Sprint 5: Affiliate-Marktplatz – detaillierte Roadmap

Zeitraum: 8.–12. Sep 2025 (5 Arbeitstage)

Ziel: Monetarisierung durch einen integrierten Shop pro Studio (Gym) in der App. Produkte können vom Studio-Admin verwaltet und von Mitgliedern gekauft werden. Die Abwicklung (Zahlung, Bestellhistorie) wird über Stripe realisiert. Nach Abschluss dieses Sprints kann das Team sich auf das Lizenzmodell (Sprint 6) konzentrieren.

1. Voraussetzungen und Kontext

- **Vorherige Sprints:** Dashboard ist online (Sprint 4), 3D-Heatmap funktioniert (Sprint 3), Offline-Sync stabil (Sprint 2). Die App verfügt über ein Branding-System und eine Admin-Rolle.
- **Bestehende Infrastruktur:** Firebase Firestore und Storage; Cloud Functions sind einsatzbereit; Auth-System unterstützt Custom Claims (Admin).
- **Externe Dienste:** Stripe-Konto muss eingerichtet sein; es sollen API-Keys (Publishable & Secret Key) sicher als GitHub-Secrets oder in Firebase Config hinterlegt werden.
- **Rechtliche Vorbereitung:** Abklärung von Steuerfragen, Datenschutz (DSGVO) und Widerrufsrecht wird parallel von einem Juristen durchgeführt (nicht Teil des Codes).
- **Team:** Zwei Frontend-Entwickler (Flutter), ein Backend-Entwickler für Stripe-Integration, ein QA-Engineer, optional UI/UX-Designer.

2. Aufgabenplan (Tag-für-Tag)

Tag 1 (Mo 8. Sep 2025) – Anforderungsdefinition & Datenmodell

Workshop mit Stakeholdern (AM):

- Welche Produktkategorien sollen verfügbar sein? (Bekleidung, Merchandise, Supplements).
- Preisgestaltung und Provision: Prozentualer Anteil für App-Betreiber vs. Studio.
- Versandoptionen: Abholung im Studio vs. Versand (inkl. Lieferkosten).
- Rückgabe/Erstattung: Prozess definieren, Schnittstelle zur Juristischen Abteilung.

Datenmodell entwerfen (AM):

- **Firestore-Sammlungen und Dokumente:**
  - `gyms/{gymId}/products/{productId}`
    Felder: `name, description, price, currency, stock, imageUrl, category, active, createdAt, updatedAt, taxRate, provisionRate`.
    Optional: `discount` (Prozentsatz oder Betrag), `tags`.
  - `gyms/{gymId}/orders/{orderId}`
    Felder: `userId, items` (List von Produkt-IDs + Menge + Preis zum Kaufzeitpunkt), `totalAmount, paymentStatus` (pending, paid, failed), `createdAt, fulfilledAt, shippingAddress, deliveryOption`.
  - `users/{userId}/orders/{orderId}` (optional Redundanz für schnelle Anzeige in Profil).
- **Storage:** Produktbilder in Cloud Storage unter `products/{gymId}/{productId}.jpg`.

Backend-API-Design (PM):

- **Product CRUD:** Via Firestore (Admins dürfen schreiben).
- **Order Placement:** Cloud Function `createOrder` (geschützt), die eine Bestellung anlegt, einen Stripe PaymentIntent erstellt und den `client_secret` zurückgibt.
- **Webhook-Endpoint:** Cloud Function `stripeWebhook` verarbeitet Stripe Events (`PaymentIntent succeeded/failed`) und aktualisiert `paymentStatus` der Bestellung.
- **Security:** Firestore-Regeln erweitern, sodass nur Admins Produkte erstellen/ändern und nur der Käufer seine Bestellungen lesen kann.

Stripe-Setup:

- Erstellen eines Test-Accounts in Stripe; Produkte sind nicht in Stripe gespeichert (Inventar wird in Firestore geführt), aber Zahlungen laufen über Stripe PaymentIntents.
- Hinterlegen der API-Keys in Firebase Functions Config (`stripe_secret_key, stripe_webhook_secret`).

Tag 2 (Di 9. Sep 2025) – Datenmodell implementieren & UI-Grundgerüst

Firestore-Implementierung (AM):

- Erstellung der Collections in Firestore; Anlegen eines Test-Product-Dokuments via Firebase Console oder Seed-Script.
- Anpassung der Firestore-Regeln:

```
match /gyms/{gymId}/products/{productId} {
  allow read: if true; // alle Nutzer können Produkte sehen
  allow write: if request.auth != null && request.auth.token.gym_id == gymId &&
    ↪ request.auth.token.role == 'admin';
}
match /gyms/{gymId}/orders/{orderId} {
  allow read, write: if request.auth != null && request.auth.token.gym_id == gymId
    ↪ && request.auth.uid == resource.data.userId;
}
```

- Erweiterung der AuthProvider um eine Methode `isAdmin()` zur besseren Abfrage im Frontend.

Model-Klassen & Provider (AM):

- **Product-Klasse** mit genannten Feldern; **Order-Klasse** inklusive Liste von `OrderItem`.
- **ProductProvider:** Laden aller aktiven Produkte eines Gyms; Suchen nach Kategorie/Tag; Caching.
- **CartProvider:** Verwalten eines Warenkorbs (`Map productId -> quantity`); Berechnung von Gesamtsumme; Methoden zum Hinzufügen/Entfernen.
- **OrderProvider:** Erstellung einer Bestellung (Kommunikation mit Cloud Function), Abfrage des Bestellstatus.

UI-Grundgerüst (PM):

- Neues Tab „Affiliate“ in der Haupt-App erweitern (derzeit Platzhalter).
- **Produktliste:** Grid- oder ListView, Darstellung von Bild, Name, Preis, „In den Warenkorb“-Button.

- **Produktdetail:** Separate Seite mit Beschreibung, großem Bild, Preis, Auswahlelement für Menge, „Hinzufügen“-Button.
- **Warenkorb-Seite:** Anzeige aller ausgewählten Produkte, Summenberechnung, Button „Zur Kasse“.
- Navigationsfluss festlegen: Affiliate-Tab → Produktliste → Detail → Warenkorb → Checkout.

Tag 3 (Mi 10. Sep 2025) – Stripe-Checkout & Zahlungsabwicklung

Cloud Function createOrder implementieren (AM):

- Eingabe: `gymId, userId, cartItems` (Liste von `productId + quantity`).
- Überprüfung: Prüfen, ob Produkte existieren, `active == true` und Lagerbestand ausreichend ist; Gesamtbetrag berechnen (inkl. Steuern, Rabatt).
- Stripe-PaymentIntent erstellen (`stripe.paymentIntents.create`); Betrag in der kleinsten Währungseinheit (z. B. Cent) übergeben; Beschreibung (z. B. „Tap’em Shop Purchase“).
- Bestellung in `gyms/{gymId}/orders/{orderId}` mit `paymentStatus = 'pending'` speichern und `clientSecret` zurückgeben.

Client-Integration (PM):

- Installation des Stripe Flutter SDK (`flutter_stripe`).
- In `CheckoutPage`: Nach Klick auf „Zur Kasse“ → Cloud Function `createOrder` aufrufen; `clientSecret` empfangen.
- Aufruf von `Stripe.instance.initPaymentSheet()` mit dem erhaltenen `clientSecret` und PublishableKey; anschließend `presentPaymentSheet()`.
- Erfolgs- und Fehlerbehandlung implementieren: Bei Erfolg Snackbar „Bezahlung erfolgreich“; bei Fehler den Warenkorb intakt lassen und Fehlermeldung anzeigen.

Webhook-Handling (PM):

- Cloud Function `stripeWebhook` registrieren und Events `payment_intent.succeeded` und `payment_intent.pa` verarbeiten.
- Bei Erfolg: Bestellstatus in Firestore auf `paid` ändern, Datum `fulfilledAt` setzen; Bestand (`stock`) der Produkte reduzieren.
- Bei Fehlschlag: Bestellstatus `failed`; optional Auto-Storno oder Hinweis an den Nutzer.
- Sicherheitsaspekt: Prüfen der `stripe_webhook_secret`, um sicherzustellen, dass Webhook authentisch ist.

Bestellhistorie:

- Optionale Anzeige der Bestellungen im Benutzerprofil (Profil-Tab) – Liste mit Datum, Gesamtpreis, Status.
- Admin-Ansicht (siehe Tag 4) zeigt alle Bestellungen für das Gym.

Tag 4 (Do 11. Sep 2025) – Admin-Panel & Rabattaktionen

Admin-Panel (AM):

- Neue Seite `ProductAdminScreen` (nur für Admins); Funktionen:
  - **Produkt erstellen:** Formular mit Name, Beschreibung, Preis, Anzahl, Kategorie; Bild hochladen mittels Image-Picker und Upload in Storage.
  - **Produkt bearbeiten:** Daten laden, ändern, speichern; inaktiv setzen (`active=false`).
  - **Bestellungen einsehen:** Tabellarische Liste der letzten Bestellungen; Spalten: Datum, Benutzer, Betrag, Status; Such-/Filterfunktionen.
- **Zugriffsschutz:** Route nur zugänglich, wenn `AuthProvider.isAdmin()`.

Gutscheine & Rabatte (PM):

- Optionales Feature: Firestore-Sammlung `gyms/{gymId}/coupons/{couponId}`: Felder `code, discountType` (percent/amount), `value, expiresAt, usageLimit`.
- Im `createOrder` Cloud Function: Prüfen, ob `couponCode` übergeben wurde; Gültigkeit prüfen, Rabatt berechnen; Coupon-Verbrauch hochzählen.
- **UI-Integration:** Eingabefeld für Gutschein-Code im Warenkorb/Checkout-Flow; Feedback bei ungültigem Code.

Provision & Umsatzberechnung:

- Implementierung einer Cloud Function oder Scheduled-Task, die am Monatsende den Umsatz pro Gym berechnet, die Provision des App-Betreibers abzieht und dem Studio einen Bericht speichert (z. B. `gyms/{gymId}/reports/{month}` mit Feldern `grossSales, provision, netSales`).
- Diese Informationen werden für Lizenzabrechnungen in Sprint 6 benötigt.

Tag 5 (Fr 12. Sep 2025) – Tests, QA & Übergabe

Unit- & Widget-Tests:

- **ProductProvider:** Laden, Filtern, Pagination.
- **CartProvider:** Hinzufügen und Entfernen von Produkten; Summenberechnung inkl. Rabatten.
- **OrderProvider:** Korrektes Anlegen von Bestellungen und Aufruf der Cloud Function.
- **ProductListWidget** und **ProductDetailWidget:** Anzeige der Informationen, Interaktion mit „In den Warenkorb“.

Integrationstests mit Mock-Payments:

- Einsatz von Stripe Test-Cards (z. B. 4242 4242 4242 4242) für erfolgreiche Zahlungen; Test-Cards mit Fehlercodes für Fehlversuche.
- Sicherstellen, dass Webhook-Events den Status der Bestellung korrekt aktualisieren; bei Fehlern bleibt Bestellung auf `failed`.
- Test, dass bei Bestandsunterschreitung (`stock = 0`) kein Kauf möglich ist.

QA-Review & Rechts-Check:

- QA-Engineer führt einen Kaufvorgang von A–Z durch, inklusive Stornierung und erneuter Bestellung.
- Jurist prüft Datenschutzhinweise im Checkout-Flow und Widerrufsrecht-Darstellung (externer Input).
- Optional: Nutzungsbedingungen und AGB anpassen; Links im UI einbinden.

Dokumentation & Deployment:

- `docs/affiliate_marketplace.md`:
  - Datenmodell, Ablauf eines Kaufs, Abwicklung über Stripe, Admin-Panel-Anleitung.
  - Hinweise zu Rabatten, Provision, Rückerstattung.
- Update der README mit Informationen zur Einrichtung der Stripe-Keys und Webhook-URL.
- Fertigstellung der Cloud Function-Deploy-Skripte; Deployment auf Firebase Functions und Firestore.

Sprint-Review & Vorbereitung auf Sprint 6:

- Live-Demo des Shops: Produkt anlegen, kaufen, Zahlung abwickeln, Bestellstatus anzeigen.
- Feedback der Stakeholder einholen; eventuell Priorisierung offener Punkte (z. B. Versandabwicklung, Produktbewertungen).
- Auflistung der Tasks für Sprint 6 (Lizenz- & Multi-Tenancy-Modell) – z. B. Berücksichtigung des Marktplatz-Umsatzes in der Lizenzabrechnung.

3. Definition of Done

- **Datenmodell implementiert:** Produkte, Bestellungen und (optional) Gutscheine in Firestore; Regeln schützen vor unbefugtem Zugriff.
- **Shop-UI verfügbar:** Nutzer können Produkte durchsuchen, Details einsehen, Artikel in den Warenkorb legen und zur Kasse gehen.
- **Zahlung über Stripe funktioniert:** Cloud Functions erstellen PaymentIntent; Client-Secret wird genutzt, um Zahlung in der App abzuschließen; Webhook aktualisiert Bestellstatus.
- **Admin-Panel:** Studio-Admins können Produkte anlegen und ändern, Bestellungen einsehen und bearbeiten.
- **Rabattcodes optional implementiert:** Gutscheine können angelegt und im Checkout eingelöst werden.
- **Tests bestanden:** Unit-, Widget- und Integrationstests decken die Kaufabwicklung ab; Zahlungs-Simulation läuft in der CI.
- **Dokumentation & Rechtliches:** Datenmodelle und Abläufe sind dokumentiert; Hinweise zu DSGVO und Widerrufsrecht sind aufgenommen; juristische Prüfung ist erfolgt.
- **Sprint-Review abgeschlossen:** Stakeholder sind zufrieden und es besteht Klarheit über die nächsten Schritte für Sprint 6.

4. Ausblick auf Sprint 6

Mit einem funktionierenden Affiliate-Shop ist der nächste Meilenstein die Multi-Tenancy & Lizenzierung (15.–19. Sep 2025). Dabei werden Lizenzen pro Gym verwaltet, aktive Nutzer gezählt und Abrechnungen automatisiert – die Einnahmen aus dem Marktplatz spielen dabei eine zentrale Rolle.