ChatGPT

MCP on Mobile: Running Model Context Protocol Servers on Android & iOS

MCP SDK Support for Android & iOS Devices

Official SDKs: The major MCP libraries have evolved to support mobile platforms. The official **C# MCP SDK** (backed by Microsoft) targets .NET 7+ and is cross-platform, meaning you can include it in a .NET MAUI app for Android or iOS. Likewise, the **Swift MCP SDK** runs natively on Apple platforms (iOS 16+ and even watchOS/tvOS) [1]. In practice, this means you *can* build MCP servers or clients directly into mobile apps using these SDKs. For example, a .NET MAUI app can use the C# SDK to implement an MCP server, and an iOS app or framework can use the Swift SDK to do the same in Swift/Objective-C.

Other Languages: Official Python and TypeScript SDKs for MCP are primarily intended for desktop/server use (running on Node.js or CPython), so they don't run on iOS out-of-the-box. Android *could* run Python/Node via third-party apps or embedded runtimes (e.g. using Termux on Android to run a Python MCP server). But for a clean **mobile-app integration**, the native SDKs (C# or Swift) are the straightforward choice. Community-driven ports exist (e.g. some Java/Kotlin examples for Android automation), but they often rely on external PC control rather than purely on-device execution.

Transport Options on Mobile: STDIO vs. HTTP/SSE

MCP is **transport-agnostic**, supporting both a **Standard I/O (stdio)** interface and network protocols like **HTTP with Server-Sent Events (SSE)** [2] [3]. On mobile devices, these transport choices have different viability:

- **STDIO:** This is ideal when an MCP server runs as a subprocess on the same host as the AI client (for example, a code editor launching a local tool). On mobile, however, you typically **cannot spawn arbitrary subprocesses** in a sandboxed app (especially on iOS). Android does allow app processes or native binaries in some cases, but in general mobile apps run in a single process. Thus, using STDIO in a mobile app context is uncommon. STDIO might be used on Android in a developer scenario (e.g. an IDE app launching a helper process), but on iOS it's practically not an option. In short, **STDIO transport is not directly practical for on-device MCP in normal mobile apps** due to platform restrictions.

- **HTTP/SSE:** Using a local HTTP server (optionally with SSE for streaming responses) is the **preferred method for mobile**. The MCP spec itself notes SSE or *"streamable HTTP"* is recommended for remote or networked scenarios [2] [3]. In the mobile case, the MCP server would run within the app and listen on a network socket. This could be `localhost` (for an on-device AI client) or the device's LAN IP (to allow external clients to connect). **SSE** enables the server to push real-time tool results or stream outputs – useful for long-running tools. In practice, community examples strongly favor SSE/HTTP for phone-hosted servers. For instance, the Tasker MCP project explicitly states that **SSE mode is "perfect for running a persistent server, for example on your phone via Termux, that can be accessed over the network by multiple clients."** In contrast, STDIO is only *"ideal for local integrations"* on a desktop where the host can spawn the server [4].

In summary, a local mobile MCP server will almost always expose an HTTP(S) endpoint (with SSE for streaming) rather than rely on stdio. If you are embedding an MCP server in a .NET MAUI app, you'd likely use an HTTP listener to handle MCP JSON-RPC requests. The OpenAI Apps SDK (ChatGPT tools) also expects network-accessible connectors, standardizing on HTTP/SSE – *"we recommend streamable HTTP"* for MCP [3].

Examples of MCP Servers Running on Mobile

Android – On-Device Servers: Android's openness has enabled real on-phone MCP servers for various use cases:

- **Tasker MCP Server (Android Automation):** A standout example is the *Tasker MCP server* by Luis Sanchez. It bridges an LLM with the Android device's native capabilities through the Tasker automation app. The server is a lightweight Go binary that runs on the phone (often launched in a Termux Linux environment or via Tasker itself) [4]. In SSE mode, it listens on a port on the phone. This allows an AI agent (e.g. running on a PC, or conceivably on the phone) to call tools that do things like toggling WiFi, reading texts, controlling apps, etc. Essentially, Tasker MCP turns your phone into an MCP server exposing **hundreds of device actions**. A guide by Skywork.ai describes it as *"the missing bridge that connects the reasoning power of LLMs to the real-world actions available on an Android device"* [5]. Users have demonstrated asking a remote AI "@phone, do X" where `@phone` directs the query to the phone's MCP endpoint – and the phone executes the task. This is done securely (the Tasker server requires an API key and runs within your network/VPN) [6] [7].

- **Mobile Automation MCP (Android & iOS via PC):** There are MCP servers like **Mobile-Next's "mobile-mcp"** and others that use Appium/UIAutomator under the hood to automate mobile UIs [8] [9]. These typically run on a computer (or cloud) and interface with devices (real or emulated) via ADB (for Android) or WebDriverAgent/XCUITest (for iOS). For example, an *Android UIAutomator2 MCP server* can run on a PC and let an AI agent control an Android device's UI by simulating taps and reading screen content [10]. Similarly, iOS UI automation MCP solutions exist (e.g. the **iHackSubhodip iOS MCP server**). An article on that project calls it *"a universal translator that allows an AI model like Claude to understand and control iOS apps, just like a human would"* [11]. However, **these are not running** on **the phone** – they require an external host (often macOS for iOS devices) to drive the automation. They demonstrate that MCP can integrate mobile platforms, but the server logic isn't fully self-contained on the phone in these cases.

- **Embedded MCP in Apps:** We're also seeing examples where the MCP server logic is part of a mobile app itself (built with cross-platform .NET or native code). For instance, the **Telerik .NET MAUI MCP server** focuses on .NET MAUI UI components. It's distributed as an NPM package that can be launched (via stdio) to provide AI assistants with knowledge of Telerik UI controls [12] [13]. In that scenario, a developer running Copilot Chat or VS can spin up the MCP server to answer questions about UI APIs. Another example is the **".NET MAUI Graphics MCP Server"** (an open-source project) which runs inside a MAUI app to let AI draw on a canvas. It exposes tools like `maui_graphics_draw_circle`, etc., and updates the app's UI in real time as the AI commands it [14] [15]. This is essentially an MCP server *inside* a running mobile app, demonstrating that a MAUI app can host the server and perform actions in-app. (In testing, developers use the MCP Inspector or Copilot to send drawing commands to the app [16].)

- **Mobile Client Apps:** (For completeness) There are also mobile *clients* for MCP. For example, *ChatMCP* or the **SystemPrompt** app allow you to connect your phone to external MCP servers

[17] [18]. These aren't servers themselves, but they indicate a growing mobile ecosystem around MCP – e.g. using your phone to voice-command various MCP tools. They illustrate that mobile usage of MCP (either as host or client) is an active area.

Bottom line: It is technically possible to run an MCP server on a phone (especially Android) and have AI tools interact with local data or apps. Android has real demos of this (Tasker MCP running **on device** with SSE access [4]), whereas iOS is currently limited to tethered solutions or simpler on-device contexts due to Apple's restrictions. No known App Store-approved app exposes a full device-control MCP server (for security reasons), but within the app's sandbox you can still use MCP to expose *that app's* functionality.

Integrating an MCP Server in a .NET MAUI Mobile App

If you're building a custom Android/iOS app with .NET MAUI and want it to host an MCP server, here are key considerations:

- **Using the C# MCP SDK:** You can add the official ModelContextProtocol NuGet package to your MAUI project and define your MCP **tools** in C#. The SDK uses attributes to mark tool methods, which can be static methods that perform actions or fetch data. At runtime, you create and run an `McpServer`. For example, in a console app one might do:

```
builder.Services.AddMcpServer()
                .WithStdioServerTransport()
                .WithToolsFromAssembly();
await builder.Build().RunAsync();
```

  which starts an MCP server over stdio [19] [20]. In a mobile app, instead of stdio, you'd likely use an **HTTP transport**. The C# SDK as of late 2025 does support HTTP transports (the exact API might be `WithHttpServerTransport(...)` or creating an `HttpServerTransport` manually). This would bind to a port on the device. You might choose a localhost-only binding if the AI client is running on the same device (less common), or a LAN-accessible IP if you want external clients (e.g. your PC or another device) to call it. Ensure to pick a port that doesn't conflict; on Android, you can listen on a high-numbered port without special permissions. On iOS, listening on a socket is allowed, but only while the app is active (more on backgrounding below).

- **App Lifecycle & Background Execution:** Mobile apps are not designed to run indefinitely in the background serving requests, so you must plan how your MCP server will run:

- **Android:** To keep the server alive when the app is not in the foreground, you should use a **Foreground Service**. This is a background service with a persistent notification, indicating to the user that your app is running in the background. Android foreground services prevent the OS from killing your process for being idle [21]. In a MAUI app, you'd implement platform-specific Android services (via DependencyService or multi-targeting) that start/stop the MCP server. The StackOverflow community confirms that *"for Android, use a foreground service... the mobile will have a notification"* to keep it alive [21]. Without a foreground service, Android may terminate your app after some time if the user navigates away, which would drop any MCP connections.

- **iOS:** iOS is far more restrictive. Apps cannot simply run arbitrary servers in the background; once your app is not frontmost, it gets suspended very quickly unless it's performing a task in a narrow set of allowed categories (e.g. playing audio, VoIP, navigation, etc.). Long-running sockets or servers are **not permitted** in general – *"Apple explicitly forbids the sort of perpetually running background process"* that a continuous MCP server would represent [22]. There are BGProcessingTasks in iOS, but those are for finite work and not real-time socket listening. In practice, an MCP server in an iPhone app will only be reachable while the app is open (or perhaps a short time after moving to background, before suspension). This means you can't rely on an iOS app to be an always-on MCP endpoint. One workaround could be using **Push Notifications or Shortcuts**: e.g. have the app wake via a push message, do a quick MCP interaction, then shut down – but that gets complex and outside MCP's normal use. So, on iOS, consider MCP servers for *interactive* use only (when user launches the app and initiates a session).

- **Embedding vs. External:** Since you're using MAUI, the assumption is the MCP server runs in-process with the app. This is different from the typical plugin scenario on desktop where an IDE launches the server as a separate process. In-app, you'll call `McpServer.RunAsync()` on perhaps a background thread or task. It's wise to start the server **after** any UI initialization (so it doesn't block the UI thread). Also, handle app closure events: on Android, you might stop the server in `OnDestroy`; on iOS in `OnSleep`. Cleanly shutting down the server (closing sockets etc.) will free the port and resources.

- **STDIO in-app:** If your MAUI app itself includes an LLM client agent, you might not need HTTP at all – you could call the tools as functions directly. (In essence, you'd be bypassing MCP's transport and just invoking methods). But if you want to adhere to MCP spec for compatibility, you could simulate a "local" client connecting via stdio. This would be unusual – it means your app spawns another process *of itself* or a helper. MAUI apps don't easily spawn child processes on iOS, and on Android it'd require JNI/exec hackery. So, realistically, you won't use stdio transport **on-device** except in contrived scenarios. Network transport is the way to go for interoperability.

- **Testing the server:** While developing your MAUI-embedded MCP server, you can use tools like the **MCP Inspector** to test it. The Inspector (via `npx @modelcontextprotocol/inspector`) can launch and connect to your server for debugging [16]. You might run your app in an emulator or device, ensure it's listening (emulator typically accessible at `10.0.2.2` from host machine if Android), and point Inspector or an AI client to that address.

Security, Sandboxing, and Permissions

Running an MCP server on a phone raises important security considerations:

- **App Sandbox:** Any MCP tools you expose can only operate within the app's privileges. For instance, a tool that reads a file must be limited to files your app can access. On **iOS**, that means the app's sandbox container (and any user-selected documents or photos via official APIs – direct file system access is otherwise sandboxed). On **Android**, your app can access its internal storage and, with permissions, certain shared data (contacts, external storage, SMS, etc.). If your MCP tools need to, say, read contacts or SMS on Android, you *must* declare the appropriate permissions in AndroidManifest.xml and request runtime permission from the user. The MCP server doesn't bypass OS security – it's subject to the same rules as any app. A malicious or misbehaving AI could in theory call a tool to, say, delete files or send messages, so you should only expose tools that you (the developer) are comfortable automating, and possibly build in

confirmations for destructive actions. Keep the principle of least privilege: don't expose a blanket "exec shell command" tool on mobile, for example, since that could be dangerous (and won't work without root on Android, and not at all on iOS).

- **Network Exposure:** If your mobile MCP server listens on a network interface, consider scope and authentication. Binding to `localhost` (loopback) means only apps on the device can talk to it – which might be appropriate if the AI client also runs on-device. Binding to the LAN IP (e.g. `192.168.x.x`) allows other devices on the WiFi network to reach it – you'd want this if your PC's IDE or ChatGPT is the client. In that case, **secure the connection**. At minimum, use a non-privileged port and perhaps an auth token. MCP has an authentication mechanism (OAuth 2 flows, etc.) in its spec for connectors. In practice, a simple solution is to require an API key or token in a header for any call to your server. The Tasker MCP server, for example, generates a random API key you must provide when launching the server, and clients must know that key to connect [6] [7]. This prevents unauthorized devices on the network from using your tools. If you're exposing your phone's MCP server over the internet (not generally recommended), absolutely put it behind a VPN or secure tunnel. Some users route traffic through a VPN like Tailscale so that the phone's MCP endpoint is only reachable to their personal devices, mitigating open exposure.

- **OAuth and User Authorization:** MCP is designed with security in mind – for instance, connecting an AI to Gmail or PayPal requires OAuth consent. For a local server, OAuth isn't applicable unless your tool calls out to a third-party API. But if your MCP server itself requires a login (say your app has user accounts), ensure the AI can only perform actions that the user is authenticated for. In a multi-user app, you wouldn't want one user's AI session accessing another's data via MCP. Treat the MCP tool calls as an extension of your app's UI – they should enforce the same permission checks. If appropriate, you could even prompt the user on the device when a sensitive tool is invoked (though this might break the AI's flow – use carefully).

- **Platform Sandboxing Quirks:** Note that iOS will **not allow** certain actions at all. For example, no iOS app (even with user permission) can read SMS or call logs, or toggle system settings directly. Those are just off-limits for all apps due to sandboxing. An Android app with the right permissions can do more (read texts, modify settings, etc.), but starting with Android 13+ even some of those are restricted or require special privileges (for instance, accessing call logs or SMS is heavily vetted by Google Play policies). So, the set of "tools" you can expose might be inherently limited by what the OS allows third-party apps to do. Always consult the platform developer documentation for what's possible and permissible.

- **Data Storage:** If your MCP server writes data or caches context, be mindful of where it stores it (use app-specific storage). And if the data is sensitive (e.g. results from a corporate database), treat the device with the same care as any client: secure storage, encryption if needed, etc. Mobile devices can be lost or stolen, so don't keep secrets in plain text on disk.

In short, **running an MCP server on mobile is as secure as the app hosting it**. Follow best practices of mobile security and you won't introduce new risks. The MCP protocol itself can be secured (e.g. via OAuth tokens) – indeed, an MCP mobile client like SystemPrompt emphasizes *"secure authentication with OAuth"* when connecting to servers [23]. For your own local server, you control the environment, so focus on not exposing it beyond intended scope and leveraging the OS security model.

Third-Party Tools & Workarounds for Mobile MCP Hosting

Because of the challenges mentioned, developers have come up with creative workarounds and helper libraries:

- **Tasker and Termux (Android):** These tools allow running custom code on Android without writing a full app. As discussed, Tasker MCP uses a combination of a Tasker script (to expose device tasks) and a native binary server. Termux can run languages like Python or Node on-device. For example, one could run the official Python FastMCP server on an Android phone via Termux. It's not an "in-app" solution, but it enables quick experimentation – essentially turning your phone into a small server host. This is great for power users; for a polished app, you'd bake the logic into the app instead of requiring Termux.

- **Appium/WebDriverAgent (iOS):** Since running an automation server *on* iOS is unfeasible under App Store rules, the common solution is running a server on a Mac/PC that communicates with the iPhone. Projects like the iOS Mobile Automation MCP server by iHackSubhodip leverage Apple's UI testing framework (XCUITest) through WebDriverAgent on the device, controlled by a Python FastMCP server on macOS [24]. In other words, the heavy lifting is off-device. If your goal is to, say, let an AI test your iPhone app's UI, you will likely need such a tethered setup. It's a "half-local" MCP: the device is being automated, but the MCP server is just outside it.

- **SwiftMCP and Native APIs:** Some community efforts (like **SwiftMCP** and **SwiftLens**) aim to integrate iOS/macOS native APIs with MCP [25] [26]. For instance, SwiftLens is an MCP server that gives AI semantic understanding of Swift code (for Xcode projects) [25]. SwiftMCP appears to facilitate JSON-RPC calls to Apple APIs via MCP [27]. These could potentially run on-device (since they're just Swift packages) inside developer apps or playgrounds, allowing, for example, an AI assistant within an iPad app to call native functionality. This is niche but shows that the community is extending MCP into native mobile development scenarios (like code analysis, IDE integration on iPad, etc.).

- **VPNs/Networking tricks:** As briefly noted, if you need to access a phone's MCP server from outside (e.g. you're away from home but want to query your phone), you'll need to deal with NAT and security. Solutions include using a VPN or tunneling service. Some users report using **Tailscale** (a peer-to-peer VPN) to connect their mobile devices to their home network securely, so that the phone's MCP service is reachable at a private IP [28] [29]. Another approach could be using NGROK or Cloudflare Tunnel to expose the server with TLS and authentication. These are not specific to MCP, just general networking solutions to bridge to a mobile device that normally sits behind cellular NAT or WiFi firewall.

- **Frameworks and Registries:** As MCP gained traction, "registries" of community-built servers (like **PulseMCP/AugmentCode** directories) have grown. You'll find many MCP servers – some oriented to mobile (e.g. Home Assistant MCP for IoT, Fastlane MCP for app deployment, etc.). While these aren't libraries to run on-device, they might give you code or inspiration to adapt. For example, the **Fastlane MCP server** allows AI to perform mobile app build/test release tasks via Fastlane [30] – not something that runs on a phone, but it could be part of a pipeline that ultimately deploys to devices.

In conclusion, **running an MCP server on Android/iOS is doable and has been demonstrated, especially on Android.** With .NET MAUI, you can embed the server right into your app using the C# SDK, given careful consideration to how the server will run (likely as an HTTP endpoint) and how to keep

it alive on mobile. Android is fairly accommodating (use a Foreground Service for long-running tasks [21]), whereas iOS will require the app to be active for any interactions (no true always-on service in the background). Always plan for security – limit your tools to what's safe, require authentication for external access, and respect the device's permission model. By leveraging the official SDKs and the community know-how (Tasker for Android, and tethered automation for iOS), you can extend AI agents into the "last mile" of mobile devices, enabling new powerful use cases at the palm of your hand.

Sources: Current MCP documentation and blogs, including Google Cloud's MCP guide [2], OpenAI's Apps SDK notes [3], the Tasker MCP guide [4] [6], Telerik's .NET MAUI MCP documentation [12], and community examples of mobile MCP servers on Android and iOS [11] [21].

---

1 GitHub - modelcontextprotocol/swift-sdk: The official Swift SDK for Model Context Protocol servers and clients.
https://github.com/modelcontextprotocol/swift-sdk

2 What is Model Context Protocol (MCP)? A guide | Google Cloud
https://cloud.google.com/discover/what-is-model-context-protocol

3 MCP
https://developers.openai.com/apps-sdk/concepts/mcp-server/

4 5 6 7 10 Tasker MCP Server: The Ultimate Guide to AI-Powered Android Automation
https://skywork.ai/skypage/en/tasker-mcp-server-ai-android-automation/1981175065464459264

8 9 GitHub - mobile-next/mobile-mcp: Model Context Protocol Server for Mobile Automation and Scraping (iOS, Android, Emulators, Simulators and Real Devices)
https://github.com/mobile-next/mobile-mcp

11 24 The Agentic Shift in Mobile Testing: A Deep Dive into iHackSubhodip's iOS Automation MCP Server
https://skywork.ai/skypage/en/agentic-shift-mobile-testing-ios-automation/1981253016629465088

12 13 .NET MAUI MCP Server - Telerik UI for .NET MAUI
https://www.telerik.com/maui-ui/documentation/ai/mcp-server

14 15 16 GitHub - jsuarezruiz/maui-graphics-mcp-server: Effortlessly craft stunning mobile UI components with AI, powered by the Model Context Protocol!
https://github.com/jsuarezruiz/maui-graphics-mcp-server

17 28 29 Best way to use MCPs on mobile iOS / Android : r/mcp
https://www.reddit.com/r/mcp/comments/1kvqaui/best_way_to_use_mcps_on_mobile_ios_android/

18 23 systemprompt MCP - Native iOS & Android MCP Server Management
https://systemprompt.io/

19 20 GitHub - modelcontextprotocol/csharp-sdk: The official C# SDK for Model Context Protocol servers and clients. Maintained in collaboration with Microsoft.
https://github.com/modelcontextprotocol/csharp-sdk

21 android - How to create a background service in .NET Maui - Stack Overflow
https://stackoverflow.com/questions/71259615/how-to-create-a-background-service-in-net-maui

22 Is it possible to maintain a persistent websocket connection ... - Reddit
https://www.reddit.com/r/iOSProgramming/comments/9jv5wq/is_it_possible_to_maintain_a_persistent_websocket/

25 Introducing SwiftLens – The first and only iOS/Swift MCP server that ...
https://www.reddit.com/r/ClaudeAI/comments/1m12pzs/introducing_swiftlens_the_first_and_only_iosswift/

26 SwiftLens - iOS/Swift MCP Server for AI Code Analysis
https://swiftlens.tools/

27 SwiftMCP: Native iOS API Integration with JSON-RPC - MCP Market
https://mcpmarket.com/server/swiftmcp

30 Fastlane MCP Server - LobeHub
https://lobehub.com/mcp/loidodev-fastlane-mcp-server