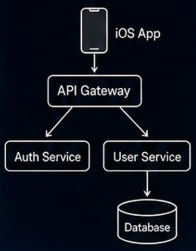
Share Your Screen.

Mastering the Senior iOS Live Coding and System Design Interview

Take-home projects, code review, and the communication strategies that pass senior and staff interviews, in Swift.

System Design



Take-Home Project Checklist

- [x] Clear Requirements
- [x] Architecture & Data Flow
- [x] Core Functionality
- [x] Edge Cases
- [x] UI/UX Polish
- [x] Testing & Documentation

Communication Strategies

- Think out loud.
- Ask clarifying questions.
- Align early, iterate together.
- Summarize and confirm.



MIKE SALARI

FREE FIELD GUIDE · FROM THE HIRING SIDE OF THE TABLE

Minute six

Where senior iOS candidates lose the live round, and what the interviewer is writing down while it happens.

MIKE SALARI

A free sample of Share Your Screen: Senior iOS Live Coding and System Design Interview

THE PROBLEM

You can build it. You still lose the round.

Two candidates get the same prompt: build a small feed that loads images into reusable cells.

The first reads it, nods, and starts typing. Ten minutes of silence. The code works, until a cell scrolls off screen mid-load and a stale image flashes into the wrong row, and the interviewer has to point it out.

The second says, before touching the keyboard: "The interesting part here is cell reuse, because a slow image can land in a recycled cell, so I want to cancel on reuse. I will come back to that on purpose."

Same skill. Different round.

At the senior level the interviewer is answering three questions. Can this person think clearly under pressure? Can this person make good decisions and explain them? Will this person be safe building production systems? The code is evidence for those answers, and so is everything you say while you write it.

This guide gives you the opening five minutes of a live round, then takes you into a real round that passed, up to the moment it was won.

Adapted from the introduction and chapter 12 of Share Your Screen.

The first five minutes, before you type

The opening shapes how the interviewer reads the rest of the hour. The instinct under a shared screen is to start typing, to prove you are productive. Resist it. Taking a moment before you code looks like thinking, not hesitation, when you make your reasoning visible. Run this sequence.

1. Read the prompt back in your own words. It confirms you understood and buys a moment to think. "So I am building a screen that shows a list of repositories with search, loading from an API, and I should handle the states that matter." It also surfaces the load-bearing words, here "search" and "from an API", where the difficulty lives.

2. Clarify only what changes your design. Two or three questions whose answers fork the build. Does search hit the server or filter a loaded list? Is offline in scope? Roughly how large is the list? Skip cosmetic questions; they signal stalling rather than scoping.

3. Name the risk out loud. "The interesting part here is the search, because typing quickly will race, so I want to handle cancellation. I will come back to that on purpose." Now the interviewer knows what to watch for, and you are the one who pointed it out.

4. Say the plan, then choose the first slice. "I will do a view, a view model with an injected service so it is testable, and I will get a vertical slice rendering first, then add search, then the loading and error states."

Pick the smallest thing that runs, usually the list rendering one page of data. A running app at minute ten gives you something to demo for the rest of the round.

5. Name your first checkpoint. "I will know the slice works when the list shows the stubbed results." It is a lightweight version of test-first thinking, and it keeps you honest about progress.

6. Build the data path before the UI. The most common opening mistake is laying out a beautiful cell before there is any data flow, which burns the clock on the easy part and leaves the state and the async for the panicked final minutes.

All of it fits in under five minutes, and by your first real line of code the interviewer has already seen how you think.

Restate › Clarify › Name the risk › Plan and first slice › Checkpoint › Data path first

From chapter 12 of Share Your Screen, which then builds a searchable article list end to end on the same clock.

EVIDENCE

A real round that passed

Forty-five minutes. A senior iOS role at a payments company, a shared editor, one interviewer. Anonymized, with the shape of the round preserved. Here is how it opened, from the transcript.

Interviewer: The task is a token manager for a payments app. It hands out an auth token, and if the token is missing or expired it refreshes it using a refresh call. I want it to be safe if several callers ask at once. Start by writing a failing test, then implement, then add more tests.

Candidate: Got it. Before I write anything, let me restate the behavior so we agree on it. There is one piece of state, the current token, and one operation, get a valid token. If we have a fresh token we return it. If not, we refresh once and everyone gets that result. And the interesting part is the "at once," which tells me concurrency is the real subject here. Let me start with the simplest failing test, that asking for a token when we have none triggers exactly one refresh.

Candidate: This will not even compile yet, which is the point. It pins down the contract before I write the type.

> **What the interviewer noticed.** Restating the behavior before typing, and naming "at once" as the real subject, told the interviewer that this candidate took the failing-test instruction as a way into the problem and not as a chore. The test counts refresh calls instead of only checking the returned string, which already hints that the race is the point.

Then, on purpose, the wrong version: a class with a `Bool isRefreshing` flag, just enough to go green. The interviewer asked one question.

> **"What happens if two parts of the app call this at once when the token is expired?"**

This is the pivot of the round. "Use an actor" is a common answer here, and it fails two questions later, because an actor alone does not guarantee a single refresh.

What the candidate said next, the analogy the interviewer named in the feedback, the code that closed the round, and the cancellation call senior candidates often get backwards are in the book.

Read the full round in case study 1 and Mock 1.

MINUTE SIX

When your mind goes blank

Every real round has a hard moment, often a few minutes in, when the first plan meets the first surprise. The interviewer watches how you handle it more than whether it happens. A solve with a narrated recovery can score as well as a clean one, because it is evidence of the thing they are hiring for. The version that reliably fails is the silent freeze.

Do not search for brilliance. Run a fixed sequence. The first two steps:

Name it, without apology. "Let me take a second to think about the cleanest approach here." That buys time, and the interviewer hears composure.

Shrink it. State the smallest version you can definitely make progress on. "Let me get the list rendering with hardcoded data first, then layer in the network." You need the next small, certain step, not the whole solution.

The rest of the sequence, what to say when a build breaks in front of them, how to ask for a hint without sounding stuck, and recovery phrases for every kind of stall, is in chapter 19 and Appendices C and J.

From chapter 19 of Share Your Screen.

READERS

What readers say

Rated 5.0 by readers on salari.dev and Gumroad.

> "The test-first opening strategy and the mock transcripts gave me a reliable script for the first five minutes, and momentum carried me through the rest."

David H., iOS Engineer

> "Most iOS interview books stop at 'here's the answer.' This one shows you the entire conversation: the interviewer's follow-ups, where candidates stall, and what a strong recovery looks like. I ran two of the annotated mocks with a colleague before my onsite, and I got the offer."

Daniel K., Senior iOS Engineer

> "What I love most is that Mike shares his real experience instead of just listing questions and answers. ... this book has already changed the way I think about live coding sessions."

Ivan G., verified purchase

> "If you have an upcoming technical interview and even if you think you are prepared you should buy Share Your Screen. ... It helped me during my interview processes but also how I structure my code better and do better development."

Aurelian G., verified purchase

You have the first five minutes. The book has the other forty.

Most interview books teach you what to answer. This one trains you for what happens after your first answer.

The follow-up that changes the problem. The failing test. The design decision you have to defend. The moment your build breaks and the interviewer is still watching.

See how strong candidates work through those moments:

> **Annotated mock interviews.** Seven full rounds, minute by minute, with what the interviewer noticed.

> **Interviewer scorecards.** The rubric for every format, plus runsheets so a friend can run you through a round.

> **Tested Swift companion.** Reference implementations with 76 tests and nine exercises you can practice with.

Read the full round. Prepare for yours.

Get the book →

Get the full book

salari.dev/books/share-your-screen-live-coding-ios
mikesalari.gumroad.com/l/share-your-screen-live-coding-ios

Share Your Screen: Senior iOS Live Coding and System Design Interview. 603 pages, PDF and EPUB, with the companion Swift package.
Mike Salari. Staff mobile engineer · 500+ technical interviews from the hiring side.