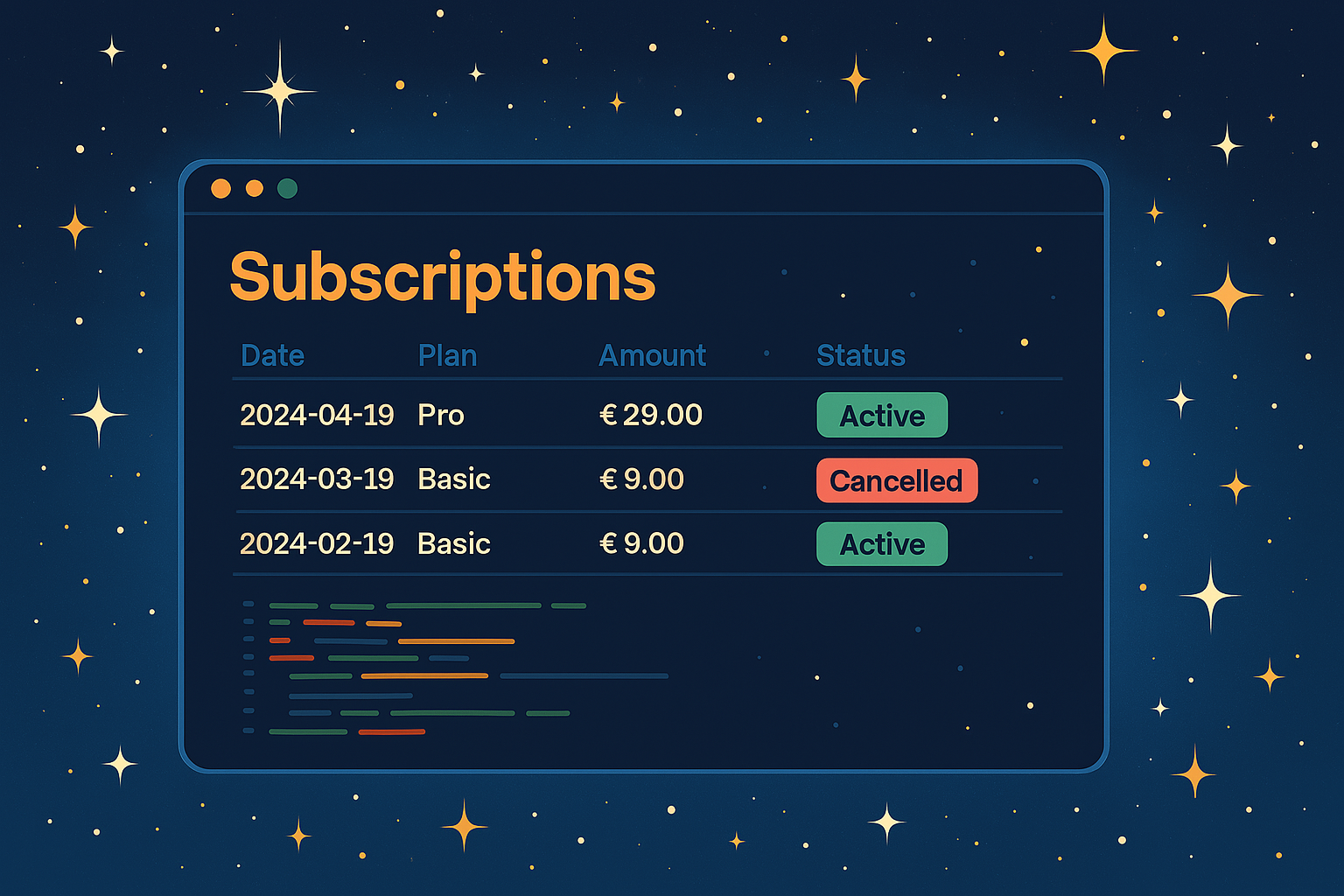
**💸 Laravel Cashier Saga – “Betaal of Betaal Niet... er is geen Probeer!”**

**Amine Ben Slimene, Didier Vanassche, Robin Caulier**

Laatste update: 05-05-2025



Welkom, jonge Padawan van de code.

In een sterrenstelsel hier niet ver vandaan, worstelden ontwikkelaars met formulieren, webhooks en mysterieuze Stripe-errors. Sommigen gaven op… anderen zochten hulp bij het Orakel van StackOverflow...

Maar jij? Jij koos het pad van de Cashier. 🧘‍♂️💳

In dit episch driedaags avontuur leer je hoe je een professioneel abonnementsysteem bouwt met **Laravel Cashier**, **Stripe**, en de kracht van **Livewire**.

Dit is geen saaie gids met grijze muren en console-commando’s zonder ziel – nee, dit is een vrolijke reis vol code, humor, en Jedi-wijsheden.

Bereid je voor op:  
 ✅ Magische migraties  
 ✅ Betalingsportalen die zelfs Jabba de Hutt jaloers maken  
 ✅ Facturen met de elegantie van een lightsaber  
 ✅ En gebruikersprofielen die duidelijk zeggen: "Ik ben Premium. Deal with it."

**🧙‍ Dag 1: De Eerste Stap naar Cashier Meesterschap – Installatie en Initiatie**

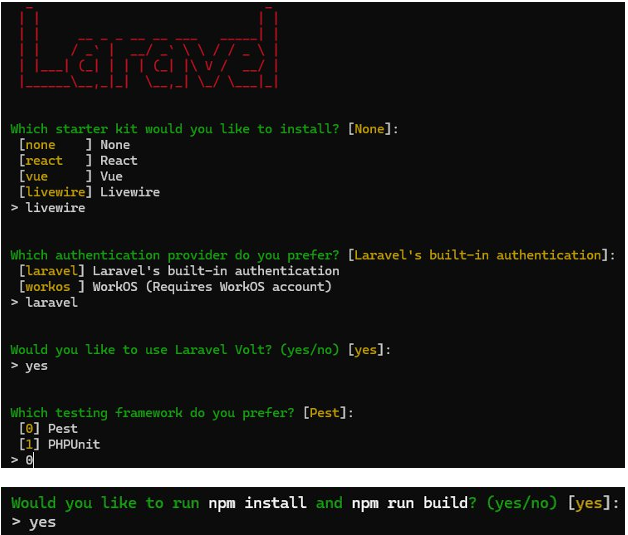
Welkom, jonge Laravel-Jedi. Vandaag beginnen we aan je training in de mysterieuze kunst van... **Laravel Cashier**. De Force is sterk in jou – maar zonder duidelijke stappen? Je eindigt in de duistere vallei van frustratie. Dus, trek je Jedi-jas aan, pak je lightsaber (of terminal) en laten we beginnen.

**✅ Stap 1: Een Nieuw Laravel-Project Aanmaken**

Gebruik deze krachtlijn in je terminal:

livewire new laravelApp

Je krijgt dan een **interactieve wizard** zoals op de afbeelding . Kies **exact** deze instellingen:



1. Which starter kit would you like to install?  
    👉 **Livewire**
2. Which authentication provider do you prefer?  
    👉 **Laravel's built-in authentication**
3. Would you like to use Laravel Volt?  
    👉 yes
4. Which testing framework do you prefer?  
    👉 0 (**Pest**)
5. Would you like to run npm install and npm run build?  
    👉 yes

⚠️ Deze keuzes zorgen ervoor dat je project **volledig klaar staat** met Livewire, Tailwind CSS, auth, Laravel Volt en tooling.

**✅ Stap 2: Open je project in PhpStorm**

Open je project in PhpStorm met deze stappen:

1. Start PhpStorm
2. Kies **"Open"**
3. Navigeer naar je map laravelApp
4. Klik op **Open**

💡 *Tip: stel in PhpStorm Laravel & Blade support in voor betere auto-completion en highlighting.*

### **⚡ Open een nieuw terminalvenster en voer uit:**

composer run dev

✅ **Stap 3: Installeer Laravel Cashier via Composer**

Voer dit bevel uit in je projectmap:

composer require laravel/cashier

🧠 *"Hmm, installeren via Composer je moet. Anders, met Cashier geen verbinding krijgen zul je..."*

✅ **Stap 4: Voeg Cashier-kolommen toe aan je users-tabel**

Cashier is veeleisend – hij wil meer data! Voeg deze kolommen toe met een nieuwe migratie:

php artisan vendor:publish --tag="cashier-migrations"

Voer daarna de migratie uit als een ware Jedi:

php artisan migrate:fresh --seed

✅ **Stap 5: Activeer de Krachten van Billable in je User Model**

Open app/Models/User.php, voeg bovenaan toe:

use Laravel\Cashier\Billable;

En dan, binnen je User class:

use Billable;

Voorbeeld:

class User extends Authenticatable  
{  
 use HasFactory, Notifiable, Billable;  
 // ...  
}

⚔️ *"Billable je bent nu... Gebruik deze kracht wijs."*

✅ **Stap 6: Configureer je Stripe-sleutels – toegang tot het universum!**

Voeg je test-sleutels toe aan .env:

STRIPE\_KEY=pk\_test\_...  
STRIPE\_SECRET=sk\_test\_...

Nog geen Stripe-account? 🧙‍ Registreer gratis: <https://dashboard.stripe.com/register>

Voeg dit toe aan config/services.php als het er nog niet staat:

'stripe' => [  
 'model' => App\Models\User::class,  
 'key' => env('STRIPE\_KEY'),  
 'secret' => env('STRIPE\_SECRET'),  
],

✅ **Stap 7: Maak producten en prijzen aan in Stripe – laat de credits rollen**

Stripe werkt met producten en prijzen. Ga naar je testomgeving: 👉 <https://dashboard.stripe.com/test/products>

Maak een product aan:

* Naam: Premium
* Prijs: €20.00
* Kopieer het prijs-ID (zoals price\_1Nhxxx...)

✅ **Stap 8: Test je Jedi-skills – maak een betaling**

Stripe probeert je gebruiker na betaling te redirecten naar een route met de naam home. Maar... die bestaat niet!  
deze home-route is **alleen bedoeld voor de test**. In een echt project kun je later een aangepaste success\_url opgeven (bijvoorbeeld een dashboard of een bedankpagina).

😱 **Oplossing**: Voeg toe in routes/web.php:

Route::get('/home', function () {  
 return 'Welkom thuis na betaling!';  
})->name('home');

En dan:

🔹 **Optie A: Gebruik abonnement (subscription)**

Route::get('/checkout', function () {  
 $user = \App\Models\User::first();  
 return $user->newSubscription('default', 'price\_1R7z98R2ThrdJtxhX3UENRCo')  
 ->checkout();  
});

👉 Route::get('/checkout', function () {

Deze regel definieert een GET-route met het pad /checkout. Wanneer iemand deze URL bezoekt in de browser, wordt de functie binnenin uitgevoerd.

👉 $user = \App\Models\User::first();

Hiermee wordt de eerste gebruiker uit de database opgehaald (vaak gebruikt als voorbeeld of voor testdoeleinden). In een echte applicatie zou je normaal de ingelogde gebruiker gebruiken, bijvoorbeeld auth()->user().

👉 return $user->newSubscription('default', 'price\_1R7z98R2ThrdJtxhX3UENRCo')

->checkout();

Deze regel start een nieuw abonnement voor de opgehaalde gebruiker:

* 'default' is de naam van het abonnement (handig als je meerdere abonnementen per gebruiker wil ondersteunen).
* 'price\_1R7z98R2ThrdJtxhX3UENRCo' is de prijs-ID van Stripe (dit correspondeert met een abonnementstype dat je in Stripe hebt aangemaakt).
* ->checkout() genereert een Stripe Checkout-link, zodat de gebruiker zijn betaling veilig kan afronden op de Stripe-betaalpagina.

🔹 **Optie B: Eenmalige betaling (one-time payment)**

Route::get('/checkout', function () {  
 $user = \App\Models\User::first();  
 return $user->checkout([  
 ['price' => 'price\_1Nhxxx...', 'quantity' => 1],  
 ]);  
});

🌐 **Testen in je browser:** Ga naar:  
 http://127.0.0.1:8000/checkout  
 en doorloop het betaalproces.

Gebruik deze testkaart:

* Nummer: 4242 4242 4242 4242
* Vervaldatum: iets in de toekomst
* CVC: 123
* Naam / E-mail: wat je wil

Na een succesvolle betaling word je doorgestuurd naar /home, waar je het testbericht ziet: **"Welkom thuis na betaling!"**

✅ **Stap 9: Activeer de Force – Webhooks!**

Stripe stuurt webhooks voor gebeurtenissen zoals:

* Betaling gelukt
* Abonnement gestopt
* Kaart geweigerd

Laravel Cashier luistert op:

/stripe/webhook

📦 Installeer de Stripe CLI: <https://stripe.com/docs/stripe-cli>

1. Download zip voor Windows
2. Pak uit
3. Verplaats stripe.exe naar C:\stripe-cli
4. Voeg toe aan je PATH
5. Open een terminal en test:

stripe version

Zie je stripe version 1.xx.x? Dan zit je goed.

🧭 **Hoe voeg je een map toe aan je Path in Windows?**

* Windows-toets + S → zoek "omgevingsvariabelen"
* Klik op "Systeemomgevingsvariabelen bewerken"
* Onder "Systeemvariabelen" → klik op Path → Bewerken
* Nieuw → voeg C:\stripe-cli toe → OK

✅ **Stap 10: Start de Stripe CLI en test je verbinding**

In je terminal:

stripe listen --forward-to <http://localhost:8000/stripe/webhook>

Je ziet iets als:

Ready! Your webhook signing secret is whsec\_...

👉 Plak die sleutel in .env:

STRIPE\_WEBHOOK\_SECRET=whsec\_...

✅ **Stap 11: Laravel voorbereiden op webhooks**

Cashier regelt veel voor jou, je hoeft alleen dit nog in .env te zetten:

STRIPE\_KEY=pk\_test\_...  
STRIPE\_SECRET=sk\_test\_...  
STRIPE\_WEBHOOK\_SECRET=whsec\_...

Laravel registreert automatisch de route /stripe/webhook. Je hoeft dus geen route toe te voegen. 🤯

🧪 **Test de webhook**

stripe trigger invoice.paid

Als het werkt zie je iets als:

[200] POST <http://localhost:8000/stripe/webhook> [evt\_1REDRAR2...]

🛑 **Mag je de terminal sluiten?** Niet als je nog Stripe gebruikt of aan het testen bent. Sluit je hem, dan werkt je webhook niet meer lokaal.

🌐 **In productie?** Dan voeg je je webhook toe via het Stripe dashboard met je echte domein. Bijvoorbeeld:

<https://jouwdomein.com/stripe/webhook>

🎉 Gefeliciteerd Padawan, Dag 1 is voltooid. De kracht groeit in jou... Neem nu een pauze, drink een space-milkshake, en laat me weten wanneer je klaar bent

**🔮 Dag 2 – Het Pad van de Plannen: Seeder Magie en Order Meesterschap**

(*"Een Jedi kent zijn abonnementen. Hij zaait de plannen in de database, en bewaart elke transactie in de kronieken van de Orders."*)

✅ **Stap 12: Producten in Seeder (Pricing plannen aanmaken in de database)**

Een Jedi betaalt niet met goudstukken. Nee, we spreken hier over abonnementen. Tijd om die op te slaan in de database.

📌 Voer uit in je command-line:

php artisan make:model Plan -m -s

Open daarna database/migrations/xxxx\_xx\_xx\_xxxxxx\_create\_plans\_table.php en vul het in met deze krachtige structuur:

Schema::create('plans', function (Blueprint $table) {  
 $table->id();  
 $table->string('name'); // Basic, Pro, etc.  
 $table->string('slug')->unique(); // 'basic', 'pro'  
 $table->string('stripe\_price\_id'); // De prijs-ID van Stripe  
 $table->decimal('price', 8, 2);  
 $table->text('features')->nullable(); // JSON of tekst  
 $table->timestamps();  
});

📌 Open daarna database/seeders/PlanSeeder.php en tover deze machtige code erin:

namespace Database\Seeders;  
  
use Illuminate\Database\Seeder;  
use App\Models\Plan;  
  
class PlanSeeder extends Seeder  
{  
 public function run(): void  
 {  
 Plan::create([  
 'name' => 'Basic',  
 'slug' => 'basic',  
 'stripe\_price\_id' => 'price\_1R7z7t...',  
 'price' => 5.00,  
 'features' => 'Access to basic features',  
 ]);  
  
 Plan::create([  
 'name' => 'Pro',  
 'slug' => 'pro',  
 'stripe\_price\_id' => 'price\_1R7z8Z...',  
 'price' => 10.00,  
 'features' => 'Everything in Basic + more',  
 ]);  
  
 Plan::create([  
 'name' => 'Premium',  
 'slug' => 'premium',  
 'stripe\_price\_id' => 'price\_1R7z98...',  
 'price' => 20.00,  
 'features' => 'All features unlocked',  
 ]);  
 }  
}

📌 Voeg de PlanSeeder toe aan DatabaseSeeder.php:

Open database/seeders/DatabaseSeeder.php en voeg toe:

$this->call([  
 PlanSeeder::class,  
]);

✅ **Stap 13: Tabel voor Orders aanmaken**

Wat is een Jedi zonder zijn archieven? Elke betaling, elke transactie, elk abonnement... we houden het bij als ware galactische geschiedschrijvers.

Voer deze heilige opdracht uit:

php artisan make:model Order -m

📁 Open de nieuwe migratie in database/migrations/ en vul het als volgt in:

Schema::create('orders', function (Blueprint $table) {  
 $table->id();  
 $table->foreignId('user\_id')->constrained()->onDelete('cascade');  
 $table->foreignId('plan\_id')->constrained()->onDelete('cascade');  
 $table->string('stripe\_transaction\_id')->unique();  
 $table->decimal('amount', 8, 2);  
 $table->enum('status', ['paid', 'unpaid'])->default('unpaid');  
 $table->timestamps();  
});

📌 En activeer de schepping van je database met:

php artisan migrate:fresh --seed

✅ **Stap 14: Modellen configureren met fillables en relaties**

Wees niet als Anakin die zijn training oversloeg. Als je je modellen niet verbindt, tast je in het duister.

📁 app/Models/Order.php

namespace App\Models;  
  
use Illuminate\Database\Eloquent\Model;  
  
class Order extends Model  
{  
 protected $fillable = [  
 'user\_id',  
 'plan\_id',  
 'stripe\_transaction\_id',  
 'amount',  
 'status',  
 ];  
  
 public function user()  
 {  
 return $this->belongsTo(User::class);  
 }  
  
 public function plan()  
 {  
 return $this->belongsTo(Plan::class);  
 }  
}

📁 app/Models/Plan.php

namespace App\Models;  
  
use Illuminate\Database\Eloquent\Model;  
  
class Plan extends Model  
{  
 protected $fillable = [  
 'name',  
 'slug',  
 'stripe\_price\_id',  
 'price',  
 'features',  
 ];  
  
 public function orders()  
 {  
 return $this->hasMany(Order::class);  
 }  
}

📁 app/Models/User.php Zet onderaan in je User-klasse:

public function orders()  
{  
 return $this->hasMany(Order::class);  
}

🎉 Dag 2 volbracht, jonge Jedi! Je hebt prijzen gecreëerd, je database voorbereid en je modellen op het pad van de Force gezet. De structuur staat. De plannen liggen op tafel. Morgen... gaan we live met Stripe Checkout en slaan we elke betaling op als een holocron in de Jedi-archieven.

Zeg het met me mee:

"Ik ben één met de Code, en de Code is met mij."

🧙‍♂️ **Dag 3 – De Kracht van Checkout & Abonnementen: Jedi Wordt een Betalingsmeester**  
 (*“Een plan gekozen jij hebt… betalen je zult!”* 🌌💸)

🎬 *Welkom bij de laatste dag van je Cashier-training, jonge Padawan. Vandaag gaan we van dromen naar daden – van plannen naar betalingen. Vandaag... worden we live.*

## **✅ Stap 15: Pricing-pagina tonen met alle plannen**

🧱 Vandaag bouwen we een toonzaal voor je abonnementen – de plaats waar gebruikers een plan kunnen bewonderen en met één klik besluiten: “Ja, dit is mijn lot!”

### **📌 1. Livewire-component aanmaken**

“Lichtzwaard bouwen je moet. Component creëren dus, hmm?”

php artisan livewire:make prices/ShowPrices

Dit maakt twee magische bestanden aan:

* app/Livewire/Prices/ShowPrices.php (je Jedi-controller)
* resources/views/livewire/prices/show-prices.blade.php (je view van de Force)

### **📌 2. Vul de component-klasse in**

In app/Livewire/Prices/ShowPrices.php:

namespace App\Livewire\Prices;  
use Livewire\Component;  
use App\Models\Plan;  
  
class ShowPrices extends Component  
{  
 public $plans;  
  
 public function mount()  
 {  
 $this->plans = Plan::all();  
 }  
  
 public function render()  
 {  
 return view('livewire.prices.show-prices ');  
 }  
}

✨ *De mount-methode is je Jedi Initiatie. Hier halen we alle plannen uit de database. Simpel, krachtig, en klaar voor actie.*

### **📌 3. De View invullen**

In resources/views/livewire/ show-prices.blade.php – gebruik een krachtig rooster met Tailwind magie:

<div class="grid grid-cols-1 md:grid-cols-3 gap-6 max-w-6xl mx-auto p-10">  
 @foreach ($plans as $plan)  
 <div class="bg-white shadow-md rounded-lg p-6 text-center">  
 <h2 class="text-xl font-bold mb-2">{{ $plan->name }}</h2>  
 <p class="text-gray-500 mb-4">{{ $plan->features }}</p>  
 <p class="text-2xl font-semibold mb-6">€{{ number\_format($plan->price, 2) }}</p>  
 <a href="{{ route('checkout', $plan->id) }}"  
 class="bg-blue-500 text-white px-4 py-2 rounded hover:bg-blue-600">  
 Abonneer nu  
 </a>  
 </div>  
 @endforeach  
</div>

📣 “Een abonnement kiezen, als een lichtzwaard – elegant en persoonlijk.”

### **📌 4. Route koppelen**

Voeg dit toe aan routes/web.php:

Route::middleware(['auth'])->group(function () {

Route::get('/pricing',\App\Livewire\Prices\ShowPrices::class)

->name('pricing');

});

**📌 5. Voeg "Abonnementen" toe aan de sidebar**

Open het bestand:  
 📁 resources/views/components/layouts/app/sidebar.blade.php

En voeg dit toe waar je de navigatie-items definieert:

<flux:navlist.item  
 icon="credit-card"  
 :href="route('pricing')"  
 :current="request()->routeIs('pricing')"  
 wire:navigate  
>  
 {{ \_\_('Pricing') }}  
</flux:navlist.item>

🧠 *“Een goede navigatie, een rustige geest weerspiegelt.”*

## **✅ Stap 16: Checkout via Stripe**

Het moment suprême: de gebruiker kiest een plan en wordt naar **Stripe Checkout** geleid om z’n galactische pas af te geven.

### **📌 1. Route: /checkout/{plan}**

use Illuminate\Support\Facades\Auth;  
use App\Models\Plan;  
  
Route::get('/checkout/{plan}', function (Plan $plan) {  
 $user = Auth::user();  
  
 return $user->newSubscription('default', $plan->stripe\_price\_id)  
 ->checkout([  
 'success\_url' => route('checkout.success', ['plan' => $plan->id]) . '?session\_id={CHECKOUT\_SESSION\_ID}',  
 'cancel\_url' => route('pricing'),  
 ]);  
})->name('checkout');

🧾 **Wat doet deze route?**

* Start de checkout bij Stripe
* Stuurt de gebruiker na betaling terug naar checkout.success
* Of terug naar pricing als hij annuleert

### **📌 2. Route: /checkout/success/{plan}**

use Stripe\Stripe;  
use Illuminate\Support\Facades\Auth;  
use App\Models\Order;  
  
Route::get('/checkout/success/{plan}', function (Plan $plan) {  
 Stripe::setApiKey(config('services.stripe.secret'));  
 $user = Auth::user();  
  
 $session = \Stripe\Checkout\Session::retrieve(request('session\_id'));  
 $stripe\_transaction\_id = $session->payment\_intent ?? $session->subscription ?? 'onbekend';  
  
 $order = Order::where('stripe\_transaction\_id', $stripe\_transaction\_id)->first();  
  
 if (! $order) {  
 Order::create([  
 'user\_id' => $user->id,  
 'plan\_id' => $plan->id,  
 'stripe\_transaction\_id' => $stripe\_transaction\_id,  
 'amount' => $plan->price,  
 'status' => 'paid',  
 ]);  
 }  
  
 return redirect()->route('dashboard')->with('success', 'Je bent nu geabonneerd op het ' . $plan->name . ' plan.');  
})->name('checkout.success');

🧠 “De gebruiker betaalt. Wij vangen het op. Wij registreren de order. En we verwelkomen hem op het dashboard met een glimlach.” 🌞

## **✅ Stap 17: Toon het abonnement van de gebruiker**

🕵️ Tijd om op het profiel van de gebruiker zijn abonnement te tonen – want elke Jedi verdient een badge van eer.

### **📌 1. Voeg relatie toe in User.php**

public function latestOrder()  
{  
 return $this->hasOne(\App\Models\Order::class)->latestOfMany();  
}

✨ Dit geeft je de nieuwste bestelling van de gebruiker. Handig als je slechts het laatste plan wil tonen.

### **📌 2. Toon het plan in het profiel** **( ressources/views/components/layouts/app/sidebar.blade.php)**

@if (auth()->user()->latestOrder)  
 <div class="mt-4 p-4 bg-white rounded shadow">  
 <p><strong>Plan:</strong> {{ auth()->user()->latestOrder->plan->name }}</p>  
 </div>  
@else  
 <p class="text-gray-500">Je hebt nog geen abonnement.</p>  
@endif

🧩 Vergeet niet: in Order.php moet dit bestaan:

public function plan()  
{  
 return $this->belongsTo(\App\Models\Plan::class);  
}

# **✅ Stap 18: Toon de Bestellingen van de Gebruiker ("Orders")**

Je gebruikers willen natuurlijk weten wat ze gekocht hebben. Tijd om een overzicht te bouwen van hun bestellingen via een krachtig Livewire-component.

## **🧩 1. Maak het Livewire-component ShowOrders**

Voer deze Artisan-opdracht uit in je terminal:

php artisan livewire:make orders/ShowOrders

Dit maakt:

* app/Livewire/Orders/ShowOrders.php
* resources/views/livewire/orders/show-orders.blade.php

## **✨ 2. Vul de Blade-view met een net overzicht**

Open resources/views/livewire/orders/show-orders.blade.php   
en plak de volgende code:

<div class="p-6">  
 <h2 class="text-2xl font-bold mb-4">Mijn bestellingen</h2>  
  
 @if ($orders->isEmpty())  
 <p>Je hebt nog geen bestellingen.</p>  
 @else  
 <table class="w-full border-collapse table-auto">  
 <thead class="bg-gray-100 dark:bg-zinc-700 text-left">  
 <tr>  
 <th class="p-2">Datum</th>  
 <th class="p-2">Plan</th>  
 <th class="p-2">Bedrag</th>  
 <th class="p-2">Transactie ID</th>  
 <th class="p-2">Status</th>  
 </tr>  
 </thead>  
 <tbody>  
 @foreach ($orders as $order)  
 <tr class="border-b">  
 <td class="p-2">{{ $order->created\_at->format('Y-m-d H:i') }}</td>  
 <td class="p-2">{{ $order->plan->name ?? '—' }}</td>  
 <td class="p-2">€{{ number\_format($order->amount, 2) }}</td>  
 <td class="p-2">{{ $order->stripe\_transaction\_id }}</td>  
 <td class="p-2">{{ ucfirst($order->status) }}</td>  
 </tr>  
 @endforeach  
 </tbody>  
 </table>  
 @endif  
</div>

## **🧠 3. Logica toevoegen aan het component**

Open app/Livewire/Orders/ShowOrders.php en voeg deze code in:

<?php  
namespace App\Livewire\Orders;  
use Livewire\Component;  
use Illuminate\Support\Facades\Auth;  
use App\Models\Order;  
class ShowOrders extends Component  
{  
 public $orders;  
 public function mount()  
 {  
 // Alleen bestellingen van de ingelogde gebruiker  
 $this->orders = Order::where('user\_id', Auth::id())->latest()->get();  
 }  
 public function render()  
 {  
 return view('livewire.orders.show-orders');  
 }  
}

## **🔐 4. Beveilig de route met een middleware**

Voeg deze route toe in routes/web.php:

Route::middleware(['auth', 'check.plan'])->group(function () {  
 Route::get('/orders', \App\Livewire\Orders\ShowOrders::class)->name('orders.index');  
});

## **🛡️ 5. Maak de middleware check.plan aan**

Voer deze Artisan-opdracht uit om een middleware te genereren:

php artisan make:middleware CheckPlan

Dan: open app/Http/Middleware/CheckPlan.php en voeg deze logica toe:

<?php  
namespace App\Http\Middleware;  
use Closure;  
use Illuminate\Http\Request;  
use Symfony\Component\HttpFoundation\Response;  
class CheckPlan  
{  
 public function handle(Request $request, Closure $next): Response  
 {  
 $user = $request->user();  
 // Als de gebruiker geen actief abonnement heeft, wordt hij doorgestuurd  
 if (! $user || ! $user->subscribed('default')) {  
 return redirect()->route('pricing')->with('error', 'Je hebt een actief abonnement nodig om deze pagina te bekijken.');  
 }  
 return $next($request);  
 }  
}

## **🛡** Voeg deze code toe in bootstrap/app.php.

->withMiddleware(function (Middleware $middleware) {  
 $middleware->alias([  
 'check.plan' => CheckPlan::class,  
 ]);  
})

## **🧭 6. Voeg een menu-item toe in de sidebar**

Open resources/views/components/layouts/app/sidebar.blade.php en voeg dit toe tussen je navigatie-items:

<flux:navlist.item  
 icon="document-text"  
 :href="route('orders.index')"  
 :current="request()->routeIs('orders.index')"  
 wire:navigate  
>  
 {{ \_\_('Orders') }}  
</flux:navlist.item>

## **🧪 7. Testen in je browser**

Start je Laravel-server indien nodig:

php artisan serve

Open dan in je browser:

<http://localhost:8000/orders>

✅ Als alles goed is, zie je een **tafel met alle bestellingen** van de ingelogde gebruiker.  
 Indien geen bestellingen? Dan krijg je het bericht: *“Je hebt nog geen bestellingen.”*

## **🧾 Stap 19: Imprimeer je bestelling – Voeg een downloadbare factuur toe**

Je gebruikers willen graag een factuur van hun bestelling downloaden. Met deze stap voeg je een downloadlink toe én genereer je een mooie PDF.

### **🧩 1. Voeg een kolom “Acties” toe aan de tabel**

Open je Blade-bestand resources/views/livewire/orders/show-orders.blade.php en **pas de <thead> aan** :

<th class="p-2">Acties</th>

Voeg vervolgens in je <tbody> bij elke bestelling de volgende cel toe :

<td class="p-2">  
 <a href="{{ route('orders.download', $order->id) }}"  
 class="text-blue-600 hover:text-blue-800"  
 title="Download factuur">  
 <svg xmlns="http://www.w3.org/2000/svg" class="h-5 w-5 inline-block" viewBox="0 0 20 20" fill="currentColor">  
 <path d="M2 6a2 2 0 012-2h5l2 2h5a2 2 0 012 2v6a2 2 0 01-2 2H4a2 2 0 01-2-2V6z"/>  
 <path stroke-linecap="round" stroke-linejoin="round" d="M14 5l7 7m0 0l-7 7m7-7H3"/>  
 </svg>  
 </a>  
</td>

### **🛣️ 2. Voeg de downloadroute toe**

Open routes/web.php en **voeg deze nieuwe route toe binnen je middlewaregroep**

use Illuminate\Support\Facades\Auth;  
  
Route::get('/orders/{order}/download', function (\App\Models\Order $order) {  
 $pdf = \Barryvdh\DomPDF\Facade\Pdf::loadView('livewire.orders.orderPDF', [  
 'order' => $order,  
 'user' => Auth::user(),  
 ]);  
  
 return $pdf->download("order-{$order->stripe\_transaction\_id}.pdf");  
})->name('orders.download');

💡 Zorg dat je bibliotheek **DomPDF** is geïnstalleerd via Composer :

composer require barryvdh/laravel-dompdf

### **🖨️ 3. Maak het PDF-sjabloon aan**

Creëer een bestand resources/views/livewire/orders/orderPDF.blade.php en plak deze nette layout:

<!DOCTYPE html>  
<html lang="en">  
<head>  
 <meta charset="UTF-8">  
 <title>Factuur #{{ $order->stripe\_transaction\_id }}</title>  
 <style>  
 body {  
 font-family: DejaVu Sans, sans-serif;  
 font-size: 14px;

color: #2d3748;  
 margin: 0;  
 padding: 40px;  
 }  
 .header {  
 display: flex;  
 justify-content: space-between;  
 align-items: center;  
 }  
 .app-name {  
 font-size: 24px;  
 font-weight: bold;  
 color: #4a5568;  
 }  
 .invoice-title {  
 font-size: 20px;  
 font-weight: bold;  
 text-align: right;  
 color: #2d3748;  
 }  
 .grid-container {  
 display: flex;  
 justify-content: space-between;  
 margin-top: 30px;  
 }  
 .section-title {  
 font-weight: bold;  
 margin-bottom: 8px;  
 color: #4a5568;  
 }  
 table {  
 width: 100%;  
 border-collapse: collapse;  
 margin-top: 40px;  
 }  
 th, td {  
 border: 1px solid #e2e8f0;  
 padding: 10px;  
 text-align: left;  
 }  
 th {  
 background-color: #f7fafc;  
 }  
 .footer {  
 margin-top: 40px;  
 font-size: 13px;  
 color: #718096;  
 }  
 </style>  
</head>  
<body>  
<div class="header">  
 <div class="app-name">{{ config('app.name', 'LaravelApp') }}</div>  
 <div class="invoice-title">Factuur</div>  
</div>  
  
<div class="grid-container">  
 <div>  
 <div class="section-title">Transactiegegevens</div>  
 <p><strong>Datum:</strong> {{ $order->created\_at->format('Y-m-d H:i') }}</p>  
 <p><strong>Transactie-ID:</strong> {{ $order->stripe\_transaction\_id }}</p>  
 <p><strong>Status:</strong> {{ ucfirst($order->status) }}</p>  
 </div>  
  
 <div>  
 <div class="section-title">Klantgegevens</div>  
 <p><strong>Naam:</strong> {{ $user->name }}</p>  
 <p><strong>E-mail:</strong> {{ $user->email }}</p>  
 </div>  
</div>  
  
<table>  
 <thead>  
 <tr>  
 <th>Beschrijving</th>  
 <th>Bedrag</th>  
 </tr>  
 </thead>  
 <tbody>  
 <tr>  
 <td>  
 Abonnement {{ $order->plan->name ?? 'Onbekend plan' }}  
 </td>  
 <td>€{{ number\_format($order->amount, 2) }}</td>  
 </tr>  
 </tbody>  
</table>  
  
<div class="footer">  
 <p>Bedankt voor je aankoop!</p>  
 <p>Als je vragen hebt, neem contact met ons op via {{ config('mail.from.address') ?? '[support@site.com](mailto:support@site.com)' }}</p>  
</div>  
</body>  
</html>

### **✅ Resultaat**

* Een nieuwe kolom “Acties” verschijnt in de tabel.
* Gebruikers kunnen nu op het downloadicoon klikken om hun factuur als **PDF-bestand** op te slaan.
* De factuur bevat duidelijke informatie over de gebruiker, bestelling en betaling.

## **🏁 Resultaat: De Keten is Voltooid**

🎉 De flow is compleet:

* Plannen? ✅
* Betalen? ✅
* Order bewaren? ✅
* Abonnement tonen in profiel? ✅

Je hebt het gedaan, Padawan. Je bent nu officieel... een **Cashier Jedi Master.**

En zo, jonge Jedi van Laravel, heb jij de weg van Cashier bewandeld.

✨ Je begon als een eenvoudige Padawan, die enkel composer require laravel/cashier kende...  
 💥 Maar nu? Nu ben je een meester van webhooks, checkout-sessies, en betaalde plannen.

Je hebt:

* ✨ Magische Stripe-sleutels getemd
* 🧾 Je gebruikers laten betalen zonder dat ze erbij weglopen
* 🛡️ Een solide Order-model gebouwd dat zelfs de Empire niet kan breken
* 👤 En je gebruikersprofiel getransformeerd in een heilige inschrijvingspagina

Maar onthoud dit, coder van het licht:

*“Met grote subscriptions komt grote verantwoordelijkheid.”* – Laravelius Maximus

