Stripe Connect Integration (Phase 1 - Basis)

📋 Übersicht

Für die Multi-Tenant-Plattform benötigen wir Stripe Connect, damit jeder Coach seine eigenen Zahlungen erhält.

🎯 Status: Vorbereitet, noch nicht implementiert

Das Datenbank-Schema ist vorbereitet mit:
- `coaches.stripe_account_id` - Stripe Connect Account ID
- `coaches.stripe_onboarding_complete` - Status des Onboardings
- `subscriptions.stripe_subscription_id` - Stripe Subscription ID
- `subscriptions.stripe_customer_id` - Stripe Customer ID

🔧 Nächste Schritte für Stripe Connect

1. Backend-Funktion erstellen (Supabase Edge Function oder externes API)

```
// stripe-connect-onboarding.js
// Erstellt einen Stripe Connect Account für den Coach

async function createConnectAccount(coachId) {
  const account = await stripe.accounts.create({
    type: 'standard',
    country: 'DE',
    email: coach.email,
    business_type: 'individual',
  });

  // Speichere account.id in coaches.stripe_account_id

  // Erstelle Onboarding-Link
  const accountLink = await stripe.accountLinks.create({
    account: account.id,
    refresh_url: 'https://your-domain.com/coach-dashboard.html',
    return_url: 'https://your-domain.com/coach-dashboard.html?stripe=success',
    type: 'account_onboarding',
  });

  return accountLink.url;
}
```

2. Dashboard-Integration

Im Coach-Dashboard unter “Einstellungen”:

```
// Wenn Coach auf "Mit Stripe verbinden" klickt
document.getElementById('connectStripeBtn').onclick = async () => {
  try {
    // Call backend function
    const response = await fetch('/api/create-connect-account', {
      method: 'POST',
      headers: { 'Content-Type': 'application/json' },
      body: JSON.stringify({ coachId: currentCoach.id })
    });

    const { url } = await response.json();

    // Redirect to Stripe Onboarding
    window.location.href = url;

  } catch (error) {
    console.error('Stripe Connect error:', error);
  }
};
```

3. Checkout-Session für Coach-Pakete

```
// stripe-create-checkout.js
// Erstellt eine Checkout-Session für ein Coach-Paket

async function createCheckoutSession(packageId, coachId, customerId) {
  const pkg = await getPackage(packageId);
  const coach = await getCoach(coachId);

  // Erstelle oder hole Stripe Price
  let stripePriceId = pkg.stripe_price_id;

  if (!stripePriceId) {
    // Erstelle neues Stripe Product & Price
    const product = await stripe.products.create({
      name: pkg.name,
      description: pkg.description,
    }, {
      stripeAccount: coach.stripe_account_id
    });

    const price = await stripe.prices.create({
      product: product.id,
      unit_amount: pkg.price * 100, // in Cents
      currency: 'eur',
      recurring: {
        interval: pkg.billing_interval
      },
    }, {
      stripeAccount: coach.stripe_account_id
    });

    stripePriceId = price.id;

    // Update package
    await updatePackage(packageId, {
      stripe_price_id: price.id,
      stripe_product_id: product.id
    });
  }

  // Erstelle Checkout Session
  const session = await stripe.checkout.sessions.create({
    mode: 'subscription',
    customer: customerId,
    line_items: [{
      price: stripePriceId,
      quantity: 1,
    }],
    success_url: 'https://your-domain.com/?coach=' + coach.slug + '&payment=success',
    cancel_url: 'https://your-domain.com/?coach=' + coach.slug + '&payment=canceled',
    payment_intent_data: {
      application_fee_amount: calculatePlatformFee(pkg.price), // z.B. 10%
    },
  }, {
    stripeAccount: coach.stripe_account_id
  });

  return session.url;
}

function calculatePlatformFee(price) {
  // Plattform-Gebühr: z.B. 10% oder fester Betrag
  const feePercentage = 0.10; // 10%
```

```
  return Math.round(price * 100 * feePercentage);
}
```

4. Webhook-Handling

```
// stripe-webhooks.js
// Handle Stripe Webhooks

app.post('/webhook/stripe', async (req, res) => {
  const sig = req.headers['stripe-signature'];
  let event;

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

  switch (event.type) {
    case 'checkout.session.completed':
      await handleCheckoutComplete(event.data.object);
      break;

    case 'customer.subscription.created':
      await handleSubscriptionCreated(event.data.object);
      break;

    case 'customer.subscription.updated':
      await handleSubscriptionUpdated(event.data.object);
      break;

    case 'customer.subscription.deleted':
      await handleSubscriptionCanceled(event.data.object);
      break;
  }

  res.json({ received: true });
});

async function handleSubscriptionCreated(subscription) {
  // Update subscriptions table
  await supabase
    .from('subscriptions')
    .update({
      stripe_subscription_id: subscription.id,
      stripe_customer_id: subscription.customer,
      status: 'active',
      current_period_start: new Date(subscription.current_period_start * 1000),
      current_period_end: new Date(subscription.current_period_end * 1000),
    })
    .eq('stripe_subscription_id', subscription.id);
}
```

🏗️ Backend-Architektur Optionen

Option 1: Supabase Edge Functions

```
supabase functions new stripe-connect-onboarding
supabase functions new stripe-create-checkout
supabase functions new stripe-webhooks
```

Vorteile:
- Nativ in Supabase integriert
- Kein separater Server nötig
- Automatisches Deployment

Option 2: Separates Backend (Node.js/Express)

```
npm install express stripe
```

Vorteile:
- Mehr Kontrolle
- Einfacheres Debugging
- Kann auf beliebigem Server gehostet werden

Option 3: Netlify Functions

```
# netlify/functions/stripe-*.js
```

Vorteile:
- Gleicher Deployment-Flow wie Frontend
- Keine zusätzliche Infrastruktur

💰 Platform Fee Struktur

Verschiedene Modelle möglich:

1. Prozentual

```
const PLATFORM_FEE_PERCENT = 0.10; // 10%
const fee = price * PLATFORM_FEE_PERCENT;
```

2. Fester Betrag

```
const PLATFORM_FEE_FIXED = 2.00; // €2 pro Transaktion
```

3. Hybrid

```
const fee = Math.max(
  price * 0.10,  // Mindestens 10%
  2.00           // Aber mindestens €2
);
```

📊 Testing

1. Stripe Test Mode

- Verwende Stripe Test Keys
- Teste mit Test-Karten: https://stripe.com/docs/testing

2. Test-Workflow

1. Coach registriert sich
2. Coach verbindet Stripe (Test-Account)
3. Erstellt Paket
4. Testuser abonniert Paket
5. Prüfe Subscription in Stripe Dashboard
6. Prüfe Subscription in Supabase

🔐 Sicherheit

API Keys schützen

```
// Nie im Frontend!
const STRIPE_SECRET_KEY = process.env.STRIPE_SECRET_KEY;

// Nur im Backend verwenden
const stripe = require('stripe')(STRIPE_SECRET_KEY);
```

Webhook-Signatur verifizieren

```
const signature = req.headers['stripe-signature'];
const event = stripe.webhooks.constructEvent(
  req.body,
  signature,
  WEBHOOK_SECRET
);
```

📝 Nächste Implementierungs-Schritte

1. **Wähle Backend-Architektur** (Empfehlung: Netlify Functions)
2. **Erstelle Stripe Connect Account**
3. **Implementiere Onboarding-Flow**
4. **Erstelle Checkout-Session-Handler**
5. **Implementiere Webhooks**
6. **Testing mit Test-Accounts**
7. **Go Live mit echten Stripe-Keys**

🚀 Quick Start (Empfohlen)

1. Erstelle Stripe-Account für die Plattform
2. Aktiviere Stripe Connect

3. Erstelle Netlify Functions:
   ```
   netlify/functions/
   ├── stripe-connect-onboarding.js
   ├── stripe-create-checkout.js
   └── stripe-webhooks.js
   ```
4. Füge Environment Variables hinzu (Netlify)
5. Deploy & Test

📚 Ressourcen

- Stripe Connect Docs (https://stripe.com/docs/connect)
- Stripe Connect Onboarding (https://stripe.com/docs/connect/collect-then-transfer-guide)
- Stripe Checkout Sessions (https://stripe.com/docs/checkout/quickstart)
- Stripe Webhooks (https://stripe.com/docs/webhooks)

Status: Dokumentation & Schema vorbereitet
Nächster Schritt: Backend-Funktionen implementieren