# Top 140 IOS Mobile Testing Interview Questions (with common XCUITest Code challenges )

**Author:** Lamhot Siagian  LinkedIn

---

# Chapter 1 – Introduction to XCUITest

1. What role do UI tests play in an iOS QA strategy, and how do they differ from unit tests?
2. Compare XCUITest and Appium—what are the pros and cons of each for iOS UI automation?
3. Describe the architecture of the XCTest framework: what are test targets, bundles, and runners?
4. How does Xcode discover and execute UI tests under the hood?
5. In your first XCUITest, how would you verify that your app has successfully launched?
6. What are the limitations of XCUITest compared to lower-level testing (e.g., unit or integration tests)?
7. Explain how the test bundle communicates with the host app at runtime.
8. How do you structure your XCUITest target within an Xcode project for maximum maintainability?
9. Describe a scenario where UI tests would catch issues that unit tests would not.
10. What's your strategy for organizing and naming XCUITest cases and methods?

---

# Chapter 2 – Environment Setup & Best Practices

1. How do you add and configure a UI-test target in Xcode for an existing app?
2. Explain the importance of signing and provisioning profiles for UI-test targets.
3. How do launch arguments and environment variables enhance test flexibility?
4. Why are accessibility identifiers critical for stable UI tests, and how do you apply them in SwiftUI vs. Storyboards?
5. Describe a best-practice folder structure for keeping test assets and data organized.
6. How would you handle cases where identifiers aren't available on a third-party SDK view?
7. What's your process for synchronizing the host app's build settings with the test target?
8. How do you ensure tests remain reliable when the app's UI layout changes?

9. Explain how you'd manage secrets or credentials required during UI tests.
10. How do you integrate code reviews or linting into your XCUITest codebase?

---

# Chapter 3 – Locating Elements

1. What are the differences between `app.buttons["…"]`, `app.staticTexts["…"]`, and `app.otherElements["…"]`?
2. How do NSPredicate queries improve element-lookup robustness?
3. Write an NSPredicate that matches any button whose label begins with "OK".
4. When would you choose coordinate-based interactions over element queries?
5. Explain the trade-offs between accessibility labels, identifiers, and traits.
6. How do you debug "element not found" errors in XCUITest?
7. Describe a fallback strategy when an element's identifier changes frequently.
8. How can you combine multiple query filters (e.g., type AND label) in code?
9. What performance considerations exist when querying large view hierarchies?
10. How do you verify that the element you located is indeed unique on screen?

---

# Chapter 4 – Basic Interactions

1. How do you simulate a text entry into a secure text field?
2. What methods exist for dismissing the system keyboard, and when should you use each?
3. Explain the difference between `.tap()` and `.press(forDuration:)`.
4. How do you verify an element is both present and hittable before interacting?
5. Describe how `.waitForExistence(timeout:)` works under the hood.
6. What's the benefit of using XCUITest assertions (`XCTAssert*`) versus manual `if-else` checks?
7. How do you retrieve and validate the value of a UI element (e.g., a text field)?
8. Discuss strategies for typing into a field that already contains placeholder text.
9. How would you handle intermittent failures due to keyboard focus issues?
10. Explain a scenario where you'd need to chain interactions (e.g., tap→type→swipe) within one test.

---

# Chapter 5 – Gestures & Advanced Actions

1. How do you programmatically perform a swipe on a `XCUIElement`?
2. Describe how `.pinch(withScale:velocity:)` differs from `.press(forDuration:thenDragTo:)`.
3. What are the common pitfalls when automating pinch-to-zoom on image views?
4. How do you scroll within a table view versus a collection view?
5. Explain how you'd calculate the coordinates for a drag-and-drop gesture.
6. How do you test multi-finger gestures if needed?
7. What strategies ensure gesture reliability across different screen sizes?
8. How can you detect and handle overscroll behavior in a scroll view?
9. Describe how you'd test a custom gesture recognizer in your app.
10. What role do velocity and duration play in advanced gesture accuracy?

---

# Chapter 6 – Synchronization & Flakiness

1. Why is `sleep()` discouraged in XCUITest, and what should you use instead?
2. Compare `waitForExistence(timeout:)` vs. `expectation(for:evaluatedWith:handler:)`.
3. How do you use `XCTNSPredicateExpectation` to wait for dynamic content?
4. Describe three common causes of flaky UI tests and how to mitigate each.
5. Explain how network delays can affect test stability and how to work around them.
6. What's the benefit of isolating animations from test execution?
7. How do you verify an activity indicator has disappeared before proceeding?
8. Describe using launch arguments to enable "fast-path" test modes in your app.
9. How do you structure retry logic for transient UI failures?
10. Explain how CI environments may exacerbate flakiness compared to local runs.

---

# Chapter 7 – Alerts, Sheets & System Prompts

1. How do you tap an alert button once it appears?
2. Explain how `addUIInterruptionMonitor` works for system dialogs.
3. What's the typical pattern for handling a location or camera permission prompt?
4. How do you trigger and verify an alert appears in your test?
5. Describe error-handling if an expected alert never shows up.
6. How can you test multiple sequential system prompts in one flow?
7. What are the limitations of `UIInterruptionMonitor`?
8. How do you return the app to a clean state after dismissing alerts?

9. Explain strategies for testing custom action sheets.
10. How do you record and replay interactions with modal sheets?

---

# Chapter 8 – Complex Controls

1. How do you adjust a `UIPickerView` wheel to a specific value?
2. Explain how `.adjust(toNormalizedSliderPosition:)` works under the hood.
3. How can you verify the selected segment in a `UISegmentedControl`?
4. Describe testing a toggle switch's on/off states.
5. What's your approach for interacting with date pickers in different locales?
6. How do you test custom controls that aren't standard UIKit elements?
7. How do you chain picker and slider interactions in one test?
8. Explain the pitfalls of testing nested container controls.
9. How do you validate that the UI reflects data changes after control adjustments?
10. What strategies ensure these interactions work on both iPhone and iPad form factors?

---

# Chapter 9 – Test Data & Mocking

1. How do you inject mock URLs or flags into your app via launch arguments?
2. Describe using local JSON bundles to stub network responses.
3. Explain how you'd integrate a lightweight mock server like Swifter.
4. How do you switch between production and test endpoints at runtime?
5. What are the benefits of dependency-injecting your network layer for tests?
6. How do you verify UI behavior when the API returns an error?
7. Describe organizing test fixtures and sample data in your repo.
8. How do you handle large datasets that don't fit into JSON bundles?
9. What strategies exist for resetting mock state between test methods?
10. Explain how you'd audit that no real network calls slip through in CI.

---

# Chapter 10 – Screenshot & Recording

1. How do you capture a screenshot on assertion failure?
2. Explain using `XCTAttachment` to embed images in test reports.
3. What setup is required to enable video recording in XCUITest?
4. How do you configure your scheme to automatically record test sessions?
5. Describe strategies for naming and organizing captured assets.
6. How can you compare screenshots to a reference baseline?
7. What are the trade-offs of embedding attachments vs. external storage?

8. How do you ensure recordings don't bloat your CI artifacts?
9. Explain how you'd capture a photo of a specific view hierarchy region.
10. What's your process for reviewing and triaging screenshot failures?

---

# Chapter 11 – Performance & Launch Metrics

1. How do you measure application launch time using `measure(metrics:)`?
2. Describe adding custom signposts for key user-flow benchmarks.
3. Explain how to interpret the results of an `XCTOSSignpostMetric`.
4. How do you configure performance tests to run only on demand?
5. What's the impact of performance tests on overall test-suite runtime?
6. How do you detect regressions in launch time across commits?
7. Describe integrating metric results into a CI dashboard.
8. How do you handle noise from background system processes?
9. Explain best practices for isolating the test device's performance.
10. How do you verify that a code change improved a measured metric?

---

# Chapter 12 – CI/CD Integration & Parallel Tests

1. How do you configure GitHub Actions (or Jenkins) to run XCUITests on simulators?
2. Explain the `-parallel-testing-enabled YES` flag and its implications.
3. How do you shard tests across multiple simulators or devices?
4. Describe handling simulator lifecycle (creation, cleanup) in CI.
5. How do you manage UDID and signing for real-device farms?
6. What strategies ensure tests remain isolated when run in parallel?
7. Explain collecting and aggregating test reports from distributed runs.
8. How do you handle flaky tests differently in CI vs. local development?
9. Describe monitoring resource usage (CPU, memory) during CI test runs.
10. How would you roll out a test-only build to your CD pipeline without impacting production?

---

# Chapter 13 – 20 common XCUITest challenges you'll often encounter in real-world iOS UI tests

1. **Verify App Launch**

   ```swift
   func testAppLaunch() {
       let app = XCUIApplication()
       app.launch()
       XCTAssertTrue(app.buttons["mainButton"].exists)
   }
   ```

2. **Tap a Button and Verify Navigation**

```
func testTapLoginButton() {
    let app = XCUIApplication()
    app.launch()
    app.buttons["loginButton"].tap()
    XCTAssertTrue(app.staticTexts["welcomeLabel"].waitForExistence(timeout: 5))
}
```

3. **Enter Text in a Text Field**

```
func testEnterUsername() {
    let app = XCUIApplication()
    app.launch()
    let usernameField = app.textFields["usernameField"]
    XCTAssertTrue(usernameField.exists)
    usernameField.tap()
    usernameField.typeText("test_user")
    XCTAssertEqual(usernameField.value as? String, "test_user")
}
```

4. **Handle Secure Text Field**

```
func testEnterPassword() {
    let app = XCUIApplication()
    app.launch()
    let passwordField = app.secureTextFields["passwordField"]
    passwordField.tap()
    passwordField.typeText("P@ssw0rd")
    // Secure fields return •••••, so check that it isn't empty
    XCTAssertFalse((passwordField.value as! String).isEmpty)
}
```

5. **Dismiss Keyboard**

```
func testDismissKeyboard() {
    let app = XCUIApplication()
    app.launch()
    let field = app.textFields["searchField"]
    field.tap()
    field.typeText("hello")
    app.keyboards.buttons["Return"].tap()
    XCTAssertFalse(app.keyboards.element.exists)
}
```

6. **Scroll a Table View to a Specific Cell**

```
func testScrollToCell() {
    let app = XCUIApplication()
    app.launch()
    let table = app.tables["mainTable"]
    let cell = table.cells.element(boundBy: 20)
    table.scrollToElement(element: cell)
    XCTAssertTrue(cell.exists)
}
// Helper extension:
extension XCUIElement {
    func scrollToElement(element: XCUIElement) {
        while !element.isHittable {
            swipeUp()
        }
    }
}
```

7. **Scroll a Collection View**

```
func testScrollCollection() {
    let app = XCUIApplication()
    app.launch()
    let collection = app.collectionViews["imageGrid"]
    let targetCell = collection.cells["imageCell_50"]
    collection.swipeUp() // or loop until hittable
    XCTAssertTrue(targetCell.waitForExistence(timeout: 5))
}
```

8. **Swipe to Delete a Cell**

```
func testSwipeToDelete() {
    let app = XCUIApplication()
    app.launch()
    let cell = app.tables["todoList"].cells["task_3"]
    cell.swipeLeft()
    cell.buttons["Delete"].tap()
    XCTAssertFalse(cell.exists)
}
```

9. **Select from a Picker Wheel**

```
func testSelectPickerValue() {
    let app = XCUIApplication()
    app.launch()
    app.buttons["showPicker"].tap()
    let picker = app.pickers.pickerWheels.element
    picker.adjust(toPickerWheelValue: "March")
    XCTAssertEqual(picker.value as? String, "March")
}
```

10. **Toggle a Switch**

```
func testToggleSwitch() {
        let app = XCUIApplication()
        app.launch()
        let darkModeSwitch = app.switches["darkModeSwitch"]
        let originalState = darkModeSwitch.value as! String
        darkModeSwitch.tap()
        XCTAssertNotEqual(darkModeSwitch.value as! String, originalState)
    }
```

11. **Adjust a Slider**

```
    func testAdjustSlider() {
        let app = XCUIApplication()
        app.launch()
        let volumeSlider = app.sliders["volumeSlider"]
        volumeSlider.adjust(toNormalizedSliderPosition: 0.7)
        XCTAssertEqual(volumeSlider.value as! String, "70%")
    }
```

12. **Use a Segmented Control**

```
    func testSegmentedControl() {
        let app = XCUIApplication()
        app.launch()
        let segment = app.segmentedControls["optionsSegment"]
        segment.buttons["Second"].tap()
        XCTAssertTrue(app.staticTexts["selectedSecond"].exists)
    }
```

13. **Handle an Alert**

```
    func testHandleAlert() {
        let app = XCUIApplication()
        app.launch()
        app.buttons["showAlert"].tap()
        let alert = app.alerts["Warning"]
        XCTAssertTrue(alert.exists)
        alert.buttons["OK"].tap()
        XCTAssertFalse(alert.exists)
    }
```

14. **UI Interruption Monitor (e.g., Permissions)**

```
    func testHandlePermissionAlert() {
        let app = XCUIApplication()
        addUIInterruptionMonitor(withDescription: "Permissions") { alert in
            if alert.buttons["Allow"].exists {
                alert.buttons["Allow"].tap()
                return true
            }
            return false
        }
        app.launch()
        app.buttons["accessCamera"].tap()
        app.tap() // trigger the monitor
        XCTAssertTrue(app.otherElements["cameraView"].exists)
    }
```

15. **Long Press Gesture**

```
func testLongPress() {
    let app = XCUIApplication()
    app.launch()
    let element = app.images["profilePicture"]
    element.press(forDuration: 2.0)
    XCTAssertTrue(app.buttons["editPhoto"].exists)
}
```

16. **Drag and Drop**

```
func testDragAndDrop() {
    let app = XCUIApplication()
    app.launch()
    let from = app.cells["item_1"]
    let to = app.cells["item_5"]
    from.press(forDuration: 1.0, thenDragTo: to)
    XCTAssertTrue(to.images["item_1"].exists)
}
```

17. **Pinch to Zoom**

```
func testPinchToZoom() {
    let app = XCUIApplication()
    app.launch()
    let map = app.images["mapView"]
    map.pinch(withScale: 2.0, velocity: 1.0)
    // verify zoom by checking some UI change
    XCTAssertTrue(app.buttons["zoomedInButton"].exists)
}
```

18. **Launch with Arguments & Environment**

```
func testLaunchInDemoMode() {
    let app = XCUIApplication()
    app.launchArguments = ["-DemoMode", "YES"]
    app.launchEnvironment = ["UITest": "1"]
    app.launch()
    XCTAssertTrue(app.staticTexts["demoBanner"].exists)
}
```

19. **Measure App Launch Performance**

```
func testAppLaunchPerformance() {
    measure(metrics: [XCTOSSignpostMetric.applicationLaunch]) {
        XCUIApplication().launch()
    }
}
```

20. **Capture and Attach a Screenshot**

```
func testTakeScreenshot() {
    let app = XCUIApplication()
    app.launch()
    // perform some actions…
    let screenshot = XCUIScreen.main.screenshot()
    let attachment = XCTAttachment(screenshot: screenshot)
    attachment.lifetime = .keepAlways
    add(attachment)
}
```