coupons, and advanced reporting as your SaaS grows.

What's Next

Continue learning about payments and subscriptions in Laravel:

- How to Integrate Stripe Payments in Laravel
- PayPal Integration in Laravel (Step by Step)
- Handling File Uploads and Image Storage in Laravel