7-Day Flutter Super App Training

(Modular Approach · WebView Mini-Apps · Platform Channels · State Mgmt · Mini-App Config · Git)

Day 1 – Flutter Super App Foundations

- Flutter & Dart refresher (null safety, isolates, async/await).
- Why Flutter for super apps (scalability, shared UI system).
- Super app patterns (shell + mini-apps): WebView vs native modules.
- Project setup: feature-first foldering, env files, flavors.
- Build a Mini-App Launcher (Grid/List, icons, deep links).

Day 2 – Architecture & Modular Approach

- Clean Architecture & MVVM for super apps (core/domain/app layers).
- Modularization: feature packages, lib/ boundaries, shared design system.
- Navigation at scale: go_router + StatefulShellRoute (app vs project nav rails).
- Dependency injection with Riverpod (providers for services, repos).
- Refactor to modules: core, auth, catalog, and miniapps.

Day 3 – Secure WebView Integration

- webview_flutter setup (Android/iOS): manifests, WKWebView config.
- Hardening: domain allowlist, block mixed content, no file access, prevent new windows.
- Navigation delegates & violation handling (cancel + fallback UI).
- JS↔Dart bridge: JavaScriptChannels / postMessage, typed payloads.
- Build a reusable SecureWebView widget (loading, error, retry, offline).

Day 4 – State Management & Data Flows

- State in a super app: Riverpod (app-level, feature-level, ephemeral vs persisted).
- Cross-module state contracts (auth/user/tenant/theme).
- Passing context to WebViews: JWT, query params, postMessage bootstrap.
- Persistence: Hive / SharedPreferences / SQLite; when to use each.
- Implement GlobalAppState (auth, tenant, locale, feature flags).

Day 5 – Auth, Security & Platform Channels

- Auth in Flutter (Firebase/Supabase/OAuth)—token lifecycle & refresh.
- Zero-trust WebViews: capability-scoped APIs, short-lived JWT per mini-app.
- RBAC in Flutter (role/tenant aware menus & guards).
- Platform Channels:
- - When to use native vs WebView.
- - Sample: expose a native SDK (e.g., device info / payments) safely to Flutter.
- - Guardrails: explicit method allowlist, parameter validation.
- Wire token passing to mini-apps + revocation & kill-switch hooks.

Day 6 – Mini-App Config, Environments & Git

- Mini-App Config (single source of truth):
- - appId, title, publisher, entryUrl, icon.
- - environments: dev / staging / prod (endpoints, feature flags, CSP).
- - capabilities: camera, payments, file access (request/deny).
- - SDK requirements (Flutter, Dart, Android/iOS targets, plugins).
- - Bridge contract: allowlisted RPCs + JSON-Schema for requests/responses.
- - Versioning: appVersion, bridgeVersion, minShellVersion.
- Environment selection UX inside the super app.
- Git & Version Control for super apps and mini-apps:
- - Repo model: mono-repo with Melos or multi-repo with shared config repo.
- - Branching: trunk-based or GitFlow.
- - Semantic versioning & compatibility matrix.
- - Tags & changelogs; signed tags for config drops.
- - Git hooks: validate miniapp_manifest.json against schema before commit.
- Implement MiniAppConfigService in Flutter.

Day 7 – Multi-Tenant, Theming & Performance

- Multi-tenant design: tenant claims in JWT, config-driven menus & visibility.
- White-labeling: theme packs, typography, icon sets via module overrides.
- Performance:
- - Pre-warm WebView instances; HTTP/2 preconnect; skeleton UIs.
- - Memory/CPU profiling for WebView vs Dart (DevTools).
- - Asset & image policies for mini-apps (responsive, cache-friendly).
- Release management (manual release checklist for shell and config/mini-apps).
- Capstone:
- - Super app shell with two third-party mini-apps, each on different envs.
- - Secure bridge + RBAC + environment selector + offline config cache.
- - Git tag the shell, bridge, and config; update compatibility matrix.