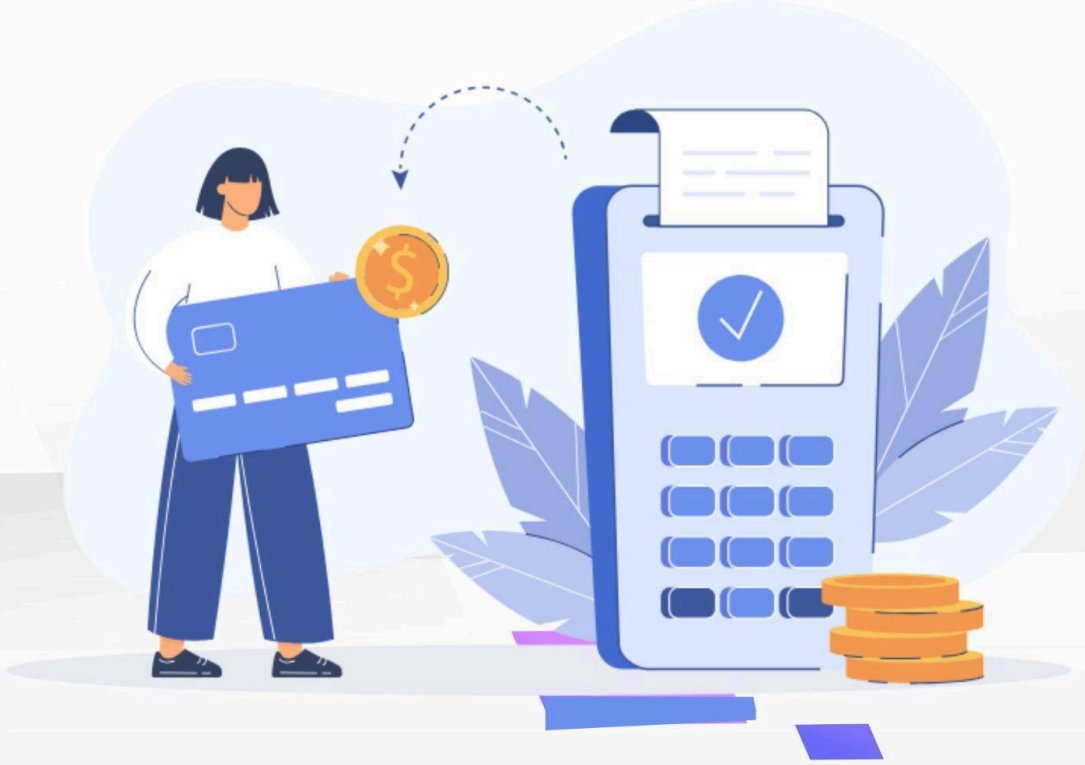
Stripe Integration in Angular

A Step-by-Step Guide to Creating Payment Functionality



Abdelfattah Ragab

Stripe Integration in Angular

A Step-by-Step Guide to Creating Payment Functionality

Abdelfattah Ragab

Introduction

Welcome to the book “Stripe Integration in Angular: A Step-by-Step Guide to Creating Payment Functionality”. In this book, I explain how to integrate Stripe into your Angular application.
Stripe is a leading payment processing platform that enables businesses to accept online payments.
By integrating payment processing into your application, you can create all kinds of e-commerce applications.
You will learn how to create the checkout session, how to use webhooks events and finally how to go live.
By the end of this book, you will be able to process payments in your Angular application and handle all kinds of scenarios.
Let us get started.

What is Stripe?

Stripe is a leading provider of payment services that enables businesses to efficiently accept and process online payments. It enables merchants to process credit and debit card transactions as well as other payment methods via a simple and secure platform. Stripe is particularly popular with start-ups and entrepreneurs as it offers a user-friendly API and robust features that enable quick integration into websites and applications

Create a Stripe account

In order to use Stripe for payment processing, it is necessary to create a Stripe account first. This account serves as the basis for accessing Stripe's services and functions.

Dashboard Management

The Stripe dashboard provides a user-friendly interface where you can manage your payments, view transaction history, process refunds and access analytics. This centralized management is essential for monitoring your company's financial activities.

API Keys

Once you have created an account, you will receive unique API keys that are required to integrate Stripe into your application. These keys authenticate your requests and ensure that your application can communicate securely with Stripe's servers.

Test Mode

Stripe offers a **test mode** that allows you to simulate transactions without processing real payments. This is important for developers to make sure their integration works correctly before it goes live.
You will receive unique API keys for testing that you can use during development.

Going Live

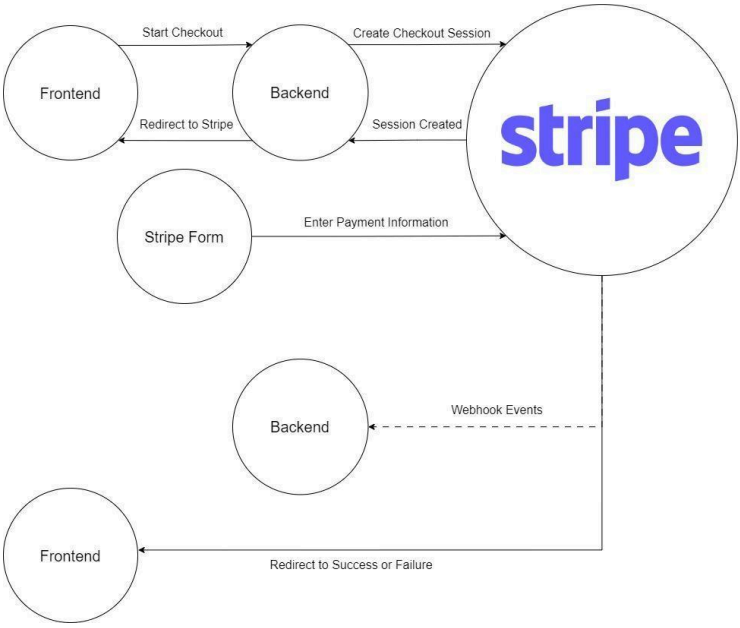
When you're ready for production, simply replace the test API keys with the live API keys. That's it, you don't need to change anything else. You don’t need to change anything in the code either. Just replace the test API keys with the live keys and you're done.
For each mode you get two keys, a public one that you can use in your frontend application and a secret one that you should only use in the backend application. Public keys begin with `pk_live_`, while secret keys begin with `sk_live_`.

You'll receive another key pair for testing.
The test keys begin with `pk_test_` and `sk_test_`.

The Backend

To effectively integrate Stripe into an application, both front-end and back-end components are required. Each plays a different role in ensuring secure and efficient payment processing.
The **frontend** is responsible for the user interface through which customers interact with the payment system. It provides users with a seamless and intuitive experience that allows them to enter their payment information and submit transactions easily.
However, the front end alone cannot process payments securely. It must communicate with the backend to complete the transaction.
The **backend** is responsible for the more sensitive aspects of payment processing. It communicates with Stripe's servers using secret API keys that should never be disclosed to the frontend. After the backend receives the response from Stripe, it sends back to the frontend to inform the user of the payment status.

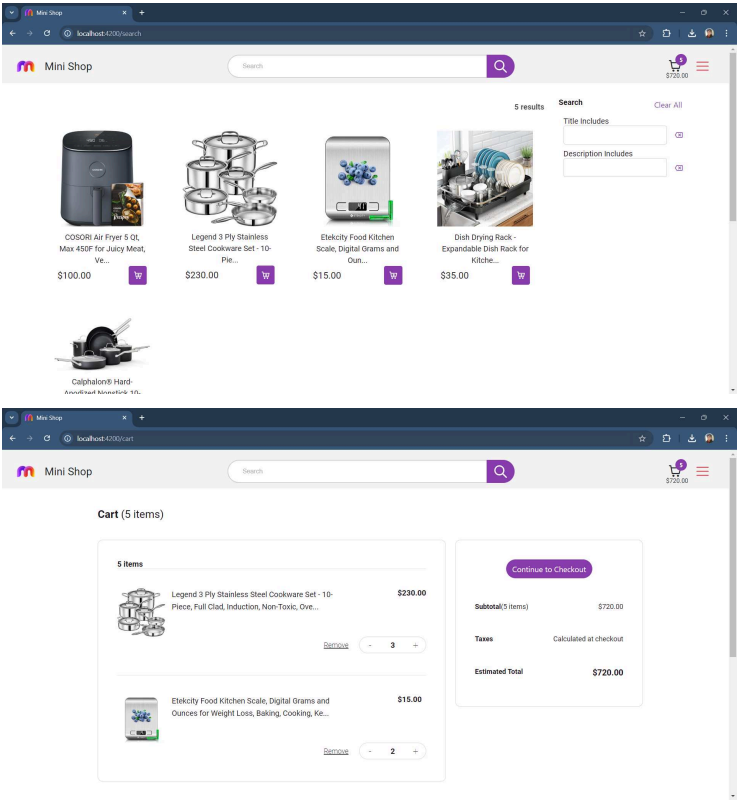
Stripe Payment Process



1. When the user clicks the checkout button, we send the items from the shopping cart to the backend and ask it to create a checkout session.
2. It sends it to the Stripe server and asks it to create a checkout session.
3. Stripe creates a session for us and returns its ID.
4. We send this ID to the frontend, which loads the Stripe form for the specified session ID.
5. The user enters their payment details and sends the form to the Stripe server.

6. Stripe informs the backend about webhook events and redirects the frontend accordingly for success or failure.

Practical Example



This Angular application consists of two pages, the search page with some products and the shopping cart page.

On the search page, the products are loaded from the backend.
When you click on the "Proceed to checkout" button, nothing happens.
But this will change soon. We will complete the checkout together.
There is another application for the backend that was created with NestJS.
It has two modules for products and orders. We don’t connect to the database, we just send an array of products to the frontend.
You can find the code example at https://books.abdelfattah-ragab.com
Download, unzip and run. Don’t forget to `npm install`.
You need to install nest cli on your computer.
To start the Nest application: `nest start`
To start the Angular application: `ng serve`
I will start by working on the backend application.

Debug Enabled

In the Nest application, you can start troubleshooting by clicking on the "Run" menu“ → “Start Debugging”.
This works because I added a **launch.json** file in the **.vscode** folder. It contains the configuration for NestJS debugging. Simply add this file to any of your NestJS applications to enable debugging.

rawBody

Add both `bodyParser` and `rawBody` to the **main.ts** file. `rawBody` is required for Stripe Webhooks. Activate cors as well. It should read as follows:

```
import { NestFactory } from
'@nestjs/core';
import { AppModule } from
'./app.module';

async function bootstrap() {
  const app = await
NestFactory.create(AppModule, {
    bodyParser: true,
    rawBody: true,
  });
  app.enableCors();
  await app.listen(3000);
}
bootstrap();
```

Install Dependencies

We need to install stripe and dotenv.

```
npm i stripe dotenv
```

Env

Add the stripe secret key to the **.env** file