PRACTICAL

INTERVIEW RUNTIME

ANDROID INTERVIEW PREPARATION

Kotlin, Coroutines, Compose, and System Design
for Senior and Staff Engineers.

MIKE SALARI

Interview Runtime

Android Interview Preparation

Kotlin, Coroutines, Compose, and System Design for Senior and Staff Engineers

- **Release:** September 10, 2026
- **Edition:** First Interview Runtime edition
- **Author:** Mike Salari
- **Official site:** interviewruntime.com

This is a practical preparation system built around observable evidence, correction, changed constraints, and retry. It is not a question bank and does not promise a hiring outcome.

Preface

Most interview books fail at the moment the interview becomes real.

They give you questions. Then more questions. Then polished answers written with unlimited time, complete information, and no interviewer interrupting halfway through. You finish a chapter recognizing every term. Recognition feels like readiness, so you keep reading.

Then the constraint changes.

The device goes offline after the user taps Pay. The process dies before the confirmation screen appears. Two collectors observe the same flow. A retry may repeat an operation that already succeeded. The interviewer asks who owns the truth, what survives, and how you would prove it.

The memorized answer has nowhere to go.

That is the old failure of technical preparation. Question banks reward recall. Topic lists treat every gap as equally important. Model answers hide the hesitation, wrong assumption, and correction that produced them. Books age around framework names while the harder engineering questions remain untouched. Most of all, they let you practice without preserving evidence. You read, nod, and feel better, but nothing tells you whether your next answer became safer, clearer, or easier to defend.

Feeling prepared and being prepared are indistinguishable from inside your own head.

I have conducted more than five hundred technical interviews from the hiring side. Capable Android engineers rarely fail because they have never heard of a library or API. They fail more quietly. A correct mechanism is attached to the wrong owner. A durable write is described as a successful submission. Cancellation is confused with failure. A clean architecture diagram says nothing about recovery. A strong production story reaches its conclusion before the candidate explains the decision that mattered.

The answer sounds solid. Nobody in the debrief can say what made it strong.

That is why this is a book, but it is built as a program.

The book gives you the explanations, examples, technical models, and interview context. The program makes you perform before coaching, preserve the attempt, score observable evidence, correct the first consequential gap, and try again after one constraint changes. Reading supplies material. The loop produces improvement.

The unit of progress is not a chapter finished. It is a repaired answer.

You will work on Kotlin, coroutines and Flow, Compose state and lifecycle, architecture, offline systems, testing, performance, security, live coding, system design, and leadership. But the names are not the curriculum. The curriculum is judgment: who owns the state, what promise the system can honestly make, which failure changes the design, what evidence would disprove your assumption, and how clearly another engineer can follow your reasoning.

Every serious attempt should leave an artifact: code, a test, a diagram, a timeline, a decision record, or a recording. That artifact keeps confidence from rewriting history. It shows what you said before the explanation made the answer feel obvious.

You do not need to read the whole book before you begin. Start with the answer you least want an interviewer to examine. Record it without help. Use the diagnostic to find the gap that changes the answer. Follow the shortest route that repairs it. Then repeat the attempt under a changed constraint.

Two honest limits matter.

This program cannot promise a job. Hiring still depends on the role, the company, the interviewers, the market, and your performance that day. It can give you something more defensible: a way to see whether your preparation changed your work.

And it will not flatter you. Your first score may be lower than expected. Keep it. The uncomfortable first attempt is not a verdict; it is the baseline. The second attempt matters only if the evidence changed.

Do not prepare until you feel ready.

Prepare until the difference is visible.

Copyright and publication notice

Interview Runtime: Android Interview Preparation First Interview Runtime edition. Released September 10, 2026.

This book is educational material. It is not legal, employment, security, or financial advice. The company and interview scenarios are original composites. They do not reproduce confidential interviews or private hiring material.

Do not upload proprietary employer code, confidential interview questions, candidate recordings, customer data, credentials, or private hiring material to public services.

The code examples are teaching material. Before using an idea in production, validate it against the Android versions, library versions, security model, accessibility requirements, performance constraints, and product contract of the system you are building.

Errata, accessibility, and updates

Corrections and accessibility reports are welcome at mike@salari.dev. Include the chapter, the relevant passage, and enough context to reproduce a technical issue.

The current edition and update information are available at interviewruntime.com.

Evidence and ethics contract

The scenarios in this book are constructed or composite practice material. They do not reproduce confidential employer questions or claim privileged access to a hiring process.

Treat a proposed test as a plan for learning, not as an executed result. Verify version-sensitive Android, Kotlin, and library behavior against current primary documentation before using it in production. The scores in this book guide deliberate practice; they do not predict a hiring decision.

Use your own work when preparing stories. Preserve confidentiality, name the boundary of your contribution, and never inflate an example to make the scope sound larger.

Table of contents

Contents: interview execution

Contents: field reference

About the author

Mike Salari is a Staff Mobile Engineer, mobile architect, and tech lead. He has spent fifteen years building production mobile systems, much of that work in fintech and other regulated environments where recovery, security, and release discipline are part of the product.

The narrower credential behind this book is his experience on the hiring side. He has conducted more than five hundred technical interviews, from early-career screens through Staff and engineering-management loops. He has watched strong engineers lose the decision because their reasoning stayed implicit, and he has written the feedback candidates rarely get to see.

This book reflects that vantage point. The Android scenarios focus on ownership, system behavior, evidence, and correction because those are the parts of an interview answer that survive beyond a memorized API name.

Mike writes about interview preparation at interviewruntime.com.

Contact

- Website and resources: interviewruntime.com
- Books and editions: mikesalari.dev
- Email: mike@salari.dev

Questions about the material, corrections, and accessibility are welcome. For technical corrections, include the chapter and the smallest example that reproduces the problem.

PART I · DIAGNOSE AND ROUTE

Start here: make your reasoning inspectable

In the next fifteen minutes, make an interview brief, record a two-minute answer, and find one weakness to investigate. Spend three minutes on the brief, two on the answer, seven reviewing it, and three choosing the next action. These are practice timeboxes, not employer standards. Mark unanswered questions for later; do not spend the session searching for them.

Begin with the interview you actually face

An Android role can involve very different work. One team needs deep client implementation. Another needs a migration across several products. Another needs a technical lead who can make delivery risks visible and settle ownership disputes. Preparing for the title alone can leave you practicing the wrong evidence.

Write down the role, the expected scope, and the information you have about the loop. Separate confirmed facts from guesses. A job description mentioning architecture does not establish that there will be a system-design round. A recruiter mentioning coding does not establish whether you will use an Android project or a plain editor.

Find out the duration and format of the rounds, the tools available, whether running tests is possible, and the rules for external assistance. Ask about expectations rather than confidential questions. Record unanswered items as unknowns.

Use this short interview brief:

Item	What to record
Target	Role, team, and expected ownership
Format	Confirmed rounds, duration, and tools
Evidence	Implementation, design, incident reasoning, or leadership examples likely to be relevant
Unknowns	Questions still awaiting confirmation
Time	Interview date and realistic preparation sessions
Rules	Documentation, internet, AI, and collaboration permissions

Update the brief when new information arrives. Your preparation plan is allowed to change.

Capture a decision before polishing it

Choose a real mobile project. It does not have to be large or famous. Use a decision for which you can explain your own responsibility.

Set a two-minute timer and answer:

> Describe a mobile engineering decision where two reasonable approaches competed. What constraint decided it, what did you personally do, and how did you evaluate the result?

Record locally or write under the same time limit. Preserve confidentiality as established in the evidence and ethics contract. If your experience is outside Android, say so; a cross-platform example can demonstrate decision-making while leaving Android knowledge to separate evidence.

Stop when the timer ends. Keep the first version.

Inspect the answer in three passes

First, find the decision. Underline the sentence that explains what you chose. If there is no such sentence, you delivered context rather than an answer. Your correction is not to eliminate all background. It is to establish the decision early enough that the background has a purpose.

Second, find the constraint. “We wanted maintainable code” does not distinguish the options. “Two teams needed to change the feature independently while the existing app remained supported” gives the decision a shape. Useful constraints eliminate or weaken plausible alternatives.

Third, find the evidence. Did you observe a result, run a comparison, inspect a failure, or acknowledge that measurement was missing? “It worked well” is a conclusion without a visible basis. Name what you observed and the limit of that observation.

Write one sentence for each pass:

- My decision was...
- The constraint that mattered most was...
- My evidence was..., with the limitation that...

If one sentence is difficult to complete, that is your first practice target.

From a component list to a decision

An incomplete answer might say:

> We introduced an offline architecture with a repository and local storage. I worked on the Android side. It improved reliability and made the feature easier to maintain.

There may be substantial work behind this answer. The listener cannot yet see it. Neither the user-visible failure nor the candidate's responsibility is clear, and the claimed improvement has no stated evidence.

A design answer can turn that vague description into a proposal:

> In this hypothetical delivery app, drivers needed to record an outcome while connectivity was unreliable. I owned the client decision about preserving a pending submission. We separated “recorded on this device” from “accepted by the service,” so the screen did not imply remote confirmation prematurely. The difficult boundary was a lost response: a retry could not assume the first request had failed. I would agree a duplicate-handling contract with the service team, then test the uncertain-response path before describing the design as reliable.

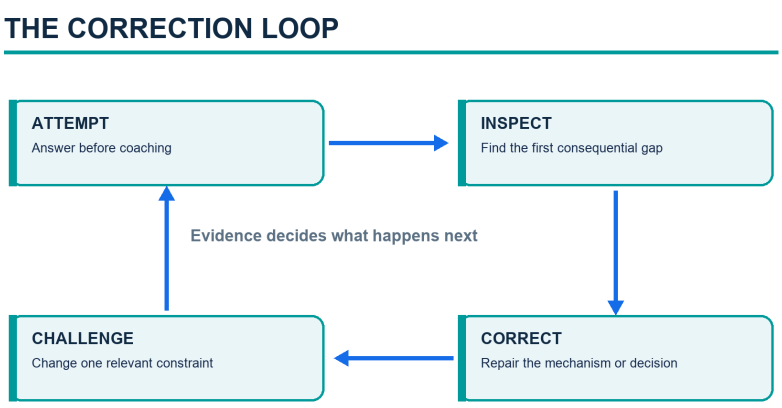
This proposal makes the reasoning easier to examine, but it is not a claim about work you personally delivered.

For your recording, use the facts of your project. Replace “improved reliability” with the observed change. Replace “I worked on the Android side” with your contribution: perhaps implementing the pending-state display, writing the migration, or finding the retry defect. Keep “we” for the team's decision and “I” for the work you can account for.

You may discover that the project is a poor example for this question. That is useful too. Choose a smaller decision you can explain accurately rather than enlarging your role in a more impressive project.

The answer is stronger because it is inspectable. The interviewer can challenge the status model, the ownership boundary, or the verification. A list of components offers fewer useful places to probe.

Use a repeatable practice cycle



The correction loop turns an uncoached attempt into evidence, a focused repair, and a changed-constraint challenge.

Attempt the problem without reading its worked answer. Preserve what you produced. Identify the first consequential gap, study the relevant reasoning, and attempt the problem again with one changed constraint.

Choose a consequential gap rather than the easiest cosmetic improvement. If an answer assumes an unknown server result is a failure, polishing the opening sentence will not resolve the design problem. If the design is sound but takes ten minutes to explain, practicing another API may not resolve the communication problem.

Change one fact on the retry. For the delivery scenario, add a second device, remove a backend capability, or require a status history. One change makes the reason for the new answer visible.

Keep a small practice record: prompt, constraints, time limit, first attempt, missing evidence, correction, retry result. One honest record is more useful than a large checklist of chapters you skimmed.

Keep technical knowledge and interview execution connected

A concise incorrect answer is still incorrect. A technically accurate answer that never addresses the user's problem is still incomplete. Practice the mechanism and the explanation together.

When you study a technical topic, ask what decision it changes. When you practice a story, ask which technical or organizational claim needs support. When you draw a system, explain the failure path before adding more boxes.

The diagnostic that follows separates these dimensions so you can locate a weakness. It does not suggest that they operate independently in a real round.

Your next action

Save the first recording and the three sentences. Write the next question you need answered about your interview. Then complete the diagnostic without studying its prompts in advance. You are establishing a baseline, not staging a demonstration.

Diagnose: find the gap that changes the answer

The purpose of this diagnostic is to decide what to practice next. It is not a certification, a ranking of engineers, or a prediction about a hiring decision. The anchors are practice tools used within this book, not an independently validated assessment.

Set aside forty-five minutes. Use the same conditions when you repeat it. If you used documentation, extra time, or hints, record that. Assistance is context, not something to hide.

The six dimensions

Keep six separate ratings: Kotlin and concurrency; Android platform judgment; system design; implementation and testing; scope and leadership; communication and execution. Do not average them into a readiness percentage. A severe misunderstanding in one area does not disappear because you speak clearly in another.

Use these anchors within each dimension:

Rating	Observable meaning
1	No workable approach, or a consequential misconception remains unresolved.
2	A partial approach appears, but important reasoning needs a hint or remains missing.
3	A defensible approach handles the stated problem and explains a relevant failure path.
4	The answer compares alternatives, explains verification, and adapts to a changed constraint.
5	The answer also identifies the limits of its evidence and establishes a reusable decision or ownership mechanism where the problem warrants one.

A score of 5 does not mean making the solution bigger. For a small coding task, a precise contract, a compact implementation, and one test that separates the competing behaviors can be enough. Judge the answer at the scale of the problem.

The diagnostic and the simulation lab use different scales for different jobs. The diagnostic rates six broad skills from 1 to 5 and helps you choose what to practice. A simulation rates eight dimensions from 0 to 3 against one longer attempt. Compare each exercise only with a later attempt of the same kind. Connect the two through the weakness you observed and the correction you made, not through arithmetic.

Run the diagnostic

Spend three minutes preparing the page and timer, six minutes on each of the five tasks below, and twelve minutes reviewing the evidence. Observe communication throughout rather than allocating a separate speech task.

For each task, spend four minutes on the initial problem and two on the changed constraint. A partner can reveal the change at minute four; working alone, cover it until then. Read the scoring notes only after preserving all five attempts. The requested artifact and the scheduled constraint change are instructions, not hints. Record extra coaching separately.

These prompts are hypothetical. They deliberately omit information. Name an assumption and explain what depends on it when an answer is unavailable. A partner should not invent a hidden fact and then penalize you for missing it. Clarification matters, but you still need to reach a provisional decision.

This is a small sample, not a complete test of each dimension. The form task cannot establish your performance, accessibility, or security judgment. Use the confirmed interview format to choose later probes. If a task cannot be attempted, record Not observed and the reason, not a fabricated rating of 1. For an interrupted task, preserve what you did produce and mark the missing portion. Restore missing evidence before comparing profiles.

Task 1: results arrive in the wrong order

A search screen issues work for each submitted query. The user submits “hotel,” then “hospital.” The first request finishes last, and the screen shows hotels while the input says hospital.

Explain the ownership of the work and the rule that determines which result may update the screen. Describe what happens if the screen leaves view.

Produce a short event timeline. It should show the two requests, the completion order, and the decision to accept or reject each result. You do not need to write code here.

Changed constraint at minute four: the underlying operation does not stop promptly when cancellation is requested. Update the timeline and explain what still prevents an obsolete result from appearing.

Task 2: A form returns in the wrong state

A customer is editing a multi-step form. After leaving the app and later returning, some fields are restored, others are missing, and the screen suggests that submission completed. The service has no confirmed submission associated with the customer.

Separate the state needed to render a screen, the customer's unsent work, and a confirmed business outcome. Ask what “returning” means in the report before assuming a particular lifecycle event. State what evidence you would collect to reproduce the problem.

Produce a state-ownership table with three columns: fact, owner, and recovery behavior.

Changed constraint at minute four: the user now expects to resume the form on another device. Revise the ownership and recovery decisions.

Task 3: an uncertain submission

A field worker records a completed inspection. The app sends it, receives no response, and later offers Retry. The product team wants no lost inspections and no duplicate records.

Define the user's intent and the uncertain state. Identify the client and service responsibilities. Explain what would make retry safe, what happens while safety is unresolved, and which user-visible status is justified.

Produce a compact decision record: chosen approach, rejected alternative, strongest assumption, and first failure test.

Changed constraint at minute four: the service team cannot change its contract before launch. State what you would now ship, defer, or investigate before deciding.

Task 4: implement a bounded contract

Use the language and editor permitted in your target interview. Write a function that receives a finite sequence of non-null string identifiers and returns the first occurrence of each identifier in input order. Comparisons are case-sensitive. Do not modify the input.

Before coding, give an example containing duplicates and state the expected output. Then implement the function and test an empty input, all-identical input, and a mixture of repeated and distinct values. Explain the additional space your approach uses. If execution is unavailable, distinguish your manual checks from tests you actually ran.

Changed constraint at minute four: identifiers now arrive as a stream that does not end. Explain the resource implications and which requirement you would negotiate. Stop coding even if the bounded version is unfinished; preserve its actual state. You do not need to implement the streaming version during this diagnostic.

Task 5: A shared migration stalls

Three teams depend on an older component. A replacement exists, but each team has a different reason to postpone adoption. The old

component still needs support, and an upcoming feature depends on a capability available only in the replacement.

Explain how you would learn the actual blockers, agree ownership, sequence adoption, and decide whether to delay the feature or carry temporary compatibility work. State what you can decide yourself and what needs agreement.

Produce a migration outline with one pilot, an observable success condition, a rollback or containment option, and a decision point.

Changed constraint at minute four: you have no reporting authority over the teams. Show which part of the plan changes. If you already assumed this, explain how the plan works without a mandate.

Review the five artifacts

Only now read these scoring notes. Use the same 1 to 5 anchors, not five different answer keys.

Task and dimension	Evidence to inspect	Do not mistake this for sufficient evidence
1: Kotlin and concurrency	Work lifetime, completion order, and the rule that rejects stale results even when cancellation is delayed	An operator name or an unsupported cancellation guarantee
2: Android platform judgment	A distinction between screen state, durable work, and confirmed outcome; a revised cross-device recovery model	Naming a state holder without explaining what survives
3: System design	An explicit uncertain state, client/service responsibilities, and a launch decision consistent with the available contract	Retry timing presented as duplicate prevention
4: Implementation and testing	First-occurrence order, case-sensitive equality, unchanged input, discriminating checks, and the streaming resource limit	A finished function without checks, or manual tracing described as execution
5: Scope and leadership	Actual blockers to discover, ownership, a bounded pilot, containment, and a decision without assumed authority	More meetings, a mandate, or an invented personal achievement

A rating of 3 needs a defensible decision and a relevant failure path. A 4 also needs comparison, verification, and adaptation; naming the new constraint is not adapting to it. A 5 must satisfy those earlier requirements, identify the evidence limit, and establish a reusable

decision or ownership mechanism where warranted. Do not demand organizational scale from a four-minute function.

Score communication across all five tasks

Listen for whether you established the problem, distinguished facts from assumptions, reached a decision, and used the remaining time to test it. Count hints and interruptions as context. If a partner asks a clarifying question because your wording is ambiguous, record what was unclear rather than automatically treating the question as disagreement.

A polished introduction does not compensate for an absent conclusion. Equally, a pause to correct a mistaken assumption can improve the answer. Judge whether the correction made the reasoning more accurate and easier to follow.

For each dimension, record a rating and one piece of evidence. A quote, a timestamp, a test result, or a specific omission is enough. “I felt confident” is not.

A worked scoring decision

Suppose the search answer proposes canceling previous work and correctly explains the screen's ownership. Under the changed constraint, the candidate says, “Then cancellation alone is not enough. I have not yet defined how to reject the old completion.” The answer has a plausible starting point and recognizes its limit, but the required rule is still missing.

On this rubric, 2 is defensible: the candidate identifies an unresolved gap without claiming it is safe. If the candidate instead insists that requesting cancellation guarantees obsolete results cannot appear, rate that unresolved consequential misconception 1. A clear explanation does not repair a false guarantee. In either case, record the exact claim, not “bad at coroutines.”

The correction is equally narrow. Draw two requests and reverse their completion order. Define the acceptance rule independently of whether either request stops. Then explain where that rule is enforced and how to test it. On the retry, add a third query rather than rehearsing the same two names.

On the retry, score the rule, the test, and the explanation actually produced. Do not automatically award 4 for repeating a correction you just supplied. Preserve both attempts and the assistance used. An independent attempt under a further constraint is stronger evidence of transfer than immediate repetition.

Choose the first correction

INTERVIEW RUNTIME 1.0 · ANDROID

Your first attempt is only the baseline.

The full edition turns interview knowledge into evidence you can inspect, correct, retry, and defend under pressure.

- The complete Android interview preparation book
- PDF and EPUB access
- 7, 14, and 30-day preparation routes
- Diagnostics, practice cycles, and simulations
- Access to the Interview Runtime dashboard

Buy the full Android edition

interviewruntime.com

Interview preparation, not a guarantee of employment or interview outcomes.