# Roadmap für KI-Features in Tap'em

Tap'em Entwicklungsteam

10. Juni 2025

## 1. Chat-Assistent für Trainingstipps

### 1.1 Zielsetzung

Ein in die App integrierter Chatbot, der auf Nutzerfragen („Wie verbessere ich meine Kniebeuge?“) prägnante, KI-generierte Empfehlungen liefert.

### 1.2 Architektur & Komponenten

- **Service Layer:** `ChatAssistantService` für Kommunikation mit der KI-API (z. B. OpenAI).
- **Domain:** UseCase `AskTrainingTipUseCase`.
- **State Management:** Provider `ChatProvider`, hält Chat-Verlauf und Lade-/Fehlerzustand.
- **Presentation:** Screen `ChatScreen` mit Input-Feld und ListView für Nachrichten.

### 1.3 Schritt-für-Schritt-Implementierung

1. **API-Schlüssel beschaffen:** - OpenAI-Account anlegen, API-Key generieren, in `.env` speichern als $\texttt{OPENAI}_{KEY}$.`Pubspec anpassen:`

2. 3. Abhängigkeiten: `http`, ggf. `flutter_toast`.

**Service erstellen:**

- `lib/features/chat/data/chat`$_{assistant_service.dart}$ : $Methode$`Future<String> ask(String prompt`
  $Completion - Endpoint, parseAntwort$.

**Domain-UseCase:**

- `lib/features/chat/domain/usecases/ask`$_{training_tip.dart}$ : $ruft$`ChatAssistantService.ask`$auf$.

**Provider anlegen:**

- `lib/core/providers/chat`$_{provider.dart}$ : $Felder$`ListMessage history`, $Status, Methoden$`sendMessa`

**UI-Komponente:**

- `lib/features/chat/presentation/screens/chat`$_{screen.dart}$ : $Eingabefeld, Senden - Button, ListViewmitBlckenlinks/rechts$.

**Router eintragen:** `AppRouter.chat = '/chat'` und `onGenerateRoute`.

**Integration ins BottomNavigationBar**: - Neuer Tab „Chat“ hinzufügen.

**Tests & QA**: - Unit-Tests für Service und UseCase. - UI-Test: Eingabe → Antwort.

## 1.4 Meilensteine & Timeline

**Woche 1** API-Integration, Service  UseCase

**Woche 2** Provider  Basic UI (Chat-Screen)

**Woche 3** Feinschliff, Tests, Dokumentation

**Woche 4** Pilot-Release an 5 Studios

# 2. Automatische Progress-Zusammenfassung

## 2.1 Zielsetzung

Monatliche KI-generierte Zusammenfassung: Fortschritt in Prozent, Trainingshäufigkeit, Highlights als Textelemente.

## 2.2 Architektur & Komponenten

- **Data Aggregation:** UseCase `GetMonthlyStatsUseCase` (basiert auf `ReportRepository`).
- **Service Layer:** `SummaryGeneratorService` nutzt eine einfache Sprache-API, um Rohdaten in Text zu verwandeln.
- **Provider:** `SummaryProvider` hält den generierten Text und Ladezustand.
- **Presentation:** Widget `ProgressSummaryCard` mit Titel, Fließtext und optional Download-Button (PDF).

## 2.3 Schritt-für-Schritt-Implementierung

1. **UseCase zur Statistik:**

- `lib/features/report/domain/usecases/get`$_m onthly_s tats.dart : Aggregation von `usageCounts` $und$

2. **Service für Text-Generierung:**

- `lib/features/summary/data/summary`$_g enerator_s ervice.dart : Methode$ `generateSummary(MapStri` $APImitPrompt`Formuliereeine Zusammenfassung...$1.

3. **SummaryProvider:**

- `lib/core/providers/summary`$_p rovider.dart : Status-Enum, Methode$ `loadSummary()` $ruft UseCas$ $Service.$

4. **UI-Widget:**

- `lib/features/summary/presentation/widgets/progress`$_s ummary_c ard.dart : CardmitCircularP$ $Review$1, $generiertemText.$

5. **Integration in ReportScreen:**
    - Unter Heatmap-Karte: `ProgressSummaryCard()` einfügen.
6. **Tests & QA**: - Unit-Tests für UseCase und Service. - UI-Check: korrekte Darstellung, Länge, Lesbarkeit.

## 2.4 Meilensteine & Timeline

**Woche 1** UseCase für Monats-Stats + Aggregation

**Woche 2** SummaryService  Prompt-Feintuning

**Woche 3** Provider  UI-Card, Tests

**Woche 4** Pilot-Release mit 5 Studios