□□□□□□□□□□□□ iOS □□

Dieser Blogbeitrag wurde mit Unterstützung von ChatGPT-4o verfasst.

Die Bedeutung von Unit-Tests

Bei LeanCloud haben wir frühzeitig in unseren Projekten Unit-Tests implementiert, was sich als äußerst wertvoll erwiesen hat. Jeder Pull-Request (PR) löst auf Jenkins Unit-Tests aus, und unser Ziel für die Testabdeckung liegt bei etwa 80%. Es gibt zwei Hauptszenarien, in denen wir Tests schreiben: zum einen, um neue Schnittstellen zu validieren, und zum anderen, um Fehler zu reproduzieren und zu beheben. Je mehr Tests wir ansammeln, desto robuster wird unsere Codebasis. Automatisierte Tests ermöglichen es uns, Code mit Zuversicht zu veröffentlichen und umzustrukturieren, ohne manuelle Überprüfungen durchführen zu müssen.

Testprozesse und praktische Anwendungen

Hier sind einige praktische Beispiele, wie Unit-Tests uns helfen können:

Testprozess 1: Ein Benutzer hat gemeldet, dass beim Speichern eines Objekts mit einem Beschreibungsschlüssel ein Fehler auftritt. Ich habe einen Test geschrieben, um dieses Problem zu reproduzieren, das Problem gefunden und behoben, und diesen Test für zukünftige Überprüfungen beibehalten.

Testprozess 2: Bei der Entwicklung einer neuen Schnittstelle habe ich nach der Implementierung des Codes entsprechende Tests geschrieben, um sicherzustellen, dass der Code ordnungsgemäß funktioniert.

Testprozess 3: Nachdem ich den Code in `AVObject.m` geändert hatte, führte ich den Test `AVObjectTest.m` aus, um zu überprüfen, ob die Änderungen zu irgendwelchen Testfehlern führten.

Testprozess 4: Das Einreichen eines PR löst automatische Tests auf Jenkins aus.

Vorteile des Schreibens von Unit-Tests

- **Reduzierung manueller Überprüfungen:** Unit-Tests sparen Zeit, indem sie manuelle Überprüfungen überflüssig machen.

- **Fehlererkennung:** Probleme, die durch Codeänderungen verursacht werden, werden frühzeitig erkannt, was verhindert, dass Fehler andere Teile des Projekts beeinflussen.
- **Kollaborative Projekte:** In Projekten mit mehreren Entwicklern stellen Unit-Tests Konsistenz und Zuverlässigkeit sicher, selbst wenn das Projekt an andere weitergegeben wird.
- **Hochwertige Open-Source-Projekte:** Beliebte Open-Source-Projekte verfügen oft über umfangreiche Unit-Tests, was zu ihrer Zuverlässigkeit und Beliebtheit beiträgt.

Wie man effektive Unit-Tests schreibt

- **Modularer Code:** Trennen Sie die Datenebene und die UI-Ebene, um das Testen zu erleichtern.
- **Maximale Abdeckung:** Erzielen Sie mit minimalem Testcode die größtmögliche Abdeckung.
- **Asynchrone Verarbeitung:** Stellen Sie sicher, dass die Tests asynchrone Operationen handhaben können.
- **Framework-Auswahl:** Wählen Sie ein Testframework, das den Anforderungen entspricht.
- **Abdeckungsberichte:** Verwenden Sie Abdeckungsberichte, um zu verstehen, welche Teile des Codes getestet wurden.

Bewertung von Testframeworks

Wir haben mehrere Frameworks evaluiert: - **Expecta:** `expect(error).not.beNil()` - **Specta:** `describe("") it("")` - **Kiwi:** `describe("") it("")` - TDD- und BDD-Frameworks haben einige Einschränkungen, wie z.B. schlechte Integration mit Xcode, keine Test-Schaltfläche und die Tatsache, dass nicht alle Unit-Tests in der Seitenleiste aufgelistet sind.

Umgang mit asynchronen Tests

Asynchrone Tests sind entscheidend für Operationen, die nicht sofort abgeschlossen sind. Stellen Sie sicher, dass Ihr Framework asynchrone Tests effektiv unterstützt. Verwenden Sie beispielsweise in XCTest `expectations`, um auf den Abschluss asynchroner Operationen zu warten, bevor Sie Assertions durchführen.

Abdeckungsberichte

Xcode 7 hat eine integrierte Funktion für Abdeckungsberichte eingeführt. Die Schritte zur Aktivierung sind wie folgt: 1. Aktivieren Sie `Gather Coverage Data` in den Scheme-Einstellungen. 2. Führen Sie Tests für das App-Target durch, nicht für das Test-Target.

Diese Funktion ermöglicht es Entwicklern, genau zu sehen, welche Codezeilen getestet wurden, und hilft dabei, nicht getestete Codeabschnitte zu identifizieren. Weitere Details finden Sie im Blog von Big Nerd Ranch.

Automatisierte Remote-Tests mit Jenkins

Die Einrichtung von Jenkins für automatisierte Tests umfasst mehrere Schritte: 1. **Jenkins installieren:** Richten Sie Jenkins auf Ihrem lokalen Rechner oder einem Server im Rechenzentrum ein. 2. **GitHub-Integration:** Verwenden Sie das GitHub PR Build-Plugin, um Tests bei der Einreichung von Pull Requests auszulösen. - Konfigurieren Sie Webhooks, um Ereignisse an Jenkins zu senden. - Stellen Sie sicher, dass Jenkins Zugriff auf den neuesten Code des Pull Requests hat. 3. **Testskripte:** Richten Sie Testskripte in Jenkins ein, um den Testprozess zu automatisieren. - Stellen Sie sicher, dass Jenkins GitHub über die Testergebnisse benachrichtigen kann. - Konfigurieren Sie Slack- oder E-Mail-Benachrichtigungen bei Testfehlern.

Die Verwendung von Jenkins für Remote-Automatisierungstests bietet alle Vorteile der automatisierten Tests, die über lokale Tests hinausgehen, indem die Tests in einer sauberen und kontrollierten Umgebung ausgeführt werden.

Remote Packaging and Deployment

Obwohl nicht alle Projekte Remote-Packaging benötigen, kann es den Bereitstellungsprozess von SDKs und anderen wiederverwendbaren Komponenten vereinfachen. Die Schritte umfassen: - Konfiguration von Jenkins, um den Code zu lesen. - Lesen der Release-Version. - Entsperren des Keychains in der Befehlszeile, um auf Signaturzertifikate zuzugreifen.

Zusätzliche Tools und Tipps

- **Reveal:** Analysieren der Benutzeroberfläche (UI) beliebiger Apps auf einem gejailbreakten iPhone.
- **Flex:** Analysieren von Netzwerkanfragen, UI, lokalen Dateien, NSUserDefaults und Protokollen auf einem gejailbreakten iPhone.

- **Pod-Verwaltung:** Lokale Pod-Nutzung, erweiterte Podfile-Konfiguration und Veröffentlichung von Pods.
- **Framework-Erstellung:** Unterschiede zwischen dynamischen und statischen Bibliotheken sowie das Erstellen von Frameworks, die sowohl für Simulatoren als auch für echte Geräte geeignet sind.
- **Xcode-Tipps:** Nützliche Tastenkombinationen wie Shift + Command + J, um eine Datei im Navigator anzuzeigen, oder Shift + Command + O, um schnell eine Datei zu öffnen.

Fazit

Automatisierte Tests und geeignete Tools haben den Entwicklungsprozess erheblich verbessert. Durch die frühzeitige Einführung von Unit-Tests, die Nutzung von asynchroner Verarbeitung und die Verwendung von Abdeckungsberichten können wir zuverlässigere und besser wartbare Anwendungen erstellen. CI/CD-Tools wie Jenkins in Kombination mit Xcode-Entwicklungswerkzeugen und einer robusten Teststrategie stellen sicher, dass qualitativ hochwertige Software ausgeliefert wird.

Danksagung

Ein besonderer Dank geht an das LeanCloud-Team und alle, die zu unserem Testprozess beigetragen haben.