KCT Menswear Frontend Integration Documentation

Complete API Integration Guide for Website Frontend

Version: 1.0
Date: August 18, 2025
Author: MiniMax Agent
Backend Admin URL: https://rtbbsdcrfbha.space.minimax.io

Table of Contents

System Overview

Architecture Components

- **Frontend Website:** Vercel-hosted customer-facing website
- **Backend Admin System:** https://rtbbsdcrfbha.space.minimax.io
- **Database:** Supabase PostgreSQL
- **API Layer:** Supabase Edge Functions
- **Integrations:** Stripe, EasyPost, SendGrid

Dual Product Architecture

Core Products (Stripe-managed):
- 28 core items + 38 bundles
- Payment processing through Stripe
- Product data synced from Stripe

Catalog Products (Supabase-managed):
- 150+ items in custom catalog
- Custom pricing and inventory
- Direct database integration

Authentication & Security

API Base URL

```
https://your-supabase-project.supabase.co/functions/v1/
```

Authentication Methods

Service Role Authentication (Recommended for Server-Side)

```
const headers = {
  'Authorization': `Bearer ${SUPABASE_SERVICE_ROLE_KEY}`,
  'apikey': SUPABASE_SERVICE_ROLE_KEY,
  'Content-Type': 'application/json'
};
```

Anonymous Key Authentication (Client-Side)

```
const headers = {
  'Authorization': `Bearer ${SUPABASE_ANON_KEY}`,
  'apikey': SUPABASE_ANON_KEY,
  'Content-Type': 'application/json'
};
```

Required Environment Variables

```
# Supabase Configuration
SUPABASE_URL=https://your-project.supabase.co
SUPABASE_ANON_KEY=your_anon_key
SUPABASE_SERVICE_ROLE_KEY=your_service_role_key

# API Keys (for backend integration)
STRIPE_SECRET_KEY=sk_...
EASYPOST_API_KEY=EZAK...
SENDGRID_API_KEY=SG...

# Email Configuration
ADMIN_EMAIL=KCTMenswear@gmail.com
FROM_EMAIL=noreply@kctmenswear.com
```

API Endpoints Reference

1. Order Management

Create Order (Core + Catalog Products)

Endpoint: `/order-management`
Method: POST

```
const createOrder = async (orderData) => {
  const response = await fetch(
    `${SUPABASE_URL}/functions/v1/order-management`,
    {
      method: 'POST',
      headers: {
        'Authorization': `Bearer ${SUPABASE_SERVICE_ROLE_KEY}`,
        'apikey': SUPABASE_SERVICE_ROLE_KEY,
        'Content-Type': 'application/json'
      },
      body: JSON.stringify({
        action: 'create_order_queue_entry',
        order_data: orderData
      })
    }
  );

  return await response.json();
};
```

Update Order Status

Endpoint: `/order-management`
Method: POST

```
const updateOrderStatus = async (orderId, newStatus, notes = '')
=> {
  const response = await fetch(
    `${SUPABASE_URL}/functions/v1/order-management`,
    {
      method: 'POST',
      headers: {
        'Authorization': `Bearer ${SUPABASE_SERVICE_ROLE_KEY}`,
        'apikey': SUPABASE_SERVICE_ROLE_KEY,
        'Content-Type': 'application/json'
      },
      body: JSON.stringify({
        action: 'update_order_status',
        order_id: orderId,
        order_data: {
          new_status: newStatus,
          notes: notes,
          changed_by: 'frontend_system'
        }
      })
    }
  );

  return await response.json();
};
```

Get Order Analytics

Endpoint: `/order-management`

Method: POST

```
const getOrderAnalytics = async (filters = {}) => {
  const response = await fetch(
    `${SUPABASE_URL}/functions/v1/order-management`,
    {
      method: 'POST',
      headers: {
        'Authorization': `Bearer ${SUPABASE_SERVICE_ROLE_KEY}`,
        'apikey': SUPABASE_SERVICE_ROLE_KEY,
        'Content-Type': 'application/json'
      },
      body: JSON.stringify({
        action: 'get_order_analytics',
        filters: filters
      })
    }
  );

  return await response.json();
};
```

2. Stripe Payment Processing

Create Payment Intent (Core Products)

Endpoint: `/stripe-payment-intent`

Method: POST

```
const createPaymentIntent = async (paymentData) => {
  const response = await fetch(
    `${SUPABASE_URL}/functions/v1/stripe-payment-intent`,
    {
      method: 'POST',
      headers: {
        'Authorization': `Bearer ${SUPABASE_SERVICE_ROLE_KEY}`,
        'apikey': SUPABASE_SERVICE_ROLE_KEY,
        'Content-Type': 'application/json'
      },
      body: JSON.stringify({
        amount: paymentData.amount,
        currency: paymentData.currency || 'usd',
        cartItems: paymentData.cartItems,
        customerEmail: paymentData.customerEmail,
        shippingAddress: paymentData.shippingAddress,
        billingAddress: paymentData.billingAddress,
        specialInstructions: paymentData.specialInstructions,
        rushOrder: paymentData.rushOrder,
        groupOrder: paymentData.groupOrder
      })
    }
  );

  return await response.json();
};
```

3. Shipping Integration

Calculate Shipping Rates

Endpoint: `/shipping-rates`
Method: POST

```
const calculateShippingRates = async (shippingData) => {
  const response = await fetch(
    `${SUPABASE_URL}/functions/v1/shipping-rates`,
    {
      method: 'POST',
      headers: {
        'Authorization': `Bearer ${SUPABASE_SERVICE_ROLE_KEY}`,
        'apikey': SUPABASE_SERVICE_ROLE_KEY,
        'Content-Type': 'application/json'
      },
      body: JSON.stringify({
        orderId: shippingData.orderId,
        toAddress: {
          name: shippingData.customerName,
          street1: shippingData.street1,
          street2: shippingData.street2,
          city: shippingData.city,
          state: shippingData.state,
          zip: shippingData.zip,
          country: shippingData.country || 'US',
          phone: shippingData.phone
        },
        weight: shippingData.weight || 16, // ounces
        dimensions: {
          length: shippingData.length || 12,
          width: shippingData.width || 9,
          height: shippingData.height || 3
        }
      })
    }
  );

  return await response.json();
};
```

Create Shipping Label

Endpoint: `/shipping-label`
Method: POST

```
const createShippingLabel = async (labelData) => {
  const response = await fetch(
    `${SUPABASE_URL}/functions/v1/shipping-label`,
    {
      method: 'POST',
      headers: {
        'Authorization': `Bearer ${SUPABASE_SERVICE_ROLE_KEY}`,
        'apikey': SUPABASE_SERVICE_ROLE_KEY,
        'Content-Type': 'application/json'
      },
      body: JSON.stringify({
        rateId: labelData.rateId,
        orderId: labelData.orderId
      })
    }
  );

  return await response.json();
};
```

4. Email Integration

Send Email Notification

Endpoint: `/send-email`
Method: POST

```
const sendEmail = async (emailData) => {
  const response = await fetch(
    `${SUPABASE_URL}/functions/v1/send-email`,
    {
      method: 'POST',
      headers: {
        'Authorization': `Bearer ${SUPABASE_SERVICE_ROLE_KEY}`,
        'apikey': SUPABASE_SERVICE_ROLE_KEY,
        'Content-Type': 'application/json'
      },
      body: JSON.stringify({
        emailType: emailData.emailType, // 'order_confirmation',
'shipping_confirmation', etc.
        orderData: emailData.orderData,
        trackingData: emailData.trackingData,
        customData: emailData.customData
      })
    }
  );

  return await response.json();
};
```

5. Order Automation

Trigger Order Automation

Endpoint: `/order-automation`
Method: POST

```
const triggerOrderAutomation = async (automationData) => {
  const response = await fetch(
    `${SUPABASE_URL}/functions/v1/order-automation`,
    {
      method: 'POST',
      headers: {
        'Authorization': `Bearer ${SUPABASE_SERVICE_ROLE_KEY}`,
        'apikey': SUPABASE_SERVICE_ROLE_KEY,
        'Content-Type': 'application/json'
      },
      body: JSON.stringify({
        action: automationData.action, // 'order_created',
'status_changed', 'shipping_label_created'
        orderData: automationData.orderData,
        previousStatus: automationData.previousStatus
      })
    }
  );

  return await response.json();
};
```

Order Creation Integration

Complete Order Flow Implementation

Step 1: Create Order and Payment Intent (For Core Products)

```
const processStripeOrder = async (cartData) => {
  try {
    // Step 1: Create Payment Intent
    const paymentIntentResponse = await createPaymentIntent({
      amount: cartData.total,
      currency: 'usd',
      cartItems: cartData.items.map(item => ({
        product_id: item.id,
        product_name: item.name,
        stripe_product_id: item.stripeProductId,
        product_source: 'core_stripe',
        quantity: item.quantity,
        price: item.price,
        size: item.size,
        color: item.color
      })),
      customerEmail: cartData.customer.email,
      shippingAddress: cartData.shippingAddress,
      billingAddress: cartData.billingAddress,
      specialInstructions: cartData.notes,
      rushOrder: cartData.isRush,
      groupOrder: cartData.isGroup
    });

    if (paymentIntentResponse.error) {
      throw new Error(paymentIntentResponse.error.message);
    }

    const { clientSecret, orderId, orderNumber } =
paymentIntentResponse.data;

    // Step 2: Process payment on frontend with Stripe Elements
    // (This happens on your frontend with Stripe.js)

    // Step 3: After successful payment, trigger automation
```

```
    await triggerOrderAutomation({
      action: 'order_created',
      orderData: {
        id: orderId,
        order_number: orderNumber,
        customer_email: cartData.customer.email,
        customer_name: cartData.customer.name,
        total_price: cartData.total,
        status: 'payment_confirmed',
        created_at: new Date().toISOString(),
        shipping_address: cartData.shippingAddress
      }
    });

    return {
      success: true,
      clientSecret,
      orderId,
      orderNumber
    };

  } catch (error) {
    console.error('Order processing failed:', error);
    return {
      success: false,
      error: error.message
    };
  }
};
```

Step 2: Create Direct Order (For Catalog Products)

```
const processCatalogOrder = async (cartData) => {
  try {
    // Create order directly in database
    const orderData = {
      order_id: `KCT-<span class="math-inline" style="display:
inline;"><math xmlns="http://www.w3.org/1998/Math/MathML"
display="inline"><mrow><mrow><mi>D</mi><mi>a</mi><mi>t</mi><mi>e</
mi><mo>&#x0002E;</mo><mi>n</mi><mi>o</mi><mi>w</mi><mo
stretchy="false">&#x00028;</mo><mo stretchy="false">&#x00029;</
mo></mrow><mo>&#x02212;</mo></mrow></math></
span>{Math.random().toString(36).substr(2, 4).toUpperCase()}`,
      customer_email: cartData.customer.email,
      customer_name: cartData.customer.name,
      customer_phone: cartData.customer.phone,
      total_amount: cartData.total,
      items: cartData.items.map(item => ({
        product_id: item.id,
        product_name: item.name,
        catalog_product_id: item.catalogId,
        product_source: 'catalog_supabase',
        quantity: item.quantity,
        price: item.price,
        size: item.size,
        color: item.color,
        sku: item.sku
      })),
      shipping_address: cartData.shippingAddress,
      billing_address: cartData.billingAddress,
      special_instructions: cartData.notes,
      rush_order: cartData.isRush,
      group_order: cartData.isGroup
    };

    // Create order queue entry
    const createResponse = await createOrder(orderData);
```

```
    if (createResponse.error) {
      throw new Error(createResponse.error.message);
    }

    // Trigger automation for order confirmation emails
    await triggerOrderAutomation({
      action: 'order_created',
      orderData: {
        ...orderData,
        id: createResponse.data.id,
        status: 'pending_payment'
      }
    });

    return {
      success: true,
      orderId: createResponse.data.id,
      orderNumber: orderData.order_id
    };

  } catch (error) {
    console.error('Catalog order creation failed:', error);
    return {
      success: false,
      error: error.message
    };
  }
};
```

Step 3: Mixed Cart Processing (Core + Catalog Products)

```
const processMixedCart = async (cartData) => {
  try {
    const coreProducts = cartData.items.filter(item => item.type
=== 'core');
    const catalogProducts = cartData.items.filter(item =>
item.type === 'catalog');

    const results = [];

    // Process Core Products if any
    if (coreProducts.length > 0) {
      const coreCartData = {
        ...cartData,
        items: coreProducts,
        total: coreProducts.reduce((sum, item) => sum +
(item.price * item.quantity), 0)
      };

      const coreResult = await processStripeOrder(coreCartData);
      results.push({ type: 'core', ...coreResult });
    }

    // Process Catalog Products if any
    if (catalogProducts.length > 0) {
      const catalogCartData = {
        ...cartData,
        items: catalogProducts,
        total: catalogProducts.reduce((sum, item) => sum +
(item.price * item.quantity), 0)
      };

      const catalogResult = await
processCatalogOrder(catalogCartData);
      results.push({ type: 'catalog', ...catalogResult });
    }
```

```
    return {
      success: results.every(r => r.success),
      results
    };

  } catch (error) {
    console.error('Mixed cart processing failed:', error);
    return {
      success: false,
      error: error.message
    };
  }
};
```

Shipping Integration

Shipping Workflow Implementation

Step 1: Calculate Shipping Rates

```
const getShippingOptions = async (orderData) => {
  try {
    const shippingRates = await calculateShippingRates({
      orderId: orderData.orderId,
      customerName: orderData.customer.name,
      street1: orderData.shippingAddress.street1,
      street2: orderData.shippingAddress.street2,
      city: orderData.shippingAddress.city,
      state: orderData.shippingAddress.state,
      zip: orderData.shippingAddress.zip,
      country: orderData.shippingAddress.country,
      phone: orderData.customer.phone,
      weight: calculateOrderWeight(orderData.items),
      dimensions: calculateOrderDimensions(orderData.items)
    });

    if (shippingRates.error) {
      throw new Error(shippingRates.error.message);
    }

    return shippingRates.data.rates;

  } catch (error) {
    console.error('Failed to get shipping rates:', error);
    return [];
  }
};

// Helper functions
const calculateOrderWeight = (items) => {
  // Default weight calculation - customize based on your products
  return items.reduce((total, item) => {
    const itemWeight = item.weight || 1; // default 1 oz per item
    return total + (itemWeight * item.quantity);
  }, 4); // 4 oz base packaging weight
```

```
};

const calculateOrderDimensions = (items) => {
  // Default dimensions - customize based on your products
  const itemCount = items.reduce((sum, item) => sum +
item.quantity, 0);

  if (itemCount <= 2) {
    return { length: 12, width: 9, height: 3 };
  } else if (itemCount <= 5) {
    return { length: 14, width: 10, height: 4 };
  } else {
    return { length: 16, width: 12, height: 6 };
  }
};
```

Step 2: Process Shipping Selection

```
const processShippingSelection = async (orderId, selectedRateId)
=> {
  try {
    // Create shipping label
    const labelResponse = await createShippingLabel({
      rateId: selectedRateId,
      orderId: orderId
    });

    if (labelResponse.error) {
      throw new Error(labelResponse.error.message);
    }

    const { shipmentId, trackingNumber, labelUrl, cost } =
labelResponse.data;

    // Update order status
    await updateOrderStatus(orderId, 'shipped',
      `Shipping label created. Tracking: ${trackingNumber}`);

    // Trigger shipping automation
    await triggerOrderAutomation({
      action: 'shipping_label_created',
      orderData: {
        id: orderId,
        tracking_number: trackingNumber,
        carrier: labelResponse.data.carrier,
        status: 'shipped'
      }
    });

    return {
      success: true,
      trackingNumber,
      labelUrl,
```

```
      cost
    };

  } catch (error) {
    console.error('Shipping processing failed:', error);
    return {
      success: false,
      error: error.message
    };
  }
};
```

Email Integration

Available Email Types

1. **order_confirmation** - Sent when order is created
2. **shipping_confirmation** - Sent when order ships with tracking
3. **delivery_confirmation** - Sent when order is delivered
4. **admin_new_order** - Admin notification for new orders

Email Implementation Examples

```
// Send order confirmation
const sendOrderConfirmation = async (orderData) => {
  return await sendEmail({
    emailType: 'order_confirmation',
    orderData: {
      id: orderData.id,
      customer_email: orderData.customer_email,
      customer_name: orderData.customer_name,
      total_price: orderData.total_amount,
      created_at: orderData.created_at,
      status: orderData.status,
      shipping_address: orderData.shipping_address
    }
  });
};

// Send shipping confirmation with tracking
const sendShippingConfirmation = async (orderData, trackingData)
=> {
  return await sendEmail({
    emailType: 'shipping_confirmation',
    orderData: {
      id: orderData.id,
      customer_email: orderData.customer_email,
      customer_name: orderData.customer_name,
      total_price: orderData.total_amount
    },
    trackingData: {
      tracking_code: trackingData.tracking_number,
      carrier: trackingData.carrier,
      estimated_delivery_date: trackingData.estimated_delivery,
      tracking_url: `https://tools.usps.com/go/TrackConfirmAction?
qtc_tLabels1=${trackingData.tracking_number}`
    }
  });
```

```
};

// Send admin notification
const sendAdminNotification = async (orderData) => {
  return await sendEmail({
    emailType: 'admin_new_order',
    orderData: {
      id: orderData.id,
      customer_name: orderData.customer_name,
      customer_email: orderData.customer_email,
      total_price: orderData.total_amount,
      created_at: orderData.created_at
    }
  });
};
```

Database Schema & Data Structures

Order Data Structure

```
interface Order {
  id: string;
  order_number: string;
  customer_email: string;
  customer_name: string;
  customer_phone?: string;
  status: OrderStatus;
  order_priority: OrderPriority;
  subtotal: number;
  tax_amount?: number;
  shipping_amount?: number;
  discount_amount?: number;
  total_amount: number;
  currency: string;

  // Stripe integration fields
  stripe_payment_intent_id?: string;
  payment_method?: string;
  payment_status?: string;

  // Address fields
  shipping_address_line_1?: string;
  shipping_address_line_2?: string;
  shipping_first_name?: string;
  shipping_last_name?: string;
  shipping_city?: string;
  shipping_state?: string;
  shipping_postal_code?: string;
  shipping_country?: string;

  billing_address_line_1?: string;
  billing_address_line_2?: string;
  billing_city?: string;
  billing_state?: string;
  billing_postal_code?: string;
```

```
  billing_country?: string;

  // Shipping fields
  shipping_rate_id?: string;
  shipping_label_url?: string;
  tracking_number?: string;
  tracking_status?: string;
  carrier?: string;
  service_type?: string;
  shipping_cost?: number;
  easypost_shipment_id?: string;

  // Order characteristics
  is_rush_order: boolean;
  is_group_order: boolean;
  special_instructions?: string;
  internal_notes?: string;

  // Timestamps
  created_at: string;
  updated_at: string;
  processed_at?: string;
  shipped_at?: string;
  delivered_at?: string;
  estimated_delivery_date?: string;
  actual_delivery_date?: string;
}

type OrderStatus =
  | 'pending_payment'
  | 'payment_confirmed'
  | 'processing'
  | 'in_production'
  | 'quality_check'
  | 'packaging'
```

```
  | 'shipped'
  | 'out_for_delivery'
  | 'delivered'
  | 'completed'
  | 'cancelled'
  | 'refunded'
  | 'on_hold'
  | 'exception';

type OrderPriority =
  | 'low'
  | 'normal'
  | 'high'
  | 'urgent'
  | 'rush'
  | 'wedding_party'
  | 'prom_group'
  | 'vip_customer';
```

Order Item Data Structure

```
interface OrderItem {
  id: string;
  order_id: string;
  product_source: 'core_stripe' | 'catalog_supabase';

  // Product identification
  stripe_product_id?: string;
  stripe_price_id?: string;
  catalog_product_id?: string;
  product_name: string;
  product_sku?: string;
  product_description?: string;

  // Product details
  size?: string;
  color?: string;
  material?: string;
  custom_measurements?: any;

  // Pricing
  quantity: number;
  unit_price: number;
  total_price: number;

  // Bundle information
  is_bundle_item: boolean;
  bundle_parent_id?: string;
  bundle_type?: string;

  // Status and notes
  item_status: OrderStatus;
  production_notes?: string;
  quality_check_notes?: string;

  // Timestamps
```

```
  created_at: string;
  updated_at: string;
}
```

Cart Item Structure (Frontend to Backend)

```
interface CartItem {
  // Required fields
  product_id: string;
  product_name: string;
  quantity: number;
  price: number;

  // Product type identification
  product_source: 'core_stripe' | 'catalog_supabase';
  stripe_product_id?: string; // For Core Products
  catalog_product_id?: string; // For Catalog Products

  // Product variants
  size?: string;
  color?: string;
  material?: string;
  sku?: string;

  // Bundle information
  is_bundle_item?: boolean;
  bundle_type?: string;

  // Custom options
  custom_measurements?: {
    chest?: number;
    waist?: number;
    sleeve?: number;
    length?: number;
    [key: string]: any;
  };
}
```

Webhook Configuration

Required Webhooks

1. Stripe Webhooks

Endpoint: `https://your-supabase-project.supabase.co/functions/v1/stripe-webhook`

Required Events:
- `payment_intent.succeeded`
- `payment_intent.payment_failed`
- `charge.succeeded`
- `charge.failed`

2. EasyPost Webhooks

Endpoint: `https://your-supabase-project.supabase.co/functions/v1/easypost-webhook`

Required Events:
- `tracker.created`
- `tracker.updated`
- `shipment.purchased`
- `shipment.delivered`

Webhook Setup Instructions

Stripe Webhook Setup

1. Go to Stripe Dashboard → Developers → Webhooks
2. Click "Add endpoint"
3. Set URL: `https://your-supabase-project.supabase.co/functions/v1/stripe-webhook`
4. Select events listed above

5. Save and copy the webhook secret

EasyPost Webhook Setup

1. Go to EasyPost Dashboard → Account → Webhooks
2. Click "Add Webhook"
3. Set URL: `https://your-supabase-project.supabase.co/functions/v1/easypost-webhook`
4. Select events listed above
5. Save configuration

Error Handling

Standard Error Response Format

```
interface APIError {
  error: {
    code: string;
    message: string;
    timestamp?: string;
  };
}
```

Common Error Codes

- `ORDER_CREATION_FAILED` - Order creation failed
- `PAYMENT_INTENT_FAILED` - Stripe payment intent creation failed
- `SHIPPING_RATES_FAILED` - Shipping rate calculation failed
- `EMAIL_SEND_FAILED` - Email sending failed
- `INVALID_REQUEST` - Request validation failed

- `AUTHENTICATION_FAILED` - Invalid API key or auth token

Error Handling Implementation

```
const handleAPIResponse = async (response) => {
  const data = await response.json();

  if (!response.ok) {
    throw new Error(data.error?.message || 'API request failed');
  }

  if (data.error) {
    throw new Error(data.error.message);
  }

  return data;
};

// Usage example
try {
  const result = await createOrder(orderData);
  console.log('Order created:', result.data);
} catch (error) {
  console.error('Order creation failed:', error.message);
  // Handle error appropriately
}
```

Testing & Implementation Examples

Complete Integration Example

```
class KCTOrderProcessor {
  constructor(config) {
    this.supabaseUrl = config.supabaseUrl;
    this.serviceRoleKey = config.serviceRoleKey;
    this.anonKey = config.anonKey;
  }

  async processOrder(cartData) {
    try {
      // Determine order type
      const hasStripeProducts = cartData.items.some(item =>
        item.product_source === 'core_stripe' ||
item.stripe_product_id
      );

      const hasCatalogProducts = cartData.items.some(item =>
        item.product_source === 'catalog_supabase' ||
item.catalog_product_id
      );

      if (hasStripeProducts && hasCatalogProducts) {
        // Mixed cart - process separately
        return await this.processMixedCart(cartData);
      } else if (hasStripeProducts) {
        // Core products only - use Stripe
        return await this.processStripeOrder(cartData);
      } else {
        // Catalog products only - direct order
        return await this.processCatalogOrder(cartData);
      }

    } catch (error) {
      console.error('Order processing failed:', error);
      return {
        success: false,
```

```
        error: error.message
      };
    }
  }

  async processStripeOrder(cartData) {
    // Create payment intent
    const paymentResponse = await this.createPaymentIntent({
      amount: cartData.total,
      cartItems: cartData.items,
      customerEmail: cartData.customer.email,
      shippingAddress: cartData.shippingAddress,
      rushOrder: cartData.isRush
    });

    if (paymentResponse.error) {
      throw new Error(paymentResponse.error.message);
    }

    // Return client secret for frontend payment processing
    return {
      success: true,
      paymentIntent: paymentResponse.data,
      requiresPayment: true
    };
  }

  async processCatalogOrder(cartData) {
    // Create order directly
    const orderResponse = await this.createOrder({
      order_id: this.generateOrderNumber(),
      customer_email: cartData.customer.email,
      customer_name: cartData.customer.name,
      total_amount: cartData.total,
      items: cartData.items,
```

```
      shipping_address: cartData.shippingAddress
    });

    if (orderResponse.error) {
      throw new Error(orderResponse.error.message);
    }

    // Send confirmation email
    await this.sendOrderConfirmation(orderResponse.data);

    return {
      success: true,
      order: orderResponse.data,
      requiresPayment: false
    };
  }

  async processMixedCart(cartData) {
    // Split cart and process each type
    const stripeItems = cartData.items.filter(item =>
      item.product_source === 'core_stripe'
    );
    const catalogItems = cartData.items.filter(item =>
      item.product_source === 'catalog_supabase'
    );

    const results = [];

    if (stripeItems.length > 0) {
      const stripeResult = await this.processStripeOrder({
        ...cartData,
        items: stripeItems,
        total: this.calculateTotal(stripeItems)
      });
      results.push({ type: 'stripe', ...stripeResult });
```

```
    }

    if (catalogItems.length > 0) {
      const catalogResult = await this.processCatalogOrder({
        ...cartData,
        items: catalogItems,
        total: this.calculateTotal(catalogItems)
      });
      results.push({ type: 'catalog', ...catalogResult });
    }

    return {
      success: results.every(r => r.success),
      orders: results
    };
  }

  // Helper methods
  generateOrderNumber() {
    return `KCT-<span class="math-inline" style="display:
inline;"><math xmlns="http://www.w3.org/1998/Math/MathML"
display="inline"><mrow><mrow><mi>D</mi><mi>a</mi><mi>t</mi><mi>e</
mi><mo>&#x0002E;</mo><mi>n</mi><mi>o</mi><mi>w</mi><mo
stretchy="false">&#x00028;</mo><mo stretchy="false">&#x00029;</
mo></mrow><mo>&#x02212;</mo></mrow></math></
span>{Math.random().toString(36).substr(2, 4).toUpperCase()}`;
  }

  calculateTotal(items) {
    return items.reduce((sum, item) => sum + (item.price *
item.quantity), 0);
  }

  async makeAPICall(endpoint, data) {
    const response = await fetch(`<span class="math-inline"
```

```
style="display: inline;"><math xmlns="http://www.w3.org/1998/Math/
MathML" display="inline"><mrow><mrow><mi>t</mi><mi>h</mi><mi>i</
mi><mi>s</mi><mo>&#x0002E;</mo><mi>s</mi><mi>u</mi><mi>p</
mi><mi>a</mi><mi>b</mi><mi>a</mi><mi>s</mi><mi>e</mi><mi>U</
mi><mi>r</mi><mi>l</mi></mrow><mo>&#x0002F;</mo><mi>f</mi><mi>u</
mi><mi>n</mi><mi>c</mi><mi>t</mi><mi>i</mi><mi>o</mi><mi>n</
mi><mi>s</mi><mo>&#x0002F;</mo><mi>v</mi><mn>1</mn><mo>&#x0002F;</
mo></mrow></math></span>{endpoint}`, {
      method: 'POST',
      headers: {
        'Authorization': `Bearer ${this.serviceRoleKey}`,
        'apikey': this.serviceRoleKey,
        'Content-Type': 'application/json'
      },
      body: JSON.stringify(data)
    });

    return await handleAPIResponse(response);
  }
}

// Usage
const orderProcessor = new KCTOrderProcessor({
  supabaseUrl: process.env.SUPABASE_URL,
  serviceRoleKey: process.env.SUPABASE_SERVICE_ROLE_KEY,
  anonKey: process.env.SUPABASE_ANON_KEY
});

// Process an order
const result = await orderProcessor.processOrder({
  customer: {
    email: 'customer@example.com',
    name: 'John Doe',
    phone: '+1234567890'
  },
```

```
  items: [
    {
      product_id: 'suit-001',
      product_name: 'Classic Navy Suit',
      product_source: 'core_stripe',
      stripe_product_id: 'prod_stripe123',
      quantity: 1,
      price: 599.99,
      size: '42R'
    }
  ],
  total: 599.99,
  shippingAddress: {
    street1: '123 Main St',
    city: 'New York',
    state: 'NY',
    zip: '10001',
    country: 'US'
  },
  isRush: false
});
```

Frontend Integration Example (React)

```
import { useState } from 'react';
import { loadStripe } from '@stripe/stripe-js';
import { Elements, CardElement, useStripe, useElements } from
'@stripe/react-stripe-js';

const stripePromise =
loadStripe(process.env.REACT_APP_STRIPE_PUBLISHABLE_KEY);

const CheckoutForm = ({ cartData, onSuccess, onError }) => {
  const stripe = useStripe();
  const elements = useElements();
  const [processing, setProcessing] = useState(false);

  const handleSubmit = async (event) => {
    event.preventDefault();

    if (!stripe || !elements) {
      return;
    }

    setProcessing(true);

    try {
      // Create payment intent
      const response = await fetch('/api/create-order', {
        method: 'POST',
        headers: {
          'Content-Type': 'application/json'
        },
        body: JSON.stringify(cartData)
      });

      const { clientSecret, orderId } = await response.json();

      // Confirm payment
```

```
      const result = await stripe.confirmCardPayment(clientSecret,
{
        payment_method: {
          card: elements.getElement(CardElement),
          billing_details: {
            name: cartData.customer.name,
            email: cartData.customer.email
          }
        }
      });

      if (result.error) {
        onError(result.error.message);
      } else {
        // Payment succeeded
        onSuccess({ orderId, paymentIntent:
result.paymentIntent });
      }

    } catch (error) {
      onError(error.message);
    } finally {
      setProcessing(false);
    }
  };

  return (
    <form onSubmit={handleSubmit}>
      <CardElement />
      <button type="submit" disabled={!stripe || processing}>
        {processing ? 'Processing...' : `Pay $${cartData.total}`}
      </button>
    </form>
  );
};
```

```
const Checkout = ({ cartData }) => {
  const handleSuccess = (result) => {
    console.log('Payment successful:', result);
    // Redirect to success page or show confirmation
  };

  const handleError = (error) => {
    console.error('Payment failed:', error);
    // Show error message to user
  };

  return (
    <Elements stripe={stripePromise}>
      <CheckoutForm
        cartData={cartData}
        onSuccess={handleSuccess}
        onError={handleError}
      />
    </Elements>
  );
};
```

Security Best Practices

API Key Security

1. **Never expose service role keys on the frontend**
2. **Use anonymous keys for client-side operations**
3. **Store sensitive keys in environment variables**
4. **Implement rate limiting on your API endpoints**

Data Validation

```
const validateOrderData = (orderData) => {
  const errors = [];

  if (!orderData.customer?.email) {
    errors.push('Customer email is required');
  }

  if (!orderData.items || orderData.items.length === 0) {
    errors.push('Order must contain at least one item');
  }

  if (!orderData.total || orderData.total <= 0) {
    errors.push('Order total must be greater than 0');
  }

  orderData.items.forEach((item, index) => {
    if (!item.product_id) {
      errors.push(`Item ${index + 1}: Product ID is required`);
    }
    if (!item.quantity || item.quantity <= 0) {
      errors.push(`Item ${index +
1}: Quantity must be greater than 0`);
    }
    if (!item.price || item.price <= 0) {
      errors.push(`Item ${index + 1}: Price must be greater than
0`);
    }
  });

  return errors;
};
```

CORS Configuration

All edge functions include proper CORS headers:

```
const corsHeaders = {
  'Access-Control-Allow-Origin': '*',
  'Access-Control-Allow-Headers': 'authorization, x-client-info,
apikey, content-type',
  'Access-Control-Allow-Methods':
'POST, GET, OPTIONS, PUT, DELETE, PATCH',
  'Access-Control-Max-Age': '86400',
  'Access-Control-Allow-Credentials': 'false'
};
```

Production Deployment Checklist

Environment Variables Setup

- [ ] SUPABASE_URL configured
- [ ] SUPABASE_SERVICE_ROLE_KEY configured
- [ ] STRIPE_SECRET_KEY configured
- [ ] EASYPOST_API_KEY configured
- [ ] SENDGRID_API_KEY configured
- [ ] ADMIN_EMAIL configured
- [ ] FROM_EMAIL configured

Webhook Configuration

- [ ] Stripe webhooks configured with correct endpoint
- [ ] EasyPost webhooks configured with correct endpoint
- [ ] Webhook secrets stored securely

- [ ] Webhook endpoints tested

Testing Requirements

- [ ] Order creation tested for Core Products
- [ ] Order creation tested for Catalog Products
- [ ] Mixed cart processing tested
- [ ] Shipping rate calculation tested
- [ ] Email sending tested
- [ ] Error handling tested

Security Verification

- [ ] API keys secured and not exposed
- [ ] RLS policies verified
- [ ] Input validation implemented
- [ ] CORS properly configured

Support & Resources

Documentation Links

- **Supabase Documentation:** https://supabase.com/docs
- **Stripe API Reference:** https://stripe.com/docs/api
- **EasyPost API Reference:** https://www.easypost.com/docs
- **SendGrid API Reference:** https://docs.sendgrid.com/

Backend Admin Dashboard

- **URL:** https://rtbbsdcrfbha.space.minimax.io

- **Login:** Use admin credentials provided separately

Contact Information

For technical support or questions about this integration:
- **Admin Email:** KCTMenswear@gmail.com
- **Backend URL:** https://rtbbsdcrfbha.space.minimax.io

This documentation provides complete integration guidelines for connecting your KCT Menswear website frontend to the backend admin system. Follow the implementation examples and ensure all security best practices are implemented before going live.