Tutoriel - iOS

Introduction

Swift est un langage de programmation développé par la société Apple, destiné à la programmation d'applications sur les systèmes d'exploitation iOS, macOS, ... Ce langage est donc propre à Apple, il est donc rare de s'en être servi si l'on n'a pas développé d'applications destinées à un dispositif Apple.
La particularité que j'ai tout de suite remarquée par rapport aux autres langages mobiles (que ce soit Frameworks multi-plateforme ou Android) est que l'interface de développement est très visuelle. L'on peut effectuer beaucoup de traitement directement depuis les "storyboard" qui sont des fenêtres contenant nos différentes vues, et les relations entre elles.

CocoaPods

Suite à différents problèmes lors de l'import de bibliothèques telle que GoogleMaps, j'ai choisi d'utiliser un logiciel de gestion de dépendances afin de m'aider dans cette tâche. Ce logiciel permet d'importer "automatiquement" les librairies demandées, et requiert un fichier "Podfile" de la forme:

```
platform :ios, '12.0'

source 'https://github.com/CocoaPods/Specs.git'
target 'PathPartout' do
  pod 'GoogleMaps'
  pod 'GooglePlaces'
end
```

Pour lancer CocoaPods, il faut utiliser la commande `pod install` puis `open App.xcworkspace`. La chose à ne pas oublier est que par la suite, il faut toujours ouvrir le fichier `.xcworkspace` et non le `.xcodeproj` car les dépendances ne sont pas gérées sinon. Le problème est que de base, XCode (éditeur de code développé par Apple spécialement pour le swift) ouvre de base les `.xcodeproj`, il faut donc bien penser à l'ouvrir manuellement à chaque fois...

Storyboard

Vues

Les différentes vues et contrôleurs doivent être créés directement depuis le fichier “storyboard” :

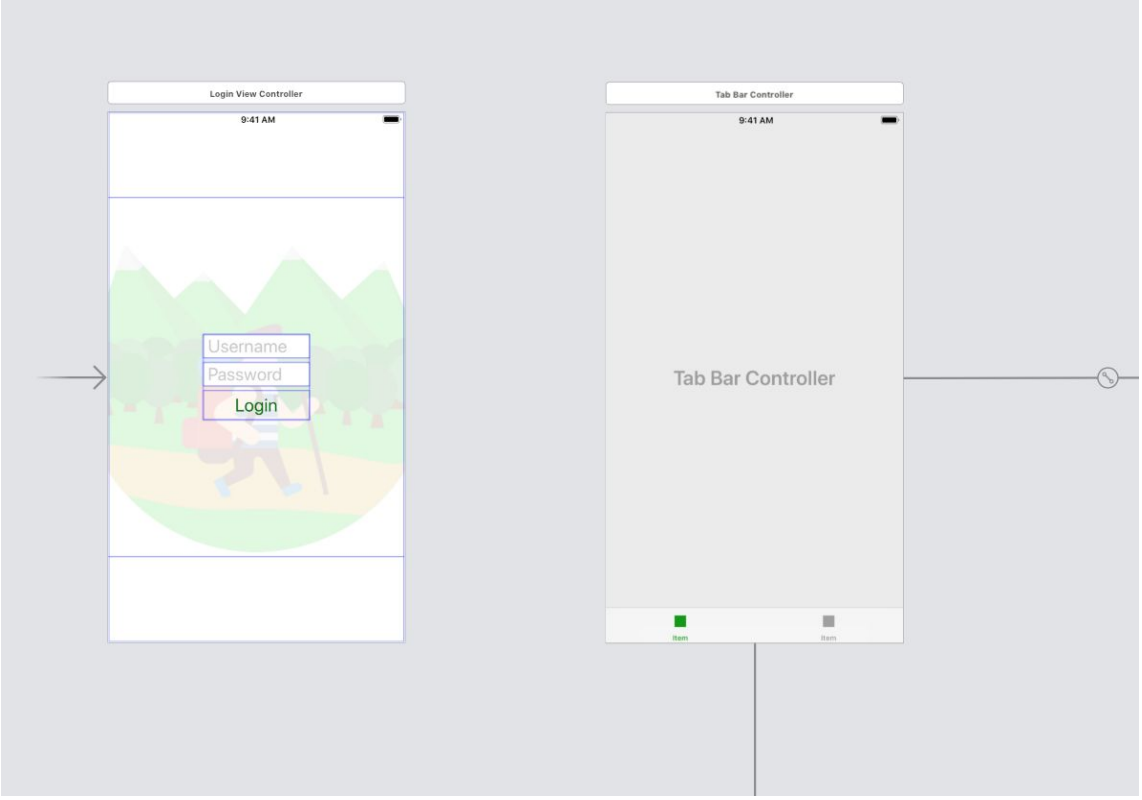

Figure 1 : storyboard

L’on peut définir depuis cette page le point d’entrée de l’application, nos vues, et les relations entre elles. Par exemple, nous pouvons choisir quelle vue s’ouvrira lors d’un click sur un bouton directement depuis l’interface graphique, en maintenant “control” en glissant vers la vue cible :

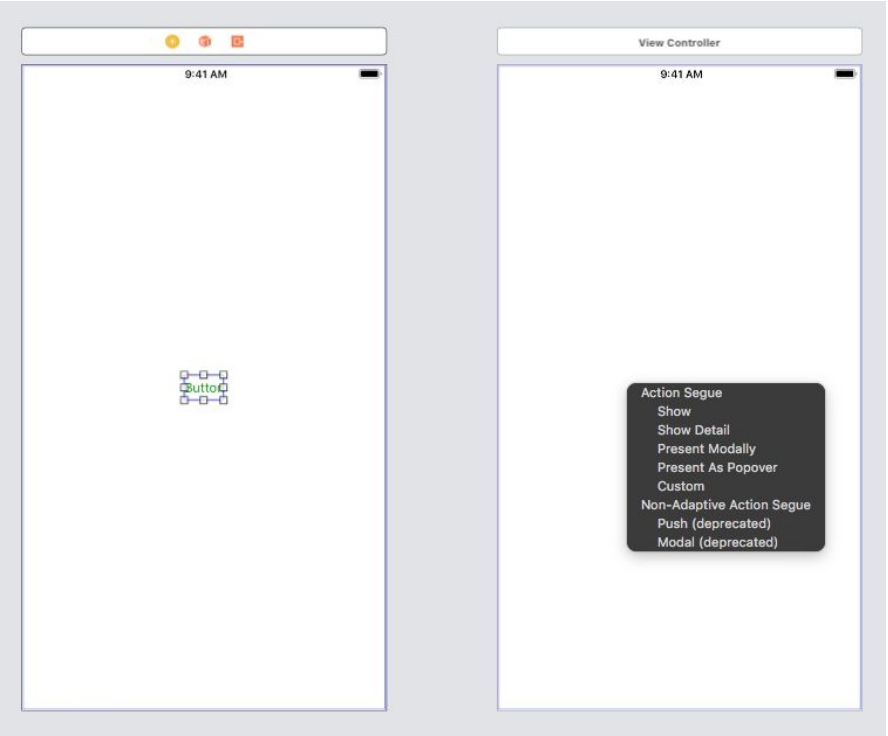

Figure 2 : Relation entre vues depuis storyboard

L'on peut ensuite choisir quel type d'actions, et même la transition directement depuis l'interface graphique. C'est un processus un peu surprenant au départ, car non initié mais finalement tout est bien pensé et on devient vite familier avec la méthode. Le processus est le même lorsqu'on souhaite faire un lien entre la vue et le contrôleur : l'on peut par exemple faire glisser un bouton depuis la vue vers le controller pour créer la variable qui lui sera assignée, ou même pour le lier à une action (onclick, ...) :

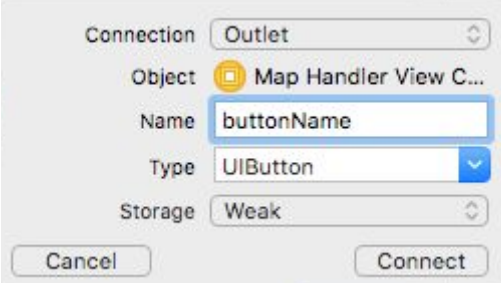

Figure 3 : Connexions vues - controller depuis storyboard

Tab Bar Controller

J'ai trouvé l'approche pour créer un Tab Bar Controller différente de ce que j'ai eu l'occasion de faire habituellement. Encore une fois, l'on peut tout faire en interface graphique et lorsqu'on créé un Tab Bar Controller, nous obtenons le résultat suivant :

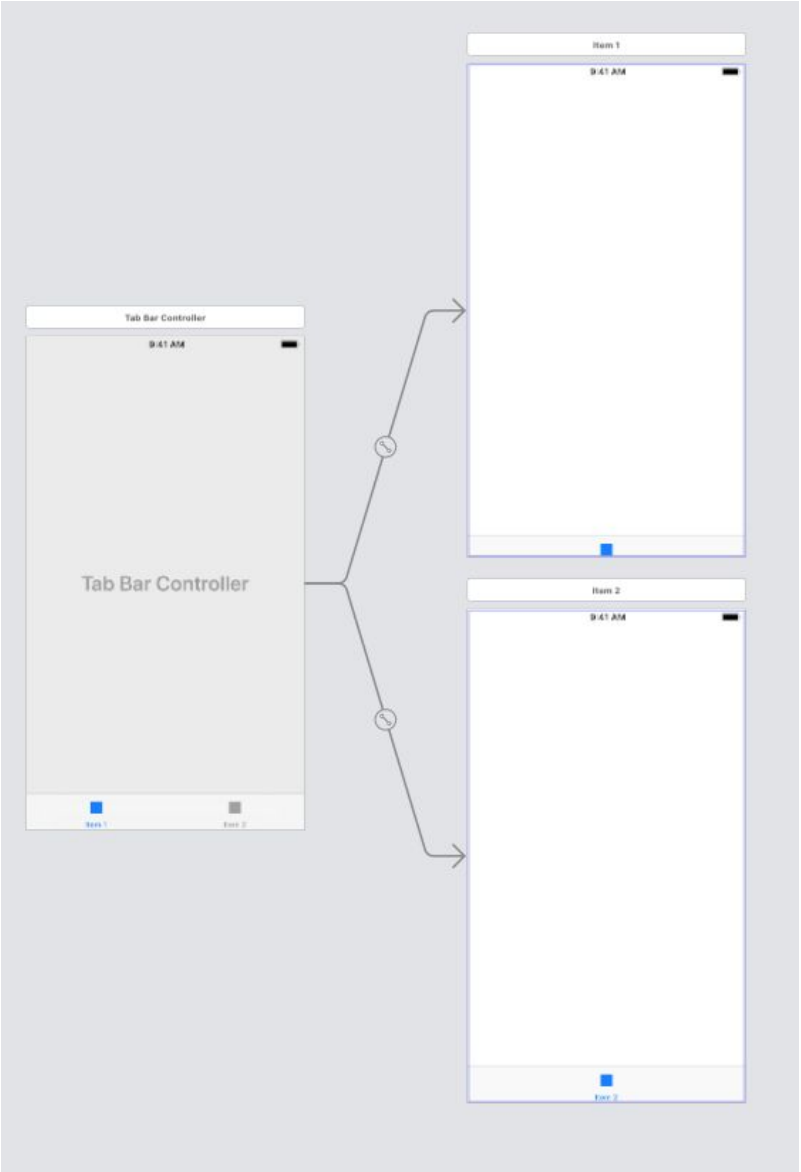

Figure 4 : Tab Bar Controller

En fait, lors de la création deux `Items` sont automatiquement créés ce qui correspond aux différentes pages affichées dans le controller. J'ai trouvé cette manière différente, et ai dans un premier temps essayé de créer une barre de navigation en bas de page, que j'aurais ensuite placée sur les différentes vues où elle doit apparaître. Ici en fait le processus est simplifié par rapport à d'autres environnements, mais différent ce qui cause parfois des pertes de temps.

Communication vues non reliées

Lors du développement de l'application, j'ai du faire communiquer des vues non reliées. C'était dans le cadre particulier de l'intégration d'une page de connexion : la page de connexion doit pouvoir être accessible à n'importe quel moment, si l'utilisateur souhaite se déconnecter. Il serait mauvais de placer sur chaque vue un bouton "se déconnecter" reliée à la vue de la connexion. C'est pourquoi, il est commun de créer une vue de connexion relié à aucune autre vue et d'utiliser un fichier "Switcher". C'est en fait un fichier global, qui pourra être accessible depuis n'importe quel vue. La fonction ci dessous montre un exemple d'utilisation du Switcher (dans cet exemple le mot de passe n'est vérifié, l'on supposera qu'il est bon) :

```
@IBAction func buttonActionLogin(_ sender: Any) {
    UserDefaults.standard.set(true, forKey: "status")
    Switcher.updateRootVC()
}
```

Figure 5 : Click on login button

Et le "Switcher" :

```
static func updateRootVC(){

    let status = UserDefaults.standard.bool(forKey: "status")
    var rootVC : UIViewController?

    if(status == true){
        rootVC = UIStoryboard(name: "Main", bundle:
            nil).instantiateViewController(withIdentifier: "tabbarvc") as!
            UITabBarController
    }else{
        rootVC = UIStoryboard(name: "Main", bundle:
            nil).instantiateViewController(withIdentifier: "loginvc") as!
            LoginViewController
    }

    let appDelegate = UIApplication.shared.delegate as! AppDelegate
    appDelegate.window?.rootViewController = rootVC

}
```

Figure 6 : Switcher

Ainsi, uniquement dans le cas où le mot de passe est bon le contenu de l'application sera dévoilé, sinon l'on restera sur la page de connexion. Pour déconnecter l'utilisateur, c'est aussi simple : l'on met le `status` à false et l'on rapelle `Switcher.updateRootVC()`. Bien sur, l'on ne prend pas en compte le backend ici, il y aurait sinon plus de traitements à prendre en compte. L'on remarque aussi que l'on peut appeler les ViewController avec leurs identifiers. Ce processus permet de réutiliser facilement les composants.

Avantages et inconvénients

Avantages

- Très simple de vérifier le design de l'application sur tous les dispositifs (pour vérifier que les différentes tailles d'écran sont bien optimisées) car peu de modèles
- L'accès au GPS, et aux autres fonctionnalitées est très facile
- XCode est vraiment optimisé pour la language : usage très plaisant
- Ensemble de tutoriels écrits par Apple bien détaillés
- Gestion des couleurs et images de manière intuitive (pas de chemins d'accès à utiliser, il suffit de les glisser dans un dossier)

Inconvénients

- Apprentissage plutôt long (pas le langage de programmation, mais tout ce qu'il y a autour)
- Utilisation de Swift qui est restreint à Apple
- Lors de recherches d'aide sur internet, beaucoup de méthodes sont inutilisables : beaucoup de modifications du langage lors des nouvelles versions
- Pas de documentation sur certaines erreurs faisant planter l'application