Introduction à Storyboard et Autolayout

Ma première app

Antoine Moevus

Table des matières

Références 1
Vidéos explicatives 2
Planification des requis 3
  Les avantages de la planification 3
Structure d'un projet iOS dans XCode 4
  AppDelegate.swift 4
  SceneDelegate.swift 4
  ViewController.swift 4
  Main.storyboard 4
  Assets.xcassets 5
  Base.lproj 5
  LaunchScreen.storyboard 5
  Info.plist 5
Ma première app: exemple le lecteur vidéo 6
  Création du projet 6
  Créer l'interface 6
  Ajouter une vidéo 6
  Importer les librairies 7
  Ajouter le `video player` 8

Références

- [1] https://developer.apple.com/documentation/avfoundation/avplayer
- [2] https://blog.shipbook.io/swiftui-vs-storyboard

Vidéos explicatives

- Commencer un projet d’application iOS
- Explication du lecteur vidéo

Planification des requis

Avant de commencer à programmer, il faut planifier votre projet. De nos jours, on va toujours essayer de planifier des livrables, et donc on va souvent décomposer une application en fonctionnalités et identifier les besoins de chaque fonctionnalité. Par exemple, pour une app qui va lire des vidéos:

- Est-ce que les vidéos sont locales ou en ligne?
- Est-ce que l'on souhaite que le lecteur vidéo soit intégré ou bien que l'app redirige vers un autre lecteur vidéo?
- Quels types de fichiers faut-il lire ?
- Est-ce que l'on va utiliser la librairie fournie par Apple ou bien les besoins du projet nécessitent d'en créer une nouvelle?

Et ainsi de suite.

Les avantages de la planification

1. Performance et réussite
    - La planification d'un projet implique d'avoir une description détaillée des objectifs, tâches, ressources, et échéances. Ensuite, on peut les organiser et assigner des rôles à l'équipe. Avec une bonne planification, les efforts de l'équipe seront améliorés et la réussite du projet sera assurée.
2. Objectifs clairs
    - Avoir des objectifs clairs de ce qui doit être livré va permettre de s'assurer que ce soit fait correctement. Ne pas savoir ce sur quoi l'on travaille va compliquer les relations entre les membres de l'équipe. Il sera alors impossible de savoir ce qui est complété ou non. Sans des objectifs clairs et des stratégies pour les atteindre, il est alors impossible pour l'équipe de concentrer ses efforts.
3. Allocation de ressources
    - Il faut savoir la quantité de matériel et de personnes nécessaires pour réussir le projet. De plus, c'est important d'estimer quand ces ressources doivent être accessibles. Ce sont donc des choses à planifier.
4. Communication
    - Avec une bonne planification, on améliore la communication entre les membres de l'équipe. Il est aussi possible de faire remonter les problèmes plus rapidement et ainsi de s'adapter aux changements.
5. Formation appliquée au projet
    - Certaines compétences vont devoir être acquises au cours du projet et pour les prévoir, il faut avoir planifié le projet.

Structure d'un projet iOS dans XCode

Lorsque vous créez un nouveau projet dans XCode, plusieurs fichiers et dossiers sont générés automatiquement pour vous. Ces fichiers constituent la structure de base de votre projet. Comprendre cette structure et ce que chaque fichier fait peut vous aider à naviguer plus efficacement dans votre projet.

L'arborescence d'un projet iOS typique dans XCode est la suivante:

```
MonProjet
+-- AppDelegate.swift
+-- SceneDelegate.swift
+-- ViewController.swift
+-- Main.storyboard
+-- Assets.xcassets
|   +-- AppIcon.appiconset
|   +-- LaunchImage.launchimage
+-- Base.lproj
|   +-- LaunchScreen.storyboard
+-- Info.plist
```

AppDelegate.swift

Ce fichier gère les événements au niveau de l'application. C'est le point d'entrée de votre application, et c'est là où des méthodes sont appelées en réponse à des événements majeurs de l'application, comme lorsqu'elle se lance, se termine, se met en arrière-plan ou revient au premier plan. Généralement, on ne le modifie pas.

SceneDelegate.swift

Avec l'introduction de iOS 13 et des scènes dans UIKit, la responsabilité de la gestion de la fenêtre de l'application est passée de `AppDelegate` à `SceneDelegate`. Cela donne plus de flexibilité pour gérer plusieurs fenêtres, surtout sur iPadOS.

ViewController.swift

Ce fichier gère une vue dans votre application. C'est là que vous ajoutez le code pour contrôler l'interface utilisateur de cette vue, y compris la gestion des interactions de l'utilisateur.

Main.storyboard

C'est là où vous construisez l'interface utilisateur de votre application en utilisant un éditeur de glisser-déposer. Vous pouvez ajouter et configurer des éléments d'interface utilisateur comme des boutons, des étiquettes, des champs de texte, etc.

Assets.xcassets

Ce dossier est utilisé pour stocker toutes les ressources graphiques de votre application, comme les images et les icônes.

Base.lproj

Ce dossier contient les fichiers de ressources localisées. Les fichiers contenus dans ce dossier servent de point de départ pour la localisation de l'application dans différentes langues. Par exemple, le fichier `Storyboard` principal et le fichier `Localizable.strings` se trouvent souvent dans ce dossier. Ces fichiers peuvent ensuite être copiés et traduits dans d'autres dossiers `.lproj` (par exemple, `fr.lproj` pour le français, `es.lproj` pour l'espagnol, etc.) pour adapter l'interface utilisateur et les textes de l'application aux différentes langues prises en charge.

LaunchScreen.storyboard

Ce fichier définit l'écran qui s'affiche lorsque votre application se lance. Il est généralement utilisé pour afficher une image ou un logo en attendant que l'application soit prête à être utilisée.

C'est une vue d'ensemble de base de la structure d'un projet XCode. Il y a beaucoup plus à explorer, notamment comment ajouter d'autres contrôleurs de vue, gérer les dépendances, ajouter des tests unitaires, etc.

Info.plist

Ce fichier est utilisé pour stocker les métadonnées de votre application. Il inclut des informations comme la version de l'application, le nom d'affichage, les autorisations nécessaires, et plus encore.

Ma première app: exemple le lecteur vidéo

On veut faire un lecteur vidéo qui :

1. Joue automatiquement une vidéo dès que la vue apparaît
2. Lis une vidéo locale nommée "test.mp4"
3. Utilise le lecteur vidéo standard d'iOS

Création du projet

Créer un nouveau projet dans XCode. Un menu de choix de template apparait alors:

- Sélectionner l'onglet "iOS"
- Choisir "App" dans la catégorie "Application"
- Cliquer sur suivant

Un menu pour choisir les options de votre projet apparait:

- Nommer le projet: `video_player`
- Entrez un nom d'organisation (la vôtre)
- Regarder la valeur du champ "Bundle Identifier"[1]
- Choisir "Storyboard" pour l'interface
- Choisir "Swift" pour le langage
- Décocher "Use Core Data"
- Cocher "Include Tests"

Ensuite il ne reste qu'à choisir le dossier où sauver votre projet. Vous pouvez cocher l'option Git.

Créer l'interface

- Si néceesaire, dans `video_player/`, ajouter un fichier de type storyboard, et le nommer `main`.

Ajouter une vidéo

- Ajouter une vidéo à votre projet XCode (par exemple celle-ci: lien).
- Renommer la vidéo "test" et conserver l'extension.
- S'assurer que le nom de la vidéo apparait bien dans "Copy Bundle Resources".
- Ensuite, afin de charger la vidéo dans l'app, copier le code suivant dans le fichier `ViewController.swift` :

```
private func findVideo() {
    guard let path = Bundle.main.path(forResource: "video", ofType:"mp4") else {
```

```
        debugPrint("Video file not found")
        return
    }
}
```

Remplacer la valeur de l'argument `forResource` par "test".

Le fichier `ViewController` final devrait ressembler au code suivant:

```
import UIKit

class ViewController: UIViewController {

    override func viewDidLoad() {
        super.viewDidLoad()
        // Do any additional setup after loading the view.
    }

    private func findVideo() {
        guard let path = Bundle.main.path(forResource: "test", ofType:"mp4") else {
            debugPrint("Video file not found")
            return
        }
    }
}
```

Importer les librairies

AVPLayer

La librairie fournie `AVPLayer` supporte la lecture des formats vidéos suivants:

- H.263
- H.264 AVC
- MPEG-4 SP
- VP8

Dans la documentation officielle [1], on peut voir plusieurs méthodes afin d'observer et contrôler la lecture de la vidéo.

Importation

Ajouter dans `ViewController` les `import` suivants:

```
import UIKit
import AVKit
```

```
import AVFoundation
```

Ajouter le `video player`

Dans la classe `ViewController` ajouter la fonction suivante:

```
override func viewDidAppear(_ animated: Bool){
    super.viewDidAppear(animated)

    // Code
}
```

Cette fonction est fournie par iOS. Elle est appelée quand votre vue vient d'apparaitre sur l'écran de l'utilisateur.

Ensuite:

- Renommer `findVideo()` en `playVideo()`
- Dans `playVideo()`:
    - Coder la création d'un instance `AVPLayer`

```
let player = AVPlayer(url: URL(fileURLWithPath: path))
let playerController = AVPlayerViewController()
playerController.player = player
```

- Afficher le lecteur de vidéo

```
//...
playerController.player = player
present(playerController, animated: true) {
    player.play()
}
```

- Appeler `playVideo()` dans `viewDidAppear()`
- Vérifier s'il y a de erreurs
- Lancer le simulateur. Bon visonnage!

À la fin, vous devez avoir un fichier `viewController.swift` similaire à celui-ci:

```
import UIKit
import AVKit
import AVFoundation

class ViewController: UIViewController {
```

```
    override func viewDidLoad() {
        super.viewDidLoad()
        // Do any additional setup after loading the view.
    }
    override func viewDidAppear(_ animated: Bool) {
        super.viewDidAppear(animated)

        playVideo()
    }

    private func playVideo() {
        guard let path = Bundle.main.path(forResource: "test", ofType:"mp4") else {
            debugPrint("Video file not found")
            return
        }
        let player = AVPlayer(url: URL(fileURLWithPath: path))
        let playerController = AVPlayerViewController()
        playerController.player = player
        present(playerController, animated: true) {
            player.play()
        }
    }
}
```

[1] Si vous souhaitez publier votre app sur l'App Store, il faut que l'identifiant de ballot soit unique. Par la suite, cet ID va permettre de mettre à jour votre app sans devoir en faire une nouvelle.