Nightlife OS - Architecture Addendum: Payments, Merch & Ticketing

Erweiterungen für Monetarisierung: Spreadshirt-Integration, Stripe Connect & Event-Kalender

Version: 1.0
Erstellt am: 3. Dezember 2025
Basis: ARCHITECTURE.md (Phase 1-8), ARCHITECTURE_ADDENDUM_INTERACTIVE_STATUS.md
Zweck: Monetarisierung über Plattform-Provision (Ticketing) & Merch-Umsatz (Spreadshirt)

Inhaltsverzeichnis

1. ÜBERBLICK & MOTIVATION
2. USER-CODE & QR – BASIS FÜR MERCH
   - 2.1 Datenmodell-Erweiterungen
   - 2.2 Personal Code-Generierung
   - 2.3 QR-Code-Generierung & Storage
   - 2.4 Funktionen & Interfaces
   - 2.5 Firestore-Schema
3. MERCH / SPREADSHIRT – KOMPLETT ÜBER SUPER ADMIN
   - 3.1 Globales Merch-Config (Super Admin)
   - 3.2 Club-spezifische Merch-Settings
   - 3.3 Spreadshirt-Anbindung
   - 3.4 Datenmodell Merch-Bestellungen
   - 3.5 UI – Visitor-App (Merch-Tab)
   - 3.6 Firestore-Schema
4. TICKETING – STRIPE CONNECT + PLATTFORM-PROVISION
   - 4.1 Globales Payment-Config
   - 4.2 ClubPaymentSettings mit Stripe-Connect
   - 4.3 Onboarding-Flow für Clubs (Stripe Connect)
   - 4.4 Datenmodell Events & Tickets
   - 4.5 Ticket-Kauf-Flow mit Provision
   - 4.6 Provision-Transparenz & Reporting
   - 4.7 Firestore-Schema
5. EVENT-KALENDER – NUR EVENTS VON BESUCHTEN CLUBS
   - 5.1 Datenmodell-Erweiterung User
   - 5.2 Event-Query-Logik
   - 5.3 UI – Events-Tab
   - 5.4 Firestore-Schema
6. SICHERHEIT, ROLLEN & FIRESTORE RULES
   - 6.1 Rollenrechte
   - 6.2 Firestore Security Rules
   - 6.3 DSGVO-Compliance

7. ZUSAMMENFASSUNG & DATEIEN
  - 7.1 Neue Strukturen (Übersicht)
  - 7.2 Betroffene Dateien
  - 7.3 Code-Snippets

---

1. ÜBERBLICK & MOTIVATION

Dieses Addendum erweitert die bestehende Nightlife OS-Architektur um **drei zentrale Monetarisierungs-Funktionen**:

Warum diese Erweiterungen?

Monetarisierung 1: Merch (100% Super Admin)

Geschäftsmodell: Spreadshirt-Shop komplett über Super Admin-Konto → 100% Umsatz geht an Plattform.

Mehrwert für Clubs:
- Clubs können Merch-Bereich aktivieren und ihr Logo einbinden
- Kostenlose Marketing-Möglichkeit für Club-Brand
- Keine Kosten, keine Verwaltung, kein Risiko

Mehrwert für User:
- Personalisierter Merch mit eigenem User-Code + QR
- Nahtlose Integration in die App
- Club-Merch von allen besuchten Clubs

Monetarisierung 2: Ticketing (Provision pro Ticket)

Geschäftsmodell: Stripe Connect → Clubs nutzen eigenes Stripe-Konto → Plattform erhält automatische Provision (7% Standard).

Mehrwert für Clubs:
- Eigenes Stripe-Konto = volle Kontrolle
- Automatische Auszahlung durch Stripe
- Keine manuelle Abrechnung
- Professionelles Ticketing-System integriert

Mehrwert für Plattform:
- Automatische Provision über `application_fee_amount`
- Keine manuelle Abrechnung nötig
- Transparente Umsatz-Reports im Super-Admin-Dashboard

Feature 3: Event-Kalender (Privacy-First)

Ansatz: User sehen nur Events von Clubs, in denen sie bereits waren (Check-In-Historie).

Mehrwert:
- Keine Spam-Events von unbekannten Clubs
- Relevante Event-Vorschläge basierend auf tatsächlichem Verhalten
- Privacy-First: Kein Tracking von fremden Clubs

Architektur-Prinzipien

1. **Super Admin hat volle Kontrolle über Merch-Umsätze**
2. **Clubs haben volle Kontrolle über Ticket-Umsätze (minus Provision)**
3. **Stripe Connect automatisiert die Provisionsverteilung**
4. **Multi-Tenancy bleibt gewahrt** (Clubs können sich nicht gegenseitig in die Karten schauen)
5. **DSGVO-konform:** Zahlungsdaten bei Drittanbietern (Spreadshirt, Stripe)

2. USER-CODE & QR – BASIS FÜR MERCH

Konzept

Jeder User erhält bei Registrierung einen **eindeutigen Personal Code** (z.B. `CFX8R3Y`), der:
- **Plattformweit eindeutig** ist (nicht nur pro Club)
- **Stabil** ist (ändert sich nie, außer bei Account-Löschung)
- Als **QR-Code** visualisiert wird
- Für **Merch-Personalisierung**, Friend-Requests, Loyalty-Programme genutzt wird

2.1 Datenmodell-Erweiterungen

PlatformUser (plattformweit)

Erweitere `platform/users/{uid}`:

```
// packages/shared-types/src/user.ts

export interface PlatformUser {
  // Bestehende Felder
  uid: string;
  email: string;
  displayName: string | null;
  photoURL: string | null;
  createdAt: number;
  lastSeenAt: number;
  isPlatformAdmin: boolean;
  ownedClubs: string[];
  memberClubs: string[];

  // NEU: Personal Code & QR
  personalCode: string;          // z.B. "CFX8R3Y" (6-8 Zeichen)
  qrCodeUrl: string;              // URL zum QR-Bild in Firebase Storage
  personalCodeCreatedAt: number;  // Timestamp der Code-Erstellung
}
```

Wichtig:
- `personalCode` wird **einmalig** bei Registrierung/erstem Login generiert

- `qrCodeUrl` wird **einmalig** erstellt und in Firebase Storage gespeichert
- Code ist **plattformweit eindeutig** (nicht club-spezifisch)

2.2 Personal Code-Generierung

Helper-Funktion

Datei: `packages/core/src/user/generatePersonalCode.ts`

```
/**
 * Generiert einen eindeutigen Personal Code für User
 * Format: 6-8 Zeichen, A-Z + 0-9, keine Verwechslungen (O vs 0, I vs 1)
 */

const SAFE_CHARS = 'ABCDEFGHJKLMNPQRSTUVWXYZ23456789'; // Ohne O, I, 0, 1

export function generatePersonalCode(length: number = 7): string {
  let code = '';
  for (let i = 0; i < length; i++) {
    const randomIndex = Math.floor(Math.random() * SAFE_CHARS.length);
    code += SAFE_CHARS[randomIndex];
  }
  return code;
}

/**
 * Prüft ob Personal Code bereits existiert (Kollisionsprüfung)
 */
export async function isPersonalCodeUnique(code: string): Promise<boolean> {
  const db = getFirestore();
  const usersQuery = query(
    collection(db, 'platform/users'),
    where('personalCode', '==', code)
  );
  const snapshot = await getDocs(usersQuery);
  return snapshot.empty;
}

/**
 * Generiert einen eindeutigen Personal Code mit Kollisionsprüfung
 * Max. 10 Versuche, danach Fehler
 */
export async function generateUniquePersonalCode(): Promise<string> {
  let attempts = 0;
  const maxAttempts = 10;

  while (attempts < maxAttempts) {
    const code = generatePersonalCode();
    const isUnique = await isPersonalCodeUnique(code);

    if (isUnique) {
      return code;
    }

    attempts++;
  }

  throw new Error('Failed to generate unique personal code after 10 attempts');
}
```

2.3 QR-Code-Generierung & Storage

Cloud Function / Backend-Service

Datei: `packages/core/src/user/generateQRCode.ts`

```
import QRCode from 'qrcode';
import { getStorage, ref, uploadString, getDownloadURL } from 'firebase/storage';

/**
 * Generiert QR-Code als Base64-Bild und speichert es in Firebase Storage
 * @param userId - Firebase UID
 * @param personalCode - Personal Code (z.B. "CFX8R3Y")
 * @returns Download-URL des QR-Codes
 */
export async function generateAndUploadQRCode(
  userId: string,
  personalCode: string
): Promise<string> {
  try {
    // QR-Code als Data-URL generieren
    const qrDataUrl = await QRCode.toDataURL(personalCode, {
      errorCorrectionLevel: 'H',
      type: 'image/png',
      width: 512,
      margin: 2,
      color: {
        dark: '#000000',
        light: '#FFFFFF'
      }
    });

    // Zu Firebase Storage hochladen
    const storage = getStorage();
    const qrRef = ref(storage, `users/${userId}/qr-code.png`);

    // Base64 zu Blob konvertieren
    const base64Data = qrDataUrl.split(',')[1];
    await uploadString(qrRef, base64Data, 'base64', {
      contentType: 'image/png',
      cacheControl: 'public, max-age=31536000' // 1 Jahr Cache
    });

    // Download-URL abrufen
    const downloadUrl = await getDownloadURL(qrRef);
    return downloadUrl;

  } catch (error) {
    console.error('Error generating QR code:', error);
    throw new Error('Failed to generate QR code');
  }
}
```

2.4 Funktionen & Interfaces

User-Registrierung mit Personal Code

Datei: `packages/core/src/user/userService.ts`

```
import { doc, setDoc, getDoc, serverTimestamp } from 'firebase/firestore';
import { generateUniquePersonalCode } from './generatePersonalCode';
import { generateAndUploadQRCode } from './generateQRCode';

export interface CreateUserResult {
  success: boolean;
  user?: PlatformUser;
  error?: string;
}

/**
 * Erstellt oder aktualisiert User-Dokument mit Personal Code
 * Wird beim ersten Login aufgerufen
 */
export async function ensureUserHasPersonalCode(
  userId: string,
  email: string,
  displayName?: string | null
): Promise<CreateUserResult> {
  try {
    const db = getFirestore();
    const userRef = doc(db, 'platform/users', userId);
    const userSnap = await getDoc(userRef);

    // Wenn User bereits Personal Code hat, nichts tun
    if (userSnap.exists() && userSnap.data().personalCode) {
      return {
        success: true,
        user: userSnap.data() as PlatformUser
      };
    }

    // Personal Code generieren
    const personalCode = await generateUniquePersonalCode();

    // QR-Code generieren & hochladen
    const qrCodeUrl = await generateAndUploadQRCode(userId, personalCode);

    // User-Dokument erstellen/updaten
    const userData: Partial<PlatformUser> = {
      personalCode,
      qrCodeUrl,
      personalCodeCreatedAt: Date.now(),
      email,
      displayName: displayName || null,
      lastSeenAt: Date.now()
    };

    if (!userSnap.exists()) {
      // Neuer User
      userData.uid = userId;
      userData.createdAt = Date.now();
      userData.isPlatformAdmin = false;
      userData.ownedClubs = [];
      userData.memberClubs = [];
    }

    await setDoc(userRef, userData, { merge: true });

    return {
      success: true,
      user: { ...userSnap.data(), ...userData } as PlatformUser
    };

  } catch (error) {
    console.error('Error ensuring user has personal code:', error);
    return {
      success: false,
      error: error instanceof Error ? error.message : 'Unknown error'
    };
  }
}
```

2.5 Firestore-Schema

Collection: `platform/users/{uid}`

```
platform/
└── users/
    └── {uid}/
        ├── uid: string
        ├── email: string
        ├── displayName: string | null
        ├── photoURL: string | null
        ├── createdAt: number
        ├── lastSeenAt: number
        ├── isPlatformAdmin: boolean
        ├── ownedClubs: string[]
        ├── memberClubs: string[]
        ├── personalCode: string              // NEU: "CFX8R3Y"
        ├── qrCodeUrl: string                 // NEU: Firebase Storage URL
        └── personalCodeCreatedAt: number     // NEU: Timestamp
```

Firebase Storage-Struktur

```
gs://nightlife-os.appspot.com/
└── users/
    └── {uid}/
        └── qr-code.png                       // QR-Code-Bild
```

---

3. MERCH / SPREADSHIRT – KOMPLETT ÜBER SUPER ADMIN

Konzept

- **Super Admin** besitzt **einen** Spreadshirt-Shop
- **Alle Umsätze** fließen an Super Admin (100%)
- **Clubs** können nur konfigurieren:
- Merch-Bereich aktivieren/deaktivieren
- Produkt-Kategorien auswählen (z.B. nur T-Shirts)

- Optional: Club-Logo für Designs hinterlegen
- **Clubs erhalten keinen Anteil** am Merch-Umsatz

3.1 Globales Merch-Config (Super Admin)

Datenmodell

Datei: `packages/shared-types/src/merch.ts`

```
export interface GlobalMerchSettings {
  spreadshirtShopId: string;              // Spreadshirt Shop-ID
  spreadshirtApiKey: string;              // Spreadshirt API-Key (verschlüsselt)
  spreadshirtApiSecret: string;           // Spreadshirt API-Secret (verschlüsselt)
  shopsByCountry?: Record<string, string>; // Optional: Land-spezifische Shops
  enabled: boolean;                        // Global aktiviert?
  baseProductIds: string[];                // Standard-Produkt-IDs (T-Shirt, Hoodie,
etc.)
  updatedAt: number;
}
```

Firestore-Schema

```
platform/
└── config/
    └── globalMerchSettings/                 // Fixed Document
        ├── spreadshirtShopId: string
        ├── spreadshirtApiKey: string
        ├── spreadshirtApiSecret: string
        ├── shopsByCountry: object
        ├── enabled: boolean
        ├── baseProductIds: string[]
        └── updatedAt: number
```

Security Rules

```
match /platform/config/globalMerchSettings {
  // Nur Super Admin kann lesen/schreiben
  allow read, write: if isSuperAdmin();
}
```

3.2 Club-spezifische Merch-Settings

Datenmodell

Datei: `packages/shared-types/src/club.ts`

```
export interface ClubMerchSettings {
  enabled: boolean;                          // Merch-Bereich aktiv?
  allowedProductCategories: string[];        // z.B. ["tshirt", "hoodie", "cap"]
  brandingLogoUrl?: string;                  // Optional: Club-Logo
  customMessage?: string;                    // Optional: Text für Produktseite
  updatedAt: number;
}

export interface Club {
  // Bestehende Felder...
  clubId: string;
  name: string;
  slug: string;
  // ...

  // NEU: Merch-Settings
  merchSettings?: ClubMerchSettings;
}
```

Firestore-Schema

```
platform/
└── clubs/
    └── {clubId}/
        ├── (bestehende Felder...)
        └── merchSettings:                   // NEU
            ├── enabled: boolean
            ├── allowedProductCategories: string[]
            ├── brandingLogoUrl?: string
            ├── customMessage?: string
            └── updatedAt: number
```

Security Rules

```
match /platform/clubs/{clubId} {
  allow read: if isSuperAdmin() || isClubOwner(clubId);

  // Club-Admin kann nur merchSettings updaten (nicht die Spreadshirt-Keys!)
  allow update: if isClubOwner(clubId) &&
    request.resource.data.diff(resource.data).affectedKeys().hasOnly(['merchSettings']
);
}
```

3.3 Spreadshirt-Anbindung

Service-Implementierung

Datei: `packages/core/src/merch/spreadshirtService.ts`

```
import axios, { AxiosInstance } from 'axios';

export interface MerchProduct {
  id: string;
  name: string;
  description: string;
  basePrice: number;             // Preis in Cent
  currency: string;
  category: string;              // "tshirt", "hoodie", "cap"
  imageUrl: string;
  availableSizes: string[];
  availableColors: string[];
  spreadshirtProductId: string;
}

export interface PersonalizedDesignResult {
  designId: string;
  checkoutUrl: string;
  productSnapshot: MerchProduct;
}

export class SpreadshirtService {
  private apiClient: AxiosInstance;
  private shopId: string;

  constructor(apiKey: string, apiSecret: string, shopId: string) {
    this.shopId = shopId;
    this.apiClient = axios.create({
      baseURL: 'https://api.spreadshirt.net/api/v1',
      headers: {
        'Authorization': `Bearer ${apiKey}`,
        'Content-Type': 'application/json'
      }
    });
  }

  /**
   * Lädt Produkte für Club (gefiltert nach allowedCategories)
   */
  async fetchProductsForClub(
    clubId: string,
    allowedCategories: string[],
    locale: string = 'de'
  ): Promise<MerchProduct[]> {
    try {
      const response = await this.apiClient.get(
        `/shops/${this.shopId}/products`,
        { params: { locale } }
      );

      const allProducts = response.data.products || [];

      // Filtern nach allowedCategories
      const filtered = allProducts
        .filter((p: any) => allowedCategories.includes(p.category))
        .map((p: any) => ({
          id: `${clubId}_${p.id}`,
          name: p.name,
          description: p.description,
          basePrice: Math.round(p.price.amount * 100), // Euro → Cent
          currency: p.price.currency,
          category: p.category,
          imageUrl: p.defaultImageUrl,
          availableSizes: p.sizes || [],
          availableColors: p.colors || [],
          spreadshirtProductId: p.id
        }));

      return filtered;

    } catch (error) {
      console.error('Error fetching Spreadshirt products:', error);
      throw new Error('Failed to fetch products from Spreadshirt');
    }
  }

  /**
   * Erstellt personalisiertes Design mit User-Code + QR
   */
  async createPersonalizedDesign(
    user: PlatformUser,
    club: Club,
    baseProductId: string,
    selectedSize: string,
    selectedColor: string
  ): Promise<PersonalizedDesignResult> {
    try {
      // 1. Design erstellen mit QR + Personal Code
      const designPayload = {
        name: `${user.personalCode} - ${club.name}`,
        elements: [
          {
            type: 'image',
            imageUrl: user.qrCodeUrl,
            position: { x: 100, y: 100 },
            size: { width: 150, height: 150 }
          },
          {
            type: 'text',
            text: user.personalCode,
            position: { x: 100, y: 270 },
            fontSize: 24,
            fontFamily: 'Arial Bold'
          }
        ]
      };

      if (club.merchSettings?.brandingLogoUrl) {
        // Optional: Club-Logo hinzufügen
        designPayload.elements.push({
          type: 'image',
          imageUrl: club.merchSettings.brandingLogoUrl,
          position: { x: 50, y: 50 },
          size: { width: 100, height: 100 }
        });
      }

      const designResponse = await this.apiClient.post(
        `/shops/${this.shopId}/designs`,
        designPayload
      );

      const designId = designResponse.data.id;

      // 2. Checkout-URL erstellen
```

```
      const checkoutUrl = `https://shop.spreadshirt.net/${this.shopId}` +
        `/product/${baseProductId}` +
        `?designId=${designId}` +
        `&size=${selectedSize}` +
        `&color=${selectedColor}`;

      // 3. Product-Snapshot für Bestellung
      const productSnapshot: MerchProduct = {
        id: baseProductId,
        name: designResponse.data.name,
        description: '',
        basePrice: 0, // Wird von Spreadshirt berechnet
        currency: 'EUR',
        category: 'custom',
        imageUrl: designResponse.data.previewUrl,
        availableSizes: [selectedSize],
        availableColors: [selectedColor],
        spreadshirtProductId: baseProductId
      };

      return {
        designId,
        checkoutUrl,
        productSnapshot
      };

    } catch (error) {
      console.error('Error creating personalized design:', error);
      throw new Error('Failed to create personalized design');
    }
  }

  /**
   * Webhook: Bestellung von Spreadshirt empfangen
   */
  async handleSpreadshirtWebhook(payload: any): Promise<void> {
    // Wird später implementiert für Order-Tracking
    console.log('Spreadshirt webhook received:', payload);
  }
}

// Singleton-Instanz
let spreadshirtServiceInstance: SpreadshirtService | null = null;

export async function getSpreadshirtService(): Promise<SpreadshirtService> {
  if (!spreadshirtServiceInstance) {
    // Lade globale Settings
    const db = getFirestore();
    const settingsDoc = await getDoc(
      doc(db, 'platform/config/globalMerchSettings')
    );

    if (!settingsDoc.exists()) {
      throw new Error('Spreadshirt settings not configured');
    }

    const settings = settingsDoc.data() as GlobalMerchSettings;

    spreadshirtServiceInstance = new SpreadshirtService(
      settings.spreadshirtApiKey,
      settings.spreadshirtApiSecret,
      settings.spreadshirtShopId
    );
  }

  return spreadshirtServiceInstance;
}
```

3.4 Datenmodell Merch-Bestellungen

Interface

Datei: `packages/shared-types/src/merch.ts`

```
export interface MerchOrder {
  id: string;
  userId: string;
  clubId?: string | null;                    // Aus welchem Club-Kontext bestellt
  spreadshirtOrderId: string;                // Spreadshirt-Order-ID
  createdAt: number;
  status: 'pending' | 'confirmed' | 'shipped' | 'canceled';
  productSnapshot: MerchProduct;             // Kopie zum Zeitpunkt der Bestellung
  totalPrice: number;                        // in Cent
  currency: string;
  trackingNumber?: string;
  shippedAt?: number;
}
```

Firestore-Schema

```
platform/
└── merchOrders/
    └── {orderId}/
        ├── id: string
        ├── userId: string
        ├── clubId?: string
        ├── spreadshirtOrderId: string
        ├── createdAt: number
        ├── status: string
        ├── productSnapshot: object
        ├── totalPrice: number
        ├── currency: string
        ├── trackingNumber?: string
        └── shippedAt?: number
```

Security Rules

```
match /platform/merchOrders/{orderId} {
  // User kann nur eigene Bestellungen sehen
  allow read: if request.auth.uid == resource.data.userId;

  // Nur Super Admin kann alle Bestellungen sehen
  allow read: if isSuperAdmin();

  // Bestellungen werden nur über Backend erstellt (kein direkter Client-Zugriff)
  allow write: if false;
}
```

3.5 UI – Visitor-App (Merch-Tab)

Komponenten-Struktur

```
apps/user-app/app/(tabs)/merch/
├── page.tsx                              # MerchTab (Hauptseite)
├── [productId]/
│   └── page.tsx                          # MerchProductDetail
└── components/
    ├── MerchProductCard.tsx              # Produkt-Karte
    ├── UserCodeDisplay.tsx               # Eigener Code + QR
    └── MerchOrderHistory.tsx             # Bestellhistorie
```

MerchTab (Hauptseite)

Datei: `apps/user-app/app/(tabs)/merch/page.tsx`

```
'use client';

import { useEffect, useState } from 'react';
import { useAuth } from '@nightlife/core/hooks/use-auth';
import { useClubData } from '@nightlife/core/hooks/use-club-data';
import { MerchProduct } from '@nightlife/shared-types';
import { getSpreadshirtService } from '@nightlife/core/merch/spreadshirtService';
import { MerchProductCard } from './components/MerchProductCard';
import { UserCodeDisplay } from './components/UserCodeDisplay';

export default function MerchTab() {
  const { user, userData } = useAuth();
  const { currentClub } = useClubData();
  const [products, setProducts] = useState<MerchProduct[]>([]);
  const [loading, setLoading] = useState(true);

  useEffect(() => {
    if (!currentClub || !currentClub.merchSettings?.enabled) {
      setLoading(false);
      return;
    }

    loadProducts();
  }, [currentClub]);

  const loadProducts = async () => {
    try {
      const service = await getSpreadshirtService();
      const clubProducts = await service.fetchProductsForClub(
        currentClub!.clubId,
        currentClub!.merchSettings!.allowedProductCategories,
        userData?.language || 'de'
      );
      setProducts(clubProducts);
    } catch (error) {
      console.error('Failed to load products:', error);
    } finally {
      setLoading(false);
    }
  };

  if (!currentClub?.merchSettings?.enabled) {
    return (
      <div className="flex flex-col items-center justify-center h-screen p-6">
        <h2 className="text-2xl font-bold text-white mb-4">
          Merch nicht verfügbar
        </h2>
        <p className="text-gray-400 text-center">
          Dieser Club bietet derzeit keinen Merch-Bereich an.
        </p>
      </div>
    );
  }

  return (
    <div className="min-h-screen bg-gradient-to-b from-slate-900 to-slate-950 pb-20">
      {/* Header mit eigenem Code */}
      <div className="sticky top-0 z-10 bg-slate-900/95 backdrop-blur-sm border-b
border-slate-800">
        <div className="p-4">
          <h1 className="text-2xl font-bold text-white mb-2">
            {currentClub.name} Merch
          </h1>
          <UserCodeDisplay
            personalCode={userData?.personalCode || ''}
            qrCodeUrl={userData?.qrCodeUrl || ''}
          />
        </div>
      </div>

      {/* Custom Message (optional) */}
      {currentClub.merchSettings.customMessage && (
        <div className="p-4 bg-cyan-600/20 border-l-4 border-cyan-500 mx-4 mt-4">
          <p className="text-cyan-100">{currentClub.merchSettings.customMessage}</p>
        </div>
      )}

      {/* Produkt-Liste */}
      <div className="p-4 space-y-4">
        {loading ? (
          <div className="flex justify-center py-12">
            <div className="animate-spin rounded-full h-12 w-12 border-4 border-
cyan-500 border-t-transparent" />
          </div>
        ) : (
          <div className="grid grid-cols-2 gap-4">
            {products.map((product) => (
              <MerchProductCard key={product.id} product={product} />
            ))}
          </div>
        )}
      </div>
    </div>
  );
}
```

MerchProductDetail

Datei: `apps/user-app/app/(tabs)/merch/[productId]/page.tsx`

```
'use client';

import { useState } from 'react';
import { useParams, useRouter } from 'next/navigation';
import { useAuth } from '@nightlife/core/hooks/use-auth';
import { useClubData } from '@nightlife/core/hooks/use-club-data';
import { getSpreadshirtService } from '@nightlife/core/merch/spreadshirtService';
import { Button } from '@nightlife/ui/components/button';
import { UserCodeDisplay } from '../components/UserCodeDisplay';

export default function MerchProductDetail() {
  const params = useParams();
  const router = useRouter();
  const { user, userData } = useAuth();
  const { currentClub } = useClubData();

  const [selectedSize, setSelectedSize] = useState('M');
  const [selectedColor, setSelectedColor] = useState('black');
  const [includePersonalization, setIncludePersonalization] = useState(true);
  const [creating, setCreating] = useState(false);

  const handleOrder = async () => {
    if (!userData || !currentClub) return;

    try {
      setCreating(true);

      const service = await getSpreadshirtService();
      const result = await service.createPersonalizedDesign(
        userData,
        currentClub,
        params.productId as string,
        selectedSize,
        selectedColor
      );

      // Öffne Spreadshirt-Checkout
      window.open(result.checkoutUrl, '_blank');

      // Optional: Speichere Bestellung in Firestore (über Backend)
      // await createMerchOrder(...)

    } catch (error) {
      console.error('Failed to create order:', error);
      alert('Bestellung konnte nicht erstellt werden');
    } finally {
      setCreating(false);
    }
  };

  return (
    <div className="min-h-screen bg-gradient-to-b from-slate-900 to-slate-950 p-6">
      {/* Produkt-Details */}
      <div className="bg-slate-800 rounded-lg overflow-hidden">
        <img
          src="/placeholder-tshirt.png"
          alt="Product"
          className="w-full h-64 object-cover"
        />

        <div className="p-6">
          <h2 className="text-2xl font-bold text-white mb-4">
```

```
            {currentClub?.name} T-Shirt
          </h2>

          {/* Größen-Auswahl */}
          <div className="mb-4">
            <label className="text-sm text-gray-400 mb-2 block">Größe</label>
            <div className="flex gap-2">
              {['S', 'M', 'L', 'XL', 'XXL'].map((size) => (
                <button
                  key={size}
                  onClick={() => setSelectedSize(size)}
                  className={`px-4 py-2 rounded-lg ${
                    selectedSize === size
                      ? 'bg-cyan-600 text-white'
                      : 'bg-slate-700 text-gray-300'
                  }`}
                >
                  {size}
                </button>
              ))}
            </div>
          </div>

          {/* Farb-Auswahl */}
          <div className="mb-4">
            <label className="text-sm text-gray-400 mb-2 block">Farbe</label>
            <div className="flex gap-2">
              {['black', 'white', 'navy', 'red'].map((color) => (
                <button
                  key={color}
                  onClick={() => setSelectedColor(color)}
                  className={`w-10 h-10 rounded-full border-2 ${
                    selectedColor === color
                      ? 'border-cyan-500'
                      : 'border-transparent'
                  }`}
                  style={{ backgroundColor: color }}
                />
              ))}
            </div>
          </div>

          {/* Personalisierung */}
          <div className="mb-6">
            <label className="flex items-center gap-3 cursor-pointer">
              <input
                type="checkbox"
                checked={includePersonalization}
                onChange={(e) => setIncludePersonalization(e.target.checked)}
                className="w-5 h-5"
              />
              <span className="text-white">
                Meinen Code & QR aufdrucken
              </span>
            </label>

            {includePersonalization && (
              <div className="mt-4 p-4 bg-slate-700 rounded-lg">
                <UserCodeDisplay
                  personalCode={userData?.personalCode || ''}
                  qrCodeUrl={userData?.qrCodeUrl || ''}
                  compact
                />
              </div>
            )}
          </div>

          {/* Bestellen-Button */}
          <Button
            onClick={handleOrder}
            disabled={creating}
            className="w-full py-4 text-lg"
          >
            {creating ? 'Wird erstellt...' : 'Jetzt bestellen'}
          </Button>
        </div>
      </div>
    </div>
  );
}
```

UserCodeDisplay Komponente

Datei: `apps/user-app/app/(tabs)/merch/components/UserCodeDisplay.tsx`

```tsx
'use client';

import { QRCodeSVG } from 'qrcode.react';
import { useState } from 'react';

interface UserCodeDisplayProps {
  personalCode: string;
  qrCodeUrl: string;
  compact?: boolean;
}

export function UserCodeDisplay({
  personalCode,
  qrCodeUrl,
  compact = false
}: UserCodeDisplayProps) {
  const [showQR, setShowQR] = useState(false);

  if (compact) {
    return (
      <div className="flex items-center gap-3">
        <div className="bg-white p-2 rounded">
          <QRCodeSVG value={personalCode} size={40} />
        </div>
        <div>
          <div className="text-xs text-gray-400">Dein Code</div>
          <div className="text-lg font-bold text-white">{personalCode}</div>
        </div>
      </div>
    );
  }

  return (
    <div className="flex items-center justify-between">
      <div>
        <div className="text-sm text-gray-400">Dein Personal Code</div>
        <div className="text-xl font-bold text-cyan-400">{personalCode}</div>
      </div>

      <button
        onClick={() => setShowQR(!showQR)}
        className="px-4 py-2 bg-slate-700 hover:bg-slate-600 rounded-lg text-white
text-sm"
      >
        {showQR ? 'QR ausblenden' : 'QR anzeigen'}
      </button>

      {showQR && (
        <div className="absolute top-20 right-4 bg-white p-4 rounded-lg shadow-xl">
          <img src={qrCodeUrl} alt="QR Code" className="w-48 h-48" />
        </div>
      )}
    </div>
  );
}
```

3.6 Firestore-Schema (Zusammenfassung)

```
platform/
├── config/
│   └── globalMerchSettings/                # NEU: Globale Spreadshirt-Config
│       ├── spreadshirtShopId
│       ├── spreadshirtApiKey
│       ├── spreadshirtApiSecret
│       ├── shopsByCountry
│       ├── enabled
│       └── baseProductIds
│
├── clubs/
│   └── {clubId}/
│       └── merchSettings:                  # NEU: Club-spezifische Einstellungen
│           ├── enabled
│           ├── allowedProductCategories
│           ├── brandingLogoUrl
│           └── customMessage
│
└── merchOrders/                            # NEU: Bestellungen
    └── {orderId}/
        ├── userId
        ├── clubId
        ├── spreadshirtOrderId
        ├── status
        ├── productSnapshot
        └── ...
```

---

4. TICKETING – STRIPE CONNECT + PLATTFORM-PROVISION

Konzept

- **Jeder Club** hat sein eigenes **Stripe-Konto** (via Stripe Connect)
- **Plattform** erhält automatische **Provision** (Standard: 7%) über `application_fee_amount`
- **Stripe** splittet Zahlungen automatisch:
- Club erhält: `amount - application_fee_amount`
- Plattform erhält: `application_fee_amount`
- **Keine manuelle Abrechnung** nötig

4.1 Globales Payment-Config

Datenmodell

Datei: `packages/shared-types/src/payment.ts`

```
export interface GlobalPaymentSettings {
  stripePublicKey: string;                   // Stripe Publishable Key
  stripeSecretKeyStoredInBackend: boolean;   // Secret Key nur im Backend
  defaultTicketCommissionPercent: number;    // Standard-Provision (z.B. 7)
  enabled: boolean;
  updatedAt: number;
}
```

Firestore-Schema

```
platform/
└── config/
    └── globalPaymentSettings/               # Fixed Document
        ├── stripePublicKey: string
        ├── stripeSecretKeyStoredInBackend: boolean
        ├── defaultTicketCommissionPercent: number
        ├── enabled: boolean
        └── updatedAt: number
```

Security Rules

```
match /platform/config/globalPaymentSettings {
  // Nur Super Admin kann lesen/schreiben
  allow read, write: if isSuperAdmin();
}
```

4.2 ClubPaymentSettings mit Stripe-Connect

Datenmodell

Datei: `packages/shared-types/src/club.ts`

```
export interface ClubPaymentSettings {
  stripeConnectedAccountId: string | null;  // "acct_xxx..." oder null
  ticketingEnabled: boolean;
  ticketCommissionPercent?: number;          // Optional: Club-spezifisch
  onboardingComplete: boolean;               // Stripe-Onboarding abgeschlossen?
  onboardedAt?: number;
  updatedAt: number;
}

export interface Club {
  // Bestehende Felder...
  clubId: string;
  name: string;
  slug: string;
  // ...

  // NEU: Payment-Settings
  paymentSettings?: ClubPaymentSettings;
}
```

Firestore-Schema

```
platform/
└── clubs/
    └── {clubId}/
        ├── (bestehende Felder...)
        └── paymentSettings:                 # NEU
            ├── stripeConnectedAccountId: string | null
            ├── ticketingEnabled: boolean
            ├── ticketCommissionPercent?: number
            ├── onboardingComplete: boolean
            ├── onboardedAt?: number
            └── updatedAt: number
```

Security Rules

```
match /platform/clubs/{clubId} {
  allow read: if isSuperAdmin() || isClubOwner(clubId);

  // Club-Admin kann nur paymentSettings.ticketingEnabled updaten
  allow update: if isClubOwner(clubId) &&
    request.resource.data.diff(resource.data).affectedKeys()
      .hasOnly(['paymentSettings.ticketingEnabled', 'paymentSettings.updatedAt']);
}
```

4.3 Onboarding-Flow für Clubs (Stripe Connect)

API-Endpoint: Onboarding-Link erstellen

Datei: `apps/club-admin/app/api/payment/stripe/connect-link/route.ts`

```
import { NextRequest, NextResponse } from 'next/server';
import Stripe from 'stripe';
import { getFirestore, doc, updateDoc } from 'firebase/firestore';

const stripe = new Stripe(process.env.STRIPE_SECRET_KEY!, {
  apiVersion: '2024-11-20.acacia'
});

export async function POST(request: NextRequest) {
  try {
    const { clubId } = await request.json();

    // Auth-Check (nur Club-Owner)
    // TODO: Implementiere Auth-Check

    const db = getFirestore();
    const clubRef = doc(db, 'platform/clubs', clubId);
    const clubSnap = await getDoc(clubRef);

    if (!clubSnap.exists()) {
      return NextResponse.json(
        { error: 'Club not found' },
        { status: 404 }
      );
    }

    const club = clubSnap.data();
    let accountId = club.paymentSettings?.stripeConnectedAccountId;

    // Wenn noch kein Account existiert, erstelle einen
    if (!accountId) {
      const account = await stripe.accounts.create({
        type: 'standard',
        country: 'DE', // TODO: Aus Club-Daten
        email: club.contact?.email,
        business_profile: {
          name: club.name,
          url: club.contact?.website
        }
      });

      accountId = account.id;

      // Speichere Account-ID
      await updateDoc(clubRef, {
        'paymentSettings.stripeConnectedAccountId': accountId,
        'paymentSettings.updatedAt': Date.now()
      });
    }

    // Erstelle Onboarding-Link
    const accountLink = await stripe.accountLinks.create({
      account: accountId,
      refresh_url: `${process.env.APP_URL}/admin/settings/payments?refresh=true`,
      return_url: `${process.env.APP_URL}/admin/settings/payments?success=true`,
      type: 'account_onboarding'
    });

    return NextResponse.json({
      url: accountLink.url,
      accountId
    });
  } catch (error) {
    console.error('Error creating Stripe Connect link:', error);
    return NextResponse.json(
      { error: 'Failed to create Connect link' },
      { status: 500 }
    );
  }
}
```

Webhook: Account-Status

Datei: `apps/club-admin/app/api/payment/stripe/webhook/route.ts`

```typescript
import { NextRequest, NextResponse } from 'next/server';
import Stripe from 'stripe';
import { getFirestore, doc, updateDoc } from 'firebase/firestore';

const stripe = new Stripe(process.env.STRIPE_SECRET_KEY!, {
  apiVersion: '2024-11-20.acacia'
});

export async function POST(request: NextRequest) {
  try {
    const body = await request.text();
    const sig = request.headers.get('stripe-signature')!;

    const event = stripe.webhooks.constructEvent(
      body,
      sig,
      process.env.STRIPE_WEBHOOK_SECRET!
    );

    const db = getFirestore();

    // Handle account.updated event
    if (event.type === 'account.updated') {
      const account = event.data.object as Stripe.Account;

      // Finde Club mit dieser Account-ID
      const clubsRef = collection(db, 'platform/clubs');
      const q = query(
        clubsRef,
        where('paymentSettings.stripeConnectedAccountId', '==', account.id)
      );
      const clubSnap = await getDocs(q);

      if (!clubSnap.empty) {
        const clubDoc = clubSnap.docs[0];

        // Prüfe ob Onboarding abgeschlossen
        const onboardingComplete =
          account.charges_enabled &&
          account.payouts_enabled;

        await updateDoc(clubDoc.ref, {
          'paymentSettings.onboardingComplete': onboardingComplete,
          'paymentSettings.onboardedAt': onboardingComplete ? Date.now() : null,
          'paymentSettings.updatedAt': Date.now()
        });
      }
    }

    return NextResponse.json({ received: true });

  } catch (error) {
    console.error('Webhook error:', error);
    return NextResponse.json(
      { error: 'Webhook processing failed' },
      { status: 400 }
    );
  }
}
```

4.4 Datenmodell Events & Tickets

Event-Typen

Datei: `packages/shared-types/src/events.ts`

```
export interface EventTicketType {
  typeId: string;                            // "standard", "fastlane", "vip"
  name: string;                              // "Standard-Ticket"
  description?: string;
  price: number;                             // in Cent (Minor Units)
  currency: string;                          // "EUR"
  maxQuantity?: number;                      // Max. Anzahl verfügbar
  soldQuantity: number;                      // Bereits verkauft
  perks?: string[];                          // z.B. ["Freier Eintritt", "1
Freigetränk"]
  highlightColor?: string;                   // z.B. "#FFD700" für VIP
  sortOrder: number;                         // Anzeigereihenfolge
}

export interface Event {
  id: string;
  clubId: string;
  name: string;
  description?: string;
  imageUrl?: string;

  // Zeitplanung
  startAt: number;                           // Event-Start (Timestamp)
  endAt?: number;                            // Event-Ende

  // Ticketing
  ticketingEnabled: boolean;
  ticketTypes: EventTicketType[];
  ticketSalesStart?: number;                 // Verkaufsstart
  ticketSalesEnd?: number;                   // Verkaufsende

  // Kapazität
  maxCapacity?: number;                      // Max. Gäste gesamt
  soldTickets: number;                       // Verkaufte Tickets gesamt

  // Meta
  createdBy: string;                         // UID des Erstellers
  createdAt: number;
  updatedAt: number;
  published: boolean;                        // Veröffentlicht?
}

export interface EventTicket {
  id: string;
  eventId: string;
  clubId: string;
  userId: string;

  // Ticket-Details
  typeId: string;                            // Referenz zu EventTicketType
  typeSnapshot: EventTicketType;             // Kopie zum Zeitpunkt des Kaufs

  // Status
  status: 'pending' | 'valid' | 'used' | 'canceled' | 'refunded';
  purchasedAt?: number;
  usedAt?: number;
  canceledAt?: number;

  // QR-Code (separater Code, nicht personalCode!)
  ticketCode: string;                        // Eindeutiger Ticket-Code
  qrCodeUrl: string;                         // QR-Bild-URL
  // Payment
  stripePaymentIntentId?: string;
  paidAmount: number;                    // Bezahlter Betrag (in Cent)

  // Meta
  createdAt: number;
  updatedAt: number;
}
```

Firestore-Schema

```
clubs/
└── {clubId}/
    ├── events/                               # NEU: Events
    │   └── {eventId}/
    │       ├── id: string
    │       ├── clubId: string
    │       ├── name: string
    │       ├── description: string
    │       ├── imageUrl: string
    │       ├── startAt: number
    │       ├── endAt: number
    │       ├── ticketingEnabled: boolean
    │       ├── ticketTypes: array
    │       ├── ticketSalesStart: number
    │       ├── ticketSalesEnd: number
    │       ├── maxCapacity: number
    │       ├── soldTickets: number
    │       ├── createdBy: string
    │       ├── createdAt: number
    │       ├── updatedAt: number
    │       └── published: boolean
    │
    └── tickets/                              # NEU: Verkaufte Tickets
        └── {ticketId}/
            ├── id: string
            ├── eventId: string
            ├── clubId: string
            ├── userId: string
            ├── typeId: string
            ├── typeSnapshot: object
            ├── status: string
            ├── purchasedAt: number
            ├── usedAt: number
            ├── ticketCode: string
            ├── qrCodeUrl: string
            ├── stripePaymentIntentId: string
            ├── paidAmount: number
            ├── createdAt: number
            └── updatedAt: number
```

4.5 Ticket-Kauf-Flow mit Provision

API-Endpoint: Payment Intent erstellen

Datei: `apps/user-app/app/api/tickets/create-payment-intent/route.ts`

```
import { NextRequest, NextResponse } from 'next/server';
import Stripe from 'stripe';
import { getFirestore, doc, getDoc } from 'firebase/firestore';

const stripe = new Stripe(process.env.STRIPE_SECRET_KEY!, {
  apiVersion: '2024-11-20.acacia'
});

export async function POST(request: NextRequest) {
  try {
    const { eventId, ticketTypeId, userId } = await request.json();

    // TODO: Auth-Check (userId muss request.auth.uid entsprechen)

    const db = getFirestore();

    // 1. Lade Event
    const eventRef = doc(db, 'clubs/{clubId}/events', eventId); // TODO: clubId ermit-
teln
    const eventSnap = await getDoc(eventRef);

    if (!eventSnap.exists()) {
      return NextResponse.json(
        { error: 'Event not found' },
        { status: 404 }
      );
    }

    const event = eventSnap.data() as Event;

    // 2. Finde Ticket-Type
    const ticketType = event.ticketTypes.find(t => t.typeId === ticketTypeId);

    if (!ticketType) {
      return NextResponse.json(
        { error: 'Ticket type not found' },
        { status: 404 }
      );
    }

    // 3. Validierungen
    if (!event.ticketingEnabled) {
      return NextResponse.json(
        { error: 'Ticketing not enabled for this event' },
        { status: 400 }
      );
    }

    const now = Date.now();
    if (event.ticketSalesStart && now < event.ticketSalesStart) {
      return NextResponse.json(
        { error: 'Ticket sales have not started yet' },
        { status: 400 }
      );
    }

    if (event.ticketSalesEnd && now > event.ticketSalesEnd) {
      return NextResponse.json(
        { error: 'Ticket sales have ended' },
        { status: 400 }
      );
    }
    if (ticketType.maxQuantity && ticketType.soldQuantity >= ticketType.maxQuantity) {
      return NextResponse.json(
        { error: 'This ticket type is sold out' },
        { status: 400 }
      );
    }

    if (event.maxCapacity && event.soldTickets >= event.maxCapacity) {
      return NextResponse.json(
        { error: 'Event is sold out' },
        { status: 400 }
      );
    }

    // 4. Lade Club + Payment Settings
    const clubRef = doc(db, 'platform/clubs', event.clubId);
    const clubSnap = await getDoc(clubRef);

    if (!clubSnap.exists()) {
      return NextResponse.json(
        { error: 'Club not found' },
        { status: 404 }
      );
    }

    const club = clubSnap.data();
    const paymentSettings = club.paymentSettings as ClubPaymentSettings;

    if (!paymentSettings?.stripeConnectedAccountId) {
      return NextResponse.json(
        { error: 'Club payment settings not configured' },
        { status: 400 }
      );
    }

    // 5. Lade globale Payment-Settings (für Provision)
    const globalPaymentRef = doc(db, 'platform/config/globalPaymentSettings');
    const globalPaymentSnap = await getDoc(globalPaymentRef);
    const globalPayment = globalPaymentSnap.data() as GlobalPaymentSettings;

    // 6. Berechne Provision
    const priceInCents = ticketType.price;
    const commissionPercent =
      paymentSettings.ticketCommissionPercent ??
      globalPayment.defaultTicketCommissionPercent;
    const applicationFee = Math.round(priceInCents * (commissionPercent / 100));

    // 7. Erstelle Stripe PaymentIntent mit application_fee_amount
    const paymentIntent = await stripe.paymentIntents.create({
      amount: priceInCents,
      currency: ticketType.currency.toLowerCase(),
      payment_method_types: ['card', 'ideal', 'sepa_debit'],

      // WICHTIG: Stripe Connect mit Provision
      application_fee_amount: applicationFee,
      transfer_data: {
        destination: paymentSettings.stripeConnectedAccountId
      },

      // Metadaten für Webhook
      metadata: {
        clubId: event.clubId,
        eventId: event.id,
        ticketTypeId: ticketType.typeId,
        userId: userId,
        ticketPrice: priceInCents.toString(),
        commission: applicationFee.toString()
      }
    });

    // 8. Response mit Client Secret
    return NextResponse.json({
      clientSecret: paymentIntent.client_secret,
      paymentIntentId: paymentIntent.id,
      ticketPreview: {
        eventName: event.name,
        ticketType: ticketType.name,
        price: priceInCents,
        currency: ticketType.currency
      }
    });

  } catch (error) {
    console.error('Error creating payment intent:', error);
    return NextResponse.json(
      { error: 'Failed to create payment intent' },
      { status: 500 }
    );
  }
}
```

Webhook: Payment erfolgt

Erweiterung von: `apps/user-app/app/api/payment/stripe/webhook/route.ts`

```
// ... (bestehender Webhook-Code)

// Handle payment_intent.succeeded
if (event.type === 'payment_intent.succeeded') {
  const paymentIntent = event.data.object as Stripe.PaymentIntent;
  const metadata = paymentIntent.metadata;

  // Erstelle Ticket
  const db = getFirestore();

  // Generiere Ticket-Code
  const ticketCode = generateTicketCode(); // z.B. "TCKT-ABC123XYZ"

  // Generiere Ticket-QR
  const qrCodeUrl = await generateAndUploadQRCode(
    metadata.userId,
    ticketCode
  );

  // Ticket-Dokument erstellen
  const ticketRef = doc(
    db,
    `clubs/${metadata.clubId}/tickets/${paymentIntent.id}`
  );

  await setDoc(ticketRef, {
    id: paymentIntent.id,
    eventId: metadata.eventId,
    clubId: metadata.clubId,
    userId: metadata.userId,
    typeId: metadata.ticketTypeId,
    typeSnapshot: {}, // TODO: Lade von Event
    status: 'valid',
    purchasedAt: Date.now(),
    ticketCode,
    qrCodeUrl,
    stripePaymentIntentId: paymentIntent.id,
    paidAmount: paymentIntent.amount,
    createdAt: Date.now(),
    updatedAt: Date.now()
  });

  // Inkrementiere soldTickets im Event
  const eventRef = doc(db, `clubs/${metadata.clubId}/events/${metadata.eventId}`);
  await updateDoc(eventRef, {
    soldTickets: increment(1),
    updatedAt: Date.now()
  });

  // Inkrementiere soldQuantity im Ticket-Type
  // TODO: Atomare Update-Logik für ticketTypes-Array

  // Sende Bestätigungs-Email / Push
  // TODO: Implementiere Notification-Service
}
```

Helper: Ticket-Code generieren

Datei: `packages/core/src/events/generateTicketCode.ts`

```
const TICKET_CHARS = 'ABCDEFGHJKLMNPQRSTUVWXYZ23456789';

export function generateTicketCode(): string {
  const prefix = 'TCKT';
  let code = '';

  for (let i = 0; i < 12; i++) {
    const randomIndex = Math.floor(Math.random() * TICKET_CHARS.length);
    code += TICKET_CHARS[randomIndex];

    // Alle 4 Zeichen ein Bindestrich (lesbarkeit)
    if ((i + 1) % 4 === 0 && i < 11) {
      code += '-';
    }
  }

  return `${prefix}-${code}`;
}

// Beispiel: "TCKT-ABXY-3489-KLMN"
```

4.6 Provision-Transparenz & Reporting

Super-Admin-Dashboard

Datei: `apps/super-admin/app/dashboard/revenue/page.tsx`

```
'use client';

import { useEffect, useState } from 'react';
import { collection, query, where, getDocs } from 'firebase/firestore';
import { getFirestore } from 'firebase/firestore';

interface ClubRevenue {
  clubId: string;
  clubName: string;
  ticketsSold: number;
  grossRevenue: number;        // Bruttoumsatz (Summe aller Ticket-Preise)
  platformCommission: number;  // Deine Provision
  clubRevenue: number;         // Club erhält
}

export default function RevenueDashboard() {
  const [clubRevenues, setClubRevenues] = useState<ClubRevenue[]>([]);
  const [loading, setLoading] = useState(true);

  useEffect(() => {
    loadRevenueData();
  }, []);

  const loadRevenueData = async () => {
    const db = getFirestore();
    const clubs: ClubRevenue[] = [];

    // Lade alle Clubs
    const clubsSnap = await getDocs(collection(db, 'platform/clubs'));

    for (const clubDoc of clubsSnap.docs) {
      const clubData = clubDoc.data();

      // Lade alle Tickets dieses Clubs
      const ticketsSnap = await getDocs(
        collection(db, `clubs/${clubDoc.id}/tickets`)
      );

      let grossRevenue = 0;
      let platformCommission = 0;

      ticketsSnap.forEach((ticketDoc) => {
        const ticket = ticketDoc.data();
        if (ticket.status === 'valid' || ticket.status === 'used') {
          grossRevenue += ticket.paidAmount;

          // Berechne Provision aus Metadata (falls gespeichert)
          // Oder neu berechnen mit globalPaymentSettings
          const commission = Math.round(
            ticket.paidAmount * 0.07 // 7% Standard
          );
          platformCommission += commission;
        }
      });

      clubs.push({
        clubId: clubDoc.id,
        clubName: clubData.name,
        ticketsSold: ticketsSnap.size,
        grossRevenue,
        platformCommission,
        clubRevenue: grossRevenue - platformCommission
      });
    }

    setClubRevenues(clubs);
    setLoading(false);
  };

  const totalCommission = clubRevenues.reduce(
    (sum, club) => sum + club.platformCommission,
    0
  );

  return (
    <div className="p-8">
      <h1 className="text-3xl font-bold text-white mb-8">
        Revenue Dashboard
      </h1>

      {/* Gesamt-Übersicht */}
      <div className="bg-gradient-to-r from-cyan-600 to-blue-600 rounded-lg p-6 mb-8">
        <div className="text-white/80 text-sm mb-2">
          Plattform-Provision (gesamt)
        </div>
        <div className="text-4xl font-bold text-white">
          {(totalCommission / 100).toFixed(2)} €
        </div>
      </div>

      {/* Club-Tabelle */}
      <div className="bg-slate-800 rounded-lg overflow-hidden">
        <table className="w-full">
          <thead className="bg-slate-700">
            <tr>
              <th className="px-6 py-3 text-left text-white">Club</th>
              <th className="px-6 py-3 text-right text-white">Tickets</th>
              <th className="px-6 py-3 text-right text-white">Bruttoumsatz</th>
              <th className="px-6 py-3 text-right text-white">Provision (7%)</th>
              <th className="px-6 py-3 text-right text-white">Club erhält</th>
            </tr>
          </thead>
          <tbody>
            {clubRevenues.map((club) => (
              <tr key={club.clubId} className="border-t border-slate-700">
                <td className="px-6 py-4 text-white">{club.clubName}</td>
                <td className="px-6 py-4 text-right text-gray-300">
                  {club.ticketsSold}
                </td>
                <td className="px-6 py-4 text-right text-gray-300">
                  {(club.grossRevenue / 100).toFixed(2)} €
                </td>
                <td className="px-6 py-4 text-right text-cyan-400 font-bold">
                  {(club.platformCommission / 100).toFixed(2)} €
                </td>
                <td className="px-6 py-4 text-right text-green-400">
                  {(club.clubRevenue / 100).toFixed(2)} €
                </td>
              </tr>
            ))}
          </tbody>
        </table>
      </div>
    </div>
  );
}
```

4.7 Firestore-Schema (Zusammenfassung)

```
platform/
├── config/
│   └── globalPaymentSettings/              # NEU: Globale Stripe-Config
│       ├── stripePublicKey
│       ├── stripeSecretKeyStoredInBackend
│       ├── defaultTicketCommissionPercent
│       └── enabled
│
├── clubs/
│   └── {clubId}/
│       └── paymentSettings:                # NEU: Stripe Connect
│           ├── stripeConnectedAccountId
│           ├── ticketingEnabled
│           ├── ticketCommissionPercent
│           ├── onboardingComplete
│           └── onboardedAt
│
clubs/
└── {clubId}/
    ├── events/                             # NEU: Events
    │   └── {eventId}/
    │       ├── name, description, imageUrl
    │       ├── startAt, endAt
    │       ├── ticketingEnabled
    │       ├── ticketTypes[]
    │       ├── maxCapacity, soldTickets
    │       └── ...
    │
    └── tickets/                            # NEU: Verkaufte Tickets
        └── {ticketId}/
            ├── eventId, userId
            ├── typeId, typeSnapshot
            ├── status
            ├── ticketCode, qrCodeUrl
            ├── stripePaymentIntentId
            ├── paidAmount
            └── ...
```

5. EVENT-KALENDER – NUR EVENTS VON BESUCHTEN CLUBS

Konzept

- User sehen **nur Events** von Clubs, in denen sie **bereits eingecheckt** waren
- Privacy-First: Kein Tracking von fremden Clubs

- Basis: `visitedClubIds` im User-Dokument

5.1 Datenmodell-Erweiterung User

PlatformUser / Club-User

Datei: `packages/shared-types/src/user.ts`

```
export interface PlatformUser {
  // Bestehende Felder...
  uid: string;
  email: string;
  displayName: string | null;
  personalCode: string;
  qrCodeUrl: string;

  // NEU: Besuchte Clubs
  visitedClubIds: string[];   // Liste aller Clubs mit Check-In-Historie
}
```

Update bei Check-In

Datei: `packages/core/src/user/checkinService.ts`

```
import { doc, updateDoc, arrayUnion } from 'firebase/firestore';

/**
 * Check-In in Club durchführen
 * Fügt clubId zu visitedClubIds hinzu (falls nicht vorhanden)
 */
export async function checkInToClub(
  userId: string,
  clubId: string
): Promise<void> {
  const db = getFirestore();

  // 1. Update club-spezifisches User-Dokument
  const clubUserRef = doc(db, `clubs/${clubId}/users/${userId}`);
  await updateDoc(clubUserRef, {
    checkedIn: true,
    checkedInAt: Date.now(),
    lastSeen: Date.now()
  });

  // 2. Update plattformweites User-Dokument
  const platformUserRef = doc(db, `platform/users/${userId}`);
  await updateDoc(platformUserRef, {
    visitedClubIds: arrayUnion(clubId), // Fügt hinzu, falls nicht vorhanden
    lastSeenAt: Date.now()
  });
}
```

5.2 Event-Query-Logik

Service-Funktion

Datei: `packages/core/src/events/eventService.ts`

```
import {
  collection,
  query,
  where,
  orderBy,
  limit,
  getDocs
} from 'firebase/firestore';

/**
 * Lädt kommende Events für User (nur von besuchten Clubs)
 */
export async function getUpcomingEventsForUser(
  userId: string,
  limitCount: number = 20
): Promise<Event[]> {
  const db = getFirestore();

  // 1. Lade User → visitedClubIds
  const userRef = doc(db, 'platform/users', userId);
  const userSnap = await getDoc(userRef);

  if (!userSnap.exists()) {
    return [];
  }

  const user = userSnap.data() as PlatformUser;
  const visitedClubIds = user.visitedClubIds || [];

  if (visitedClubIds.length === 0) {
    return []; // User hat noch keine Clubs besucht
  }

  // 2. Query Events von besuchten Clubs
  // Firestore 'in'-Operator unterstützt max. 10 Werte
  // Falls mehr Clubs, muss in Batches gequeried werden

  const now = Date.now();
  const events: Event[] = [];

  // Batch-Query (max 10 Clubs pro Query)
  for (let i = 0; i < visitedClubIds.length; i += 10) {
    const batch = visitedClubIds.slice(i, i + 10);

    const eventsQuery = query(
      collectionGroup(db, 'events'),
      where('clubId', 'in', batch),
      where('published', '==', true),
      where('startAt', '>', now),
      orderBy('startAt', 'asc'),
      limit(limitCount)
    );

    const snapshot = await getDocs(eventsQuery);
    snapshot.forEach((doc) => {
      events.push(doc.data() as Event);
    });
  }

  // Sortieren nach Datum (da aus mehreren Queries)
  events.sort((a, b) => a.startAt - b.startAt);
  return events.slice(0, limitCount);
}
```

Wichtig: Firestore `collectionGroup` query erlaubt Suche über alle `events` -Subcollections.

5.3 UI – Events-Tab

Komponenten-Struktur

```
apps/user-app/app/(tabs)/events/
├── page.tsx                                  # EventsTab (Liste)
├── [eventId]/
│   └── page.tsx                              # EventDetailScreen
└── components/
    ├── EventCard.tsx                         # Event-Vorschau
    ├── EventTicketTypes.tsx                  # Ticket-Auswahl
    └── EventCountdown.tsx                    # Countdown bis Event
```

EventsTab

Datei: `apps/user-app/app/(tabs)/events/page.tsx`

```
'use client';

import { useEffect, useState } from 'react';
import { useAuth } from '@nightlife/core/hooks/use-auth';
import { getUpcomingEventsForUser } from '@nightlife/core/events/eventService';
import { Event } from '@nightlife/shared-types';
import { EventCard } from './components/EventCard';

export default function EventsTab() {
  const { user } = useAuth();
  const [events, setEvents] = useState<Event[]>([]);
  const [loading, setLoading] = useState(true);

  useEffect(() => {
    if (user) {
      loadEvents();
    }
  }, [user]);

  const loadEvents = async () => {
    try {
      const upcomingEvents = await getUpcomingEventsForUser(user!.uid);
      setEvents(upcomingEvents);
    } catch (error) {
      console.error('Failed to load events:', error);
    } finally {
      setLoading(false);
    }
  };

  if (loading) {
    return (
      <div className="flex justify-center items-center h-screen">
        <div className="animate-spin rounded-full h-12 w-12 border-4 border-cyan-500
border-t-transparent" />
      </div>
    );
  }

  if (events.length === 0) {
    return (
      <div className="flex flex-col items-center justify-center h-screen p-6">
        <h2 className="text-2xl font-bold text-white mb-4">
          Keine Events gefunden
        </h2>
        <p className="text-gray-400 text-center">
          Checke in einem Club ein, um Events zu sehen!
        </p>
      </div>
    );
  }

  return (
    <div className="min-h-screen bg-gradient-to-b from-slate-900 to-slate-950 pb-20">
      <div className="sticky top-0 z-10 bg-slate-900/95 backdrop-blur-sm border-b
border-slate-800 p-4">
        <h1 className="text-2xl font-bold text-white">Kommende Events</h1>
      </div>

      <div className="p-4 space-y-4">
        {events.map((event) => (
          <EventCard key={event.id} event={event} />
          ))}
        </div>
      </div>
    );
  }
```

EventCard Komponente

Datei: `apps/user-app/app/(tabs)/events/components/EventCard.tsx`

```tsx
'use client';

import Link from 'next/link';
import { Event } from '@nightlife/shared-types';
import { format } from 'date-fns';
import { de } from 'date-fns/locale';

interface EventCardProps {
  event: Event;
}

export function EventCard({ event }: EventCardProps) {
  const minPrice = Math.min(
    ...event.ticketTypes.map(t => t.price)
  );

  return (
    <Link href={`/events/${event.id}`}>
      <div className="bg-slate-800 rounded-lg overflow-hidden hover:bg-slate-750 transition-colors">
        {event.imageUrl && (
          <img
            src={event.imageUrl}
            alt={event.name}
            className="w-full h-48 object-cover"
          />
        )}

        <div className="p-4">
          <h3 className="text-xl font-bold text-white mb-2">
            {event.name}
          </h3>

          <div className="flex items-center gap-2 text-gray-400 text-sm mb-2">
            <span>📅</span>
            <span>
              {format(event.startAt, 'EEEE, dd. MMMM yyyy', { locale: de })}
            </span>
          </div>

          <div className="flex items-center gap-2 text-gray-400 text-sm mb-3">
            <span>🕐</span>
            <span>{format(event.startAt, 'HH:mm')} Uhr</span>
          </div>

          {event.ticketingEnabled && (
            <div className="flex items-center justify-between">
              <span className="text-cyan-400 font-bold">
                ab {(minPrice / 100).toFixed(2)} €
              </span>

              <span className="text-sm text-gray-500">
                {event.maxCapacity && event.soldTickets >= event.maxCapacity ? (
                  <span className="text-red-400">Ausverkauft</span>
                ) : (
                  <span className="text-green-400">Tickets verfügbar</span>
                )}
              </span>
            </div>
          )}
        </div>
      </div>
    </Link>
  );
}
```

5.4 Firestore-Schema (Zusammenfassung)

```
platform/
└── users/
    └── {uid}/
        ├── (bestehende Felder...)
        └── visitedClubIds: string[]         # NEU: Liste besuchter Clubs

clubs/
└── {clubId}/
    └── events/                              # Siehe Sektion 4.4
        └── {eventId}/
            ├── name, description
            ├── startAt, endAt
            ├── ticketingEnabled
            ├── published                     # Nur published=true wird angezeigt
            └── ...
```

---

6. SICHERHEIT, ROLLEN & FIRESTORE RULES

6.1 Rollenrechte

Übersicht

Rolle	Zugriff auf	Rechte
Super Admin	Alles	- GlobalMerchSettings (R/W) - GlobalPaymentSettings (R/W) - Alle Clubs (R/W) - Alle Merch-Orders (R) - Revenue-Reports (R)
Club Admin	Eigener Club	- ClubMerchSettings (R/W) - ClubPaymentSettings (R/W) - Events erstellen/bearbeiten (R/W) - Tickets einsehen (R) - Stripe-Onboarding starten
User	Eigene Daten	- Eigener personalCode (R) - Eigene Tickets (R/W) - Eigene Merch-Orders (R) - Events von visitedClubs (R)

6.2 Firestore Security Rules

Erweiterungen

```
rules_version = '2';
service cloud.firestore {
  match /databases/{database}/documents {

    // ===== HELPER FUNCTIONS (bereits definiert) =====
    function isAuthenticated() { ... }
    function isSuperAdmin() { ... }
    function isClubOwner(clubId) { ... }
    function hasRole(clubId, role) { ... }

    // ===== NEU: GLOBAL MERCH SETTINGS =====
    match /platform/config/globalMerchSettings {
      // Nur Super Admin
      allow read, write: if isSuperAdmin();
    }

    // ===== NEU: GLOBAL PAYMENT SETTINGS =====
    match /platform/config/globalPaymentSettings {
      // Nur Super Admin
      allow read, write: if isSuperAdmin();
    }

    // ===== NEU: CLUB MERCH SETTINGS =====
    match /platform/clubs/{clubId} {
      // Club-Admin kann merchSettings updaten
      allow update: if isAuthenticated() &&
        isClubOwner(clubId) &&
        request.resource.data.diff(resource.data).affectedKeys()
          .hasOnly(['merchSettings', 'paymentSettings']);
    }

    // ===== NEU: MERCH ORDERS =====
    match /platform/merchOrders/{orderId} {
      // User kann nur eigene Orders sehen
      allow read: if isAuthenticated() &&
        (request.auth.uid == resource.data.userId || isSuperAdmin());

      // Nur Backend kann schreiben (via Admin SDK)
      allow write: if false;
    }

    // ===== NEU: EVENTS =====
    match /clubs/{clubId}/events/{eventId} {
      // Alle authentifizierten User können published Events lesen
      allow read: if isAuthenticated() && resource.data.published == true;

      // Admins können alle Events lesen
      allow read: if hasRole(clubId, 'admin');

      // Admins können Events erstellen/updaten
      allow create, update, delete: if hasRole(clubId, 'admin');
    }

    // ===== NEU: TICKETS =====
    match /clubs/{clubId}/tickets/{ticketId} {
      // User kann nur eigene Tickets sehen
      allow read: if isAuthenticated() &&
        (request.auth.uid == resource.data.userId ||
         hasRole(clubId, 'admin'));

      // Nur Backend kann Tickets erstellen (via Webhook)
      allow create: if false;
      // User kann status updaten (für Scanning)
      // Staff kann status updaten
      allow update: if isAuthenticated() &&
        (request.auth.uid == resource.data.userId ||
         hasRole(clubId, 'admin') ||
         hasRole(clubId, 'door'));
    }

    // ===== PLATFORM USER (mit personalCode) =====
    match /platform/users/{uid} {
      // User kann eigenes Dokument lesen
      allow read: if isAuthenticated() &&
        (request.auth.uid == uid || isSuperAdmin());

      // User kann updaten (außer personalCode, qrCodeUrl, visitedClubIds)
      allow update: if request.auth.uid == uid &&
        !request.resource.data.diff(resource.data).affectedKeys()
          .hasAny(['personalCode', 'qrCodeUrl', 'isPlatformAdmin']);

      // visitedClubIds wird nur via Backend/Cloud Function gesetzt
    }
  }
}
```

6.3 DSGVO-Compliance

Zahlungsdaten

Merch (Spreadshirt)

- **Wo:** Adresse & Zahlungsdaten liegen bei **Spreadshirt**
- **Wir speichern:** Nur `spreadshirtOrderId`, `productSnapshot`, `status`
- **Keine personenbezogenen Zahlungsdaten** in unserer DB

Tickets (Stripe)

- **Wo:** Zahlungsdaten liegen bei **Stripe**
- **Wir speichern:** Nur `stripePaymentIntentId`, `paidAmount`, `status`
- **Keine Kreditkarten-Daten** in unserer DB

User-Daten

Personal Code & QR

- **Personenbezogen:** Ja (ist mit User-Account verknüpft)
- **Zweck:** Merch-Personalisierung, Loyalty, Friend-Requests
- **Löschung:** Bei Account-Löschung wird auch QR-Code aus Storage gelöscht
- **Rechtsgrundlage:** Art. 6 Abs. 1 lit. b DSGVO (Vertragserfüllung)

VisitedClubIds

- **Personenbezogen:** Ja
- **Zweck:** Relevante Event-Vorschläge
- **Löschung:** Bei Account-Löschung
- **Rechtsgrundlage:** Art. 6 Abs. 1 lit. f DSGVO (berechtigtes Interesse)

Datenlöschung

Service-Funktion: `packages/core/src/user/deleteUserData.ts`

```
export async function deleteUserData(userId: string): Promise<void> {
  const db = getFirestore();
  const storage = getStorage();

  // 1. Lösche Platform User
  await deleteDoc(doc(db, 'platform/users', userId));

  // 2. Lösche QR-Code aus Storage
  const qrRef = ref(storage, `users/${userId}/qr-code.png`);
  await deleteObject(qrRef);

  // 3. Lösche Club-User-Dokumente
  // TODO: Iteriere über memberClubs

  // 4. Lösche Tickets (Status → 'canceled')
  // TODO: Query tickets by userId

  // 5. Lösche Merch-Orders (Status → 'canceled')
  // TODO: Query merchOrders by userId
}
```

---

7. ZUSAMMENFASSUNG & DATEIEN

7.1 Neue Strukturen (Übersicht)

✅ User-Code & QR

- [x] `personalCode` pro User (plattformweit eindeutig)
- [x] `qrCodeUrl` in Firebase Storage
- [x] Generierung bei Registrierung/Login
- [x] Security: Nur eigener Code sichtbar

✅ Merch (100% Super Admin)

- [x] `GlobalMerchSettings` (Spreadshirt-Keys)
- [x] `ClubMerchSettings` (Kategorien, Branding)
- [x] Spreadshirt-Service (Produkte laden, Design erstellen)
- [x] `MerchOrder` Collection (Order-Tracking)
- [x] UI: MerchTab, ProductDetail, UserCodeDisplay

✅ Ticketing (Provision über Stripe Connect)

- [x] `GlobalPaymentSettings` (Stripe Public Key, Provision %)
- [x] `ClubPaymentSettings` (Stripe Connected Account)
- [x] Stripe Connect Onboarding-Flow
- [x] `Event` & `EventTicket` Datenmodell
- [x] PaymentIntent mit `application_fee_amount`
- [x] Webhook: payment_intent.succeeded → Ticket erstellen

- [x] Super-Admin: Revenue-Dashboard

✅ Event-Kalender (Privacy-First)

- [x] `visitedClubIds` im User-Dokument
- [x] Update bei Check-In
- [x] Event-Query: Nur von besuchten Clubs
- [x] UI: EventsTab, EventCard, EventDetail

7.2 Betroffene Dateien

Neue Dateien

```
packages/
├── shared-types/src/
│   ├── merch.ts                           # NEU: MerchProduct, MerchOrder, Glob-
alMerchSettings
│   ├── payment.ts                         # NEU: GlobalPaymentSettings
│   └── events.ts                          # NEU: Event, EventTicket, EventTicket-
Type
│
├── core/src/
│   ├── user/
│   │   ├── generatePersonalCode.ts        # NEU: Code-Generierung
│   │   ├── generateQRCode.ts              # NEU: QR-Code-Generierung & Upload
│   │   ├── userService.ts                 # ERWEITERT: ensureUserHasPersonalCode
│   │   ├── checkinService.ts              # ERWEITERT: visitedClubIds updaten
│   │   └── deleteUserData.ts              # NEU: DSGVO-Löschung
│   │
│   ├── merch/
│   │   └── spreadshirtService.ts          # NEU: Spreadshirt-Integration
│   │
│   └── events/
│       ├── eventService.ts                # NEU: getUpcomingEventsForUser
│       └── generateTicketCode.ts          # NEU: Ticket-Code-Generierung
│
apps/
├── user-app/app/
│   ├── (tabs)/
│   │   ├── merch/
│   │   │   ├── page.tsx                   # NEU: MerchTab
│   │   │   ├── [productId]/page.tsx       # NEU: MerchProductDetail
│   │   │   └── components/
│   │   │       ├── MerchProductCard.tsx   # NEU
│   │   │       ├── UserCodeDisplay.tsx    # NEU
│   │   │       └── MerchOrderHistory.tsx  # NEU
│   │   │
│   │   └── events/
│   │       ├── page.tsx                   # NEU: EventsTab
│   │       ├── [eventId]/page.tsx         # NEU: EventDetailScreen
│   │       └── components/
│   │           ├── EventCard.tsx          # NEU
│   │           ├── EventTicketTypes.tsx   # NEU
│   │           └── EventCountdown.tsx     # NEU
│   │
│   └── api/
│       ├── tickets/
│       │   └── create-payment-intent/
│       │       └── route.ts               # NEU: Stripe PaymentIntent
│       │
│       └── payment/
│           └── stripe/
│               └── webhook/
│                   └── route.ts           # ERWEITERT: payment_intent.succeeded
│
├── club-admin/app/
│   ├── api/
│   │   └── payment/
│   │       └── stripe/
│   │           └── connect-link/
│   │               └── route.ts           # NEU: Stripe Connect Onboarding
│   │
│   └── dashboard/
│       ├── settings/
│       │   ├── merch/page.tsx             # NEU: Merch-Settings
│       │   └── payments/page.tsx          # NEU: Stripe-Onboarding-UI
│       │
│       └── events/
│           ├── page.tsx                   # NEU: Event-Verwaltung
│           └── create/page.tsx            # NEU: Event erstellen
│
└── super-admin/app/
    └── dashboard/
        ├── merch/
        │   └── page.tsx                   # NEU: Merch-Config (Spreadshirt)
        │
        ├── payments/
        │   └── page.tsx                   # NEU: Payment-Config (Stripe)
        │
        └── revenue/
            └── page.tsx                   # NEU: Revenue-Dashboard
```

Erweiterte Dateien

```
packages/
├── shared-types/src/
│   ├── user.ts                            # ERWEITERT: personalCode, qrCodeUrl, vi
sitedClubIds
│   └── club.ts                            # ERWEITERT: merchSettings, paymentSet-
tings
│
firestore.rules                           # ERWEITERT: Rules für Merch, Payment, Ev
ents

firebase-storage.rules                    # ERWEITERT: Rules für QR-Codes
```

7.3 Code-Snippets

1. PaymentIntent mit Provision erstellen

```
// Zentrale Stelle: apps/user-app/app/api/tickets/create-payment-intent/route.ts

const priceInCents = ticketType.price; // z.B. 2000 (20 EUR)
const commissionPercent =
  paymentSettings.ticketCommissionPercent ??
  globalPayment.defaultTicketCommissionPercent; // z.B. 7

const applicationFee = Math.round(priceInCents * (commissionPercent / 100));
// applicationFee = 140 (1,40 EUR)

const paymentIntent = await stripe.paymentIntents.create({
  amount: priceInCents,                      // 2000 Cent = 20 EUR
  currency: 'eur',
  payment_method_types: ['card', 'ideal', 'sepa_debit'],

  // WICHTIG: Stripe Connect mit Provision
  application_fee_amount: applicationFee,    // 140 Cent = 1,40 EUR (7%)
  transfer_data: {
    destination: paymentSettings.stripeConnectedAccountId // Club-Konto
  },

  metadata: {
    clubId: event.clubId,
    eventId: event.id,
    ticketTypeId: ticketType.typeId,
    userId: userId,
    ticketPrice: priceInCents.toString(),
    commission: applicationFee.toString()
  }
});

// Resultat:
// - Club erhält: 2000 - 140 = 1860 Cent (18,60 EUR)
// - Plattform erhält: 140 Cent (1,40 EUR)
```

2. Spreadshirt-Service: Personalisiertes Design erstellen

```typescript
// Zentrale Stelle: packages/core/src/merch/spreadshirtService.ts

async createPersonalizedDesign(
  user: PlatformUser,
  club: Club,
  baseProductId: string,
  selectedSize: string,
  selectedColor: string
): Promise<PersonalizedDesignResult> {
  // 1. Design-Payload zusammenbauen
  const designPayload = {
    name: `${user.personalCode} - ${club.name}`,
    elements: [
      {
        type: 'image',
        imageUrl: user.qrCodeUrl,            // QR-Code vom User
        position: { x: 100, y: 100 },
        size: { width: 150, height: 150 }
      },
      {
        type: 'text',
        text: user.personalCode,             // "CFX8R3Y"
        position: { x: 100, y: 270 },
        fontSize: 24,
        fontFamily: 'Arial Bold'
      }
    ]
  };

  // Optional: Club-Logo hinzufügen
  if (club.merchSettings?.brandingLogoUrl) {
    designPayload.elements.push({
      type: 'image',
      imageUrl: club.merchSettings.brandingLogoUrl,
      position: { x: 50, y: 50 },
      size: { width: 100, height: 100 }
    });
  }

  // 2. Design via Spreadshirt-API erstellen
  const designResponse = await this.apiClient.post(
    `/shops/${this.shopId}/designs`,
    designPayload
  );

  const designId = designResponse.data.id;

  // 3. Checkout-URL generieren
  const checkoutUrl = `https://shop.spreadshirt.net/${this.shopId}` +
    `/product/${baseProductId}` +
    `?designId=${designId}` +
    `&size=${selectedSize}` +
    `&color=${selectedColor}`;

  return {
    designId,
    checkoutUrl,
    productSnapshot: { ... }
  };
}
```

3. Event-Query für visitedClubIds

```typescript
// Zentrale Stelle: packages/core/src/events/eventService.ts

export async function getUpcomingEventsForUser(
  userId: string,
  limitCount: number = 20
): Promise<Event[]> {
  const db = getFirestore();

  // 1. Lade User → visitedClubIds
  const userRef = doc(db, 'platform/users', userId);
  const userSnap = await getDoc(userRef);

  if (!userSnap.exists()) {
    return [];
  }

  const user = userSnap.data() as PlatformUser;
  const visitedClubIds = user.visitedClubIds || [];

  if (visitedClubIds.length === 0) {
    return []; // User hat noch keine Clubs besucht
  }

  // 2. Query Events (collectionGroup über alle Clubs)
  const now = Date.now();
  const events: Event[] = [];

  // Firestore 'in' unterstützt max. 10 Werte → Batch-Query
  for (let i = 0; i < visitedClubIds.length; i += 10) {
    const batch = visitedClubIds.slice(i, i + 10);

    const eventsQuery = query(
      collectionGroup(db, 'events'),        // Über alle clubs/{clubId}/events
      where('clubId', 'in', batch),         // Nur besuchte Clubs
      where('published', '==', true),       // Nur veröffentlichte Events
      where('startAt', '>', now),           // Nur zukünftige Events
      orderBy('startAt', 'asc'),
      limit(limitCount)
    );

    const snapshot = await getDocs(eventsQuery);
    snapshot.forEach((doc) => {
      events.push(doc.data() as Event);
    });
  }

  // 3. Sortieren & Limitieren
  events.sort((a, b) => a.startAt - b.startAt);
  return events.slice(0, limitCount);
}
```

Ende des Addendums

Status: ✅ Vollständig dokumentiert

Nächste Schritte:
1. Implementation in Phasen (empfohlen):
- Phase 1: User-Code & QR-System
- Phase 2: Merch-Integration (Spreadshirt)
- Phase 3: Ticketing (Stripe Connect)
- Phase 4: Event-Kalender

1. Testing:
   - Stripe Connect Onboarding testen (Test-Mode)
   - Spreadshirt API testen (Sandbox)
   - QR-Code-Generierung testen
   - Event-Query Performance testen (mit vielen besuchten Clubs)

2. Legal:
   - AGB erweitern (Provision-Modell)
   - Datenschutzerklärung erweitern (Spreadshirt, Stripe)
   - Impressum prüfen (Händler-Status bei Merch)

Dokumentations-Version: 1.0
Letztes Update: 3. Dezember 2025
Autor: DeepAgent für Nightlife OS