# Stripe Payment Integration in ES Modules Format

Here's your payment integration system refactored using ES Modules syntax:

## 1. Stripe Service (`services/stripe.js`)

```javascript

import Stripe from 'stripe';

const stripe = new Stripe(process.env.STRIPE\_SECRET\_KEY);

export default stripe;

```

## 2. User Model (`models/User.js`)

```javascript

import mongoose from 'mongoose';

const { Schema } = mongoose;

const userSchema = new Schema({

email: String,

name: String,

// Payment fields

stripeCustomerId: String,

paymentMethods: [{

paymentMethodId: String,

cardBrand: String,

cardLast4: String,

isDefault: Boolean,

addedAt: { type: Date, default: Date.now }

}],

transactions: [{

type: Schema.Types.ObjectId,

ref: 'Payment'

}]

}, { timestamps: true });

export default mongoose.model('User', userSchema);

```

## 3. Payment Model (`models/Payment.js`)

```javascript

import mongoose from 'mongoose';

const { Schema } = mongoose;

const paymentSchema = new Schema({

user: {

type: Schema.Types.ObjectId,

ref: 'User',

required: true

},

amount: {

type: Number,

required: true

},

currency: {

type: String,

default: 'USD'

},

status: {

type: String,

enum: ['pending', 'success', 'failed', 'refunded'],

default: 'pending'

},

paymentMethod: String,

stripePaymentIntentId: String,

stripeChargeId: String,

paymentGatewayResponse: Object,

description: String,

metadata: Object

}, { timestamps: true });

export default mongoose.model('Payment', paymentSchema);

```

## 4. Payment Controller (`controllers/paymentController.js`)

```javascript

import stripe from '../services/stripe.js';

import User from '../models/User.js';

import Payment from '../models/Payment.js';

export const createPaymentIntent = async (req, res) => {

try {

const { userId, amount, currency, paymentMethod } = req.body;

const user = await User.findById(userId);

if (!user) {

return res.status(404).json({ error: 'User not found' });

}

if (!user.stripeCustomerId) {

const customer = await stripe.customers.create({

email: user.email,

name: user.name,

metadata: { userId: user.\_id.toString() }

});

user.stripeCustomerId = customer.id;

await user.save();

}

const paymentIntent = await stripe.paymentIntents.create({

amount: Math.round(amount \* 100),

currency: currency || 'usd',

customer: user.stripeCustomerId,

payment\_method\_types: ['card', 'upi'],

metadata: { userId: user.\_id.toString() }

});

const payment = new Payment({

user: user.\_id,

amount,

currency: currency || 'usd',

status: 'pending',

stripePaymentIntentId: paymentIntent.id,

paymentMethod

});

await payment.save();

res.json({

clientSecret: paymentIntent.client\_secret,

paymentId: payment.\_id

});

} catch (error) {

console.error('Error creating payment intent:', error);

res.status(500).json({ error: error.message });

}

};

export const handlePaymentSuccess = async (req, res) => {

try {

const { paymentId, paymentIntentId } = req.body;

const paymentIntent = await stripe.paymentIntents.retrieve(paymentIntentId);

const payment = await Payment.findById(paymentId).populate('user');

if (!payment) {

return res.status(404).json({ error: 'Payment record not found' });

}

payment.status = paymentIntent.status === 'succeeded' ? 'success' : 'failed';

payment.stripeChargeId = paymentIntent.charges.data[0]?.id;

payment.paymentGatewayResponse = paymentIntent;

await payment.save();

if (payment.status === 'success') {

const user = payment.user;

user.transactions.push(payment.\_id);

if (req.body.savePaymentMethod) {

const paymentMethod = await stripe.paymentMethods.retrieve(

paymentIntent.payment\_method

);

user.paymentMethods.push({

paymentMethodId: paymentMethod.id,

cardBrand: paymentMethod.card?.brand,

cardLast4: paymentMethod.card?.last4,

isDefault: user.paymentMethods.length === 0

});

}

await user.save();

}

res.json({ success: true, payment });

} catch (error) {

console.error('Error handling payment success:', error);

res.status(500).json({ error: error.message });

}

};

export const stripeWebhook = async (req, res) => {

const sig = req.headers['stripe-signature'];

let event;

try {

event = stripe.webhooks.constructEvent(

req.rawBody,

sig,

process.env.STRIPE\_WEBHOOK\_SECRET

);

} catch (err) {

console.error('Webhook signature verification failed:', err);

return res.status(400).send(`Webhook Error: ${err.message}`);

}

switch (event.type) {

case 'payment\_intent.succeeded':

await handleSuccessfulPayment(event.data.object);

break;

case 'payment\_intent.payment\_failed':

await handleFailedPayment(event.data.object);

break;

default:

console.log(`Unhandled event type ${event.type}`);

}

res.json({ received: true });

};

async function handleSuccessfulPayment(paymentIntent) {

const payment = await Payment.findOne({

stripePaymentIntentId: paymentIntent.id

});

if (payment) {

payment.status = 'success';

payment.stripeChargeId = paymentIntent.charges.data[0]?.id;

payment.paymentGatewayResponse = paymentIntent;

await payment.save();

}

}

async function handleFailedPayment(paymentIntent) {

const payment = await Payment.findOne({

stripePaymentIntentId: paymentIntent.id

});

if (payment) {

payment.status = 'failed';

payment.paymentGatewayResponse = paymentIntent;

await payment.save();

}

}

export default {

createPaymentIntent,

handlePaymentSuccess,

stripeWebhook

};

```

## 5. Routes (`routes/paymentRoutes.js`)

```javascript

import express from 'express';

import paymentController from '../controllers/paymentController.js';

const router = express.Router();

router.post('/create-payment-intent', paymentController.createPaymentIntent);

router.post('/payment-success', paymentController.handlePaymentSuccess);

router.post('/webhook', paymentController.stripeWebhook);

export default router;

```

## 6. Main Application File (`app.js` or `server.js`)

```javascript

import express from 'express';

import mongoose from 'mongoose';

import dotenv from 'dotenv';

import paymentRoutes from './routes/paymentRoutes.js';

dotenv.config();

const app = express();

// Middleware

app.use(express.json());

app.use(express.urlencoded({ extended: true }));

// Database connection

mongoose.connect(process.env.MONGODB\_URI)

.then(() => console.log('Connected to MongoDB'))

.catch(err => console.error('MongoDB connection error:', err));

// Routes

app.use('/api/payments', paymentRoutes);

// Error handling middleware

app.use((err, req, res, next) => {

console.error(err.stack);

res.status(500).send('Something broke!');

});

const PORT = process.env.PORT || 3000;

app.listen(PORT, () => {

console.log(`Server running on port ${PORT}`);

});

```

## 7. Flutter Client Implementation

For your Flutter app, you'll need to:

1. Add the `stripe\_sdk` package to your `pubspec.yaml`

2. Create a service class to handle API calls:

```dart

// lib/services/payment\_service.dart

import 'dart:convert';

import 'package:http/http.dart' as http;

class PaymentService {

final String baseUrl;

PaymentService({required this.baseUrl});

Future<Map<String, dynamic>> createPaymentIntent({

required String userId,

required double amount,

String currency = 'USD',

String paymentMethod = 'card',

}) async {

final response = await http.post(

Uri.parse('$baseUrl/api/payments/create-payment-intent'),

headers: {'Content-Type': 'application/json'},

body: json.encode({

'userId': userId,

'amount': amount,

'currency': currency,

'paymentMethod': paymentMethod,

}),

);

if (response.statusCode == 200) {

return json.decode(response.body);

} else {

throw Exception('Failed to create payment intent');

}

}

Future<void> confirmPaymentSuccess({

required String paymentId,

required String paymentIntentId,

bool savePaymentMethod = false,

}) async {

final response = await http.post(

Uri.parse('$baseUrl/api/payments/payment-success'),

headers: {'Content-Type': 'application/json'},

body: json.encode({

'paymentId': paymentId,

'paymentIntentId': paymentIntentId,

'savePaymentMethod': savePaymentMethod,

}),

);

if (response.statusCode != 200) {

throw Exception('Failed to confirm payment');

}

}

}

```

## Key Differences from CommonJS:

1. Uses `import/export` syntax instead of `require/module.exports`

2. File extensions must be included in imports (`.js`)

3. Top-level `await` requires wrapping in async functions or using module wrappers

4. More modern and standardized approach supported in Node.js 12+

Remember to:

1. Add `"type": "module"` to your `package.json`

2. Update your `start` script to use ES modules

3. Ensure all file imports include the `.js` extension

4. Use the latest stable version of Node.js for best support

can you show me how the flow of the features work . like first this button in app triggers which route and so on

Here's a clear flow of how the payment feature works from the Flutter app to the Node.js backend:

## Payment Flow Diagram

```

Flutter App UI → Node.js Backend → Stripe API → Database

```

## Step-by-Step Flow

### 1. Initiate Payment (Flutter UI)

- \*\*User Action\*\*: Clicks "Pay Now" button

- \*\*Flutter Code\*\*:

```dart

ElevatedButton(

onPressed: () async {

final result = await PaymentService().createPaymentIntent(

userId: 'user123',

amount: 99.99,

paymentMethod: 'gpay' // or 'card', 'paypal'

);

// Proceed to payment sheet

},

child: Text('Pay Now'),

)

```

### 2. Create Payment Intent (Backend)

- \*\*Route\*\*: `POST /api/payments/create-payment-intent`

- \*\*Backend Flow\*\*:

1. Receives `{ userId, amount, paymentMethod }`

2. Checks if user exists

3. Creates Stripe customer if needed

4. Creates payment intent with Stripe

5. Saves preliminary payment record to MongoDB

6. Returns `{ clientSecret, paymentId }`

### 3. Display Payment Sheet (Flutter)

```dart

await Stripe.instance.initPaymentSheet(

paymentSheetParameters: SetupPaymentSheetParameters(

paymentIntentClientSecret: result['clientSecret'],

merchantDisplayName: 'My App',

googlePay: PaymentSheetGooglePay(

merchantCountryCode: 'US',

currencyCode: 'USD',

testEnv: true

)

)

);

await Stripe.instance.presentPaymentSheet();

```

### 4. Payment Processing (Stripe)

- Stripe handles:

- Card authentication

- 3D Secure if needed

- UPI/GPay redirects

- PayPal authentication

### 5. Payment Result Handling

#### Success Case:

- \*\*Flutter\*\*:

```dart

try {

await Stripe.instance.confirmPaymentSheetPayment();

await PaymentService().confirmPaymentSuccess(

paymentId: result['paymentId'],

paymentIntentId: getIntentIdFromClientSecret(result['clientSecret'])

);

showSuccessUI();

} catch (e) {

showErrorUI();

}

```

- \*\*Backend\*\* (`POST /api/payments/payment-success`):

1. Verifies payment with Stripe

2. Updates payment record status to "success"

3. Adds transaction to user document

4. Saves payment method if requested

#### Failure Case:

- Automatically handled by Stripe SDK

- Error shown to user

- Backend still records failed attempt via webhook

### 6. Webhook Events (Async Updates)

- \*\*Stripe → Backend\*\*: `POST /api/payments/webhook`

- Handles:

- Delayed payment confirmations

- Disputes

- Refunds

- Subscription renewals

### 7. Database Updates

```mermaid

sequenceDiagram

participant Flutter

participant NodeJS

participant Stripe

participant MongoDB

Flutter->>NodeJS: POST /create-payment-intent

NodeJS->>Stripe: Create PaymentIntent

Stripe-->>NodeJS: clientSecret

NodeJS->>MongoDB: Save Payment (pending)

NodeJS-->>Flutter: Return clientSecret

Flutter->>Stripe: Present payment UI

Stripe-->>Flutter: Payment complete

Flutter->>NodeJS: POST /payment-success

NodeJS->>Stripe: Verify payment

NodeJS->>MongoDB: Update Payment (success)

NodeJS->>MongoDB: Update User (add transaction)

Stripe->>NodeJS: Webhook event

NodeJS->>MongoDB: Sync payment status

```

## Key Endpoints Summary

| Flutter Action | Backend Route | Description |

|---------------|--------------|-------------|

| Initiate payment | `POST /create-payment-intent` | Creates Stripe intent and DB record |

| Confirm success | `POST /payment-success` | Finalizes successful payments |

| - | `POST /webhook` | Handles async Stripe events |

## Error Handling Flow

1. \*\*Frontend Errors\*\*:

- Display user-friendly messages

- Log technical details

- Allow retry

2. \*\*Backend Errors\*\*:

- Always update payment status (even failures)

- Preserve Stripe error codes

- Send appropriate HTTP status codes

This flow ensures:

- Real-time UI feedback

- Reliable payment recording

- Audit trail for all transactions

- Support for multiple payment methods

so im confused like when the pay now button gets clicked it will hit the router.post('/create-payment-intent', paymentController.createPaymentIntent); route. so when should the front end hit router.post('/payment-success', paymentController.handlePaymentSuccess);

router.post('/webhook', paymentController.stripeWebhook); these routes

Let me clarify the complete flow with precise timing for each API call. Here's exactly when each route should be called:

---

### \*\*Complete Payment Flow with Route Triggers\*\*

#### \*\*1. "Pay Now" Button Clicked (Initial Payment Setup)\*\*

- \*\*Flutter Action\*\*: User clicks payment button

- \*\*API Call\*\*: `POST /create-payment-intent`

- \*\*When\*\*: Immediately after button click, \*\*before showing payment UI\*\*

- \*\*Purpose\*\*:

- Creates a Stripe PaymentIntent

- Saves a preliminary payment record in MongoDB (`status: pending`)

- Gets `clientSecret` for client-side confirmation

```dart

// Flutter Code

onPressed: () async {

// Step 1: Call your backend to create intent

final response = await http.post(

Uri.parse('$BASE\_URL/create-payment-intent'),

body: jsonEncode({

'userId': '123',

'amount': 100,

'currency': 'USD'

})

);

final clientSecret = response.data['clientSecret'];

final paymentId = response.data['paymentId'];

// Now show Stripe payment sheet with this clientSecret

}

```

---

#### \*\*2. Payment UI Confirmation (Client-Side)\*\*

- \*\*Flutter Action\*\*:

- Shows Stripe payment sheet using `clientSecret`

- User completes payment (card/GPay/PayPal)

- \*\*API Call\*\*: None yet (pure client-side Stripe interaction)

- \*\*When\*\*: After receiving `clientSecret` but \*\*before payment succeeds/fails\*\*

```dart

// Flutter Code

await Stripe.instance.initPaymentSheet(

paymentSheetParameters: SetupPaymentSheetParameters(

paymentIntentClientSecret: clientSecret,

// ... other config

)

);

// Present the payment sheet to user

await Stripe.instance.presentPaymentSheet();

// This is where user actually pays

```

---

#### \*\*3. Payment Success (Immediate Frontend Confirmation)\*\*

- \*\*Flutter Action\*\*:

- Detect successful payment client-side

- \*\*Optional\*\*: Show temporary success UI

- \*\*API Call\*\*: `POST /payment-success`

- \*\*When\*\*: Immediately after Stripe confirms success \*\*but before showing final success UI\*\*

- \*\*Purpose\*\*:

- Verify payment with Stripe (anti-fraud)

- Update MongoDB records (`status: success`)

- Save payment method if requested

```dart

try {

// Step 2: Client-side confirmation

await Stripe.instance.confirmPaymentSheetPayment();

// Step 3: Notify backend of success

await http.post(

Uri.parse('$BASE\_URL/payment-success'),

body: jsonEncode({

'paymentId': paymentId,

'paymentIntentId': extractId(clientSecret),

'savePaymentMethod': saveForFuture

})

);

// Only now show permanent success UI

showSuccessScreen();

} catch (e) {

showErrorUI();

}

```

---

#### \*\*4. Webhook (Async Backend Verification)\*\*

- \*\*Trigger\*\*: Automatically by Stripe (not called by Flutter)

- \*\*API Call\*\*: `POST /webhook`

- \*\*When\*\*: 1-5 seconds after payment completes (Stripe's timing)

- \*\*Purpose\*\*:

- \*\*Redundant verification\*\* (in case frontend call fails)

- Handle disputes/refunds later

- Process subscriptions/recurring payments

```javascript

// Node.js Webhook Handler

router.post('/webhook', (req, res) => {

// Stripe sends events here AUTOMATICALLY

// Not triggered by your Flutter app!

});

```

---

### \*\*Key Timing Diagram\*\*

```mermaid

sequenceDiagram

participant Flutter

participant NodeJS

participant Stripe

Flutter->>NodeJS: POST /create-payment-intent (when button clicked)

NodeJS-->>Flutter: clientSecret

Flutter->>Stripe: presentPaymentSheet() (user pays)

alt Success

Stripe-->>Flutter: Payment complete

Flutter->>NodeJS: POST /payment-success (before showing final UI)

Stripe->>NodeJS: POST /webhook (async, later)

else Failure

Stripe-->>Flutter: Error shown immediately

end

```

---

### \*\*When to Use Each Route\*\*

| Route | Called By | Timing | Purpose |

|-------|----------|--------|---------|

| `/create-payment-intent` | Flutter | On button click | Setup payment |

| `/payment-success` | Flutter | After client-side success | Confirm payment |

| `/webhook` | \*\*Stripe\*\* (auto) | 1-5s after payment | Backup verification |

---

### \*\*Why Two Success Paths?\*\*

1. \*\*Frontend-initiated (`/payment-success`)\*\*:

- Faster user feedback (update UI immediately)

- But less secure (client could fake this call)

2. \*\*Webhook (Stripe-initiated)\*\*:

- More reliable (Stripe always calls this)

- Handles edge cases (e.g., network failures)

- Required for subscriptions/refunds

\*\*Best Practice\*\*: Use both for redundancy!

- Update UI optimistically via `/payment-success`

- Let webhook correct any mismatches later

---

### \*\*Example Scenario\*\*

1. User pays successfully → Flutter calls `/payment-success`

2. Network fails → MongoDB still shows "pending"

3. Stripe webhook arrives later → updates to "success"