Améliorer le développement iOS grâce aux tests automatisés et aux outils

Cet article de blog a été rédigé avec l'aide de ChatGPT-4o.

L'importance des tests unitaires

Chez LeanCloud, nous avons mis en place des tests unitaires dès les premières étapes du projet, ce qui s'est avéré extrêmement précieux. Chaque demande de tirage (PR) déclenche des tests unitaires sur Jenkins, avec un objectif de couverture d'environ 80%. Il existe deux scénarios principaux pour écrire des tests : valider de nouvelles interfaces et reproduire et corriger des bugs. Plus nous accumulons de tests, plus notre base de code devient robuste. Les tests automatisés nous permettent de publier et de refactoriser du code en toute confiance, sans avoir besoin de validation manuelle.

Processus de test et applications pratiques

Voici quelques exemples concrets de la manière dont les tests unitaires peuvent nous aider :

Processus de test 1 : Un utilisateur a signalé une erreur lors de la sauvegarde d'un objet avec une clé de description. J'ai écrit un test pour reproduire ce problème, identifié et corrigé l'erreur, puis j'ai conservé ce test pour une validation future.

Processus de test 2 : Lors du développement d'une nouvelle interface, après avoir implémenté le code, j'ai écrit les tests correspondants pour m'assurer que le code fonctionne correctement.

Processus de test 3 : Après avoir modifié le code de `AVObject.m`, j'ai exécuté le test `AVObjectTest.m` pour vérifier si les modifications ont entraîné des échecs de test.

Processus de test 4 : La soumission d'une PR déclenche des tests automatisés sur Jenkins.

Avantages de l'écriture de tests unitaires

- **Réduction de la validation manuelle:** Les tests unitaires permettent de gagner du temps en éliminant la nécessité de vérifications manuelles.

- **Détection des erreurs:** Ils permettent de détecter précocement les problèmes causés par des modifications du code, empêchant ainsi que les erreurs n'affectent d'autres parties du projet.
- **Projets collaboratifs:** Dans les projets impliquant plusieurs développeurs, les tests unitaires assurent une cohérence et une fiabilité, même si le projet est repris par d'autres personnes.
- **Projets open source de haute qualité:** Les projets open source populaires disposent généralement d'une vaste couverture de tests unitaires, ce qui contribue à leur fiabilité et à leur popularité.

Comment rédiger des tests unitaires efficaces

- **Code modulaire:** Séparez la couche de données de la couche UI pour faciliter les tests.
- **Maximiser la couverture:** Utilisez un minimum de code de test pour atteindre une couverture maximale.
- **Gestion asynchrone:** Assurez-vous que les tests peuvent gérer les opérations asynchrones.
- **Choix du framework:** Sélectionnez un framework de test adapté à vos besoins.
- **Rapport de couverture:** Utilisez des rapports de couverture pour savoir quelles parties du code ont été testées.

Évaluation des frameworks de test

Nous avons évalué plusieurs frameworks : - **Expecta:** `expect(error).not.beNil()` - **Specta:** `describe("") it("")` - **Kiwi:** `describe("") it("")` - Les frameworks TDD et BDD présentent certaines limitations, comme une intégration médiocre avec Xcode, l'absence de bouton de test, et le fait que la barre latérale ne liste pas tous les tests unitaires.

Gestion des tests asynchrones

Les tests asynchrones sont essentiels pour les opérations qui ne se terminent pas immédiatement. Assurez-vous que votre framework prend en charge efficacement les tests asynchrones. Par exemple, dans XCTest, utilisez des `expectations` pour attendre la fin des opérations asynchrones avant de procéder aux assertions.

Rapport de couverture

Xcode 7 a introduit une fonctionnalité intégrée de rapport de couverture. Voici les étapes pour l'activer : 1. Activez `Gather Coverage Data` dans les paramètres du schéma. 2. Effectuez les tests sur la cible de l'application, et non sur la cible de test.

Cette fonctionnalité permet aux développeurs de voir exactement quelles lignes de code ont été testées, ce qui aide à identifier les parties du code qui n'ont pas été testées. Pour plus de détails, visitez le blog de Big Nerd Ranch.

Automatisation des tests à distance avec Jenkins

La configuration de Jenkins pour les tests automatisés implique plusieurs étapes : 1. **Installation de Jenkins :** Configurez Jenkins sur votre machine locale ou sur un serveur dans un centre de données. 2. **Intégration avec GitHub :** Utilisez le plugin GitHub PR Build pour déclencher les tests lors de la soumission d'une demande de pull. - Configurez les Webhooks pour envoyer les événements à Jenkins. - Assurez-vous que Jenkins peut accéder au code le plus récent de la demande de pull. 3. **Scripts de test :** Configurez des scripts de test dans Jenkins pour automatiser le processus de test. - Assurez-vous que Jenkins peut notifier GitHub des résultats des tests. - Configurez des notifications Slack ou par e-mail en cas d'échec des tests.

L'utilisation de Jenkins pour les tests automatisés à distance offre tous les avantages des tests automatisés, surpassant les tests locaux, en exécutant les tests dans un environnement propre et contrôlé.

Empaquetage et déploiement à distance

Bien que tous les projets n'aient pas besoin d'un empaquetage à distance, cela peut simplifier le processus de déploiement des SDK et d'autres composants réutilisables. Les étapes incluent : - Configurer Jenkins pour lire le code. - Lire la version de publication. - Déverrouiller le trousseau en ligne de commande pour accéder aux certificats de signature.

Outils et Astuces Supplémentaires

- **Reveal:** Analysez l'interface utilisateur de n'importe quelle application sur un iPhone jailbreaké.

- **Flex:** Analysez les requêtes réseau, l'interface utilisateur, les fichiers locaux, NSUserDefaults et les logs sur un iPhone jailbreaké.
- **Gestion des Pods:** Utilisation locale des Pods, configuration avancée du Podfile et publication des Pods.
- **Création de Frameworks:** Différences entre les bibliothèques dynamiques et statiques, et comment empaqueter un framework compatible à la fois avec les simulateurs et les appareils physiques.
- **Astuces Xcode:** Raccourcis utiles, comme Shift + Command + J pour afficher le fichier dans le navigateur, et Shift + Command + O pour ouvrir rapidement un fichier.

Conclusion

Les tests automatisés et les outils appropriés améliorent considérablement le processus de développement. En introduisant tôt des tests unitaires, en exploitant le traitement asynchrone et en utilisant des rapports de couverture, nous pouvons construire des applications plus fiables et maintenables. Des outils CI/CD comme Jenkins, combinés aux outils de développement Xcode et à une stratégie de test robuste, assurent une livraison de logiciels de haute qualité.

Remerciements

Un grand merci à l'équipe de LeanCloud ainsi qu'à toutes les personnes qui ont contribué à notre processus de test.