Cours n°8 — Debug, tests et documentation

Debuggage des applications

Dans le cadre du développement de manière générale, avoir des outils qui permettent d'inspecter l'état d'une application peut s'avérer extrêmement utile. Le debuggage « primitif » qui consiste à utiliser des instructions `print` pour afficher certaines valeurs au cours de l'exécution peut toujours s'avérer pratique mais peut vite être limité pour de grosses applications.

Par défaut, les programmes Swift peuvent être débuggués avec l'outil LLDB. Celui-ci n'est pas spécifique à Swift, il est donc possible de trouver beaucoup de documentation à son sujet sur internet.

En ce qui concerne les applications iOS, LLDB est complètement intégré à Xcode en mode debuggage et ce dernier va afficher tout un tas de métriques et d'éléments liés à l'exécution en cours d'une application.

Pour débugguer une application il suffit de créer un point d'arrêt (ou *breakpoint).* (c.f. diapositive 3) En lançant l'application, une fois que le programme rencontre la ligne qui est indiquée comme point d'arrêt, l'exécution est mise en pause et l'interface de Xcode va s'adapter en conséquences. (c.f. diapositive 4)

Les sections pertinentes dans Xcode se trouvent à gauche et en bas. La partie gauche s'appelle le *debug navigator* et la partie basse la *debug console*.

Dans le *debug navigator*, on trouve en premier lieu les jauges. De cliquer sur ces jauges va afficher dans le panneau central plus de détails sur la métrique concernée (utilisation du CPU, entrée/sortie réseaux, etc.) (c.f. diapositives 6 à 9) En second lieu, on trouve la *backtrace* affichant chaque pile d'appels pour les différents threads exécutés dans l'application.

En ce qui concerne la *debug console*, elle possède en premier lieu une barre contenant diverses icônes utiles pour contrôler l'exécution en phase de debuggage. Ces icônes sont expliquées dans les diapositives 10 à 19.

En second lieu, à gauche de la partie basse, on trouve la liste des différentes variables à porté du point d'arrêt, et finalement dans la partie droite de la console, la console LLDB, permettant d'inspecter l'état du programme. Les commandes `expression` et `po` s'avèrent extrêmement utiles mais il est également possible de contrôler de nombreuses autres choses comme les points d'arrêts, inspecter les threads, etc.

Un liens vers un article relativement complet montre quelques possibilités pratiques de LLDB dans le cadre des applications iOS au niveau de la diapositive 20.

Tests

Pour faciliter l'ajout de nouvelles fonctionnalités où s'assurer qu'un bug corrigé ne refasse pas surface, il est essentiel d'écrire des tests. Apple fourni XCTest comme framework permettant d'en écrire.

Une application va être testée à deux niveaux : au niveau unitaire (test des classes Swift en elles-mêmes) et au niveau de l'interface (tests sur simulateur des interactions dans l'application).

Chaque test peut être réalisé en tant que test fonctionnel, c'est à dire testant le déroulement d'une méthode ou d'un pan de l'application, ou en tant que test de performance.

Lors de la création d'une application, il faut s'assurer de cocher la ou les cases pour que Xcode génère des fichiers de tests de base. (c.f. diapositives 24 et 25) Deux nouvelles cibles (*targets*) vont être ajoutées au projet, une pour chaque type de test. (c.f. diapositive 26)

La création de tests, les conventions de nommage ainsi que la gestion du cycle de vie des tests sont expliquées dans les diapositives 27 à 32.

En ce qui concerne les tests de performance, ces derniers sont quelques peux différents des tests fonctionnels : ils ne contiennent généralement pas d'assertions. Un appel à la méthode `measure` à laquelle on va passer une *closure* contenant le code à mesurer doit être réalisé.

Xcode va automatiquement détecter l'appel à cette méthode et va réaliser un échantillon de mesures du temps pour le bloc de code fourni. L'interface va ensuite permettre de définir cet échantillon comme référence (*baseline*). (c.f. diapositive 34) Ainsi, les futures exécutions vont s'assurer que les performances du bloc se situent dans cet échantillon.

En ce qui concerne les tests d'interface, il est très facile de se heurter à des problèmes lors de leur utilisation. Les diapositives 35 à 37 proposent une brève introduction pour écrire de tels tests.

À noter que la fonction d'enregistrement de Xcode est loin d'être parfaite.

Documentation

Petit plus, Xcode intègre parfaitement la documentation des classes fournies dans le SDK mais également celles du projet courant.

Les diapositives 38 à 40 montrent comment consulter facilement la documentation dans Xcode. Il est bon de noter que l'application de documentation, au delà de contenir la documentation des classes du SDK, contient également tous les documents que l'on peut trouver sur le site de la documentation Apple, notamment de nombreux tutoriels expliquant l'utilisation de certaines fonctionnalités.

Finalement, en diapositives 42 à 44, quelques exemples montrent comment écrire de la documentation au sein d'un projet. Il est possible d'utiliser le format Markdown pour mettre en forme les commentaires de documentation ; de très nombreuses ressources en ligne sont disponibles concernant ce format.