Mejorando el desarrollo de iOS con pruebas automatizadas y herramientas

Esta entrada del blog fue organizada con la ayuda de ChatGPT-4o.

La importancia de las pruebas unitarias

En LeanCloud, implementamos pruebas unitarias desde las primeras etapas del proyecto, lo que ha demostrado ser extremadamente valioso. Cada solicitud de extracción (PR) activa pruebas unitarias en Jenkins, y nuestro objetivo de cobertura es de aproximadamente el 80%. Hay dos escenarios principales para escribir pruebas: validar nuevas interfaces y reproducir y corregir errores. Cuantas más pruebas acumulamos, más robusto se vuelve nuestro código base. Las pruebas automatizadas nos permiten lanzar y refactorizar código con confianza, sin necesidad de verificación manual.

Flujo de Pruebas y Aplicación Práctica

Aquí hay algunos ejemplos prácticos de cómo las pruebas unitarias pueden ayudarnos:

Flujo de Prueba 1: Un usuario reportó que al guardar un objeto con una clave de descripción, se producía un error. Escribí una prueba para reproducir este problema, encontré y corregí el error, y luego conservé esta prueba para futuras verificaciones.

Proceso de Pruebas 2: Al desarrollar una nueva interfaz, después de implementar el código, escribí las pruebas correspondientes para asegurarme de que el código funcionara correctamente.

Proceso de Prueba 3: Después de modificar el código en `AVObject.m`, ejecuté `AVObjectTest.m` para verificar si los cambios causaron algún fallo en las pruebas.

Flujo de Pruebas 4: La creación de un PR desencadena pruebas automatizadas en Jenkins.

Beneficios de Escribir Pruebas Unitarias

- **Reducción de la Verificación Manual:** Las pruebas unitarias ahorran tiempo al eliminar la necesidad de verificación manual.

- **Detección de Errores:** Detectan problemas causados por cambios en el código de manera temprana, evitando que los errores afecten otras partes del proyecto.
- **Proyectos Colaborativos:** En proyectos con múltiples desarrolladores, las pruebas unitarias aseguran consistencia y confiabilidad, incluso si el proyecto es transferido a otra persona.
- **Proyectos de Código Abierto de Alta Calidad:** Los proyectos de código abierto populares suelen tener pruebas unitarias extensas, lo que contribuye a su confiabilidad y popularidad.

Cómo escribir pruebas unitarias efectivas

- **Código modular:** Separa la capa de datos y la capa de UI para facilitar las pruebas.
- **Maximizar la cobertura:** Utiliza el mínimo código de prueba para lograr la máxima cobertura.
- **Manejo de asincronía:** Asegúrate de que las pruebas puedan manejar operaciones asíncronas.
- **Elección del framework:** Selecciona un framework de pruebas que se ajuste a tus necesidades.
- **Informes de cobertura:** Utiliza informes de cobertura para entender qué partes del código han sido probadas.

Evaluación de Frameworks de Pruebas

Hemos evaluado varios frameworks: - **Expecta:** `expect(error).not.beNil()` - **Specta:** `describe("") it("")` - **Kiwi:** `describe("") it("")` - Los frameworks TDD y BDD tienen algunas limitaciones, como una integración deficiente con Xcode, la falta de un botón de prueba, y que la barra lateral no enumera todas las pruebas unitarias.

Manejo de Pruebas Asíncronas

Las pruebas asíncronas son cruciales para operaciones que no se completan de inmediato. Asegúrate de que tu framework soporte eficazmente las pruebas asíncronas. Por ejemplo, en XCTest, utiliza `expectations` para esperar a que se completen las operaciones asíncronas antes de realizar las aserciones.

Informe de Cobertura

Xcode 7 introdujo una función integrada de informes de cobertura. Los pasos para habilitarla son los siguientes: 1. Activa `Gather Coverage Data` en la configuración del esquema. 2. Realiza las pruebas en el objetivo de la aplicación (App Target), no en el objetivo de pruebas (Test Target).

Esta función permite a los desarrolladores ver exactamente qué líneas de código han sido probadas, lo que ayuda a identificar las partes del código que no han sido testeadas. Para más detalles, visita el blog de Big Nerd Ranch.

Automatización de Pruebas Remotas con Jenkins

Configurar Jenkins para realizar pruebas automatizadas implica varios pasos: 1. **Instalación de Jenkins:** Configura Jenkins en tu máquina local o en un servidor del centro de datos. 2. **Integración con GitHub:** Utiliza el plugin de GitHub PR Build para activar pruebas cuando se envíen solicitudes de extracción (pull requests). - Configura Webhooks para enviar eventos a Jenkins. - Asegúrate de que Jenkins pueda acceder al código más reciente de la solicitud de extracción. 3. **Scripts de Pruebas:** Configura scripts de pruebas en Jenkins para automatizar el proceso de pruebas. - Asegúrate de que Jenkins pueda notificar a GitHub los resultados de las pruebas. - Configura notificaciones en Slack o por correo electrónico para fallos en las pruebas.

El uso de Jenkins para pruebas automatizadas remotas ofrece todos los beneficios de las pruebas automatizadas, superando las pruebas locales al ejecutar las pruebas en un entorno limpio y controlado.

Empaquetado y despliegue remoto

Aunque no todos los proyectos requieren empaquetado remoto, este puede simplificar el proceso de despliegue de SDKs y otros componentes reutilizables. Los pasos incluyen: - Configurar Jenkins para leer el código. - Leer la versión de lanzamiento. - Desbloquear el llavero en la línea de comandos para acceder a los certificados de firma.

Herramientas y Consejos Adicionales

- **Reveal:** Analiza la interfaz de usuario de cualquier aplicación en un iPhone con jailbreak.

- **Flex:** Analiza solicitudes de red, interfaz de usuario, archivos locales, NSUserDefaults y registros en un iPhone con jailbreak.
- **Gestión de Pods:** Uso local de Pods, configuración avanzada de Podfile y publicación de Pods.
- **Creación de Frameworks:** Diferencias entre bibliotecas dinámicas y estáticas, y cómo empaquetar frameworks que funcionen tanto en simuladores como en dispositivos reales.
- **Consejos de Xcode:** Atajos útiles, como Shift + Command + J para mostrar el archivo en el navegador, y Shift + Command + O para abrir rápidamente un archivo.

Conclusión

Las pruebas automatizadas y las herramientas adecuadas mejoran significativamente el proceso de desarrollo. Al introducir pruebas unitarias desde el principio, aprovechar el procesamiento asíncrono y utilizar informes de cobertura, podemos construir aplicaciones más confiables y mantenibles. Herramientas de CI/CD como Jenkins, junto con las herramientas de desarrollo de Xcode y una estrategia de pruebas sólida, garantizan la entrega de software de alta calidad.

Agradecimientos

Un agradecimiento especial al equipo de LeanCloud y a todas las personas que contribuyeron a nuestro proceso de pruebas.