```
import Stripe from 'stripe';
import { stripe } from '../stripe/config';
import { Tables } from '@/types_db';
import { supabaseAdmin } from '../supabase/admin';

async function findUnpaidInvoice(
  transaction: Tables<'swarm_cloud_billing_transcations'>,
) {
  if (!transaction.payment_successful) {
    const invoice = await stripe.invoices.retrieve(
      transaction.invoice_id as string,
    );
    if (invoice.status === 'open') {
      return invoice;
    }
  }
}

async function attemptAutomaticCharge(invoiceId: string) {
  try {
    const invoice = await stripe.invoices.retrieve(invoiceId);

    // Check if invoice is still open (unpaid)
    if (invoice.status !== 'open') {
      console.log('Invoice already paid:', invoiceId);
      return true;
```

```
}

const customerId = invoice?.customer as string;

const customer = (await stripe.customers.retrieve(
  customerId,
)) as Stripe.Customer;

if (!customer || !customer.invoice_settings.default_payment_method) {
  console.error('No default payment method found for invoice:', invoiceId);
  return false;
}

// Attempt charge with the default payment method
const paymentResult = await stripe.invoices.pay(invoiceId);

if (paymentResult.paid) {
  console.log('Invoice successfully charged automatically:', invoiceId);
  const { error: updateError } = await supabaseAdmin
    .from('swarm_cloud_billing_transcations')
    .update({ payment_successful: true })
    .eq('invoice_id', invoiceId);

  if (updateError) {
    console.error('Error updating billing transaction:', updateError);
  }
```

```
      return true;
    } else {
      console.error(
        'Failed to charge invoice automatically:',
        invoiceId,
        paymentResult.last_finalization_error?.message,
      );

      const sendInvoiceResult = await stripe.invoices.sendInvoice(invoiceId);

      if (sendInvoiceResult.last_finalization_error) {
        console.error(
          'Error sending invoice for manual payment:',
          sendInvoiceResult.last_finalization_error?.message,
        );
      }
      return false;
    }
  } catch (error) {
    console.error('Error attempting automatic charge:', error);
    return false;
  }
}

async function getLatestBillingTransaction(userId: string): Promise<{
```

```
  status: number;
  message: string;
  transaction: Tables<'swarm_cloud_billing_transcations'> | null;
}> {
  if (!userId) {
    return {
      status: 400,
      message: 'User session not found',
      transaction: null,
    };
  }

  const { data: latestTransaction, error } = await supabaseAdmin
    .from('swarm_cloud_billing_transcations')
    .select('*')
    .eq('user_id', userId)
    .order('created_at', { ascending: false })
    .limit(1)
    .single();

  if (error) {
    if (error.code === 'PGRST116') {
      return {
        status: 200,
        message: 'No billing transactions found',
        transaction: null,
```

```
      };
    }

    console.error('Error fetching latest billing transaction:', error);
    return {
      status: 400,
      message: 'Error fetching latest billing transaction',
      transaction: null,
    };
  }

  return {
    status: 200,
    message: 'Success',
    transaction: latestTransaction ?? null,
  };
}

export {
  findUnpaidInvoice,
  attemptAutomaticCharge,
  getLatestBillingTransaction,
};
```