Sprint A, Tag 1 (Sa 9. August) — Ausführung mit Codex & Checklisten

Tap'em — Pilot Roadmap

Stand: 9. August 2025

Ziel des Tages

- **Repo aufräumen, Secrets prüfen, `.gitignore` fixen**
- **Roadmap/Backlog im GitHub-Issue-Tracker anlegen (nur Pilot-Scope)**

Übersicht

Dieser Leitfaden enthält:

A) einen **vollständigen Codex-Prompt** (1:1 in deinen Codex-Agenten kopieren),

B) eine **kurze To-do-Liste für manuelle Schritte** (Entscheidungen/Erlaubnisse, die nicht rein durch Code erledigt werden können).

1 (A) Vollständiger Codex-Prompt (Copy–Paste, 1–zu–1)

Kontext für dich, Codex Du hast Schreibzugriff auf das GitHub-Repo der Flutter-App „Tap'em" (Flutter Mobile + ggf. Flutter Web, Firebase (Firestore/Storage/Functions), iOS/Android). Erstelle heute einen Maintenance-PR, der das Repo aufräumt, Secrets-Risiken eliminiert, eine korrekte `.gitignore` einführt und einen Issues-Backlog für den **Pilot-Scope** anlegt. Führe alles versioniert in einem neuen Branch durch und eröffne danach einen PR mit sauberer Beschreibung, Checkliste und Labels.

0) Branch, Meta, Hausordnung

- Erstelle einen Branch: `chore/2025-08-09-repo-cleanup-secrets-ignore`.
- PR-Titel: **`chore: repo cleanup, secrets scan, .gitignore, pilot backlog (2025-08-09)`**.
- PR-Body soll enthalten:
  - Zweck (Cleanup + Secrets + `.gitignore` + Pilot-Backlog),
  - Checkliste (siehe unten),
  - Links zu neuen CI-Workflows und Reports,
  - Anleitung zum manuellen Triggern von Gitleaks (`workflow_dispatch`),
  - „How to rotate“ Abschnitt (falls Secrets gefunden werden).

1) `.gitignore`: anlegen/ersetzen (Flutter + Android + iOS + Web + Firebase Functions)

Erzeuge/ersetze die Datei `.gitignore` im Repo-Root **mit genau diesem Inhalt**:

```
# --- Flutter / Dart ---
.dart_tool/
.packages
build/
.flutter-plugins
.flutter-plugins-dependencies
.melos_tool
pubspec.lock        # HINWEIS: Nur ignorieren, falls dieses Repo ein Package ist. Fr Apps i. d. R. committen.

# --- IDE ---
.idea/
.android/
.vscode/
*.iml

# --- Android ---
**/local.properties
**/captures/
**/*.keystore
**/*.jks
**/app/*.apk
**/app/*.aab
**/gradle-wrapper.jar

# --- iOS / macOS ---
**/Pods/
**/Podfile.lock
**/.symlinks/
**/DerivedData/
**/.generated/
**/Runner.xcworkspace/
**/Flutter/ephemeral/
```

```
# CocoaPods artifacts
**/Pods
**/Frameworks
**/*.xcuserstate
**/*.xcworkspace
**/*.xcuserdatad

# --- Web ---
web/build/
build/web/

# --- Firebase / Google ---
# NIEMALS Service-Accounts o. App-Privatkeys committen
serviceAccount*.json
**/google-services.json
**/GoogleService-Info.plist
**/.firebaserc.local
**/.runtimeconfig.json
.firebase/
# Firebase emulators local state
**/firebase-debug.log

# --- Node / Functions ---
functions/node_modules/
functions/.env
functions/.env.*
functions/.secret
functions/.DS_Store

# --- OS / Misc ---
.DS_Store
Thumbs.db
*.log
*.tmp
*.swp
```

Wichtig: Wenn dieses Repo eine **App** ist (keine wiederverwendbare Dart-Library), dann **entferne** die Zeile `pubspec.lock` aus `.gitignore` und **committe** die vorhandene `pubspec.lock`. (Wenn unklar: Entscheidung im PR-Body transparent vermerken.)

2) Secrets-Scan & Schutz (Gitleaks + optional TruffleHog) via GitHub Actions

Gitleaks-Konfiguration `.gitleaks.toml`

```
title = "Tapem Gitleaks Config"
[allowlist]
  description = "Allowed files/lines"
  files = [
    '''firebase_options\.dart''',       # enthlt keine echten Secrets
  ]

[[rules]]
```

```
id = "FirebaseServiceAccount"
description = "Do not commit Firebase service account keys"
regex = '''(?i)\"type\":\s*\"service_account\"'''
tags = ["key","firebase","security"]
```

Workflow `.github/workflows/gitleaks.yml`

```yaml
name: Security - Gitleaks
on:
  pull_request:
  push:
    branches: [ main ]
  workflow_dispatch:

jobs:
  gitleaks:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4
        with:
          fetch-depth: 0  # volle Historie scannen
      - name: Run Gitleaks
        uses: zricethezav/gitleaks-action@v2
        with:
          args: detect --source . --config .gitleaks.toml --report-path gitleaks-report.json --report-format json
      - name: Upload Report
        uses: actions/upload-artifact@v4
        if: always()
        with:
          name: gitleaks-report
          path: gitleaks-report.json
```

(Optional) Lege zweiten Workflow `.github/workflows/trufflehog.yml` an (gleiche Trigger) für einen ergänzenden Scan.

SECURITY_REPORT.md Erzeuge im PR eine Datei `SECURITY_REPORT.md` mit:

- Anleitung zum manuellen Triggern (Actions → „Security - Gitleaks“ → „Run workflow“),
- Speicherort der Reports (Artifacts),
- Vorgehen bei Findings > 0 (Datei entfernen, Keys rotieren; Historie neu schreiben nur nach Rücksprache).

3) Repo-Aufräumen (ohne Funktionalität zu verändern)

- Scanne nach sensiblen Dateien: `google-services.json`, `GoogleService-Info.plist`, `serviceAccount*.json`, `.env*`, `.p12`, `.pem`.
- Wenn gefunden:
  - Entferne sie aus dem Repo,

  - ersetze sie durch `*-example` Stubs (z. B. `GoogleService-Info.example.plist`) mit Kommentaren,
  - ergänze im `README` die Setup-Schritte (lokale Generierung/Kopierpfade).
- Lege unter `docs/` an:
  - `docs/secrets-policy.md` (Do/Don't, Rotation, keine Keys im Code),
  - `docs/environment-setup.md` (Firebase CLI, `flutterfire configure`, Speicherorte lokaler Dateien).
- Passe `README.md` an:
  - Abschnitt „Erste Schritte": `flutter pub get`, ggf. `dart run build_runner`, Hinweis `flutterfire configure`,
  - Hinweis auf `.gitignore` und `docs/secrets-policy.md`.

4) GitHub Issues: Backlog für Pilot-Scope automatisch anlegen

Ziel Issues für Sprints A–G (nur Pilot-relevante Tasks), Labels, Templates. Nutze GitHub API oder `gh` CLI. Erzeuge Skript + Manifest für Reproduzierbarkeit.

Labels Erzeuge `.github/labels.json` (z. B. `type:feature`, `type:chore`, `type:security`, `type:docs`, `prio:P0/P1/P2`, `sprint:A..G`). Implementiere `scripts/apply_labels.ts` *oder* `scripts/apply_labels.py` + README-Abschnitt für Ausführung (per `gh` oder PAT).

Issue-Templates Lege `.github/ISSUE_TEMPLATE/feature.yml` und `bug.yml` an.

Backlog-Manifest Erzeuge `project/roadmap/issues_pilot.json` mit Feldern `title`, `body`, `labels`, `milestone` (z. B. „Pilot v1.0 (Oct 2025)"). Implementiere `scripts/create_issues.py` (Python, `requests`, nutzt `GITHUB_TOKEN`) zum Anlegen der Issues aus dem Manifest. Lege Milestone „Pilot v1.0 (Oct 2025)" an.

Inhalte für `issues_pilot.json` (Pilot-Scope) Sprint A (Security-/Rules/App Check/RC): A1 App Check (P0), A2 Firestore Rules+Tests (P0), A3 Storage Rules+Tests (P0), A4 Remote Config (P1), A5 Claims+`assignRole` (P1).
Sprint B (Feedback/Survey+Push): B1 Kategorie+Foto (P1), B2 FCM/APNs+lokal (P1), B3 serverseitige Topic-Subscription (P0), B4 Functions+Reminder (P1), B5 Report-Badges+Tests (P2).
Sprint C (NFC/Offline/QR): C1 Session-Schema+Idempotenz (P0), C2

Offline Queue+Sync+Debounce (P0), C3 NDEF Nonce+Mapping/Revoke (P0), C4 QR-Fallback (P1), C5 Sync-Status-UI+E2E (P1).
Sprint D (Serverseitige XP/Aggregate): D1 Aggregat-Schema (P0), D2 `onSetCreate` XP-Delta idempotent (P0), D3 Leaderboards aus Aggregaten (P1), D4 Simple Reports (P2), D5 Tests/Monitoring (P1).
Sprint E (Branding Light/Onboarding/Stores): E1 Branding Light (P2), E2 Onboarding Guides (P1), E3 Pilot Whitelist via RC (P1), E4 Store Drafts/Privacy (P1), E5 TestFlight/Play-Internal+CI (P1), E6 Consent+Export/Löschung Stub (P0), E7 DPIA/Privacy/Impressum (P0).
Sprint F (Hardening/Licensing/Dry-Run): F1 Minimal Licensing Gate (P0), F2 E2E-Suite (P1), F3 Pen-Tests (P0), F4 Performance Low-End (P2), F5 Bugbash/Crashlytics (P1), F6 Pilot Dry-Run (P0).
Sprint G (Buffer/Launch): G1 RC Build+Flags (P0), G2 Pilot-Rollout (P0), G3 Monitoring+Hotfix (P1), G4 KPI+Retro (P2).

PR-Checkliste (als Markdown im PR-Body)

- `.gitignore` ersetzt/ergänzt; Entscheidung zu `pubspec.lock` dokumentiert
- Gitleaks Workflow + Config erstellt; `SECURITY_REPORT.md` hinzugefügt
- Repo gescannt; sensible Dateien entfernt/ersetzt; `docs/secrets-policy.md` + `docs/environment-setup.md` + `README` aktualisiert
- Labels/Issue-Templates angelegt; Milestone „Pilot v1.0 (Oct 2025)“ erstellt
- `project/roadmap/issues_pilot.json` + `scripts/create_issues.py` hinzugefügt
- (Optional) TruffleHog Workflow
- CI grün, Gitleaks Report artefaktisiert

Jetzt ausführen (Codex) Führe alle oben beschriebenen Änderungen durch, pushe sie im Branch `chore/2025-08-09-repo-cleanup-secrets-ignore`, eröffne den PR, poste den Gitleaks-Run-Link in den PR-Body. Warte auf Review.

2 (B) Manuelle To-do-Liste für dich

- **Entscheidung zu `pubspec.lock` (App vs. Package):**
  App ⇒ `pubspec.lock` committen. Package ⇒ `pubspec.lock` ignorieren.
  (Wenn unsicher: sehr wahrscheinlich App ⇒ committen.)
- **GitHub Token/Permissions (für Issues-Skript):**
  Falls Codex kein PAT mit „repo“ + „project/issue“ Rechten hat: Fine-grained PAT (nur dieses Repo, minimale Rechte) erstellen und

als Repo Secret `GITHUB_TOKEN_PILOT_ISSUES` hinterlegen *oder* Codex GitHub-App Zugriff auf Issues erteilen.
Alternativ kannst du das Skript später lokal mit deinem Token ausführen.

- **Falls Secrets gefunden werden:**
  Nicht force-pushen, sondern:
  a) Datei im aktuellen Stand entfernen (`.gitignore` schützt künftig),
  b) **Keys rotieren** (Firebase/Stripe/Apple etc.),
  c) Historie neu schreiben (BFG/`git filter-repo`) nur im Ausnahmefall und nach Abstimmung.
- **Review & Merge:**
  PR lesen, Gitleaks-Workflow einmal manuell triggern (Actions → „Security - Gitleaks“ → „Run workflow“). Wenn grün ⇒ mergen.
- **Issues anlegen (falls Codex das nicht automatisch konnte):**
  Im Repo-Root ausführen:
  `python3 scripts/create_issues.py -token $GITHUB_TOKEN_PILOT_ISSUES`

Hinweis: Wenn du möchtest, liefere ich dir als Nächstes den Codex-Prompt für **So 10. Aug (optional)**: Rules-Draft + Storage-Rules + Emulator-Tests (ready to paste).