Setting Up Stripe with ngrok for Webhook Integration

July 3, 2025

1 Introduction

This document provides a comprehensive guide for setting up Stripe with ngrok in a new Node.js project to handle webhook events, such as payment confirmations. It includes best practices, step-by-step instructions, necessary links, and the configuration required for the `.env` file. The goal is to ensure a smooth integration and avoid common issues encountered during setup, such as webhook delivery failures or configuration mismatches.

2 Prerequisites

Before starting, ensure you have the following:

- A Node.js project with Express and Sequelize (or a similar ORM).
- A Stripe account (create one at `https://dashboard.stripe.com/register`).
- ngrok installed (download from `https://ngrok.com/download`).
- PostgreSQL or another database configured for your project.
- Basic familiarity with Node.js, Express, and REST APIs.

3 Important Considerations

To avoid common issues when integrating Stripe with ngrok, keep the following in mind:

- **ngrok URL Stability**: ngrok generates a new URL each time you start a session in the free plan. Ensure the URL remains active during testing, and update it in the Stripe Dashboard if it changes.
- **Webhook Endpoint**: Stripe webhooks must be sent to a specific endpoint (e.g., `/webhook`). Ensure the endpoint is correctly defined in your server and accessible via the ngrok URL.

- **Webhook Secret**: The Stripe webhook secret must match between your `.env` file and the Stripe Dashboard to verify webhook signatures.
- **Transaction Management**: When saving data (e.g., `stripeSessionId`) in the database, ensure updates are performed within a transaction to avoid data loss due to race conditions or rollbacks.
- **Logging**: Add sufficient logging in your webhook handler to debug issues like missing bookings or signature verification failures.
- **Avoid Redundant Updates**: Do not update the same database field (e.g., `stripeSessionId`) in multiple places to prevent overwriting or nullifying values.
- **Testing Locally**: Use the Stripe CLI (`https://stripe.com/docs/stripe-cli`) to test webhooks locally before deploying to production.
- **Production Considerations**: ngrok is suitable for development and testing but is not stable for production. Consider using a fixed domain (e.g., via Heroku, AWS, or another hosting service) for production environments.

4 Step-by-Step Setup Instructions

Follow these steps to set up Stripe and ngrok in your Node.js project:

4.1 Step 1: Install and Configure ngrok

1. **Download and Install ngrok**: Download ngrok from `https://ngrok.com/download` and follow the installation instructions for your operating system.
2. **Authenticate ngrok**: Run the following command to authenticate your ngrok account (required for the free plan):

   ```
   ngrok authtoken <your-auth-token>
   ```

   Replace `<your-auth-token>` with the token from your ngrok dashboard (`https://dashboard.ngrok.com`).
3. **Start ngrok**: Run ngrok to forward your local server (e.g., running on port 3000) to a public URL:

   ```
   ngrok http 3000
   ```

   This will generate a public URL like `https://<random-id>.ngrok-free.app`. Copy this URL.
4. **Verify ngrok**: Open the ngrok dashboard at `http://127.0.0.1:4040/inspect/http` to confirm the URL is active and receiving requests.

4.2 Step 2: Set Up Stripe

1. **Create a Stripe Account**: Sign up at https://dashboard.stripe.com/register if you don't have an account.
2. **Get API Keys**: In the Stripe Dashboard, go to Developers > API Keys (https://dashboard.stripe.com/test/apikeys) and copy the Secret Key (starts with sk_test_...).
3. **Configure Webhook Endpoint**:
   - Go to Developers > Webhooks (https://dashboard.stripe.com/test/webhooks).
   - Click Add endpoint.
   - Set the Endpoint URL to https://<your-ngrok-url>/webhook (e.g., https://37e4-41-46-44-230.ngrok-free.app/webhook).
   - Select the events to listen to, such as checkout.session.completed and charge.succeeded.
   - Save the endpoint and copy the Signing Secret (starts with whsec_...).
4. **Test Webhook Locally**: Install the Stripe CLI (https://stripe.com/docs/stripe-cli) and run:

   ```
   stripe listen --forward-to https://<your-ngrok-url>/webhook
   ```

   This will provide a new Signing Secret for local testing. Use this secret in your .env file during development.

4.3 Step 3: Configure the .env File

Add the following environment variables to your .env file:

```
# Stripe Configuration
STRIPE_SECRET_KEY=sk_test_51... (your Stripe Secret Key)
STRIPE_WEBHOOK_SECRET=whsec_... (your Webhook Signing Secret from Stripe

# Server Configuration
PORT=3000
NODE_ENV=development
BASE_URL=https://<your-ngrok-url> (e.g., https://37e4-41-46-44-230.ngrok-
```

- Replace sk_test_... with your Stripe Secret Key.
- Replace whsec_... with the Webhook Signing Secret.
- Update BASE_URL whenever the ngrok URL changes.

4.4 Step 4: Set Up the Express Server

1. **Install Dependencies**: Ensure you have the required Node.js packages:

```
npm install express stripe dotenv sequelize express-async-han
```

2. **Configure Webhook Endpoint**: In your `server.js`, define the webhook endpoint before other middlewares:

```
const express = require("express");
const webhookController = require("./controllers/webhookController");
const app = express();

// Webhook endpoint (must use raw body parser for Stripe)
app.post("/webhook", express.raw({ type: "application/json" }),
    webhookController.stripeWebhookHandler);

// Other middlewares
app.use(express.json({ limit: "20kb" }));
```

3. **Start the Server**: Ensure the server runs on the port specified in `.env` (e.g., 3000).

4.5 Step 5: Implement the Webhook Handler

Create a `webhookController.js` file to handle Stripe webhook events:

```
// webhookController.js
const stripe = require("stripe")(process.env.STRIPE_SECRET_KEY);
const expressAsyncHandler = require("express-async-handler");
const { Booking } = require("../models");

exports.stripeWebhookHandler = expressAsyncHandler(async (req, res) => {
    const sig = req.headers["stripe-signature"];
    try {
        const event = stripe.webhooks.constructEvent(
            req.body,
            sig,
            process.env.STRIPE_WEBHOOK_SECRET
        );
        if (event.type === "checkout.session.completed" ||
            event.type === "charge.succeeded") {
            const session = event.data.object;
            const stripeSessionId = session.id;
            console.log("Webhook received:", event.type);
            console.log("Stripe Session ID:", stripeSessionId);

            const booking = await Booking.findOne({ where: { stripeSessio
            if (!booking) {
                console.log("No booking found for stripeSessionId:", stri
                return res.status(404).send("Booking not found");
            }
```

```
            await booking.update({
                paymentStatus: "paid",
                status: "confirmed",
            });
            console.log(`Booking ${booking.id} marked as paid`);
        }
        res.status(200).json({ received: true });
    } catch (error) {
        console.error("Error in webhook:", error.message);
        return res.status(500).send("Server error");
    }
});
```

- Ensure the webhook handler logs all events for debugging.
- Verify the `stripeSessionId` matches the one stored in your database.

4.6 Step 6: Create a Stripe Checkout Session

In your booking controller (e.g., `bookingController.js`), create a Stripe Checkout Session and save the `stripeSessionId` in the database within a transaction:

```
// bookingController.js
const { createStripeSession } = require("../services/paymentService");
const { Booking, sequelize } = require("../models");

const createBooking = expressAsyncHandler(async (req, res, next) => {
    const { roomId, checkInDate, checkOutDate, paymentMethod } = req.body
    const transaction = await sequelize.transaction();
    try {
        const booking = await Booking.create(
            { /* booking details */ },
            { transaction }
        );
        let paymentUrl = null;
        if (paymentMethod === "card") {
            const session = await createStripeSession(booking, room, req)
            paymentUrl = session.url;
            await booking.update({ stripeSessionId: session.id }, { trans
            console.log("Updated booking with stripeSessionId:", session.
        }
        await transaction.commit();
        res.status(201).json({ status: "success", data: { booking }, paym
    } catch (error) {
        await transaction.rollback();
        next(error);
    }
});
```

- Save the `stripeSessionId` within the transaction to ensure data consistency.
- Avoid updating the `stripeSessionId` outside the transaction to prevent overwriting.

4.7 Step 7: Test the Integration

1. **Start the Server and ngrok**:

```
node server.js
ngrok http 3000
```

2. **Update Stripe Webhook**: Ensure the Stripe Dashboard webhook URL matches the current ngrok URL with `/webhook`.
3. **Create a Test Booking**: Send a POST request to create a booking with `paymentMethod: "card"`.
4. **Complete Payment**: Use the Stripe Checkout URL returned in the response to complete the payment.
5. **Monitor Logs**: Check the server logs and Stripe CLI for webhook events. Verify that the booking is updated to `paymentStatus: "paid"` and `status: "confirmed"`.
6. **Resend Webhook (if needed)**: If the webhook fails, resend it from the Stripe Dashboard (`https://dashboard.stripe.com/test/events`).

5 Links to Use

- Stripe Dashboard: `https://dashboard.stripe.com`
- Stripe API Keys: `https://dashboard.stripe.com/test/apikeys`
- Stripe Webhooks: `https://dashboard.stripe.com/test/webhooks`
- Stripe CLI: `https://stripe.com/docs/stripe-cli`
- ngrok Download: `https://ngrok.com/download`
- ngrok Dashboard: `http://127.0.0.1:4040/inspect/http`

6 Conclusion

By following these steps and best practices, you can successfully integrate Stripe with ngrok in a Node.js project for handling webhook events. Always ensure the ngrok URL and webhook secret are correctly configured, use transactions for database updates, and add sufficient logging to debug issues. For production, consider replacing ngrok with a stable domain to ensure reliable webhook delivery.