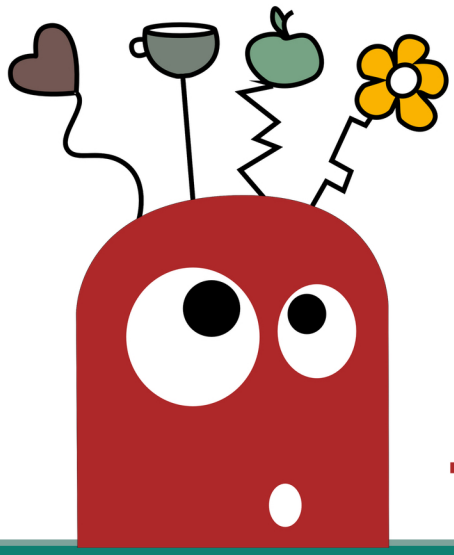
2026 edition

over 300 questions & answers



TANASCHITA.COM

Preparing for a technical

iOS Job Interview

Questions & Answers

covering

Swift | SwiftUI | Architecture | Persistence
Design Patterns | Concurrency | Combine
Authentication | Server Communication
AI & Machine Learning | Automated Testing
Dependency Management | Accessibility
Working with Legacy Code

by Natascha Fadeeva

Table of Contents

6. Server Communication

7. Concurrency

8. Persisting Data

9. Security

10. Automated Testing

11. Architecture & Design Patterns

12. Accessibility

13. Dependency Management

14. AI & Machine Learning

15. Working with Legacy Code: UIKit

16. Working with Legacy Code: Objective-C

Final Notes

Introduction

About this book

Having been a freelance iOS dev for over a decade now, I participated in quite a few hiring interviews for various projects and have also been in a position of leading the technical interviews myself.

With this book, I want to give you a guide for technical iOS questions you might be asked in such an interview. These questions aren't meant to be memorized or used as a checklist of must-know topics. Instead, think of them as guiding stars, designed to help you reflect on your knowledge, spark curiosity, and uncover areas you may want to explore more deeply.

Some questions will feel immediately relevant. Others might dip into edge cases or advanced areas. Not every role requires mastery of every topic, and you are certainly not expected to know everything.

That said, while iOS development, and software development in general, is more about understanding and critical thinking than having all the answers, it can still be calming to refresh your memory before an interview. Skimming through questions like these can warm up your technical thinking, surface things you already know, and help you walk into the conversation with more confidence. At least, I hope it does.

I wish you a happy learning, and may your journey in iOS development be as exciting as it is rewarding.

Structure of this book

Each chapter starts with a questions overview that allows you to check your knowledge without seeing the answers. All questions are numerated so you can quickly find the answer when needed. The following topics are covered in this book:

Swift

Being the main programming language for the iOS platform, a deep knowledge in Swift is expected from an iOS developer. This chapter provides questions and answers on Swift and general programming language concepts.

SwiftUI

SwiftUI is Apple's main framework for building user interfaces for iOS. This chapter provides questions and answers on the main aspects of SwiftUI.

Xcode

Being the main iOS development tool, a confident usage of Xcode is expected from an iOS developer. This chapter provides questions and answers on working and debugging with Xcode.

Combine

The Combine framework is Apple's native solution for handling asynchronous events and data streams in Swift. It enables developers to write functional reactive code, streamlining tasks like chaining, transforming, and combining data. This chapter provides questions and answers on Combine and on functional reactive programming concepts in general.

Server Communication

Server communication is part of almost every iOS application. This chapter provides questions and answers on general computer networking concepts and on specific iOS networking topics.

Concurrency

This chapter focuses on Swift's concurrency model with async/await. It also provides questions and answers on more low-level technologies like Grand Central Dispatch and Operations. Since they were used in iOS applications before async/await, knowledge around these technologies may be required for an iOS position.

Persisting Data

Persisting pieces of data is required in many applications. Apple offers solutions to store data for different scenarios. This chapter contains questions and answers on persistence technologies and frameworks such as File System, UserDefaults, SwiftData and Core Data.

Security

Many applications need to manage some kind of sensitive user data. To ensure a high level of security Apple provides different technologies to be used by developers. This chapter provides questions and answers on specific security-related iOS technologies such as keychain or shared web credentials, general cryptography concepts, and privacy-related requirements for apps and third-party SDKs.

Automated Testing

Testing is an important part in every software development process. Apple offers tools to write unit, UI and performance tests for iOS applications. This chapter provides questions and answers on general testing concepts and specific iOS testing tools.

Architecture & Design Patterns

Architectural principles and design patterns are the foundation of robust, maintainable, and scalable iOS applications. This chapter explores key architectural concepts and widely used design patterns in iOS development, providing practical examples and insights into their strengths and weaknesses.

Accessibility

Accessibility is a crucial aspect of iOS development that ensures an app can be used by people with disabilities. This chapter covers the fundamental concepts and implementation techniques for making iOS applications accessible to all users.

Dependency Management

Most iOS apps depend on code outside the app target itself. This can include Swift packages, binary SDKs, local modules, or older dependency managers such as CocoaPods and Carthage.

Dependencies can speed up development, but they also affect build times, app size, privacy, security, maintenance, and long-term flexibility. This chapter covers how dependencies work in modern iOS projects and what to consider when adding, updating, or relying on third-party code.

AI & Machine Learning

AI and machine learning are an increasingly important part of software development and engineering jobs. Machine learning techniques are used to build models that can learn patterns in data and make predictions, classifications, or generations without being explicitly programmed for every case.

On Apple platforms, this includes Apple Intelligence and the Foundation Models framework, classic machine learning workflows with Core ML and Create ML, and newer on-device AI capabilities with Core AI. This chapter starts with the AI and machine learning tools iOS developers are most likely to use in apps, then covers the machine learning concepts that are useful to know when working with these technologies.

Working with Legacy Code: UIKit

With SwiftUI pushing UIKit into the background since 2019, UIKit may still be part of existing iOS applications. A deep knowledge of UIKit may be required for an iOS position. This chapter provides questions and answers around key aspects of UIKit.

Working with Legacy Code: Objective-C

Objective-C is becoming more and more obsolete after Swift was released in 2014. However, with existing apps written in Objective-C still out there, knowledge on Objective-C may be required for an iOS position. This chapter provides questions and answers on Objective-C basics and the interoperability between Swift and Objective-C.

Copyright

7th Edition, released in June 2026. Self-published.

Contact: **tanaschita@gmail.com**

Website: **tanaschita.com**

Author: **Natascha Fadeeva**

2. Swift

Being the main programming language for the iOS platform, a deep knowledge in Swift is expected from an iOS developer. This chapter provides questions and answers on Swift and general programming language concepts.

Questions Overview

2.1 Swift Language Features

2.1.1 Swift is a type-safe language. What does type safety mean, and why is it important?

2.1.2 Swift supports type inference. What does it mean, and why is it useful?

2.1.3 What are optionals in Swift and what are their benefits?

2.1.4 Swift supports extensions. Which benefits do extensions have?

2.1.5 What is the difference between value and reference types in Swift?

2.1.6 How do classes differ from structs in Swift, and what considerations apply when choosing between them?

2.1.7 What is protocol-oriented programming in Swift, and how do protocols contribute to building modular and flexible code?

2.1.8 Which benefits do generics have? How can we define a generic function in Swift?

2.1.9 What are Swift macros, and what role do they play in Swift programming?

2.2 Swift Variables & Properties

2.2.1 What is the difference between `let` and `var` in Swift, and how does it impact code design?

2.2.2 What is a lazy stored property in Swift, and when is it useful?

2.2.3 What are computed properties in Swift, and what benefits do they provide?

2.2.4 What are property wrappers in Swift?

2.1 Swift Language Features

2.1.1 Swift is a type-safe language. What does type safety mean, and why is it important?

Type safety means that the compiler enforces rules to prevent type errors by ensuring values are used consistently with their declared types. For example, if we declare a variable as an `Int`, we cannot assign a `String` value to it. The compiler checks for type mismatches during compilation and provides errors, helping developers catch and fix issues early in the development process.

This feature improves code reliability and reduces runtime errors because many potential bugs are identified before the app runs. Type safety also enhances code readability and maintainability, as the expected data types are clear from the code.

```
var age: Int = 10
age = "ten" // Compiler error: Cannot assign a value of type 'String' to a variable of type 'Int'
```

By enforcing type safety, Swift encourages robust and predictable programming practices, making code safer and easier to debug.

2.1.2 Swift supports type inference. What does it mean, and why is it useful?

Type inference is the ability of the Swift compiler to automatically determine the type of a variable or expression based on the assigned value or context, without requiring the programmer to explicitly specify the type.

For example:

```
var age = 10
```

In this case, we did not specify the type of `age`. The compiler infers that age has the type `Int` because the assigned value is an integer.

Type inference makes code more concise and readable by reducing the need for explicit type annotations. At the