Avant-propos

Comme des millions de gens, je me suis instantanément pris de passion pour mon iPhone en le découvrant. Au début, il n'existait pas d'autre moyen que les applications web pour se procurer des applications personnalisées sur ce périphérique. Et cela me convenait parfaitement, puisque j'étais moi-même développeur web. Quelques mois plus tard, l'annonce de l'ouverture de l'Apple Store fut un véritable choc pour moi.

Je me suis précipité pour acheter tous les livres disponibles sur le langage Objective-C. Quelques-unes de mes applications web avaient déjà rencontré un certain succès et je pensais qu'il me suffirait de les réécrire sous forme d'applications natives, de les placer dans l'Apple Store et d'aller me la couler douce sur une île pour milliardaires...

Ma désillusion ne s'est pas fait attendre. Je me suis rendu compte que l'apprentissage d'Objective-C était assez complexe, mais ce qui m'a le plus chagriné, c'est que ce langage avait assez peu d'utilité en dehors de la programmation pour Mac. Certes, Xcode et Interface Builder étaient bien sympathiques, mais il ne s'agissait pas de mon environnement de développement habituel et j'avais du mal à m'y faire. J'étais exaspéré par la peine que je devais me donner rien que pour configurer mon application et l'iPhone pour les tests. Le processus d'inclusion de l'application dans l'Apple Store était encore plus laborieux. Après une ou deux semaines à batailler, je me suis demandé pourquoi je me donnais toute cette peine. Après tout, mes applications web n'étaient-elles pas déjà disponibles partout dans le monde ? Pourquoi se soucier à ce point de figurer au catalogue de l'Apple Store ?

Par-dessus le marché, il se trouve qu'Apple a la possibilité de rejeter certaines applications (et le fait parfois effectivement). C'est leur prérogative, certes, et sans doute ont-ils de bonnes raisons de procéder ainsi, mais vu de l'extérieur, ce fonctionnement semble pour le moins arbitraire et capricieux. Mettez-vous à la place de cette personne (et sachez qu'elle a bel et bien existé) : vous passez une centaine d'heures à apprendre le langage Objective-C. Vous en passez cent autres à écrire une application native pour l'iPhone. Votre application est enfin prête et vous parvenez à vous dépatouiller du processus de soumission à l'Apple Store. Que se passe-t-il ensuite ?

Vous attendez. Et attendez encore. Et encore un peu. On parle là de semaines, et parfois même de mois. Ah ! Enfin, une réponse ! Votre application… est rejetée. Bon, et maintenant ? Eh bien, vos efforts n'ont abouti à rien. Pfuit. Comme un gros ballon de baudruche.

Mais attendez, c'est que vous n'êtes pas au bout de vos peines. Supposons que votre application ait finalement été retenue. Des centaines, voire des milliers de personnes la téléchargent.

Vous n'avez pas encore touché le moindre centime, mais vous êtes déjà au septième ciel ! Soudain, les rapports de bogues commencent à vous parvenir. Vous localisez et réparez les erreurs en quelques minutes, renvoyez votre application sur iTunes et attendez qu'Apple approuve la révision.

Et vous attendez. Et attendez encore. Les clients agacés vous laissent d'horribles commentaires sur l'Apple Store. Vos ventes vacillent. Et vous attendez toujours. Vous envisagez de proposer un remboursement aux clients mécontents, mais l'Apple Store ne propose aucun moyen de le faire. Vous êtes donc contraint d'attendre et d'observer votre cote qui s'effondre alors que le bogue est pourtant corrigé depuis des jours, voire des semaines.

Cette histoire est tirée de l'expérience réelle d'un développeur. C'est peut-être un cas extrême et je n'ai pas de données quantitatives pour étayer ma thèse, mais le problème reste entier : nous, les développeurs, n'avons aucun accès aux données d'Apple et aux véritables arcanes du processus d'approbation de l'Apple Store. Tant que cela n'a pas changé, il est toujours risqué de créer une application native avec Objective-C.

Mais il existe heureusement une autre solution. Vous pouvez créer une application web à l'aide de technologies web standardisées et open source, la soumettre sous forme d'application web, la déboguer et la tester auprès de véritables utilisateurs. Une fois que vous êtes fin prêt, vous pouvez utiliser PhoneGap pour convertir votre application web en une application native iPhone et l'envoyer à l'Apple Store. Au bout du compte, si elle est rejetée, ce n'est pas un drame, parce que vous pouvez continuer de proposer l'application web. Si elle est approuvée, génial ! Vous pouvez alors commencer à ajouter des fonctionnalités qui améliorent votre application web en tirant parti des fonctionnalités matérielles uniques disponibles sur l'appareil. C'est un peu le beurre et l'argent du beurre…

À qui s'adresse ce livre ?

Je pars du principe que vous possédez déjà suffisamment d'expérience pour lire et écrire du code HTML, CSS et JavaScript (jQuery en particulier). J'inclurai du code SQL simple aux chapitres 5 et 6. Quelques rudiments en syntaxe SQL seraient ainsi appréciables, mais ils ne sont cependant pas indispensables.

Ce qu'il vous faut pour ce livre

Dans ce livre, nous éviterons le SDK iPhone autant que faire se peut. Pour suivre la grande majorité des exemples de l'ouvrage, vous aurez simplement besoin d'un éditeur de texte et de la version la plus récente de Safari (ou mieux encore, WebKit, qui est une version plus avant-gardiste de Safari, disponible pour Mac et Windows à l'adresse *http://webkit.org*). Il vous faudra de fait utiliser un Mac pour toute la partie qui concerne PhoneGap au chapitre 7, lorsque j'explique comment convertir votre application web en une application native qui peut être envoyée à l'Apple Store.

Conventions utilisées dans ce livre

Les conventions typographiques suivantes sont utilisées dans ce livre :

`Police à chasse fixe`

Cette police à chasse fixe est utilisée pour les listings de code, les noms de fichiers, les éléments de programme comme les noms de variable ou de fonction, les bases de données, les types de données, les variables d'environnement, les instructions et les mots-clés.

`Police à chasse fixe en gras`

Cette police signale les commandes ou du texte que l'utilisateur doit saisir littéralement ou attire l'attention sur des portions de code dans les listings.

`Police à chasse fixe en italique`

Cette police est utilisée pour signaler du texte devant être remplacé par des valeurs fournies par l'utilisateur ou déterminées par le contexte.

> **Info**
> Cette présentation est employée pour les notes, les astuces, les suggestions, les avertissements, les commentaires d'ordre général ou encore les mises en garde.

Utiliser les exemples de code

Ce livre a pour but de vous aider à faire votre travail. De manière générale, vous pouvez utiliser le code qu'il contient pour vos programmes et votre documentation. Vous n'avez pas besoin de nous contacter pour obtenir d'autorisation, à moins que vous ne reproduisiez des portions significatives de ce code. Ainsi, il n'est pas nécessaire de demander une autorisation pour écrire un programme qui utiliserait plusieurs parties du code de ce livre. En revanche, la vente ou la distribution de CD-Rom d'exemples de cet ouvrage requiert une autorisation. Le fait de répondre à une question en citant ce livre et de citer du code d'exemple ne requiert pas d'autorisation, mais l'intégration d'une portion significative du code de ce livre dans la documentation de votre produit demande une autorisation.

Nous vous serons reconnaissants si vous citez ce livre comme source, mais nous ne l'exigeons pas. En général, ce type de référence inclut le titre, l'auteur, l'éditeur et le numéro ISBN de l'ouvrage. Par exemple : Jonathan Stark, *Applications iPhone en HTML, CSS et JavaScript – Conversion en natifs avec PhoneGap*, Eyrolles, 2010, n° ISBN : 978-2-212-12745-4.

Notez par ailleurs que vous pouvez télécharger les exemples de code de l'ouvrage sur le site des éditions Eyrolles, *www.editions-eyrolles.com*.