Roadmap zur Umsetzung der skalierbaren GymApp

Zielsetzung

Aufbau einer modularen Flutter-App als White-Label/Multi-Tenant-Lösung, die für beliebig viele Studios personalisierte Instanzen ermöglicht. Datenisolation, Branding und CI/CD sind von Anfang an vorgesehen.

Phase 1: Projektsetup & Infrastruktur

a) *Versionsverwaltung*

- Git-Repository mit Hauptbranch `main` und Entwicklungszweig `develop` initialisieren.
- Branch-Policy definieren: Code-Reviews PR-Templates.

b) *Multi-Tenant-Grundgerüst*

- Ordnerstruktur anlegen: `lib/core/tenant/` für TenantService & Modelle. `lib/core/theme/` für dynamisches Branding.
- Shared Preferences konfigurieren, um `currentGymId` lokal zu speichern.

c) *CI/CD-Basics*

- GitHub Actions Workflow für `flutter analyze` und `flutter test` einrichten.
- SonarCloud oder Codacy integrieren für statische Code-Analyse.

d) *Firebase-Projektstruktur*

- Zentrales Firebase-Projekt anlegen mit Subcollections pro Studio: `gyms/{gymId}/...`
- Firestore-Persistence aktivieren (`settings.persistenceEnabled = true`).

Phase 2: Datenmodell & Security

a) *Datenklassen & Serialisierung*

- `json_serializable` + `build_runner` für alle Firestore-Modelle (GymConfig, Device, TrainingSession, User, etc.).
- DTOs vs. Domain-Modelle trennen.

b) *TenantService*

- Implementierung eines Singleton `TenantService` mit Methoden: `init(gymId)`, `getGymConfig()`, `switchGym()`.
- Anbieter-abhängige Streams (z. B. Gerätestatus) in Tenant-Kontext kapseln.

c) *Firestore Security Rules*

- Regeln: `match /gyms/{gymId}/sub=** { allow read, write:  if request.auth.tok == gymId; }`
- Custom Claims im Auth-Token setzen (z. B. bei Registrierung).

d) *Automatisierte Tests der Rules*

- `firebase-tools emulators:start` + `@firebase/rules-unit-testing` für CI.

Phase 3: Onboarding & Branding

a) *Onboarding-Flow*

- Screen zur Eingabe/Scannen des Gym-Codes (QR-Code).
- Nach erfolgreicher Validierung in Firestore: `TenantService.init(gymId)` aufrufen.

b) *Dynamisches Theme*

- Gym-spezifische Farben, Logos, Schriften aus `gyms/{gymId}/config` laden.
- `ThemeData` zur Laufzeit aktualisieren, `Provider/ChangeNotifier` für Theme.

c) *Asset-Management*

- Optional: Gym-Logos in Firebase Storage, per URL geladen und gecached.
- Fallback-Logos im Assets-Ordner.

d) *White-Label-Flavors vorbereiten*

- Android `productFlavors` und iOS Schemes definieren, aber initial nur Basis-Flavor nutzen.

Phase 4: Feature-Implementierung & Architektur

a) *Services modularisieren*

- Aus `ApiService` separieren: `AuthService`, `DeviceService`, `TrainingService`, `UserService`, `AffiliateService`.
- Jeder Service erhält Gym-Kontext via Dependency Injection (Riverpod / get_it).

b) *State Management verbessern*

- Wechsel zu `flutter_bloc` oder `Riverpod` für globale/tenantspezifische States.
- BLoC-Tests für Business-Logik.

c) *Screen-Architektur*

- Clean Architecture: Presentation (Widgets), Domain (UseCases), Data (Repositories).
- Beispiel-Implementierung: `GetAllDevicesUseCase`, `TrainingRepository`.

d) *UI-Komponenten*

- Wiederverwendbare Widgets im Tenant-Kontext (z. B. Device-List, Trainingsübersicht).
- Accessibility-Checks (Kontrast, Screen-Reader).

Phase 5: Tests, CI/CD & Deployment

a) *Unit- & Widget-Tests*

- Services mit `mockito` oder Fake-Firestore testen.
- Widget-Tests für Onboarding-, Login-, Dashboard-Flows.

b) *End-to-End-Tests*

- `integration_test` mit Firebase-Emulator für kompletten Nutzerpfad.

c) *CI/CD-Pipeline erweitern*

- Matrix-Build: Ein Build pro Flavor (Gym) in GitHub Actions.
- Automatisches Deployment:
  - Google Play Internal Track / TestFlight per Fastlane.

d) *Monitoring & Analytics*

- Firebase Crashlytics für Fehler-Monitoring.
- Firebase Analytics mit Gym-Property (GymId) als User-Property.

Phase 6: Skalierung & Weiterentwicklung

a) *White-Label Automation*

- Skript (z. B. in CI) zum Erzeugen neuer Flavors: `flutter build apk -flavor gymX`.

b) *Backend-Services auslagern*

- Optional: Node.js/Cloud Functions für komplexe Business-Logik (z. B. Abos, In-App-Käufe).

c) *Erweiterte Features*

- Push-Notifications (Trainingserinnerungen).
- Gamification (Challenges, Badges).
- Web-Admin-Panel für Studio-Administratoren (Flutter Web oder React).

d) *Wartung & Support*

- Versionsverwaltung pro Studio (Release-Tags).
- SLA-Definitionen und Support-Workflows etablieren.