🔄 TAILORMARKET - STRATEGIC REFACTORING ROADMAP

Version: 2.0 - Pivot to Premium Suit Brand **Aktueller Stand:** Beta Testing Phase (generische Plattform) **Ziel:** Transformation zu fokussierter Maßanzug-Marke mit Vietnam-Fokus **Geschätzte Dauer:** 6-8 Wochen Refactoring **Letztes Update:** 2025-12-21

📋 EXECUTIVE SUMMARY - WAS ÄNDERT SICH?

Strategische Änderungen (aus Business-Gespräch)

VON (Alt):

- ❌ Generischer Marktplatz für alle Kleidungsstücke
- ❌ Schneider aus verschiedenen Ländern
- ❌ Schneider bestimmen eigene Preise
- ❌ Jeder Schneider uploaded eigenes Sortiment
- ❌ Neutraler Marktplatz-Approach

ZU (Neu):

- ✅ **Fokus: NUR Maßanzüge**
- ✅ **Land: NUR Vietnam (Phase 1)**
- ✅ **Marke first, Plattform second**
- ✅ **Zentrale Preissteuerung (550-750€ pro Anzug)**
- ✅ **Kuratierte Stoff-Bibliothek (10-20 Stoffe)**
- ✅ **3-5 Standard-Anzugmodelle**
- ✅ **Klare Markenidentität**

🎯 REFACTORING PHILOSOPHY

Prinzipien für Claude Code:

1. **Nicht alles wegwerfen**
    - Auth, Zahlungssystem, Order Management → behalten
    - UI Components → behalten
    - Datenbank-Struktur → anpassen, nicht neu
2. **Schrittweise Migration**
    - Alte Daten kompatibel halten
    - Feature-Flags nutzen
    - Keine Breaking Changes auf einen Schlag
3. **Marken-Orientierung**
    - Alle UI-Texte: "Maßanzug", nicht "Produkt"
    - Vietnam-Story überall sichtbar
    - Premium-Positionierung im Design

📊 BESTANDSAUFNAHME - WAS IST DA?

Aktuell funktioniert (✅ behalten):

- [x] Next.js 16 + React 19 + TypeScript Setup
- [x] Supabase Auth + Database + RLS
- [x] Prisma ORM
- [x] shadcn/ui Components
- [x] Stripe Connect Integration
- [x] Order Management System
- [x] Review System
- [x] Email Templates (Resend)
- [x] Shopping Cart
- [x] Measurement Tool (Mock/Manual Provider)
- [x] Deployment auf Vercel

Muss angepasst werden (🔄):

- [x] Database Schema (Product Model)
- [x] UI/UX (zu generisch)
- [x] Pricing Logic (aktuell frei)
- [x] Product Categories (zu viele)
- [x] Tailor Onboarding (keine Stoff-Bibliothek)
- [x] Marketing Copy (nicht fokussiert)

Muss neu (➕):

- [ ] Suit Configuration System (Modelle, Stoffe)
- [ ] Fabric Library Management
- [ ] Vietnam-Branding
- [ ] Price Control System
- [ ] Suit-spezifische Measurement UI

🗺️ REFACTORING ROADMAP - PHASEN

PHASE R1: DATABASE & DATA MODEL REFACTORING (Woche 1)

Ziel: Datenbank-Schema an Anzug-Fokus anpassen, ohne Downtime

R1.1 Product Model erweitern

Status: [ ] Todo **Dauer:** 3-4h **Dateien:** `prisma/schema.prisma`

Aufgabe: Erweitere das Product Model um Anzug-spezifische Felder.

Prompt für Claude Code:

```
Erweitere das Product Model in prisma/schema.prisma für Maßanzüge:

NEU hinzufügen:
- suitModel: String (z.B. "Classic", "Business", "Premium")
- fabricId: String (Relation zur Fabric Bibliothek)
- fitType: String (z.B. "Slim Fit", "Regular Fit", "Relaxed Fit")
- lapelStyle: String (z.B. "Notch", "Peak", "Shawl")
- ventStyle: String (z.B. "Single Vent", "Double Vent", "No Vent")
- buttonCount: Int (default: 2)
- pocketStyle: String (z.B. "Flap", "Patch", "Welted")

BESTEHEND anpassen:
- category: enum auf ["suit"] einschränken (später)
- description: Text anpassen an Anzug-Kontext

Migration erstellen:
npx prisma migrate dev --name add_suit_specific_fields

WICHTIG:
- Bestehende Produkte NICHT löschen
- Neue Felder nullable oder mit Defaults
- RLS Policies anpassen falls nötig
```

Test Checklist:

- [ ] Migration läuft ohne Fehler
- [ ] Bestehende Daten bleiben intakt
- [ ] Neue Felder in Prisma Client verfügbar

R1.2 Fabric Library Model erstellen

Status: [ ] Todo **Dauer:** 2-3h **Dateien:** prisma/schema.prisma

Aufgabe: Erstelle ein zentrales Fabric-Modell für kuratierte Stoffe.

Prompt für Claude Code:

```
Erstelle ein neues Fabric Model in prisma/schema.prisma:

model Fabric {
  id           String    @id @default(cuid())
  name         String    // z.B. "Navy Blue Wool 120s"
  description  String?
  material     String    // z.B. "100% Wool", "Wool/Cashmere Blend"
  weight       String?   // z.B. "260g/m²"
  pattern      String?   // z.B. "Solid", "Pinstripe", "Check"
  color        String    // z.B. "Navy Blue"
  season       String?   // z.B. "All Season", "Summer", "Winter"
  imageUrl     String?
  priceCategory String    // z.B. "standard", "premium", "luxury"
  priceAdd     Float     @default(0) // Aufpreis zu Basispreis
  isActive     Boolean   @default(true)
  position     Int       @default(0) // Sortierung
  createdAt    DateTime  @default(now())
  updatedAt    DateTime  @updatedAt
  products     Product[]

  @@index([isActive, position])
}

Aktualisiere Product Model:
- Füge Relation hinzu:
  fabric   Fabric? @relation(fields: [fabricId], references: [id])
  fabricId String?

Migration:
npx prisma migrate dev --name add_fabric_library

RLS Policy für Fabric:
- SELECT: true (public, alle können Stoffe sehen)
- INSERT/UPDATE/DELETE: nur Admin (später implementieren)
```

Test Checklist:

- [ ] Fabric Model erstellt
- [ ] Relation zu Product funktioniert
- [ ] Dummy Fabrics können angelegt werden

R1.3 Suit Model Categories festlegen

Status: [ ] Todo **Dauer:** 2h **Dateien:** `lib/constants/suit-models.ts` (neu), `types/suit.ts` (neu)

Aufgabe: Definiere die 3-5 Standard-Anzugmodelle als TypeScript Constants.

Prompt für Claude Code:

Erstelle neue Datei: lib/constants/suit-models.ts

WICHTIG: Diese Datei definiert die EINZIGEN 3-5 Anzugmodelle, die es gibt.

```
export const SUIT_MODELS = [
  {
    id: 'classic',
    name: 'Classic Suit',
    description: 'Zeitloser Business-Anzug mit Regular Fit. Perfekt für formelle Anlässe.',
    basePrice: 590, // EUR
    features: [
      'Regular Fit',
      'Notch Lapel',
      'Single Vent',
      '2 Buttons',
      'Flap Pockets'
    ],
    imageUrl: '/suits/classic.jpg', // Placeholder
    isActive: true
  },
  {
    id: 'business',
    name: 'Business Suit',
    description: 'Moderner Business-Anzug mit Slim Fit. Elegant und zeitgemäß.',
    basePrice: 650,
    features: [
      'Slim Fit',
      'Peak Lapel',
      'Double Vent',
      '2 Buttons',
      'Welted Pockets'
    ],
    imageUrl: '/suits/business.jpg',
    isActive: true
  },
  {
    id: 'premium',
    name: 'Premium Suit',
    description: 'Exklusiver Maßanzug mit luxuriösen Details und erstklassiger Verarbeitung.',
    basePrice: 750,
    features: [
      'Custom Fit',
      'Choice of Lapel Style',
      'Choice of Vent Style',
      '1-3 Buttons',
      'Premium Details'
    ],
    imageUrl: '/suits/premium.jpg',
    isActive: true
```

```
  }
] as const;

export type SuitModelId = 'classic' | 'business' | 'premium';

Erstelle auch: types/suit.ts

export interface SuitConfiguration {
  modelId: SuitModelId;
  fabricId: string;
  fitType: 'slim' | 'regular' | 'relaxed';
  lapelStyle: 'notch' | 'peak' | 'shawl';
  ventStyle: 'single' | 'double' | 'none';
  buttonCount: 1 | 2 | 3;
  pocketStyle: 'flap' | 'patch' | 'welted';
  measurements: SuitMeasurements;
}

export interface SuitMeasurements {
  // Von Measurement Tool
  chest: number;
  waist: number;
  hips: number;
  shoulderWidth: number;
  sleeveLength: number;
  jacketLength: number;
  pantWaist: number;
  pantInseam: number;
  pantOutseam: number;
  unit: 'cm' | 'inch';
}
```

Test Checklist:

- [ ] Constants sind korrekt definiert
- [ ] Types können importiert werden
- [ ] 3 Modelle sind klar unterscheidbar

PHASE R2: PRICING & BUSINESS LOGIC (Woche 1-2)

Ziel: Zentrale Preissteuerung implementieren

R2.1 Price Calculation Engine

Status: [ ] Todo **Dauer:** 4-5h **Dateien:** `lib/pricing/suit-pricing.ts` (neu)

Aufgabe: Erstelle eine zentrale Preisberechnungs-Logik.

Prompt für Claude Code:

Erstelle neue Datei: lib/pricing/suit-pricing.ts

Diese Datei berechnet den FINALEN Endkundenpreis eines Anzugs.

```
import { SUIT_MODELS } from '@/lib/constants/suit-models';
import { prisma } from '@/lib/db';

interface PriceCalculationInput {
  suitModelId: string;
  fabricId: string;
  customizations?: {
    lining?: boolean; // +50€
    monogram?: boolean; // +30€
    extraTrousers?: boolean; // +120€
  };
}

interface PriceBreakdown {
  basePrice: number;        // z.B. 650€ für Business Suit
  fabricAdd: number;        // z.B. +100€ für Premium-Stoff
  customizationAdd: number; // z.B. +80€ für Extras
  totalPrice: number;       // Final price für Kunde

  // Aufteilung intern (für uns, nicht für Kunde sichtbar)
  tailorShare: number;      // 60% von totalPrice
  platformFee: number;      // 25% von totalPrice
  riskBuffer: number;       // 15% von totalPrice
}

export async function calculateSuitPrice(
  input: PriceCalculationInput
): Promise<PriceBreakdown> {
  // 1. Hole Suit Model Base Price
  const model = SUIT_MODELS.find(m => m.id === input.suitModelId);
  if (!model) throw new Error('Invalid suit model');

  const basePrice = model.basePrice;

  // 2. Hole Fabric Price Add
  const fabric = await prisma.fabric.findUnique({
    where: { id: input.fabricId }
  });
  if (!fabric) throw new Error('Invalid fabric');

  const fabricAdd = fabric.priceAdd;

  // 3. Berechne Customizations
  let customizationAdd = 0;
  if (input.customizations?.lining) customizationAdd += 50;
```

```
  if (input.customizations?.monogram) customizationAdd += 30;
  if (input.customizations?.extraTrousers) customizationAdd += 120;

  // 4. Total Price
  const totalPrice = basePrice + fabricAdd + customizationAdd;

  // 5. Aufteilung (fix, nicht verhandelbar)
  const tailorShare = Math.round(totalPrice * 0.60);
  const platformFee = Math.round(totalPrice * 0.25);
  const riskBuffer = Math.round(totalPrice * 0.15);

  return {
    basePrice,
    fabricAdd,
    customizationAdd,
    totalPrice,
    tailorShare,
    platformFee,
    riskBuffer
  };
}

WICHTIG:
- Schneider sehen NUR tailorShare, nicht die Aufteilung
- Kunde sieht NUR totalPrice
- Prozentsätze sind fix (später konfigurierbar machen)
```

Test Checklist:

- [ ] Preisberechnung funktioniert
- [ ] Aufteilung ergibt immer 100%
- [ ] Edge Cases getestet (ungültige IDs)

R2.2 Admin-Seite für Fabric Library

Status: [ ] Todo **Dauer:** 5-6h **Dateien:** `app/(admin)/admin/fabrics/*` (neu)

Aufgabe: Erstelle Admin-Interface zum Verwalten der Stoff-Bibliothek.

Prompt für Claude Code:

Erstelle Admin-Sektion für Fabric Management:

Struktur:
app/(admin)/
  admin/
    fabrics/
      page.tsx   # Übersicht aller Fabrics
      new/
        page.tsx   # Neuen Stoff anlegen
      [id]/
        edit/
          page.tsx   # Stoff bearbeiten

app/(admin)/layout.tsx # Admin Layout mit Auth Check

WICHTIG:
- Nur für Admin-User (role: 'admin')
- Redirect wenn nicht admin
- Tabelle mit allen Fabrics:
  - Name, Material, Color, Price Add, Active Status
  - Edit/Delete Actions
  - Sortierung per Drag & Drop (position field)

Form für Fabric:
- Name (Text)
- Description (Textarea)
- Material (Text)
- Weight (Text, optional)
- Pattern (Select: Solid, Pinstripe, Check, Herringbone)
- Color (Text)
- Season (Select: All Season, Summer, Winter, Spring/Fall)
- Price Category (Select: standard, premium, luxury)
- Price Add (Number, EUR)
- Image Upload (Supabase Storage)
- Active (Checkbox)

API Routes:
- GET /api/admin/fabrics
- POST /api/admin/fabrics
- PATCH /api/admin/fabrics/[id]
- DELETE /api/admin/fabrics/[id]

Auth Check in allen Routes:
- Prüfe auth.uid()
- Prüfe User.role === 'admin'
- Return 403 wenn nicht authorized

shadcn/ui Components nutzen:
- Table

- Form
- Input
- Select
- Checkbox
- Button
- Dialog (für Delete Confirm)

Test Checklist:

- [ ] Nur Admin kann zugreifen
- [ ] CRUD funktioniert für Fabrics
- [ ] Image Upload funktioniert
- [ ] Sortierung funktioniert

PHASE R3: UI/UX REFACTORING - BRAND FIRST (Woche 2-3)

Ziel: Generische Plattform → fokussierte Anzug-Marke

R3.1 Brand Identity & Naming

Status: [ ] Todo **Dauer:** 3-4h **Dateien:** `lib/constants/brand.ts`, diverse UI Components

Aufgabe: Definiere klare Marken-Identity und ersetze generische Texte.

Prompt für Claude Code:

Erstelle lib/constants/brand.ts:

```
export const BRAND = {
  name: 'TailorMarket',
  tagline: 'Maßanzüge aus Vietnam. Fair. Hochwertig. Erschwinglich.',
  mission: 'Wir verbinden talentierte Schneider aus Vietnam mit Menschen, die Qualität und Fairness schätzen.',

  values: [
    {
      title: 'Fairness',
      description: 'Schneider erhalten 60% des Verkaufspreises – deutlich über dem Marktstandard.'
    },
    {
      title: 'Qualität',
      description: 'Jeder Anzug wird von erfahrenen Schneidern mit 10+ Jahren Erfahrung gefertigt.'
    },
    {
      title: 'Transparenz',
      description: 'Du siehst genau, wer deinen Anzug fertigt und was er dafür verdient.'
    }
  ],

  pricing: {
    min: 550,
    max: 750,
    average: 650,
    savingsVsLocal: '50-70%'
  },

  vietnam: {
    why: 'Vietnam hat eine jahrhundertelange Schneidertradition und ist bekannt für erstklassige Handwerkskunst.',
    quality: 'Vietnamesische Schneider fertigen auch für internationale Luxusmarken.',
    fairness: 'Faire Bezahlung in Vietnam bedeutet: Ein Schneider verdient 3-4x lokales Durchschnittseinkommen.'
  }
} as const;
```

DANN: Ersetze überall in der App:

VORHER:
- "Produkte" → "Maßanzüge"
- "Artikel" → "Anzüge"
- "Kategorie" → "Anzugmodell"
- "Schneider entdecken" → "Unsere Schneider"
- "Jetzt kaufen" → "Maßanzug konfigurieren"

NACHHER:
- Überall anzug-spezifische Sprache
- Vietnam erwähnen wo sinnvoll
- Fairness-Message hervorheben

Dateien aktualisieren:
- app/(marketplace)/page.tsx (Homepage)
- app/components/layout/Header.tsx
- app/components/layout/Footer.tsx
- app/(marketplace)/products/page.tsx → umbenennen zu suits/page.tsx
- app/(marketplace)/tailors/page.tsx (Texte anpassen)

Test Checklist:

- [ ] Keine generischen "Produkt"-Texte mehr
- [ ] Vietnam-Story ist sichtbar
- [ ] Marken-Werte sind kommuniziert

R3.2 Homepage Refactoring

Status: [ ] Todo **Dauer:** 4-5h **Dateien:** app/(marketplace)/page.tsx

Aufgabe: Homepage zu fokussierter Anzug-Landingpage umbauen.

Prompt für Claude Code:

Komplett überarbeiten: app/(marketplace)/page.tsx

NEUE STRUKTUR:

1. Hero Section
   - Headline: "Dein Maßanzug aus Vietnam. Fair gefertigt. Perfekt sitzt er."
   - Subline: "Hochwertige Handarbeit von erfahrenen Schneidern – zu 50-70% günstigeren Preisen als in Deutschland."
   - CTA: "Anzug konfigurieren" (Link zu /suits/configure)
   - Background: Großes Bild (Schneider bei der Arbeit)

2. Trust Signals
   - "100% Maßanfertigung"
   - "Faire Bezahlung garantiert"
   - "14 Tage Rückgaberecht"
   - "Passform-Garantie"

3. Wie es funktioniert (3 Steps)
   - 1. Modell & Stoff wählen
   - 2. Maße digital erfassen
   - 3. Anzug wird in Vietnam gefertigt & geliefert

4. Warum Vietnam?
   - Text aus BRAND.vietnam
   - Bilder von Werkstätten
   - Testimonial eines Schneiders

5. Preistransparenz
   - "Ein Maßanzug bei uns: 550-750€"
   - "Vergleich Deutschland: 1.200-2.500€"
   - Breakdown zeigen (optional):
     * 60% gehen an Schneider
     * 25% Plattform & Logistik
     * 15% Qualitätssicherung

6. Unsere Schneider (Preview)
   - Grid mit 3-4 Schneider-Cards
   - Link zu "Alle Schneider"

7. Social Proof
   - Reviews (wenn vorhanden)
   - "Bereits 47 Anzüge gefertigt" (dynamisch)

8. Final CTA
   - "Jetzt deinen Maßanzug konfigurieren"

Design:
- Clean, minimalistisch
- Viel Weißraum
- Große, emotionale Bilder

- Premium-Feel (nicht billig!)
- Responsive

shadcn/ui nutzen:
- Button
- Card
- Badge

Test Checklist:

- [ ] Homepage fühlt sich premium an
- [ ] Vietnam-Story ist prominent
- [ ] CTAs sind klar
- [ ] Mobile optimiert

R3.3 Suit Configuration Flow (KERN-FEATURE)

Status: [ ] Todo **Dauer:** 8-10h **Dateien:** `app/(marketplace)/suits/configure/*` (neu)

Aufgabe: Erstelle den Haupt-Flow für Anzug-Konfiguration.

Prompt für Claude Code:

WICHTIG: Dies ist das Herzstück der neuen Plattform!

Erstelle Multi-Step Configuration Flow:

Struktur:
app/(marketplace)/suits/
  configure/
    page.tsx   # Step 1: Modell wählen
    [modelId]/
      fabric/
        page.tsx   # Step 2: Stoff wählen
      measurements/
        page.tsx   # Step 3: Maße eingeben
      customizations/
        page.tsx   # Step 4: Extras wählen
      review/
        page.tsx   # Step 5: Zusammenfassung

FLOW DETAILS:

=== STEP 1: MODELL WÄHLEN ===
URL: /suits/configure

UI:
- Zeige alle 3 SUIT_MODELS
- Große Cards mit:
  * Bild
  * Name
  * Features (Liste)
  * Basispreis (z.B. "ab 590€")
- Auswahl → weiter zu Step 2

State Management:
- Nutze URL Params + React State
- Speichere Auswahl in localStorage (für Zurück-Navigation)

=== STEP 2: STOFF WÄHLEN ===
URL: /suits/configure/business/fabric (Beispiel für Business Model)

UI:
- Filter links:
  * Material (Wool, Wool/Cashmere, etc.)
  * Pattern (Solid, Pinstripe, Check)
  * Color
  * Season
  * Price Category
- Grid mit Fabric Cards:
  * Großes Bild
  * Name

  * Material + Weight
  * Pattern + Color
  * Preis-Aufschlag (z.B. "+100€")
- Auswahl → weiter zu Step 3

Laden:
- GET /api/fabrics?active=true
- Filtern clientseitig

=== STEP 3: MASSE EINGEBEN ===
URL: /suits/configure/business/measurements

UI:
- Integration vom bestehenden Measurement Tool
- ABER: Anzug-spezifische Maße:
  * Jacket: Chest, Waist, Shoulders, Sleeve Length, Jacket Length
  * Pants: Waist, Hips, Inseam, Outseam
- Einheit: cm (default) oder inch
- Hilfe-Icons mit Mess-Anleitungen

Optional:
- "Maße von vorheriger Bestellung übernehmen" (wenn logged in)

Validation:
- Alle Pflichtfelder
- Plausibilitäts-Checks (z.B. Chest > Waist)

=== STEP 4: CUSTOMIZATIONS ===
URL: /suits/configure/business/customizations

UI:
- Optional Extras:
  * Futter (Lining) - Checkbox, +50€
  * Monogramm - Checkbox + Text Input, +30€
  * Extra Hose - Checkbox, +120€
- Jede Option zeigt:
  * Beschreibung
  * Preis-Aufschlag
  * Bild/Icon

=== STEP 5: REVIEW & ADD TO CART ===
URL: /suits/configure/business/review

UI:
- Zusammenfassung ALLES:
  * Gewähltes Modell (Bild + Name)
  * Gewählter Stoff (Bild + Details)
  * Eingegebene Maße (Tabelle)
  * Customizations (Liste)

- Preis-Breakdown (transparent):
  * Basispreis: 650€
  * Stoff-Aufschlag: +100€
  * Customizations: +80€
  * ─────────────────
  * GESAMT: 830€

- CTA: "In den Warenkorb" (nutzt bestehenden Cart)

Navigation:
- Jeden Schritt zurück editieren können
- Progress Bar oben (1/5, 2/5, etc.)

State Management:
- Nutze React Context oder Zustand
- Speichere Config in localStorage
- Bei "In den Warenkorb":
  * Erstelle Product (virtuell, nicht in DB)
  * Füge zu Cart hinzu mit allen Config-Details

WICHTIG:
- Bestehende Cart-Logik NICHT ändern
- Config-Daten in OrderItem.customizations speichern (JSON)

Test Checklist:

- [ ] Flow ist komplett durchlaufbar
- [ ] Zurück-Navigation funktioniert
- [ ] Preis wird korrekt berechnet
- [ ] Config wird in Cart übernommen
- [ ] Mobile-optimiert

PHASE R4: TAILOR ONBOARDING REFACTORING (Woche 3-4)

Ziel: Schneider können sich nicht mehr "frei" registrieren, sondern werden kuratiert

R4.1 Tailor Application System

Status: [ ] Todo **Dauer:** 5-6h **Dateien:** `app/(public)/apply/page.tsx` (neu)

Aufgabe: Erstelle Bewerbungs-Formular für Schneider (nicht direkte Registrierung).

Prompt für Claude Code:

WICHTIG: Schneider sollen sich NICHT selbst registrieren können!

Erstelle:
1. Public Page: /apply (Bewerbungsformular)
2. Admin Page: /admin/applications (Bewerbungen verwalten)

=== BEWERBUNGSFORMULAR ===
app/(public)/apply/page.tsx

Formular:
- Name (Text)
- Email (Email)
- Phone (Text)
- Land (Select, default: Vietnam)
- Stadt (Text)
- Jahre Erfahrung (Number)
- Spezialisierung (Checkbox: Anzüge, Hemden, Hosen, Kleider, etc.)
- Portfolio-Links (Text, optional)
- Warum möchtest du bei uns mitmachen? (Textarea)
- Bilder hochladen (max 5, Beispiele deiner Arbeit)

Submit:
- POST /api/tailor-applications
- Speichert in neuer Tabelle: TailorApplication

Prisma Schema erweitern:

```
model TailorApplication {
  id              String   @id @default(cuid())
  name            String
  email           String
  phone           String
  country         String
  city            String
  yearsExperience Int
  specialties     String[] // JSON array
  portfolioLinks  String?
  motivation      String
  imageUrls       String[] // Supabase Storage URLs
  status          String   @default("pending") // pending, approved, rejected
  createdAt       DateTime @default(now())
  reviewedAt      DateTime?
  reviewedBy      String?  // Admin User ID
  notes           String?  // Admin notes
}
```

UI nach Submit:
- "Vielen Dank für deine Bewerbung! Wir melden uns innerhalb von 5 Werktagen."
- Email an Applicant (optional)

=== ADMIN APPLICATIONS PAGE ===
app/(admin)/admin/applications/page.tsx

Tabelle mit allen Applications:
- Name, Email, Land, Jahre Erfahrung, Status, Datum
- Filter nach Status (pending, approved, rejected)
- Sortierung

Detail-View (Modal oder separate Page):
- Alle Infos
- Portfolio-Bilder
- Actions:
  * Approve → erstellt Tailor Account + sendet Zugangsdaten
  * Reject → Status auf rejected, Email senden
  * Add Notes

Approve-Flow:
1. Erstelle User (random password generieren)
2. Erstelle Tailor Profile (mit Daten aus Application)
3. Sende Email mit Login-Daten
4. Update Application.status = "approved"

API Routes:
- GET /api/admin/applications
- PATCH /api/admin/applications/[id]/approve
- PATCH /api/admin/applications/[id]/reject

Auth Check:
- Nur Admin

Test Checklist:

- [ ] Bewerbung kann submitted werden
- [ ] Admin kann Bewerbungen sehen
- [ ] Approve erstellt Tailor Account
- [ ] Email wird versendet

R4.2 Tailor Dashboard anpassen

Status: [ ] Todo **Dauer:** 3-4h **Dateien:** `app/(tailor-dashboard)/*`

Aufgabe: Passe Tailor Dashboard an: Keine freie Produkt-Erstellung, sondern Fabric-Auswahl.

Prompt für Claude Code:

WICHTIG: Schneider können NICHT mehr beliebige Produkte hochladen!

Änderungen in Tailor Dashboard:

1. ENTFERNEN:
   - "Neues Produkt erstellen" Button
   - Product Management komplett

2. ERSETZEN durch:
   - "Meine verfügbaren Stoffe"
   - Schneider sehen die zentrale Fabric Library
   - Können angeben: "Ich kann diesen Stoff besorgen" (Checkbox)

3. NEUE Seite:
   app/(tailor-dashboard)/dashboard/fabrics/page.tsx

   UI:
   - Tabelle aller Fabrics
   - Spalte: "Verfügbar?" (Checkbox)
   - Wenn checked: "Ich kann diesen Stoff in X Tagen besorgen" (Number Input)

   Speichern in:
   model TailorFabric {
     id   String   @id @default(cuid())
     tailorId   String
     fabricId   String
     isAvailable   Boolean @default(false)
     daysToSource   Int?   // Wie lange braucht er, um Stoff zu besorgen
     notes   String?
     createdAt   DateTime @default(now())
     updatedAt   DateTime @updatedAt

     tailor   Tailor   @relation(fields: [tailorId], references: [id])
     fabric   Fabric   @relation(fields: [fabricId], references: [id])

     @@unique([tailorId, fabricId])
   }

4. Order Zuweisung (wichtig):
   - Wenn Kunde einen Anzug bestellt:
     * System findet Tailor, der diesen Fabric verfügbar hat
     * Priorität: kürzeste daysToSource
   - Vorerst: Manuelle Zuweisung durch Admin
   - Später: Automatische Zuweisung

5. Tailor Order View:
   - Zeigt zugewiesene Orders
   - Kann Status updaten
   - Sieht tailorShare (z.B. "390€ für dich")

- Sieht NICHT platformFee

Test Checklist:

- [ ] Tailor kann keine Produkte mehr erstellen
- [ ] Tailor kann Fabrics als verfügbar markieren
- [ ] Order View zeigt korrekte Infos

PHASE R5: CHECKOUT & PAYMENT ANPASSUNGEN (Woche 4-5)

Ziel: Checkout-Flow an Anzug-Kontext anpassen

R5.1 Checkout anpassen

Status: [ ] Todo **Dauer:** 4-5h **Dateien:** `app/checkout/page.tsx`

Aufgabe: Passe Checkout an Anzug-Bestellungen an.

Prompt für Claude Code:

Checkout Flow anpassen:

WICHTIG:
- Nutze bestehenden Stripe Checkout
- Nutze bestehende Order Creation
- ABER: UI-Texte anzug-spezifisch

Änderungen:

1. Cart Summary:
  - Nicht: "Produkt von Schneider X"
  - Sondern:
    * "Business Suit"
    * "Stoff: Navy Blue Wool 120s"
    * "Customizations: Lining, Monogramm 'NK'"
    * "Geschätzte Lieferzeit: 4-6 Wochen"

2. Zusätzliche Info-Box:
  - "Dein Anzug wird maßgefertigt von [Schneider Name] in [Stadt], Vietnam"
  - "Produktionsstart: Nach Zahlungseingang"
  - "Lieferung: Ca. 4-6 Wochen (inkl. Versand)"

3. Passform-Garantie Info:
  - "Passform-Garantie: Bis 100€ für lokale Anpassungen"
  - "Rückgaberecht: 14 Tage (Details in AGB)"

4. Order Creation erweitern:
  - OrderItem.customizations speichert:
    {
      suitModelId: "business",
      fabricId: "xyz",
      measurements: {...},
      customizations: {...},
      tailorId: "abc" // später: auto-assigned
    }

5. Email nach Bestellung:
  - Nicht generisch
  - Sondern:
    * "Dein Maßanzug wird gefertigt!"
    * "Schneider [Name] hat deine Bestellung erhalten"
    * "Maße: [Übersicht]"
    * "Nächster Schritt: Produktionsstart (1-2 Tage)"

Test Checklist:

- [ ] Checkout zeigt Anzug-Details
- [ ] Order wird mit Config gespeichert
- [ ] Email ist anzug-spezifisch

PHASE R6: CONTENT & MARKETING (Woche 5-6)

Ziel: Vietnam-Story, Fairness, Premium-Positionierung überall sichtbar

R6.1 Statische Seiten aktualisieren

Status: [ ] Todo **Dauer:** 4-5h **Dateien:** `app/(marketplace)/{about,how-it-works}/page.tsx`

Aufgabe: Content-Seiten neu schreiben mit Fokus auf Anzüge + Vietnam.

Prompt für Claude Code:

Überarbeite:

1. /about
   - Mission: Faire Maßanzüge aus Vietnam
   - Vision: Globaler Zugang zu Handwerkskunst
   - Werte: Fairness, Qualität, Transparenz
   - Team (optional)
   - Vietnam: Warum gerade Vietnam?

2. /how-it-works
   - Schritt 1: Modell & Stoff wählen
   - Schritt 2: Maße digital erfassen
   - Schritt 3: Bestellung & Zahlung
   - Schritt 4: Fertigung in Vietnam (4-6 Wochen)
   - Schritt 5: Lieferung & Passform-Check

3. NEU: /vietnam
   - Dedicated Page über Vietnam
   - Schneidertradition
   - Qualität & Handwerk
   - Faire Bezahlung Kontext
   - Bilder von Werkstätten

4. NEU: /quality
   - Qualitätsversprechen
   - Material-Info
   - Passform-Garantie Details
   - Anpassungs-Service Erklärung

5. Footer Links aktualisieren:
   - Über uns
   - Wie es funktioniert
   - Warum Vietnam?
   - Qualität
   - Schneider werden (→ /apply)
   - AGB, Datenschutz, Impressum

Test Checklist:

- [ ] Alle Seiten haben anzug-spezifischen Content
- [ ] Vietnam-Story ist stark
- [ ] Links im Footer funktionieren

R6.2 SEO & Meta Tags

Status: [ ] Todo **Dauer:** 2-3h **Dateien:** Diverse `page.tsx` Files

Aufgabe: Aktualisiere alle Meta Tags für SEO.

Prompt für Claude Code:

```
Aktualisiere Metadata in allen wichtigen Pages:

Beispiel für Homepage (app/(marketplace)/page.tsx):

export const metadata: Metadata = {
  title: 'TailorMarket – Maßanzüge aus Vietnam | Fair & Hochwertig',
  description: 'Hochwertige Maßanzüge von erfahrenen Schneidern aus Vietnam. 50-70% günstiger als in Deutschland.
Fair produziert. Perfekte Passform garantiert.',
  keywords: ['Maßanzug', 'Vietnam', 'Fair Fashion', 'Anzug maßgeschneidert', 'Schneider Vietnam'],
  openGraph: {
    title: 'TailorMarket – Dein Maßanzug aus Vietnam',
    description: 'Fair gefertigt. Perfekt sitzt er. 550-750€.',
    images: ['/og-image.jpg'],
  },
};

Für alle Pages:
- /suits/configure
- /tailors
- /about
- /how-it-works
- /vietnam
- etc.

WICHTIG:
- Fokus auf: Maßanzug, Vietnam, Fair, Hochwertig
- Preisbereich nennen (550-750€)
- USP kommunizieren
```

Test Checklist:

- [ ] Alle Pages haben spezifische Metadata
- [ ] OG Images vorhanden
- [ ] Keywords sind relevant

PHASE R7: DATA MIGRATION & CLEANUP (Woche 6)

Ziel: Alte Daten bereinigen, Fokus herstellen

R7.1 Product Data Migration

Status: [ ] Todo **Dauer:** 3-4h **Dateien:** `scripts/migrate-to-suits.ts` (neu)

Aufgabe: Bestehende Produkte entweder löschen oder zu Anzügen konvertieren.

Prompt für Claude Code:

Erstelle Migration Script: scripts/migrate-to-suits.ts

Optionen:
A) Alle alten Produkte löschen (wenn Beta-Daten nicht wichtig)
B) Produkte mit category="suit" behalten, Rest löschen
C) Alle Produkte zu Standard-Anzug konvertieren (als Placeholder)

Empfehlung: A (Clean Slate)

Script:

```
import { prisma } from '@/lib/db';

async function migrateToSuits() {
  console.log('Starting migration...');

  // 1. Alte Orders & OrderItems analysieren
  const oldOrders = await prisma.order.count();
  console.log(`Found ${oldOrders} existing orders`);

  if (oldOrders > 0) {
    console.log('WARNING: Existing orders found. Consider archiving before deleting products.');
    // Optional: Orders archivieren
  }

  // 2. Produkte löschen (außer category="suit")
  const deleted = await prisma.product.deleteMany({
    where: {
      category: {
        not: 'suit'
      }
    }
  });
  console.log(`Deleted ${deleted.count} non-suit products`);

  // 3. Remaining Suit Products: Update to new schema
  const suits = await prisma.product.findMany({
    where: { category: 'suit' }
  });

  for (const suit of suits) {
    await prisma.product.update({
      where: { id: suit.id },
      data: {
        suitModel: 'classic', // Default
        fitType: 'regular',
        lapelStyle: 'notch',
        ventStyle: 'single',
        buttonCount: 2,
```

```
        pocketStyle: 'flap'
      }
    });
  }
  console.log(`Updated ${suits.length} suit products`);

  console.log('Migration complete!');
}

migrateToSuits();

Ausführen:
npx tsx scripts/migrate-to-suits.ts
```

Test Checklist:

- [ ] Script läuft ohne Fehler
- [ ] Nur relevante Daten bleiben
- [ ] Suit Products haben neue Fields

R7.2 Seed Realistic Data

Status: [ ] Todo **Dauer:** 3-4h **Dateien:** `prisma/seed-suits.ts` (neu)

Aufgabe: Erstelle realistische Demo-Daten für Anzüge.

Prompt für Claude Code:

Erstelle Seed Script: prisma/seed-suits.ts

Inhalt:

1. Erstelle 10-15 Fabrics:
   - 5x Solid Colors (Navy, Charcoal, Black, Light Gray, Dark Gray)
   - 3x Pinstripe
   - 2x Check
   - Mix aus standard/premium/luxury

2. Erstelle 3-5 Demo Tailors:
   - Namen: Nguyen Van Anh, Tran Thi Mai, Le Hoang Nam, etc.
   - Land: Vietnam
   - Städte: Hanoi, Ho Chi Minh City, Da Nang
   - Bio: Realistisch, z.B. "15 Jahre Erfahrung, spezialisiert auf Business-Anzüge"
   - Verified: true
   - Rating: 4.5-5.0

3. Erstelle Admin User:
   - Email: admin@tailormarket.com
   - Password: (gehashed, temporär)
   - Role: admin

4. KEINE Demo Products!
   - Products werden nur über Config-Flow erstellt
   - Fabrics sind das Sortiment

Ausführen:
npx prisma db seed

Test Checklist:

- [ ] Seed läuft durch
- [ ] Fabrics sind sichtbar
- [ ] Tailors sind angelegt
- [ ] Admin Login funktioniert

PHASE R8: TESTING & QA (Woche 6-7)

Ziel: Gesamtes System durchtesten

R8.1 E2E Tests aktualisieren

Status: [ ] Todo **Dauer:** 4-5h **Dateien:** `tests/e2e/*`

Aufgabe: Playwright Tests an neuen Flow anpassen.

Prompt für Claude Code:

```
Aktualisiere E2E Tests für Suit Flow:

1. Homepage Test:
   - Prüfe: Headline enthält "Maßanzug"
   - Prüfe: CTA "Anzug konfigurieren" vorhanden
   - Prüfe: Vietnam erwähnt

2. Configuration Flow Test:
   - Step 1: Modell wählen (Business)
   - Step 2: Fabric wählen (erster verfügbarer)
   - Step 3: Measurements eingeben
   - Step 4: Customizations (keine)
   - Step 5: Review → Add to Cart
   - Prüfe: Cart zeigt korrekte Config
   - Checkout → Success

3. Tailor Application Test:
   - Öffne /apply
   - Fülle Formular aus
   - Submit
   - Prüfe Success Message

4. Admin Test:
   - Login als Admin
   - Öffne /admin/fabrics
   - Erstelle neuen Fabric
   - Prüfe in Frontend sichtbar

Tests laufen lassen:
npm run test:e2e

Erwartung:
- 100% Pass Rate
- Keine Console Errors
```

Test Checklist:

- [ ] Alle E2E Tests passen
- [ ] Tests grün
- [ ] Core-Flows funktionieren

R8.2 Manual QA Checklist

Status: [ ] Todo **Dauer:** 4-6h

Aufgabe: Manuelle Tests aller Flows.

QA CHECKLIST:

Customer Flow:

- [ ] Homepage lädt, Texte sind anzug-spezifisch
- [ ] "Anzug konfigurieren" Link funktioniert
- [ ] Step 1: Alle 3 Modelle werden angezeigt
- [ ] Step 2: Fabrics laden, Filter funktionieren
- [ ] Step 3: Measurements Form validiert korrekt
- [ ] Step 4: Customizations add Preis korrekt
- [ ] Step 5: Review zeigt alle Daten, Preis stimmt
- [ ] Add to Cart funktioniert
- [ ] Cart zeigt Config Details
- [ ] Checkout → Stripe → Success
- [ ] Order in Dashboard sichtbar
- [ ] Email erhalten

Tailor Flow:

- [ ] Kann sich NICHT direkt registrieren
- [ ] /apply Formular funktioniert
- [ ] Application in Admin sichtbar
- [ ] Admin kann approven → Tailor erhält Zugang
- [ ] Tailor Login funktioniert
- [ ] Fabric Management: Kann Fabrics als verfügbar markieren
- [ ] Sieht zugewiesene Orders
- [ ] Kann Order Status updaten

Admin Flow:

- [ ] Admin Login
- [ ] Fabric CRUD funktioniert
- [ ] Applications Management funktioniert
- [ ] Approve/Reject sendet Emails

Content:

- [ ] /about zeigt Vietnam-Story
- [ ] /how-it-works erklärt Prozess
- [ ] /vietnam Page funktioniert
- [ ] Footer Links alle korrekt

Mobile:

- [ ] Homepage responsive
- [ ] Config Flow auf Mobile bedienbar
- [ ] Checkout auf Mobile funktioniert

PHASE R9: DEPLOYMENT & ROLLOUT (Woche 7-8)

Ziel: Neue Version live schalten

R9.1 Environment Vorbereitung

Status: [ ] Todo **Dauer:** 2-3h

Aufgabe: Produktions-Environment vorbereiten.

CHECKLIST:

Vercel:

- [ ] Environment Variables aktualisiert
- [ ] Build Test erfolgreich
- [ ] Preview Deployment getestet

Supabase:

- [ ] Migrations auf Production ausgeführt
- [ ] RLS Policies aktualisiert
- [ ] Seed Data (Fabrics, Admin) importiert

Stripe:

- [ ] Test Mode: Alles funktioniert
- [ ] Live Mode: Keys vorbereitet (nicht aktivieren vor Launch)

Email:

- [ ] Resend Templates aktualisiert
- [ ] Test-Emails versendet

R9.2 Soft Launch

Status: [ ] Todo **Dauer:** 1 Woche

Aufgabe: Schrittweise Rollout mit Feedback-Loop.

PLAN:

Tag 1-2: Internal Testing

- Team testet komplett
- Bugs fixen

Tag 3-4: Beta Tester (Bekannte)

- 10-20 Personen einladen
- Feedback sammeln
- Kritische Bugs fixen

Tag 5-7: Erste echte Kunden

- Invite-Only
- Stripe Live Mode aktivieren
- Monitoring intensiv

Tracking:

- Plausible Analytics aktiviert
- Conversion Funnel beobachten:
  - Homepage → Config Start
  - Config Start → Add to Cart
  - Cart → Checkout
  - Checkout → Success

📊 FORTSCHRITT TRACKING

Refactoring Phasen:

- [ ] R1: Database & Data Model (0/3 Steps)
- [ ] R2: Pricing & Business Logic (0/2 Steps)
- [ ] R3: UI/UX Refactoring (0/3 Steps)
- [ ] R4: Tailor Onboarding (0/2 Steps)
- [ ] R5: Checkout Anpassungen (0/1 Steps)
- [ ] R6: Content & Marketing (0/2 Steps)
- [ ] R7: Data Migration (0/2 Steps)
- [ ] R8: Testing & QA (0/2 Steps)
- [ ] R9: Deployment (0/2 Steps)

Gesamtfortschritt: 0/20 Steps (0%)

🎯 NEXT STEPS - WO ANFANGEN?

Empfohlene Reihenfolge:

1. **WOCHE 1: Database Foundation**
   - R1.1 → R1.2 → R1.3
   - Dann: R2.1 (Pricing Engine)
   - Ziel: Daten-Modell steht
2. **WOCHE 2: Core Flow**
   - R3.3 (Configuration Flow) - DAS HERZSTÜCK
   - R2.2 (Fabric Admin) parallel
   - Ziel: Flow ist nutzbar
3. **WOCHE 3: UI Polish**
   - R3.1 → R3.2 (Brand + Homepage)
   - R6.1 (Content Pages)
   - Ziel: Sieht nach Marke aus
4. **WOCHE 4-5: Ecosystem**
   - R4.1 → R4.2 (Tailor System)
   - R5.1 (Checkout)
   - Ziel: Komplettes System funktioniert
5. **WOCHE 6: Cleanup**
   - R7.1 → R7.2 (Migration + Seed)
   - R8.1 → R8.2 (Testing)
   - Ziel: Production-ready
6. **WOCHE 7-8: Launch**
   - R9.1 → R9.2
   - Ziel: Live

💡 KRITISCHE HINWEISE FÜR CLAUDE CODE

Beim Refactoring beachten:

1. **NIEMALS** alte Daten einfach löschen ohne Backup
2. **IMMER** migrations testen bevor production
3. **FEATURE FLAGS** nutzen wenn möglich (für schrittweisen Rollout)
4. **BACKWARDS COMPATIBILITY** so lange wie möglich erhalten
5. **TESTS** schreiben BEVOR du refactorst

Bei Unklarheiten:

- Frag nach bei komplexen Business-Entscheidungen
- Dokumentiere Annahmen im Code (Kommentare)
- Erstelle TODO-Listen für offene Punkte

Performance:

- Bestehende Optimizations NICHT kaputt machen
- Neue Queries indexieren
- Image Optimization beibehalten

📎 REFERENZEN

Wichtige Dokumente:

- `ROADMAP.md` (alte Roadmap - Referenz)
- `specs.md` (Tech Specs - weiterhin gültig)
- `PLATFORM_LOGIC.md` (falls vorhanden)
- Business-Gespräch PDF (strategische Grundlage)

Externe Dependencies:

- Stripe Connect Docs: https://stripe.com/docs/connect
- Supabase RLS: https://supabase.com/docs/guides/auth/row-level-security
- Prisma Migrations: https://www.prisma.io/docs/concepts/components/prisma-migrate

---

ENDE DER REFACTORING ROADMAP

Nächster Schritt: Beginne mit Phase R1.1 (Database Schema erweitern)

Bei Fragen: Referenziere dieses Dokument und frage spezifisch nach einzelnen Steps.