Construire son UI

Une bonne app part de là...

Construire son UI

- Les différents moyens
- UIKit : Storyboards
- UIKit : organisation
- Adaptabilité de l’interface
- Objets courants
- Mode sombre

Les différents moyens

Les différents moyens

Les différents moyens

UIKit	Code (impératif)	Fichier xib	Fichier Storyboard
SwiftUI	Swift (descriptif)		

SwiftUI

- Framework UI déclaratif
- Disponible sur toutes les plateformes Apple
- Remplace ou complète UIKit / AppKit / WatchKit
- 100% natif

SwiftUI

- Compatible iOS 13+
- Encore en évolution, la connaissance de UIKit parfois nécessaire
- SwiftUI et UIKit sont compatibles entre eux

- Sûrement le futur de la conception d’UI

SwiftUI

```swift
import SwiftUI

struct Content : View {

    @State var model = Themes.listModel

    var body: some View {
        List(model.items, action: model.selectItem) { item in
            Image(item.image)
            VStack(alignment: .leading) {
                Text(item.title)
                Text(item.subtitle)
                    .color(.gray)
            }
        }
    }
}
```

UIKit : Storyboards

- Façon de concevoir son UI
- Fonctionne en reliant des scènes via des segues
- Chaque scène représente une instance de UIViewController
  - Un UIViewController présente une vue, ou parfois encapsule un autre UIViewController

Storyboards

Storyboards

Storyboards

```
prepare(for segue:, sender:)
```

```
override func prepare(for segue: UIStoryboardSegue, sender: Any?) {

    if segue.identifier == "displayDetails" {
        var nextViewController = segue.destination
        nextViewController.title = "The title of the next viewController"
    }
}
```

Storyboards

UIKit : organisation

UIKit : organisation

- UIKit → Framework
- Contient toutes les classes pour vos UI
  - UIWindow, UIButton, UILabel…
- Travail simplifié pour le développeur

UIKit : organisation

- Une grande variété d'objets disponibles
- Accessibles en drag'n drop, ou via le code

- UIResponder
  - Gestion des touches
    - `func touchesBegan(_ touches: NSSet, withEvent event: UIEvent)`
    - `func touchesCancelled(_ touches: NSSet!, withEvent event: UIEvent!)`
    - `func touchesEnded(_ touches: NSSet, withEvent event: UIEvent)`
    - `func touchesMoved(_ touches: NSSet, withEvent event: UIEvent)`

[Obj-C]

UIResponder

- Gestion des touches
  - `touchesBegan:(NSSet *)touches withEvent:(UIEvent *)event`
  - `touchesCancelled:(NSSet *)touches withEvent:(UIEvent *)event`
  - `touchesEnded:(NSSet *)touches withEvent:(UIEvent *)event`
  - `touchesMoved:(NSSet *)touches withEvent:(UIEvent *)event`

UIWindow

- Point de départ de l'interface
- Une app iOS ne possède, en général qu'une fenêtre

UIView

- Zone rectangulaire permettant d'afficher du contenu
- Par défaut : un rectangle rempli avec une couleur de fond
- Peut être sous-classé pour être personnalisé
- Une vue peut contenir des sous-vues
  - `var subviews: [UIView] { get }`
- Une vue peut posséder une vue parente
  - `var superview: UIView? { get }`

[Obj-C]

UIView

- Zone rectangulaire permettant d'afficher du contenu
- Par défaut : un rectangle rempli avec une couleur de fond
- Peut être sous-classé pour être personnalisé
- Une vue peut contenir des sous-vues
  - `-(NSArray *) subviews;`
- Une vue peut posséder une vue parente
  - `-(UIView *) superview;`

UIControl

- Sous-classe de UIView
- Permet de réagir simplement aux taps en envoyant des actions (IBAction)

Adaptabilité de l’interface

Adaptabilité de l’interface

- iOS fonctionne sur une grande variété d'appareils
- Chaque appareil peut être utilisé en portrait ou paysage
- Cela donnerait énormément d'écrans à concevoir
  - Auto Layout, les classes de tailles et les contrôleurs adaptatifs nous aident à réduire ce travail

Coordonnées

- CGFloat : type défini à partir de float ou de double utilisé dans Core Graphics
- CGPoint : structure représentant un point
  - `(CGFloat x, CGFloat y)`
- CGSize : structure représentant une taille
  - `(CGFloat width, CGFloat height)`
- CGRect : structure représentant un rectangle
  - `(CGPoint origin, CGSize size)`

Coordonnées

- Origine du système de coordonnées en haut à gauche
- Coordonnées en *points*. Les *points* sont liés aux *pixels* en fonction de la densité de l'écran.

Adaptabilité de l’interface

Points : (3,2)

@3x

Auto Layout

- Par défaut, nos éléments sont positionnés de manière fixe dans le système de coordonnées

Auto Layout

- Par défaut, nos éléments sont positionnés de manière fixe dans le système de coordonnées
- Auto Layout nous permet d'exprimer les positions grâce à des contraintes qui seront évaluées dynamiquement
- Auto Layout définit des relations mathématiques entre nos objets graphiques.
- On doit définir suffisamment de contraintes pour placer les objets sans ambiguïtés

Demo
iPhone 6s
Running Demo on iPhone SE
Demo
Demo
Main.storyboard
Main.storyboard (Base)
View Controller Scene
View Controller
View
L Label
View Controller Scene
View Controller
Top Layout Guide
Bottom Layout Guide
View
L Label
First Responder
Exit
Storyboard Entry Point
Label
Add New Constraints
0
0
626
Spacing to nea
Constrain
Width
Height
Equal Widths
Equal Heights
Aspect Ratio
Align
Leading E
Update Frames
None
Add Con
Selected Views
Update Frames
Update Constraints
Add Missing Constraints
Reset to Suggested Constraints
Clear Constraints
All Views in View Controller
Update Frames
Update Constraints
Add Missing Constraints
Reset to Suggested Constraints
Clear Constraints
Filter
View as: iPhone 6s (wC hR)
100 %
Demo

Auto Layout

- Possibilité de définir les contraintes dans le code
- Contraintes définies sous forme d'une équation linéaire
  - `élément1.attribut (=, <=, >=) élément2.attribut x multiplieur + constante`
- `NSLayoutConstraint`

Objets courants

Boutons

- UIButton
  - `init(type: UIButtonType) -> UIButton`
  - `var titleLabel: UILabel? { get }`
  - `var imageView: UIImageView? { get }`

Boutons

- UIButton → UIButtonTypeCustom
  - `func setImage(_ image: UIImage?, for state: UIControlState)`
- UIControlState
  - `.normal`
  - `.highlighted`
  - `.disabled`
  - `.selected`

Table View

- Affiche une liste d'éléments
- Organisation en section
- Différents styles d'affichage
- Contenu statique ou dynamique

UIKit : UITableView | SwiftUI : List

Navigation

Navigation Bar

Back button

- Permet la navigation hiérarchique entre les écrans
- Affiche le titre et le bouton précédent en fonction des titres des écrans
- Gère les animations et les gestes

UIKit : UINavigationController | SwiftUI : NavigationView

Tab Bar

- Permet l'affichage d'un menu pour l'app
- Gère le passage d'un écran à l'autre
- Affiche une image monochrome et un titre pour chaque écran
- Un TabBar devrait toujours rester visible

Tab Bar item

Tab Bar

UIKit : UITabBarController | SwiftUI : TabView

Collection View

- UICollectionView
  - UITableView aux stéroïdes
  - Affiche une collection d'éléments
  - Complètement personnalisable

Stack View

- Gère le layout d'une collection de vues
- Différentes orientions possibles
- Composant unique en UIKit, multiples en SwiftUI
- Propriétés UIStackView
  - `axis, distribution, spacing, alignment`

UIKit : UIStackView | SwiftUI : HStack/VStack/ZStack

Mode sombre

Mise en place

- Simple à mettre en place si pensé dès le départ
- Utilisation de couleurs sémantiques
  - labelColor, secondaryLabelColor, backgroundColor, etc.
- Utilisation des couleurs nommées dans le Asset catalog
- Si utilisation d'images, il faut penser aux différentes apparences
  - L'utilisation de symboles ou images templates simplifient le support

Supporting Dark Mode in Your Interface

Mode sombre

Test
iPad Pro
Test: Ready | Today at 22:44
Test
Test
Assets.xcassets
customColor
Universal Dark Appearance
AppIcon
customColor
customColor
Color
Any Appearance
Light Appearance
Dark Appearance
Universal
Filter
Show Slicing
Color Set
Name
customColor
Devices
Universal
iPhone
iPad
Mac Catalyst
CarPlay
Apple Watch
Apple TV
Mac
Appearances
Any, Light, Dark
High Contrast
Gamut
Any
Localization
Localize...
Color
Content
sRGB
Input Method
8-bit (0-255)
Red
255
Green
147
Blue
0
Opacity
100,0 %
Show Color Panel

Opt-out

- Possibilité de ne pas supporter le mode sombre
  - Sur certains écrans
    - `overrideUserInterfaceStyle = .light`
  - Ou toute l'app
    - Clé `UIUserInterfaceStyle` dans Info.plist avec la valeur `Light`
    - Pas recommandé !

Supporting Dark Mode in Your Interface

Pour aller plus loin...

- Designing for iOS
- UI Elements
- iOS Human Interface Guidelines (iBook)