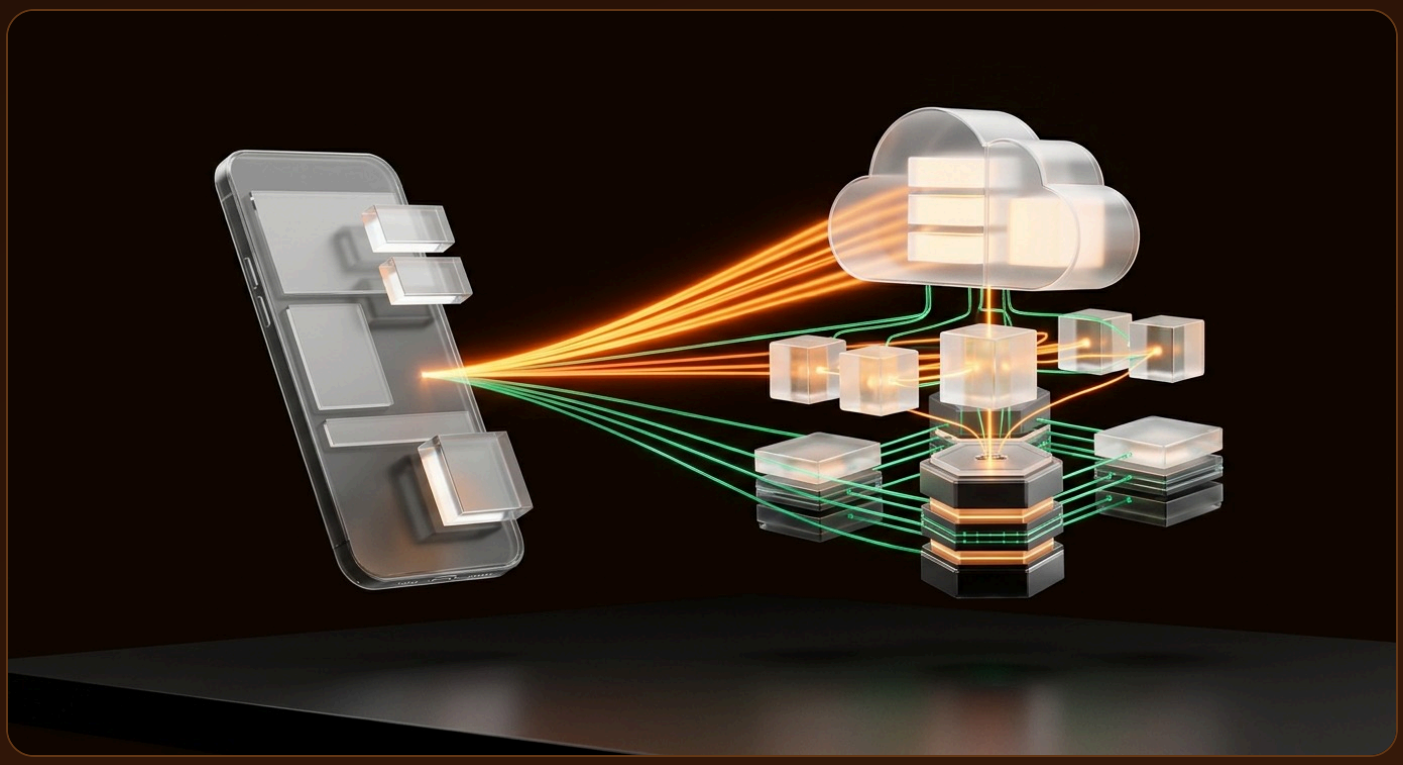
GOOGLE AI & CREATIVE STUDIOS • FIELD GUIDE

Agentic Firebase & Android CLI Workflows

The complete 12-page visual handbook to full-stack AI mobile engineering—uniting scoped Firebase MCP backends, App Check-protected Gemini 3.8 Kotlin apps, headless Compose Preview rendering, and automated Android Device Hub journeys.

Firebase MCP & Agent Skills | Gemini 3.8 Flash & Live | Play Integrity App Check | android CLI & Delta Builds | Compose Preview Sidecar | Device Hub & XML Journeys



Cloud-to-Code Backend

Query Firestore, inspect Data Connect GraphQL, and diagnose App Hosting logs safely via scoped `firebase mcp`.

Protected Gemini 3.8 Apps

Ship Android Kotlin experiences on `gemini-3.8-flash` and `gemini-3.8-live` gated by Firebase App Check.

Closed-Loop Device QA

Audit Compose `bounds_url` trees, stream AVDs in Device Hub, and run strict sequential XML `<journey>` tests.

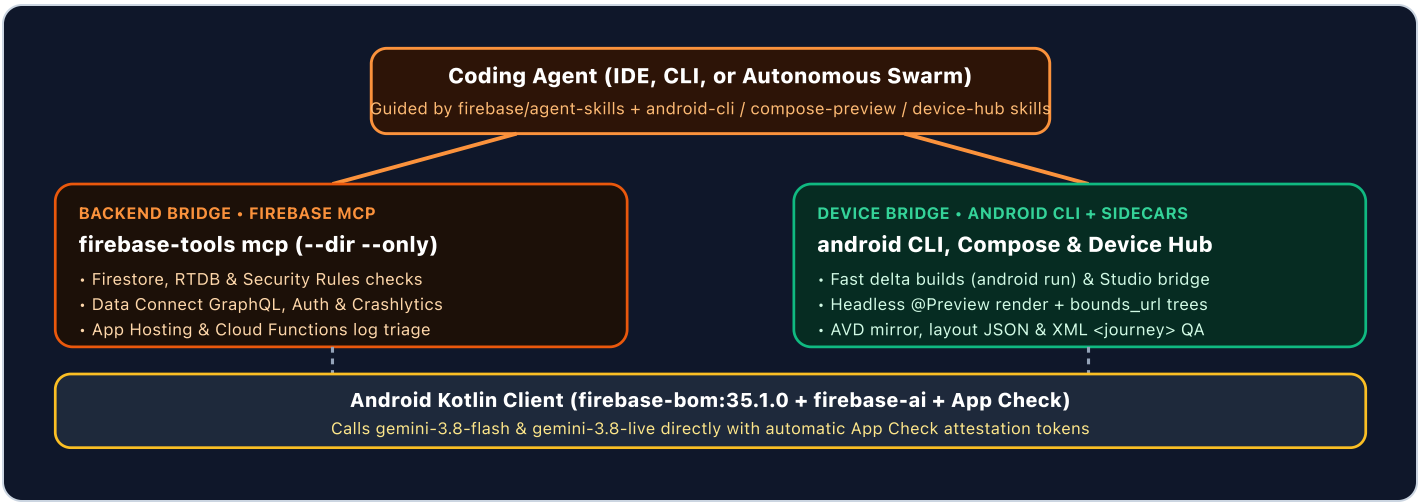
The Full-Stack Mental Model: Cloud Meets Pixels

Modern mobile agents no longer stop at writing Kotlin files in the dark. By pairing **Firebase MCP** on the cloud side with the `android` **CLI** and its live UI sidecars on the device side, an agent can verify schemas, builds, Compose bounds, and tap-by-tap user journeys in one continuous loop.

Quick Navigation

1. Full-Stack Mental Model p. 2
2. Install, Auth & Tool Matrix p. 3
3. Annotated Cockpit UI Tour p. 4
4. Firebase MCP & Data Connect p. 5
5. AI Logic Kotlin & Config Anatomy p. 6
6. Troubleshooting & Pre-Flight p. 12
7. `android` CLI & Compose Sidecar p. 7
8. Device Hub & XML Journeys p. 8
9. 5 Copy-Paste Production Recipes p. 9
10. App Check & Multi-Device Ops p. 10
11. Cross-Tool Synergy Pipeline p. 11

Architecture: Two Bridges Closing the Mobile Loop



Why Pairing Skills With Tools Matters

Raw MCP servers expose dozens of capabilities without telling an agent *when* to use each one. Pairing `npx skills add firebase/agent-skills` with `android init` and `android skills add` teaches your agent idiomatic Security Rules patterns, Play Integrity guards, and fast UI inspection rules before it writes a single line of code.

Installation, Auth & Backend-to-Device Matrix

Set up both toolchains in under three minutes. Keep Firebase MCP tightly scoped per project directory while initializing the `android` CLI globally for fast SDK, emulator, and Studio access.

1 Install & Initialize `android` CLI

Install the native binary for your host OS, then run `android init` to configure directories and default skills:

```
# macOS Apple Silicon (darwin_arm64)
curl -fsSL
https://dl.google.com/android/cli/latest/darwin_arm64/install.sh | bash
# Linux x86-64
curl -fsSL
https://dl.google.com/android/cli/latest/linux_x86_64/install.sh | bash
# Windows Command Prompt
curl -fsSL
https://dl.google.com/android/cli/latest/windows_x86_64/install.cmd -o "%TEMP%\i.cmd" && "%TEMP%\i.cmd"
android init && android auth login
```

Discover domain skills anytime via `android skills find compose` and install with `android skills add <id>`.

2 Authenticate & Scope Firebase MCP

Firebase MCP runs as a built-in subcommand of `firebase-tools`. Refresh OAuth credentials, add agent skills, and pin MCP flags:

```
# 1. Re-authenticate & install Firebase agent skills
npx -y firebase-tools@latest login --reauth
npx skills add firebase/agent-skills
# 2. Register stdio MCP server (or claude mcp add)
claude mcp add firebase npx -- -y firebase-tools@latest mcp \
  --dir /abs/path/to/project \
  --only firestore,auth,dataconnect
```

Pinning `--dir` and `--only` prevents accidental edits across unrelated Firebase projects.

Android Tooling Decision Matrix: Pick the Fastest Layer

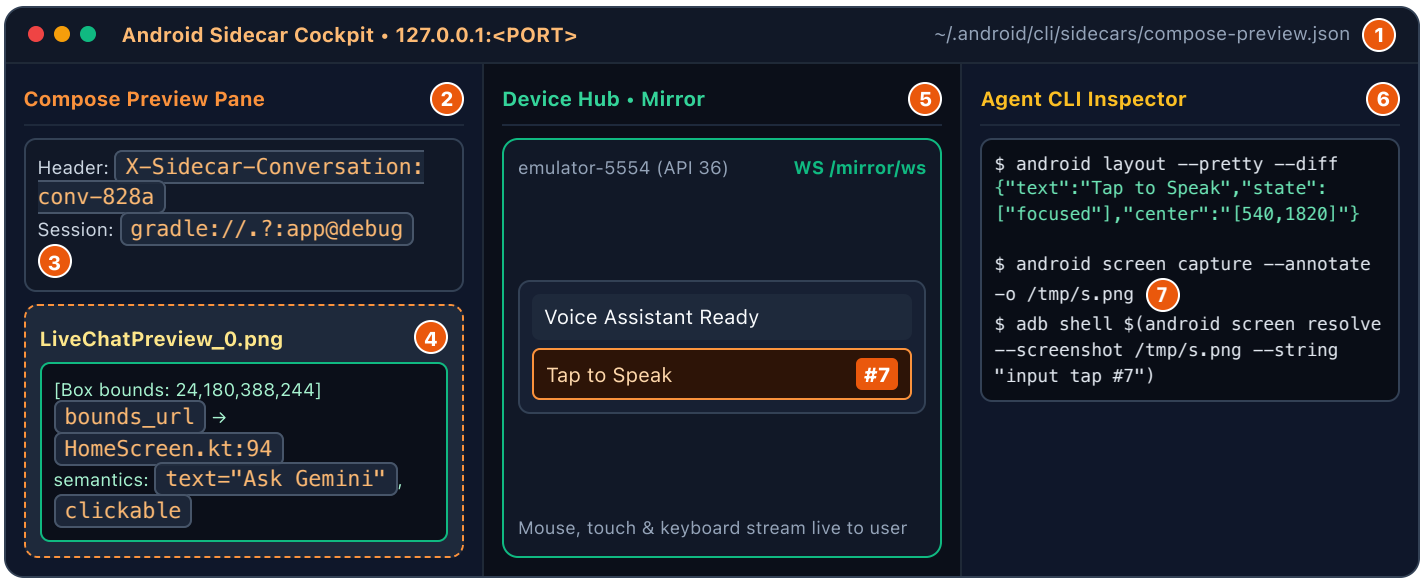
Layer	Primary Strength	When to Use It	What to Avoid
Raw `adb`	Low-level shell, input injection, logcat, port tunnels	Sending taps, swipes, `input text`, or checking `getprop sys.boot_completed`	Slow full-APK installs or manual UI hierarchy XML parsing
`android` CLI	Delta installs, JSON layout, screen resolve, SDK & docs	`android run --use-delta-install`, `layout`, `screen capture --annotate`, `docs search`	Running `android preview render` directly outside the sidecar
UI Sidecars	Persistent HTTP servers + live IDE visual panes	Inspecting `bounds_url` Compose trees and mirroring live devices in Device Hub	Sending browser `Origin` headers or omitting `X-Sidecar-Conversation`
Android Studio Bridge	Live PSI/AST symbol index & inspection engine	`android studio analyze-file`, `find-declaration`, `find-usages`, `version-lookup`	Assuming Studio is open without checking `android studio check` first

JSON Snippet for Any Standard MCP Client

In Antigravity or custom MCP configs, declare Firebase as: `"command": "npx"` with `"args": ["-y", "firebase-tools@latest", "mcp", "--dir", "/abs/path", "--only", "firestore,auth,dataconnect"]`.

Annotated Android Device Hub & Compose Preview Cockpit

When both Android sidecar plugins run alongside your terminal, you get instant pixel feedback in the IDE while your agent inspects exact coordinates, composable bounds, and incremental layout diffs over loopback HTTP and CLI.



Seven-Point Cockpit Reference Legend

1 **Sidecar Discovery File:** Read `~/.android/cli/sidecars/compose-preview.json` to get the active dynamic loopback port, then verify with `GET /api/status`.

2 **Conversation Isolation Header:** Always send `X-Sidecar-Conversation: <id>` on `POST /api/session`, `POST /api/render`, and `GET /api/state` so the pane renders your thread.

3 **Session Configuration:** `POST /api/session` sets `project_dir`, Gradle/Bazel `reference`, `fqns`, and `"wait": true` to build and render cleanly.

4 **Component Tree (`bounds_url`):** Inspect `sourceLocation`, `semantics`, and `modifiers` in JSON to audit padding and accessibility without guessing.

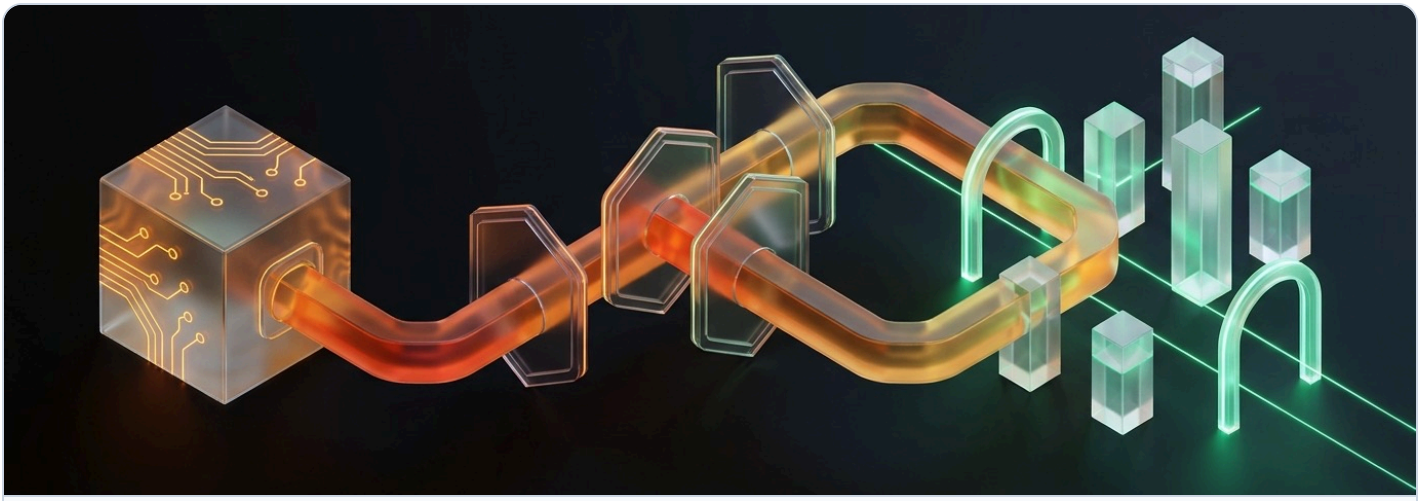
5 **User Mirror WebSocket (`/mirror/ws`):** Streams live AVD or USB device video and input in the Device Hub UI; agents inspect and drive devices via CLI and `adb`.

6 **Layout JSON & Diffs:** Run `android layout --pretty --diff` first to capture exact element bounds, focus states, and changes in minimal tokens.

7 **Annotated Screen Resolve:** When WebViews or custom canvases hide from the layout tree, run `android screen capture --annotate` and substitute box `#N` centers with `android screen resolve`.

Firebase MCP: Inspecting Live Cloud State & Rules

Running `firebase-tools@latest mcp` gives your coding agent a live, least-privilege window into your Firebase backend—so database schemas, Security Rules, GraphQL operations, and deploy logs stay aligned with your Android client.



Cloud-to-Client Verification Bridge: Scoped `firebase mcp` routes agent queries through directory-pinned permissions and Security Rules checks into Firestore, Data Connect GraphQL, and App Check gates.

Core Firebase MCP Capability Groups (`--only`)

Capability Group	What the Agent Can Inspect & Validate	Android Synergy Payoff
Firestore & RTDB + Rules	Query documents, inspect collection indexes, and validate Security Rules logic before deploy	Matches Kotlin data classes and ensures queries will not fail with `PERMISSION_DENIED`
Data Connect (GraphQL)	Author PostgreSQL-backed GraphQL schemas, mutations, and generated client connectors	Generates strongly typed Kotlin SDK calls ready for Compose UI state flows
App Hosting & Functions	Tail build failures, Cloud Functions runtime logs, and rollout environment config	Diagnoses backend webhook or server-side token errors in seconds without browser tabs
Auth, Crashlytics, FCM & RC	Look up Auth users, triage top Crashlytics stack traces, inspect Remote Config & FCM	Pairs real crash frames with `android studio find-declaration` to patch root causes
AI Logic Schema Consulting	Verify Gemini Developer API / Vertex AI backend setup, App Check enforcement, and quotas	Prevents client SDK mismatches before testing `gemini-3.8-flash` or `gemini-3.8-live`

Pair With `firebase/agent-skills` for Idiomatic Generation

Install `npx skills add firebase/agent-skills` in your workspace so the agent loads battle-tested guidance on Firestore indexing, Data Connect query security, and App Check token flows automatically whenever Firebase files appear.

Firebase AI Logic + App Check: Kotlin & Prompt Anatomy

Firebase AI Logic lets your Android app invoke Gemini directly from Kotlin without standing up a custom proxy server—while **Firebase App Check** proves every request originates from your genuine app binary.

1 Gradle Dependencies (BoM `35.1.0`)

```
// app/build.gradle.kts
dependencies {
  implementation(platform("com.google.firebase:firebase-bom:35.1.0"))
  implementation("com.google.firebase:firebase-ai")
  // Local debug vs Play Store attestation
  debugImplementation("com.google.firebase:firebase-appcheck-debug")
  releaseImplementation("com.google.firebase:firebase-appcheck-playintegrity")
}
```

2 App Check Init + Console SHA-256

Register your app's **SHA-256 fingerprint** in Firebase Console under Play Integrity, and install the matching provider in `Application.onCreate()`:

```
Firebase.initialize(this)
Firebase.appCheck.installAppCheckProviderFactory(
  if (BuildConfig.DEBUG)
DebugAppCheckProviderFactory.getInstance()
  else PlayIntegrityAppCheckProviderFactory.getInstance()
)
```

3. Invoking Gemini 3.8 Flash & Gemini 3.8 Live in Kotlin

```
// Text & multimodal generation (automatically attaches App Check tokens)
val flashModel = Firebase.ai.generativeModel(modelName = "gemini-3.8-flash")
val reply = flashModel.generateContent("Summarize today's field inspection checklist.")

// Real-time bidirectional voice & streaming session via Gemini 3.8 Live
val liveSession = Firebase.ai.liveModel(modelName = "gemini-3.8-live").connect()
liveSession.send("Start guided voice walkthrough for step one.")
```

Anatomy of a Complete Full-Stack Agent Prompt

Use Firebase MCP (--only firestore,dataconnect) to verify the InspectionLog schema and rules, enable App Check with DebugAppCheckProviderFactory for debug and PlayIntegrity for release, wire a Kotlin ViewModel calling gemini-3.8-flash for summaries and gemini-3.8-live for voice mode, render InspectionCardPreview via the Compose Preview sidecar to confirm clickable semantics in bounds_url, deploy with android run --use-delta-install onto emulator-5554, and verify the end-to-end XML <journey> using android layout --diff.

Backend MCP Scope · App Check Security · Gemini 3.8 Models · Compose Bounds QA · Delta Deploy · Journey Assertion

`android` CLI Mastery & Compose Preview Sidecar Loop

Replace slow trial-and-error Gradle cycles with fast delta deployments, live Android Knowledge Base queries, and headless Compose Preview rendering that exposes every composable's coordinates and accessibility semantics.

1 Project, SDK, Docs & Fast Delta Run

- **SDK & Scaffolding:** Manage toolchains with `android sdk list --all` / `install`, scaffold templates via `android create`, and inspect build targets with `android describe`.
- **Fast Delta Deploy:** `android install` and `android run --use-delta-install` ship only changed dex and resource deltas.
- **Wear & Multi-Surface Types:** Pass `--type` for `ACTIVITY`, `WATCH_FACE`, `TILE`, `COMPLICATION`, `DECLARATIVE_WATCH_FACE`, or `WEAR_WIDGET`.
- **Authoritative Docs:** Query current APIs via `android docs search "App Check"` and read full pages with `android docs fetch kb://...`.

2 Discovering Targets & Flavor Variants

Discover every `@Preview` and MultiPreview function in JSON before rendering:

```
android preview list \
  --reference gradle://.?:app@debug \
  --all --format=json
```

Flavor Gotcha: Modules with product flavors have no bare `debug` variant. Always include flavors first, such as `gradle://.?:feature:home@demoDebug`.

3. Compose Preview Sidecar Loop (Save First, Then Render!)

```
# 1. Read port from ~/.android/cli/sidecars/compose-preview.json & configure session with conversation header
curl -s -X POST http://127.0.0.1:$PORT/api/session \
  -H "X-Sidecar-Conversation: $CONVERSATION_ID" -H "Content-Type: application/json" \
  -d '{"project_dir":"/abs/MyApp","reference":"gradle://.?:app@debug","fqns":
["com.example.ui.HomeScreenKt.HomePreview"],"wait":true}'

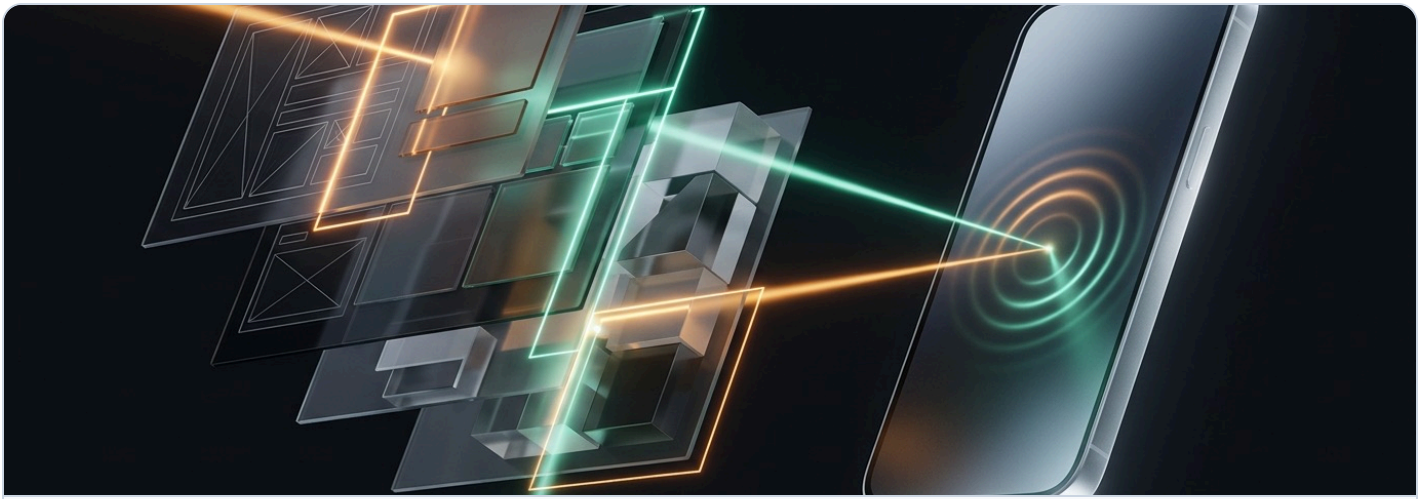
# 2. Edit Kotlin UI files -> wait until ALL saves finish -> trigger re-render (never parallelize edit + render!)
curl -s -X POST http://127.0.0.1:$PORT/api/render \
  -H "X-Sidecar-Conversation: $CONVERSATION_ID" -H "Content-Type: application/json" -d '{"wait":true}'
```

Accessibility & Layout QA Without Guessing Pixels via `bounds_url`

When `--component-tree` is active (on by default), `GET /api/state` returns a `bounds_url` JSON tree for every rendered PNG. Read `sourceLocation` (`fileName`, `lineNumber`) to jump straight to the composable, check `modifiers` (`padding`, `clickable`, `size`), and verify `semantics` (`text`, `contentDescription`, `role`, `testTag`) to catch inaccessible icon buttons automatically.

Device Hub Sidecar, Visual Targeting & XML Journeys

Previews confirm static layout; real journeys prove behavior. Combine the Device Hub sidecar and `android emulator` CLI with JSON-first inspection, visual box resolution, and strict sequential XML journey scripts.



Closed UI Inspection Loop: Layered architectural Compose planes feed exact coordinates into the live glass device, pulsing touch targets back through `android layout` and `screen resolve`.

1 Boot Check & JSON-First Inspection

- **AVD Control:** Use `android emulator list|start|stop|create|remove` or Device Hub `GET /api/devices` (never start duplicate jobs on one device).
- **Wait for Full Boot:** Online emulators may still be booting. Run: `adb -s <serial> wait-for-device shell 'while [ -z "$(getprop sys.boot_completed)" ]; do sleep 1; done'`
- **JSON First, Pixels Second:** Start with `android layout --device <serial> --pretty` (add `--diff` after actions). Switch to `android screen capture --annotate -o /tmp/screen.png` for WebViews or custom canvases, then inspect the PNG visually.

2 Precision Input & Strict XML `<journey>`

- **Tap Resolved Boxes:** `adb -s <serial> shell $(android screen resolve --screenshot /tmp/screen.png --string "input tap #7")`
- **Safe Text & Slow Scrolls:** Confirm `"focused"` in element `state`, encode spaces as `%s` (`"Hello%sworld"`), and scroll slowly over 500ms (`adb shell input swipe 250 400 250 200 500`).
- **Sequential XML Journeys:** Execute each `<action>` strictly as written in order; stop immediately on failure or crash and emit a Markdown report marking steps ✅ or ❌.

5 Real-World Copy-Paste Recipes

Drop these five battle-tested command and code recipes straight into your terminal, Gradle config, or agent prompt to eliminate repetitive boilerplate.

Recipe & Goal	Copy-Paste Snippet / Command	Expected Outcome
1. Scoped Firebase MCP + Skills Bootstrap	`npx -y firebase-tools@latest login --reauth && npx skills add firebase/agent-skills && claude mcp add firebase npx -- -y firebase-tools@latest mcp --dir /abs/app --only firestore,auth,dataconnect`	Installs Firebase idioms and pins MCP strictly to your target directory and services.
2. App Check Protected `gemini-3.8-live` Screen	`Firebase.initialize(this); Firebase.appCheck.installAppCheckProviderFactory(DebugAppCheckProviderFactory.getInstance()); val live = Firebase.ai.liveModel(modelName = "gemini-3.8-live").connect()`	Establishes a real-time Live session with automatic App Check attestation headers.
3. Fast Delta Run + Studio Analysis Loop	`android studio check && android studio analyze-file --file app/src/main/java/ChatVm.kt && android run --device emulator-5554 --use-delta-install`	Catches Kotlin lint/type issues via Studio PSI before shipping a delta APK to the AVD.
4. Headless Compose Accessibility Audit	`PORT=$(jq -r .port ~/.android/cli/sidecars/compose-preview.json) && curl -s -X POST http://127.0.0.1:$PORT/api/session -H "X-Sidecar-Conversation: $CID" -H "Content-Type: application/json" -d '{"project_dir":"/abs/app","reference":"gradle://.?:app@debug","fqns":["com.app.HomeKt.HomePreview"],"wait":true}'`	Renders the preview in your conversation pane and writes `bounds_url` semantics JSON.
5. XML Journey Visual & Diff Audit	`android screen capture --device emulator-5554 --annotate -o /tmp/s.png && adb -s emulator-5554 shell $(android screen resolve --screenshot /tmp/s.png --string "input tap #7") && android layout --device emulator-5554 --diff --pretty`	Resolves visual box `#7` to screen coordinates, taps it, and returns only changed UI nodes.

Always Wait for Edits Before Calling `POST /api/render`

Never issue a file edit tool call and a `POST /api/render` request in the same parallel batch—Wait until all Kotlin file writes finish on disk so Gradle does not compile stale code.

App Check Enforcements, Least-Privilege MCP & Fleet Ops

Production AI mobile apps require tight boundaries at every hop—limiting what your coding agent can mutate in Firebase, proving app identity at runtime, and testing safely across remote hardware.

1 Least-Privilege Firebase MCP Rules

- **Always Pin `--dir`:** Pass an absolute project path so agents never read or write the wrong Firebase alias.
- **Scope With `--only`:** Enable only the tool groups needed for the sprint (for example `firestore,auth,dataconnect`) instead of exposing destructive project-wide surfaces.
- **Review Rules Before Deploy:** Ask the agent to validate Firestore and Storage Security Rules before updating client queries.

2 Debug-to-Play Integrity Rollout

- **Local Debug Tokens:** Register the Logcat token printed by `firebase-appcheck-debug` in Firebase Console for emulator tests.
- **Release Attestation:** Swap to `firebase-appcheck-playintegrity` and add your Play App Signing **SHA-256** fingerprint in Console.
- **Enforce Last:** Monitor App Check metrics for `gemini-3.8-flash` and `gemini-3.8-live` traffic before flipping enforcement on.

3 Android Studio Live Bridge

When Android Studio is open, let the agent tap directly into its live PSI index rather than grepping blindly:

- `android studio check` — verify running IDE instances.
- `android studio find-declaration` & `find-usages` — trace Compose callbacks and ViewModels accurately.
- `android studio version-lookup` — fetch latest Maven artifacts like `firebase-bom:35.1.0` and SDK tools.

4 Remote Devices & Secure Capture Rules

Scale beyond local emulators when testing real sensors, Play Integrity, and WebViews:

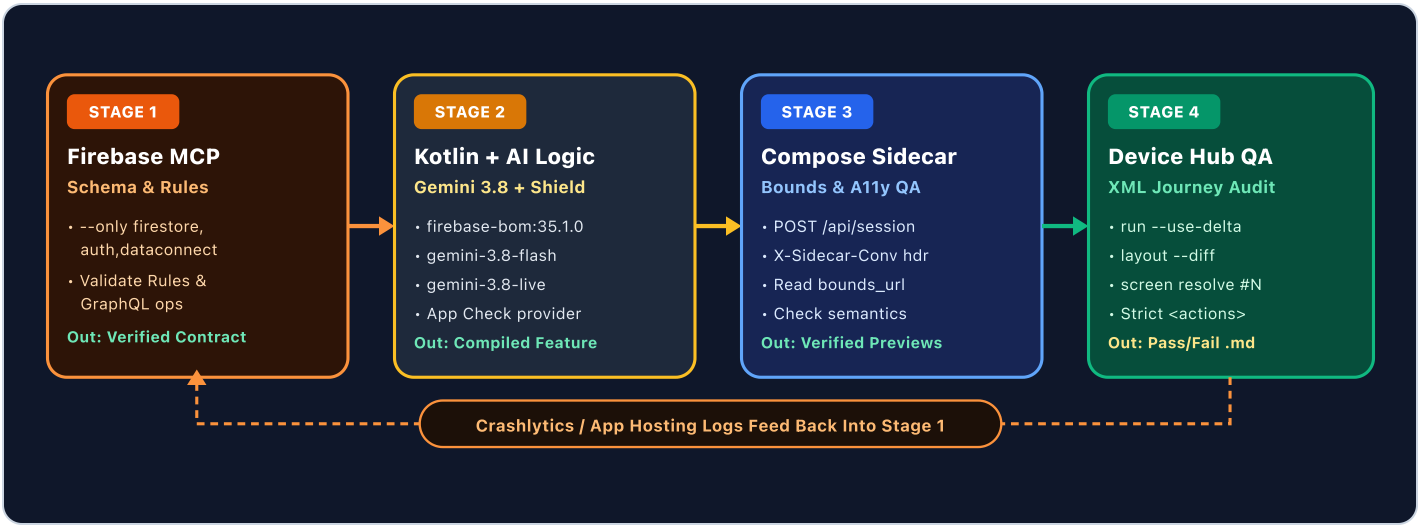
- **Remote Physical Lab:** Reserve and connect to cloud hardware via `android device remote`.
- **Tunnel Latency:** Screen captures over tunneled devices take longer; re-capture only after state-changing actions.
- **DRM & Banking Blackouts:** `FLAG_SECURE` windows capture strictly black in both screenshots and Device Hub mirrors.

One ADB Server Shared Across CLI and Sidecar

Because Device Hub and the `android` CLI share the same local ADB server, remote devices attached via `adb connect 127.0.0.1:6555` appear automatically in `GET /api/devices` and can be mirrored live by the user while you inspect them.

Cross-Tool Synergy: From Cloud Schema to Verified APK

Watch how all four toolchains interlock in a single sprint—passing validated schemas, protected Gemini 3.8 clients, accessibility-checked Compose trees, and sequential XML journey reports cleanly from stage to stage.



Stage Handoff Checklist for Teams & Multi-Agent Swarms

A Backend-to-Client Handoff

Keep one agent or phase on Firebase MCP to finalize Firestore indexes, Data Connect mutations, and Security Rules before the UI agent generates Kotlin callers. This prevents broken field names and silent permission failures during device runs.

B Preview-to-Journey Handoff

Run Compose Preview bounds checks *before* booting an AVD for journey QA. Catching missing `contentDescription` labels or clipped padding in 10 seconds inside the sidecar saves minutes of emulator interaction.

Troubleshooting Clinic & 6-Point Pre-Flight Checklist

Hit a blank pane, dropped Live session, or misbehaving ADB command? Match your symptom below for a 10-second root-cause fix.

Symptom / Failure	10-Second Root-Cause Fix
1. Compose Preview pane stays blank after `POST /api/session`	You omitted the `X-Sidecar-Conversation: <id>` header (or used a query parameter). Re-send with the exact HTTP header.
2. `android preview render` fails or renders stale UI edits	Never invoke `android preview render` directly, and never call `POST /api/render` in parallel with file edits. Save all edits first, then call `POST /api/render`.
3. Gradle preview lookup fails on `:feature@debug`	Modules with product flavors have no bare `debug` variant. Supply the flavor-first reference, e.g. `gradle://.?:feature:home@demoDebug`.
4. `gemini-3.8-live` drops immediately on physical release builds	App Check rejected an unverified token. Ensure `firebase-appcheck-playintegrity` is installed and your Play App Signing **SHA-256** is registered in Console.
5. `adb shell input text` drops words after the first space	Raw spaces truncate Android text injection, or the field lacks focus. Check `"focused"` in `android layout` state and replace spaces with `%s`.
6. `android layout` returns empty inside a login or WebView	WebViews and custom canvases hide from accessibility trees. Run `android screen capture --annotate -o /tmp/s.png`, view it, and tap via `screen resolve`.
7. Firebase MCP exposes unrelated projects or risky writes	Pin the workspace with `--dir /abs/path` and restrict active tool families using `--only firestore,auth,dataconnect`.

6-Point Pre-Flight Checklist Before Shipping

- ☐ **1. Scoped MCP & Skills:** `firebase mcp --dir --only` paired with `firebase/agent-skills` and `android init`.
- ☐ **2. Current Models Only:** Kotlin calls target `gemini-3.8-flash` and `gemini-3.8-live` via BoM `35.1.0`.
- ☐ **3. App Check Parity:** Debug provider locally; Play Integrity + Console SHA-256 verified for release.
- ☐ **4. Clean Compose Bounds:** `bounds_url` verified for `clickable` semantics and `contentDescription`.
- ☐ **5. Delta Deploy Ready:** `android run --use-delta-install` boots cleanly on AVD with `sys.boot_completed`.
- ☐ **6. Sequential Journey Pass:** All XML `<actions>` pass in order with a signed-off ✅ Markdown audit.