Stripe Payment Gateway Integration Report

# 1. Integration and Deployment of Stripe Payment Gateway Using Node.js

Stripe is a widely adopted online payment processing platform, and its integration in web applications enables secure and seamless transactions. For this assignment website, the Stripe payment gateway has been deployed using a Node.js backend service. This Node.js service acts as an API server that receives cart data, creates Stripe checkout sessions, and verifies transaction results.  
  
The integration relies on Express for handling HTTP requests and Stripe’s official SDK for securely interacting with Stripe’s payment infrastructure. Upon initiating a payment, the backend transforms cart items into Stripe-compatible line items and initiates a checkout session. Upon payment success, Stripe redirects the user to a predefined PHP page. Session verification is also handled on the backend via Stripe’s RESTful API.  
  
Deployment involves hosting this service locally at port 4242 and using a process manager (e.g. PM2) to keep the Node.js server running continuously.

# 2. Illustration of Stripe Backend Integration Using Code Examples (stripe.js)

The backend logic for Stripe integration is encapsulated within the stripe.js file. It sets up an Express server, loads environmental variables using dotenv, and instantiates the Stripe client. CORS and JSON parsing are configured to allow the frontend to communicate effectively.  
The /checkout endpoint receives cart item data from the frontend. Each item is mapped to Stripe’s expected line item format. The amount is calculated in cents (as Stripe requires it in the smallest currency unit). A checkout session is then created, and the user is redirected to Stripe's hosted checkout page.  
  
The /verify-Stripe endpoint is used post-payment to confirm if a session is complete by querying Stripe with the session ID passed via the success URL.

// Creates a checkout session based on cart data  
app.post('/checkout', async (req, res) => {  
 const { items } = req.body;  
 const lineItems = items.map(item => ({  
 price\_data: {  
 currency: 'hkd',  
 product\_data: {  
 name: item.name,  
 },  
 unit\_amount: Math.round(item.price \* 100),  
 },  
 quantity: item.quantity,  
 }));  
 const session = await stripe.checkout.sessions.create({  
 line\_items: lineItems,  
 mode: 'payment',  
 success\_url: 'http://localhost/Web%20Assignment/payment\_completed.php?session\_id={CHECKOUT\_SESSION\_ID}',  
 cancel\_url: 'http://localhost/Web%20Assignment/cart.php',  
 });  
 res.json({ url: session.url });  
});  
  
// Verifies Stripe session after checkout  
app.get('/verify-Stripe', async (req, res) => {  
 const sessionId = req.query.session\_id;  
 const session = await stripe.checkout.sessions.retrieve(sessionId);  
 if (session.payment\_status === "paid") {  
 res.json({ verified: true, session });  
 } else {  
 res.status(403).json({ verified: false });  
 }  
});

The above functions are essential for converting purchase data into Stripe’s format and for verifying payment success. This setup allows the PHP frontend to offload all sensitive payment handling to the Node.js backend, ensuring both security and compliance with Stripe's requirements.

# 3. Illustration of Stripe Frontend Integration Using Code Examples (cart.js)

On the client side, the cart.js script is responsible for handling the checkout interaction. When the 'checkout' button is clicked, the script reads the local cart items, retrieves full product details via API, and transforms each entry into a format suitable for the backend API. The formatted data is then POSTed to the /checkout endpoint on the Node.js server.

document.getElementById("checkoutButton").addEventListener("click", async () => {  
 const cartItems = JSON.parse(localStorage.getItem("cartItems")) || [];  
  
 const formattedItems = await Promise.all(cartItems.map(async item => {  
 const product = await fetchProductById(item.pid);  
 return {  
 name: product.cardTitle ?? "Untitled Product",  
 price: product.itemPrice ?? 0,  
 quantity: item.quantity ?? 1  
 };  
 }));  
  
 const response = await fetch('http://localhost:4242/checkout', {  
 method: 'POST',  
 headers: {  
 'Content-Type': 'application/json'  
 },  
 body: JSON.stringify({ items: formattedItems })  
 });  
  
 const data = await response.json();  
 if (data.url) {  
 window.location.href = data.url;  
 }  
});

This frontend logic ensures that item information is always up-to-date by fetching the latest product details before checkout. It also simplifies the handoff between the PHP-based site and the Node.js backend by adhering to a consistent JSON structure. After receiving the checkout session URL from the backend, the browser is redirected to the Stripe payment page.

# 4. Conclusion

The integration of the Stripe payment gateway via a dedicated Node.js backend has enabled the website to securely and reliably handle checkout transactions. The system separates payment logic from the frontend and PHP-based Firestore interaction, ensuring a modular and maintainable structure. By structuring the communication flow between frontend JavaScript, RESTful endpoints, and Stripe’s secure APIs, the payment experience remains both user-friendly and secure. This architecture also allows the platform to be extended easily to support advanced features, such as billing history, downloadable receipts, or subscription-based models in the future. The current setup provides a strong foundation for further e-commerce functionality.

# Read on GitHub

<https://github.com/Test-Plus-XD/Web-Assignment>