Great! Since you already have **FoodScan** (from CodeCanyon) as your base, integrating **Stripe Connect Express** involves adding a few essential components on top of the existing self-ordering flow.

Below is a clear roadmap for the developer.

Stripe Connect Express Integration Checklist

1. Stripe Platform Setup (One-Time Configuration)

- **Create a Stripe account** (already done).
- Enable **Stripe Connect** in dashboard.
- Generate your **client ID** and **API keys**.

---

2. Restaurant Onboarding (OAuth Flow)

To collect payouts on behalf of restaurants:

- Implement **Express Account onboarding** using Stripe-hosted flow:
  - Backend endpoint: /create-stripe-account-link
  - Frontend: Button in admin panel: “Connect Stripe”
  - This will redirect the restaurant to Stripe’s Express onboarding.
- Save the resulting account_id (acct_XXXX) in your database for that restaurant.

Code Reference:

```
stripe.AccountLink.create(
  account=restaurant.stripe_account_id,
  refresh_url='https://yourapp.com/reauth',
  return_url='https://yourapp.com/connected',
  type='account_onboarding'
)
```

---

3. Customer Payment Flow

Update the existing **checkout logic** in FoodScan to:

- Create a **PaymentIntent** on your platform account.

- Set transfer_data[destination] = restaurant.stripe_account_id
- Include your **platform fee** using application_fee_amount.

Example:

```
stripe.PaymentIntent.create(
  amount=10500,  # AED 105.00
  currency='aed',
  payment_method_types=['card'],
  application_fee_amount=500,  # AED 5.00 platform fee
  transfer_data={
    'destination': restaurant.stripe_account_id,
  },
)
```

---

4. Delay Payouts (T+3)

Stripe Express allows you to configure a **custom payout schedule** for each connected account.

- Set the **payout schedule to "manual"** via API or Stripe Dashboard (once).
- Use Stripe CLI or dashboard to configure the T+3 delay globally or via onboarding settings.
- Stripe will **automatically trigger payouts T+3** without your manual input.

Your developer doesn't need to manually delay anything — this is handled by Stripe based on account settings.

---

5. Refunds (Optional, But Recommended)

Allow restaurants or platform to initiate refunds:

- You'll use the stored payment_intent_id to refund via API.
- Set rules like: allow refund within 6 hours or same-day only.

Refund Example:

```
stripe.Refund.create(
  payment_intent='pi_XXXXXXX',
  amount=10500  # full refund in fils
)
```

---

6. Webhooks (To Track Payments & Payouts)

Implement webhooks to track:

- payment_intent.succeeded
- charge.refunded
- account.updated
- payout.paid

Stripe will notify your app when payments complete, refunds are made, and payouts are sent.

---

7. (Optional) UI Updates

- Show Stripe connection status in restaurant dashboard.
- Show past earnings, payout history, or refund options.
- Optionally allow customer to pay a **“service fee”** via frontend at checkout.

Summary of What Developer Needs to Build

Task	**Details**
Stripe Account Setup	Use your platform account with Connect enabled
Onboarding Flow	Implement Express onboarding with account_link
PaymentIntent	Use application_fee_amount and transfer_data[destination]
Payout Logic	Configure Express accounts to delay payouts (T+3)
Refunds	Backend endpoint to create refunds
Webhooks	To track payment, refund, and payout events
UI Adjustments	Connect button, earnings view, refund button

Here’s a complete **Stripe Connect Express integration sample backend**, using **Node.js + Express** (as it's common and fast to set up). It includes:

1. **Create account onboarding link**
2. **Create a PaymentIntent with platform fee + destination**
3. **Trigger a refund**
4. **Set payout delay**
5. **Webhook listener**

---

1. Stripe Setup (Install)

```
npm install express stripe body-parser dotenv
```

Create a .env file:

```
STRIPE_SECRET_KEY=sk_test_...
STRIPE_CLIENT_ID=ca_...
```

---

2. app.js or index.js

```
require('dotenv').config();
const express = require('express');
const stripe = require('stripe')(process.env.STRIPE_SECRET_KEY);
const bodyParser = require('body-parser');

const app = express();
app.use(bodyParser.json());

/**
 * STEP 1: Create Stripe Express Account Link
 */
app.post('/create-account-link', async (req, res) => {
  const account = await stripe.accounts.create({
   type: 'express',
   capabilities: {
     card_payments: { requested: true },
     transfers: { requested: true }
   },
   business_type: 'restaurant',
  });

  const accountLink = await stripe.accountLinks.create({
   account: account.id,
   refresh_url: 'https://yourapp.com/reauth',
   return_url: 'https://yourapp.com/connected',
   type: 'account_onboarding'
  });

  // Save account.id to your DB with restaurant
  res.json({ url: accountLink.url });
});

/**
 * STEP 2: Create PaymentIntent with platform fee and transfer to restaurant
 */
app.post('/create-payment-intent', async (req, res) => {
  const { amount, restaurantStripeAccountId } = req.body;

  const paymentIntent = await stripe.paymentIntents.create({
   amount: amount, // in fils or cents (e.g., 10500 for AED 105)
   currency: 'aed',
    payment_method_types: ['card'],
    application_fee_amount: Math.floor(amount * 0.05), // 5% platform fee
    transfer_data: {
      destination: restaurantStripeAccountId,
    }
  });

  res.json({ clientSecret: paymentIntent.client_secret });
});

/**
 * STEP 3: Create a Refund
 */
app.post('/refund', async (req, res) => {
  const { paymentIntentId, amount } = req.body;

  const refund = await stripe.refunds.create({
    payment_intent: paymentIntentId,
    amount: amount, // optional, for partial refund
  });

  res.json({ refund });
});

/**
 * STEP 4: Webhooks (optional but strongly recommended)
 */
```

```
app.post('/webhook', bodyParser.raw({ type: 'application/json' }), (request, response) =>
{
  const sig = request.headers['stripe-signature'];
  let event;

  try {
    event = stripe.webhooks.constructEvent(request.body, sig,
process.env.STRIPE_WEBHOOK_SECRET);
  } catch (err) {
    return response.status(400).send(`Webhook Error: ${err.message}`);
  }

  switch (event.type) {
    case 'payment_intent.succeeded':
      console.log('Payment succeeded:', event.data.object.id);
      break;
    case 'charge.refunded':
      console.log('Charge refunded:', event.data.object.id);
      break;
    case 'payout.paid':
      console.log('Payout completed.');
      break;
    default:
      console.log(`Unhandled event type ${event.type}`);
  }

  response.json({ received: true });
});
```

```
app.listen(4242, () => console.log('Server running on port 4242'));
```

---

3. Configure Payout Delay (Once per connected account)

This must be done manually via Stripe Dashboard or API after onboarding.

Or use API:

```
await stripe.accounts.update('acct_XXXX', {
 settings: {
  payouts: {
   schedule: {
    interval: 'daily',
    delay_days: 3,
   }
  }
 }
});
```

---

4. Frontend Summary (Minimal Tasks)

- Call /create-account-link to get onboarding URL.
- Call /create-payment-intent and pass clientSecret to Stripe.js for checkout.
- Display success/failure.
- For refunds, trigger the /refund endpoint.

Perfect. Here's a **step-by-step developer guide** tailored specifically to **integrating Stripe Connect Express** into the **FoodScan CodeCanyon source code**, broken down by:

PART 1: What to Do in the FoodScan Source Code

1. Add a "Connect Stripe" Option in Restaurant Dashboard

- In the **admin panel or restaurant login** (likely under profile/settings):
    - Add a **"Connect Stripe"** button.
    - When clicked, it should call your backend endpoint (/create-account-link) and redirect to the Stripe Express onboarding link.

2. Store Stripe Account ID

- After successful onboarding, Stripe redirects to your app with account_id.
- Save stripe_account_id in the restaurant's database profile (you might need to add a column if not present).

3. Modify Checkout Flow

- When a user places an order:
    - On backend, call /create-payment-intent and pass restaurant.stripe_account_id and order amount.
    - Use the returned client_secret with Stripe.js or Mobile SDK to complete payment.

4. Add Refund Handling

- Add a backend route /refund triggered via admin panel.
- Optionally add a UI in the admin interface to trigger refund per order.

PART 2: What to Host on Your Server

These endpoints can be hosted on a **small VPS (like 1vCPU, 1GB RAM)** running Node.js, Laravel, Python, etc.

Endpoint	**Purpose**
/create-account-link	Creates Stripe Express onboarding link

Endpoint	Purpose
/create-payment-intent	Creates the customer payment intent
/refund	Triggers full/partial refunds
/webhook	Listens to events like payments, refunds, payouts

Add SSL via Let's Encrypt, and secure all endpoints with auth tokens or IP whitelisting if needed.

PART 3: Stripe Dashboard Setup

- Go to Stripe Dashboard:
  - **Enable Connect** from the settings.
  - Copy your **API keys** and **Client ID**.
  - Set up **webhook URL** in Stripe dashboard (https://yourdomain.com/webhook).
  - Configure default **payout delay (T+3)** via Connect settings or per account.

PART 4: Additional Integrations / Locations

Integration	Where to Add in FoodScan
Stripe.js script	Frontend (checkout or payment page)
Backend integration	New Node.js/Laravel/PHP API layer
Webhook handler	On the hosted backend server
DB field stripe_account_id	Add to restaurants table/model

PART 5: Best Practices for Efficiency & Scalability

- **Keep FoodScan core logic untouched** as much as possible. Build integration in a modular way (e.g., Stripe service file or helper).
- **Offload webhook processing** to background worker if volume increases.
- **Add caching** for Stripe account check to avoid repeat API calls.
- **Avoid saving credit card details**; use only Stripe's PCI-compliant SDK.

- **Minimal infrastructure required**:
  - 1 small backend server
  - No DB scaling needed
  - Stripe handles most complexity: payment processing, refunds, payout delays

---

Optional Enhancements (Low Lift)

- Auto-hide “Pay Now” button if Stripe not connected.
- Show “Last payout” and “Next payout” status via Stripe API.
- Let restaurant add a refund reason (saved locally).
- Add logs for all webhook events in DB for transparency.

Here's a **developer handover document** followed by an **architecture/flow diagram description** in text.

Developer Handover Document for Stripe Connect Express Integration with FoodScan

Overview

Integrate Stripe Connect Express into the FoodScan CodeCanyon script to enable restaurant-specific payouts, T+3 delays, and self-managed refunds—while maintaining lightweight, secure infrastructure.

Backend Requirements

Tech Stack Recommendation: Node.js (or PHP/Laravel if your stack matches FoodScan)

Host the following endpoints on a small VPS:

Endpoint	**Description**
/create-account-link	Starts onboarding of a restaurant
/create-payment-intent	Creates customer payment with platform fee and transfer
/refund	Issues full/partial refund
/webhook	Receives Stripe events

.env Example:

```
STRIPE_SECRET_KEY=sk_live_...

STRIPE_CLIENT_ID=ca_...

STRIPE_WEBHOOK_SECRET=whsec_...
```

FoodScan Code Modifications

1. Add Stripe Connect Button

- Add in restaurant/settings.php or similar.
- When clicked: Call /create-account-link → redirect to Stripe onboarding.

2. Store Stripe Account ID

- On return, save account_id to restaurants table (stripe_account_id column).

3. Modify Checkout Process

- In the order placement logic:
  - Call /create-payment-intent
  - Inject returned client_secret into Stripe.js (frontend)
  - Confirm card payment from client-side

4. Add Refund Option

- In admin panel (order details page), add “Refund” button.
- Connect to backend /refund endpoint.

Stripe Dashboard Configuration

- Enable **Connect** > **Express**.
- Set **Payout Delay to 3 Days (T+3)**.
- Setup **webhook URL** in Stripe Dashboard.
- Optionally enable **manual refunds only by platform**.

Webhook Setup

- Secure /webhook route with Stripe’s webhook secret.
- Listen for:
  - payment_intent.succeeded
  - charge.refunded
  - payout.paid
  - account.updated

Data Model Additions

In your restaurants table:

ALTER TABLE restaurants ADD stripe_account_id VARCHAR(255);

Architecture & Payment Flow Diagram (Text Description)

Actors:

- Customer
- Restaurant Admin
- Platform Server
- Stripe Connect

1. Onboarding Flow

Restaurant Admin clicks "Connect Stripe"

→ Platform backend → Stripe `/account_links` API

→ Redirect to Stripe Express onboarding page

→ Stripe redirects back to your app

→ Platform saves `stripe_account_id`

---

2. Customer Payment Flow

Customer places order on FoodScan

→ Platform backend calls Stripe `PaymentIntent.create`

with:

- full amount

- application_fee_amount (your 5%)

- transfer_data.destination = restaurant_account_id

→ Returns `client_secret`

→ Frontend uses Stripe.js to confirm payment

→ Payment split automatically (Stripe handles)

→ Payout sent to restaurant T+3

---

3. Refund Flow

Admin panel triggers `/refund` endpoint

→ Platform calls Stripe `Refund.create` using `payment_intent_id`

→ Stripe returns refund status

→ Customer notified automatically

→ Funds reversed from restaurant (automatically)

Infrastructure

- **Frontend**: Existing FoodScan UI + minimal Stripe.js
- **Backend**: 1 small VPS with Node.js or PHP for APIs
- **Database**: FoodScan's existing MySQL/PostgreSQL
- **Storage/Backups**: No new storage/load balancing required
- **Stripe**: Handles heavy lifting (payouts, compliance, refund, security)

Cloudways Setup (Simplest & Easiest)

Step-by-Step Guide

1. **Sign Up on Cloudways**
   - Choose **DigitalOcean** as provider.
   - Select **1GB RAM, 1 vCPU** ($12/mo plan).
   - Select **PHP Stack** or **Laravel App**.
2. **Launch Server**
   - Name your server & app (e.g., foodscan, stripe-api).
   - Deploy app in ~10 minutes.
3. **Deploy FoodScan Script**
   - Use SFTP or Git to upload the FoodScan PHP code to /public_html.
   - Configure .env file, DB connections, etc.
4. **Install Node.js for Stripe Backend**
   - SSH into server → install Node.js:

bash

CopyEdit

```
curl -fsSL https://deb.nodesource.com/setup_18.x | sudo -E bash -
sudo apt install -y nodejs
```

   - Create /stripe-api folder and host Express server (or Laravel API if using PHP).
5. **Enable SSL (Free Let's Encrypt)**
   - Go to Cloudways → Application → SSL Management
   - Enter your domain/subdomain, e.g., api.yourdomain.com
   - Apply SSL in 1 click.
6. **Enable Daily Backups**
   - Go to Server → Backups → Enable (costs ~$2/month)
7. **Configure Stripe Webhooks**
   - Use https://api.yourdomain.com/webhook as endpoint in Stripe Dashboard

Here's a **simple and ready-to-use provisioning script** for your developer to quickly set up a VPS (like Cloudways, Hetzner, or any Ubuntu-based server) to host:

- Your **FoodScan PHP web app**
- A **Node.js backend** for Stripe Connect
- With **SSL**, **MySQL**, **Nginx**, and other basics

Provisioning Script: Ubuntu 22.04+ VPS

Filename: setup_server.sh
Usage: SSH into your server and run:

```
curl -sSL https://yourdomain.com/setup_server.sh | sudo bash
```

Script Begins

```
#!/bin/bash


# Update server

apt update && apt upgrade -y


# Install core packages

apt install -y nginx mysql-server php php-cli php-mysql php-curl php-zip php-mbstring php-xml php-common php-fpm unzip git curl ufw


# Install Node.js (v18 LTS)

curl -fsSL https://deb.nodesource.com/setup_18.x | bash -

apt install -y nodejs


# Install PM2 (Node process manager)

npm install -g pm2


# Enable Firewall & Open Ports

ufw allow 'Nginx Full'
```

ufw allow OpenSSH

ufw enable

# Configure MySQL

mysql_secure_installation

# Restart services

systemctl restart nginx

systemctl restart php8.1-fpm

systemctl enable php8.1-fpm mysql nginx

echo "Provisioning Complete. You can now deploy FoodScan and Node backend."

What Developer Should Do Next:

1. Upload FoodScan App

- Upload it to /var/www/foodscan
- Configure nginx server block to serve the app
- Set correct permissions:
- chown -R www-data:www-data /var/www/foodscan
- chmod -R 755 /var/www/foodscan

2. Setup Stripe Node.js Backend

- Create folder /var/www/stripe-api
- Add your index.js Express server (for /create-payment-intent, /webhook, etc.)
- Start with PM2:
- pm2 start index.js --name stripe-api
- pm2 save
- pm2 startup

3. Configure Nginx Reverse Proxy

Sample /etc/nginx/sites-available/api.yourdomain.com:

```
server {
  listen 80;
  server_name api.yourdomain.com;

  location / {
    proxy_pass http://localhost:3000;
    proxy_http_version 1.1;
    proxy_set_header Upgrade $http_upgrade;
    proxy_set_header Connection 'upgrade';
    proxy_set_header Host $host;
    proxy_cache_bypass $http_upgrade;
  }
}
```

Enable with:

```
ln -s /etc/nginx/sites-available/api.yourdomain.com /etc/nginx/sites-enabled/
nginx -t && systemctl reload nginx
```

4. Install SSL for Free

Use Certbot:

```
apt install certbot python3-certbot-nginx -y
certbot --nginx -d yourdomain.com -d api.yourdomain.com
```

Your Project Structure After Setup

```
/var/www/
├── foodscan/       → PHP frontend (CodeCanyon)
└── stripe-api/     → Node.js backend for Stripe
```

If your developer is more comfortable doing things manually:

That’s completely fine — they just need to:

1. **Install packages**: PHP, MySQL, Node.js, NGINX, etc.
2. **Secure server**: UFW firewall, SSL setup
3. **Deploy apps**: Upload FoodScan + Node.js backend
4. **Configure NGINX**: For both frontend and API
5. **Set up services**: Use pm2 for running Node backend
6. **Enable backups**: If your host doesn’t provide by default

Bottom line:

The script is just automation. If your developer understands these steps, **manual setup works just as well**.

Here's your **complete manual step-by-step guide** to set up a VPS server for hosting:

1. Your **FoodScan web app (PHP-based)**
2. Your **Stripe Connect Express API (Node.js backend)**
3. With **MySQL**, **NGINX**, **SSL**, **PM2**, and firewall for security

---

Manual Setup Guide for Ubuntu 22.04 VPS

Step 1: Connect to Your Server

SSH into your server (Hetzner, Cloudways custom VPS, etc.):

```
ssh root@your-server-ip
```

Step 2: Update System

```
apt update && apt upgrade -y
```

Step 3: Install Server Stack (PHP, MySQL, NGINX)

```
apt install -y nginx mysql-server php php-cli php-mysql php-curl php-zip php-mbstring
php-xml php-fpm unzip git curl ufw
```

Step 4: Install Node.js + PM2

```
curl -fsSL https://deb.nodesource.com/setup_18.x | bash -

apt install -y nodejs

npm install -g pm2
```

Step 5: Secure MySQL

```
mysql_secure_installation
```

Follow prompts:

- Set root password
- Remove anonymous users
- Disallow remote root login

- Remove test DB

Step 6: Configure UFW Firewall

```
ufw allow 'OpenSSH'

ufw allow 'Nginx Full'

ufw enable
```

Step 7: Upload Your Projects

FoodScan Web App

- Upload to: /var/www/foodscan
- Set correct permissions:
- chown -R www-data:www-data /var/www/foodscan
- chmod -R 755 /var/www/foodscan

Stripe Node.js Backend

- Upload or git clone to: /var/www/stripe-api
- Install packages inside that folder:
- cd /var/www/stripe-api
- npm install

Step 8: Set Up PM2 to Run Node Server

```
pm2 start index.js --name stripe-api

pm2 save

pm2 startup
```

Step 9: Configure NGINX

FoodScan Site

Create file /etc/nginx/sites-available/foodscan

```
server {
    listen 80;
    server_name yourdomain.com;

    root /var/www/foodscan;
    index index.php index.html;

    location / {
        try_files $uri $uri/ /index.php?$query_string;
    }

    location ~ \.php$ {
        include snippets/fastcgi-php.conf;
        fastcgi_pass unix:/run/php/php8.1-fpm.sock;
    }

    location ~ /\.ht {
        deny all;
    }
}
```

Stripe API

Create /etc/nginx/sites-available/api.yourdomain.com

```
server {
    listen 80;
    server_name api.yourdomain.com;

    location / {
        proxy_pass http://localhost:3000;
        proxy_http_version 1.1;
        proxy_set_header Upgrade $http_upgrade;
        proxy_set_header Connection 'upgrade';
        proxy_set_header Host $host;
        proxy_cache_bypass $http_upgrade;
    }
}
```

Enable both:

```
ln -s /etc/nginx/sites-available/foodscan /etc/nginx/sites-enabled/
ln -s /etc/nginx/sites-available/api.yourdomain.com /etc/nginx/sites-enabled/
nginx -t && systemctl reload nginx
```

Step 10: Install Free SSL with Certbot

```
apt install certbot python3-certbot-nginx -y
certbot --nginx -d yourdomain.com -d api.yourdomain.com
```

Step 11: Optional - Set Up Daily Backups

If on Hetzner: enable snapshot backups in the Hetzner console
If using Cloudways: toggle backups in the dashboard (cost ~$2/month)

Final Folder Structure

```
/var/www/
 ├── foodscan/       → Web app from CodeCanyon (PHP)
 └── stripe-api/     → Node.js backend for Stripe Connect
```

You’re Now Ready

Your developer can:

- Connect Stripe webhooks to https://api.yourdomain.com/webhook
- Launch your platform live

- Manage everything without needing DevOps help
