ChatGPT

Technical Interview Questions for Mid-Level and Senior Android Developers (2025)

Jetpack Compose and UI

- **Jetpack Compose vs. Android View System:** What are the key differences between Jetpack Compose's declarative UI approach and the traditional XML/View system (i.e. declarative vs imperative UI)? [1]
- **Composable Functions & Recomposition:** What is a Composable function in Jetpack Compose, and how does **recomposition** work when the state changes? [2]
- **State Management in Compose:** How is UI state managed in Jetpack Compose? For example, explain the use of state in Compose (and the difference between **stateful** and **stateless** composables) [3] .
- **Side-Effects in Compose:** What are side-effect APIs in Compose, and when would you use effects like `LaunchedEffect` vs. `DisposableEffect`? [4]
- **Remember vs. RememberSaveable:** What's the difference between using `remember` and `rememberSaveable` in Compose for preserving state across configuration changes? [5]
- **State Hoisting:** What is *state hoisting* in Jetpack Compose and why is it considered a best practice for building reusable components? [6]
- **Interop with Views:** Can you use Jetpack Compose UI components and traditional Android Views within the same app or screen? How do they integrate (e.g. using `ComposeView` or vice-versa)? [6]
- **Navigation in Compose:** How do you handle navigation between screens in a Jetpack Compose app (for example, using the Jetpack Navigation component or `NavHost` in Compose)? [7]

Kotlin Language and Concurrency

- **Coroutines Overview:** What are Kotlin **coroutines** and why are they useful for Android development (versus using traditional threads)? [8]
- **Suspend Functions:** What does the `suspend` keyword do in Kotlin? Explain how *suspend functions* work in coroutines and why we use them [9] .
- **Launch vs Async vs RunBlocking:** What is the difference between `runBlocking`, `launch`, and `async` in Kotlin coroutines, and when would you use each? [10]
- **Dispatchers:** When working with coroutines, when should you use `Dispatchers.IO` vs `Dispatchers.Default` (what kinds of work is each dispatcher suited for)? [11]
- **Coroutine Scope in Android:** Explain the difference between a general `CoroutineScope` and Android's **lifecycle-aware** scopes like `lifecycleScope` and `viewModelScope`. Why are the lifecycle-aware scopes recommended in Android? [12]
- **Kotlin Flow Basics:** What is a **Flow** in Kotlin and how do you use it? For example, how do operators like `collect`, `map`, and `filter` work in a Flow pipeline? [13]
- **Flow vs LiveData:** What are the differences between Kotlin Flow and LiveData for reactive data updates in Android? When might you choose one over the other? [14]

- **One-Time Events:** How can you handle one-off events (such as navigation events or showing a toast/snackbar) in an MVVM architecture? *(For instance, patterns to send events from a ViewModel without using LiveData "peek" hacks)* [15]
- **Sealed Classes vs Enums:** What are **sealed classes** in Kotlin and how do they differ from enums? Give a real-world use case of sealed classes (e.g. representing UI states) [16] .
- **Inline Functions:** What are **inline functions** in Kotlin and when should you use them? Discuss the benefits (and potential downsides) of marking functions as `inline` [17] .

Architecture and Modern Android Practices

- **MVI Architecture:** Describe the Model-View-Intent (MVI) architecture pattern in Android. What are its benefits compared to MVP/MVVM, and what trade-offs or challenges come with using MVI? [18]
- **Dependency Injection (Hilt):** How do you implement dependency injection with Hilt in Android? For example, how are dependencies provided to **ViewModels** or **Fragments/Activities** using `@HiltViewModel`, `@Inject`, and `@AndroidEntryPoint`? [19]
- **WorkManager vs Coroutines/Services:** When should you prefer using **WorkManager** instead of a coroutine, `AsyncTask` (deprecated), or foreground service for background work? In what scenarios is WorkManager the appropriate solution? [20]
- **DataStore vs SharedPreferences:** What is Jetpack **DataStore** and why is it recommended over `SharedPreferences` for storing user preferences or small data? [21]
- **Navigation Component:** How does the Jetpack **Navigation Component** help manage fragment transactions and navigation compared to manual fragment management? What benefits does it provide (e.g. handling the back stack, type-safe arguments)? [22]
- **Avoiding Memory Leaks:** Android apps can suffer memory leaks if not careful. How do you avoid memory leaks when using components like Fragments or other long-lived objects? *(For example, patterns like nullifying bindings in* `onDestroyView`*, avoiding static context references, etc.)* [23]

Testing and Performance

- **Testing ViewModel (Coroutines):** How do you unit test ViewModel logic that uses Kotlin coroutines? (Describe strategies such as using a `TestCoroutineDispatcher` or Kotlin's `runTest` with a test scope to control coroutine timing) [24]
- **Testing Compose UIs:** How would you write tests for Jetpack Compose UI components? *(For instance, verifying UI state changes using the Compose Testing framework – e.g.* `ComposeTestRule` *and testing flows or state with libraries like Turbine)* [25]
- **Performance Monitoring (Jank):** How do you monitor and optimize performance in a Compose-heavy Android application? For example, what tools or techniques would you use to detect **jank** or slow rendering, and how might you optimize rendering in Compose? [26]
- **Memory Leak Detection:** What is a **memory leak** in Android, and how can you detect and fix such leaks (for example, using tools like LeakCanary or Android Profiler)? [27]

Each of these questions reflects current Android development best practices and challenges. They cover core Kotlin language features, modern UI development with Jetpack Compose, recommended Jetpack libraries (and how they improve upon legacy approaches), as well as architecture patterns and concerns (concurrency, testability, performance) that mid-level and senior Android engineers are expected to handle in real-world projects. All are focused on technical knowledge and problem-solving, excluding any behavioral or soft-skill questions.

[1] [2] [3] [4] [5] [6] Jetpack Compose Interview Questions | by Amit Shekhar | Outcome School | Medium
https://medium.com/outcomeschool/jetpack-compose-interview-questions-c3bd7f5d947f

[7] 50+ Jetpack Compose Interview Questions and Answers
https://www.index.dev/interview-questions/jetpack-compose

[8] [9] [10] [11] [12] [13] [14] [15] [16] [17] [18] [19] [20] [21] [22] [23] [24] [25] [26] [27] 125 Android/Kotlin Interview Questions for 2025
https://www.curotec.com/interview-questions/125-android-kotlin-interview-questions/