

Présentation de Swift 5 et d'iOS 13

**Attention ce fichier comporte des videos, il doit
être ouvert avec un logiciel adapté**

Sommaire:

-XCODE

- Apparence et navigation
- Editeur de code
- Simulateur
- Swift Package Manager

-Dark mode

- Les couleurs automatiques
- Les couleurs personnalisées
- Les variantes d'images
- Le mode sombre dans le code
- Désactiver le mode sombre

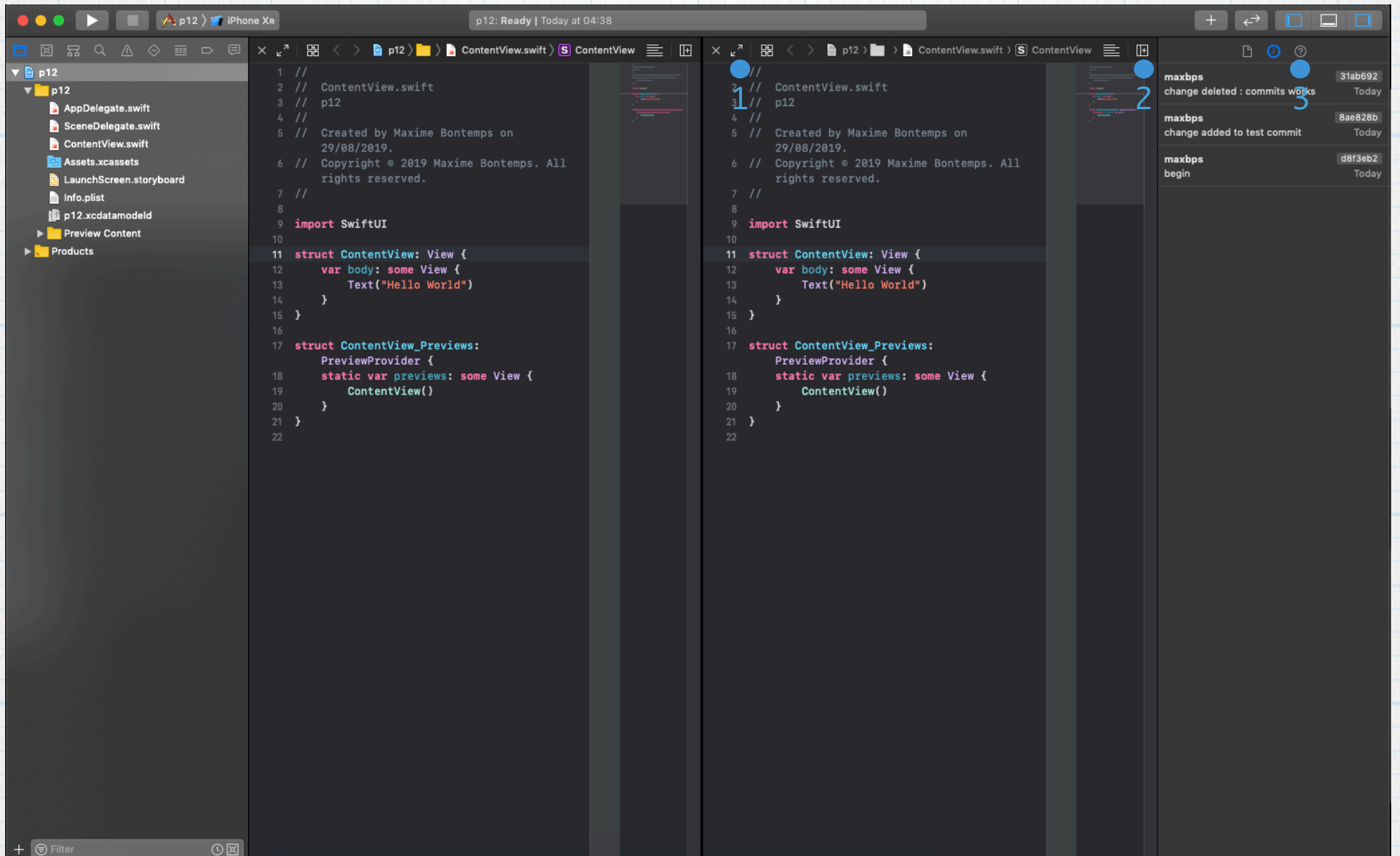
-SF Symbols

- L'app SF Symbols
- Utiliser depuis le StoryBoard
- Personnaliser l'affichage

-Swift UI

- UIKIT / SwiftUI
- Utiliser Xcode avec SwiftUI
- Les modificateurs
- VStack, Hstack , ZStack
- Spacer
- Créer un composant
- Naviguer entre deux écrans

Apparence et navigation



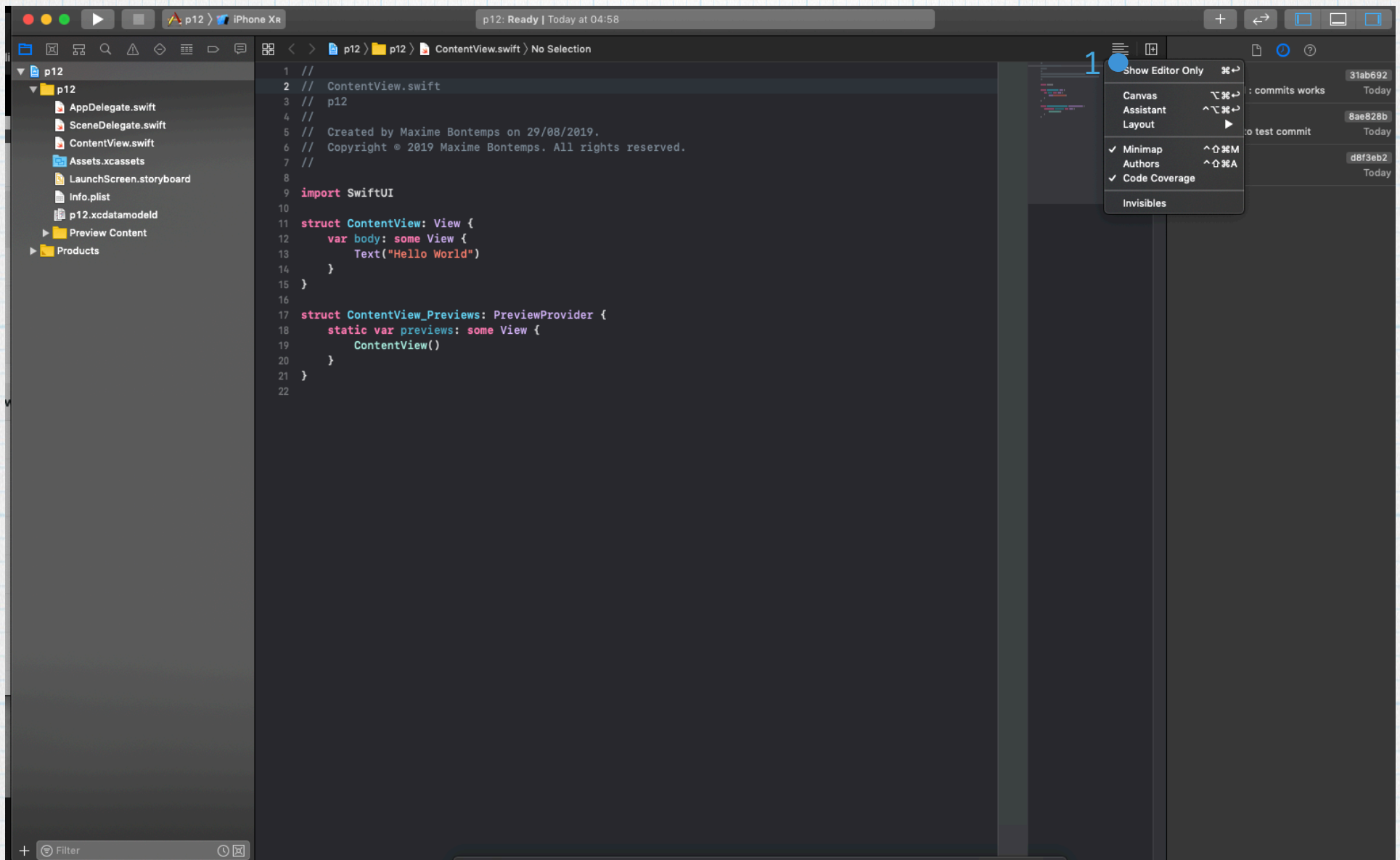
1: Bouton qui permet d'étirer simplement une fenêtre en plein écran, en appuyant de nouveau, la fenêtre revient à son emplacement, cela conserve toutes les fenêtres.

2: Bouton qui permet d'ajouter une fenêtre, les fenêtres peuvent être rajouter verticalement ou horizontalement et on peut y afficher le fichier de notre choix.

3: Un nouvel onglet est rajouté à l'inspecteur. Il permet d'avoir un visuel sur les différents commits du fichier concerné.

De manière générale XCODE a été amélioré pour nous permettre d'optimiser notre espace de travail. Par exemple on a un choix plus complet de thèmes que dans les versions précédentes.

Editeur de code



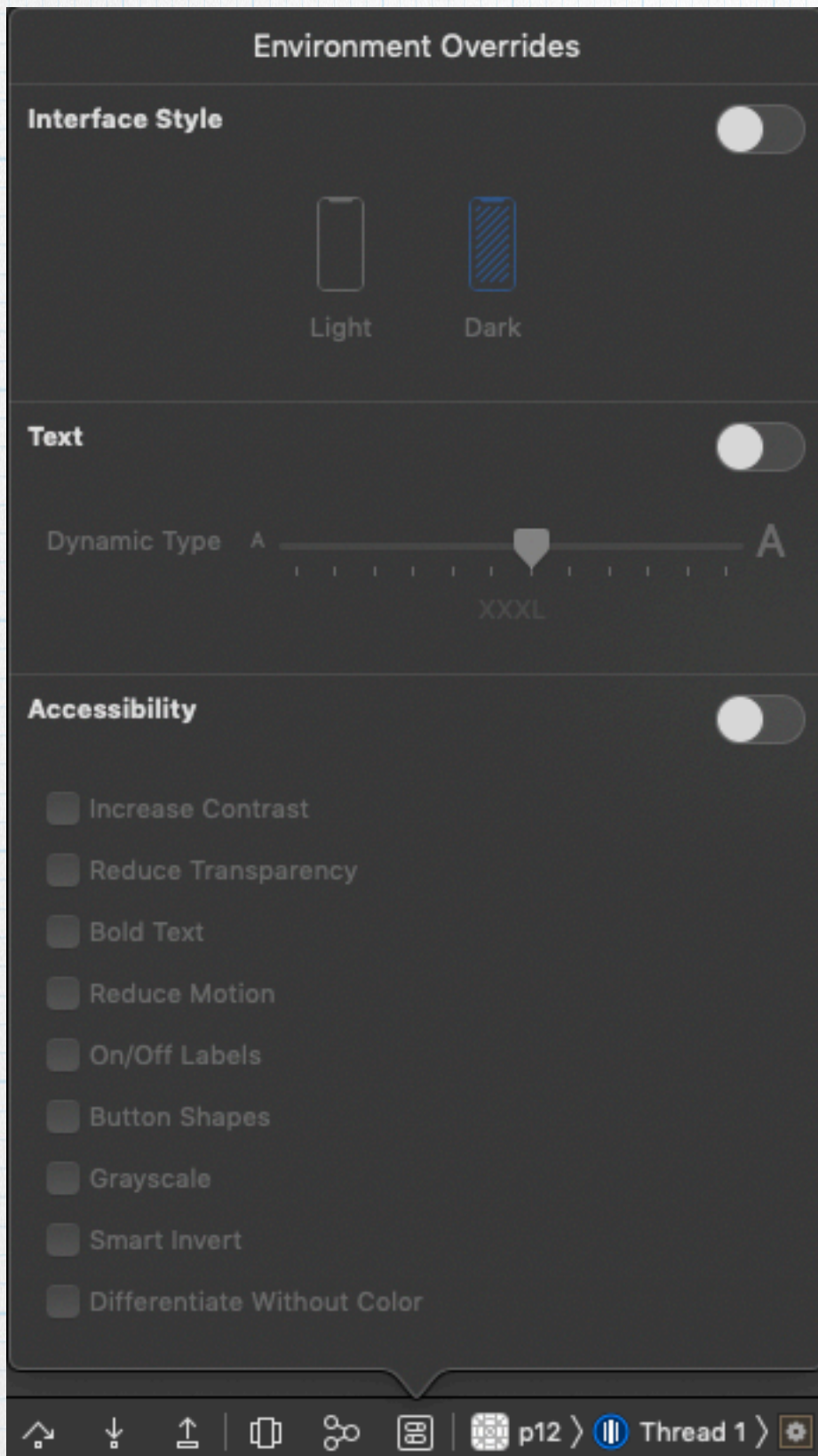
1: Bouton qui ajoute des options sur l'éditeur de code:

- minimap : bandeau avec résumé du code (avec le raccourcis cmd on a encore plus de précision)
- author : permet de voir l'auteur du dernier commit

La gestion des commentaires a été largement améliorée, grâce à « add documentation » les commentaires sont gérés avec Xcode en même temps que le reste du code.

Simulateurs

Les performances du simulateur ont nettement été améliorées, il consomme beaucoup moins de CPU que sur les anciennes versions.

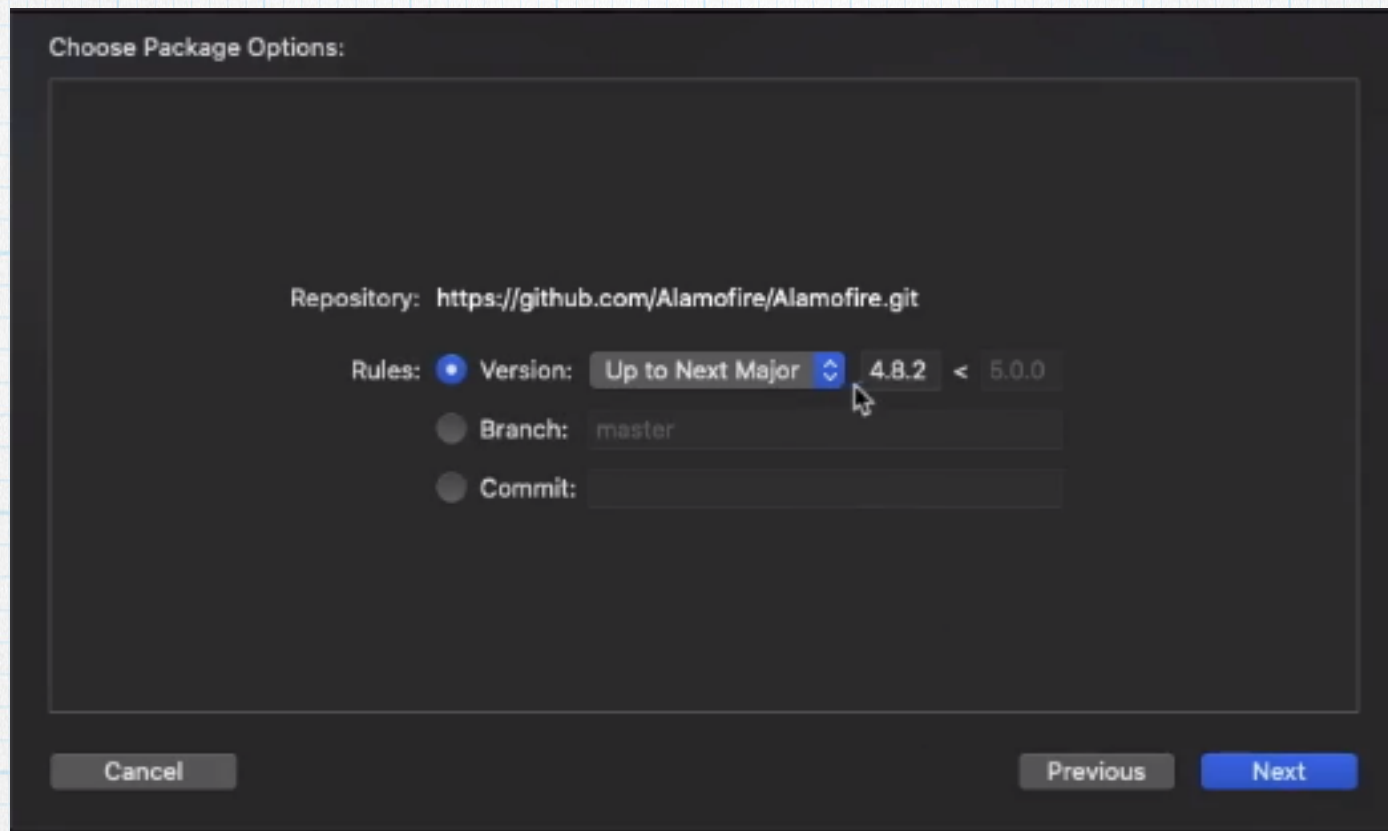


Grâce au bouton Environment Overrides on va directement pouvoir effectuer des changements sur le simulateur (passer du mode clair au sombre par exemple).

Xcode

Swift package manager

Swift package manager existait déjà mais il est désormais directement intégré dans Xcode.













- + Plus besoin d'utiliser CocoaPods, on peut laisser Xcode gérer les dépendances
- + Les mises à jour sont faites automatiquement, jusqu'à une Maj majeur.

Dark Mode

L'année dernière le monde sombre est sorti sur macOS mais c'est avec l'arrivée d'iOS 13 que l'on va voir débarquer le mode sombre sur nos iPhones. Cela implique pour le développeur de gérer les images et les couleurs (quelles couleurs en mode clair et quelles couleurs en mode sombre). On peut néanmoins le désactiver, sur des pages spécifiques ou sur toute l'application.

Les couleurs automatiques

Light	Dark	Name	API
 R 0 G 122 B 255	 R 10 G 132 B 255	Blue	systemBlue
 R 142 G 142 B 147	 R 152 G 152 B 157	Gray	systemGray
 R 52 G 199 B 89	 R 48 G 209 B 88	Green	systemGreen
 R 88 G 86 B 214	 R 94 G 92 B 230	Indigo	systemIndigo
 R 255 G 149 B 0	 R 255 G 159 B 10	Orange	systemOrange

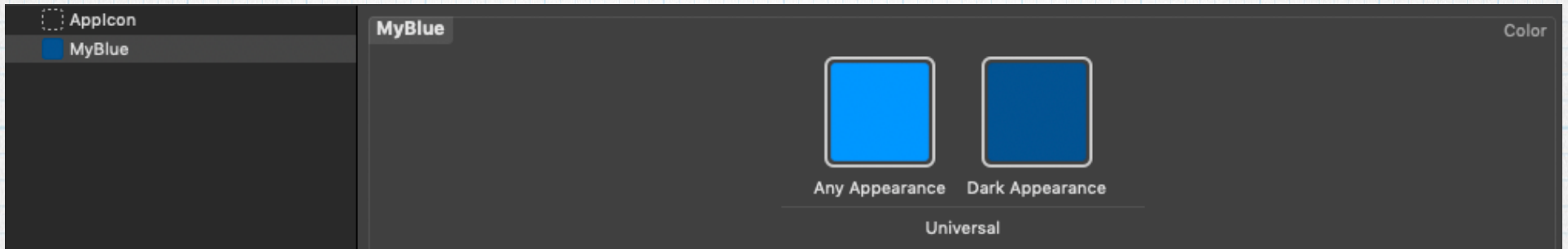
Apple a ajouté beaucoup de nouvelles couleurs. On peut voir qu'une légère nuance est présente entre la couleur « light » et la couleur « dark ». Toutefois, on peut choisir n'importe quelle autre couleur (celle-ci ne changera pas en fonction du mode).

Source : <https://developer.apple.com/design/human-interface-guidelines/ios/visual-design/color/>

Dark Mode

Les couleurs personnalisées

On peut souhaiter ne pas utiliser les couleurs fournies par Apple et créer son propre jeu de couleur. Pour cela il suffit de se rendre dans le Assets.xcassets et cliquer sur « new color set ». A partir de là on pourra choisir notre couleur qui aura une teinte différente selon notre mode, sombre ou clair.



Les variantes d'images

De la même manière que pour les couleurs on peut vouloir changer l'apparence d'une image en fonction du mode, sombre ou clair. Pour cela il suffit de faire le même procédé que pour les couleurs après avoir cliqué sur new image set.



Dark Mode

Le mode sombre dans le code

```
import UIKit

class ViewController: UIViewController {

    override func viewDidLoad() {
        super.viewDidLoad()

        if self.traitCollection.userInterfaceStyle == .dark {

        } else {

        }
    }

    override func traitCollectionDidChange(_ previousTraitCollection: UITraitCollection?) {
        if self.traitCollection.hasDifferentColorAppearance(comparedTo: previousTraitCollection) {

        }
    }
}
```

On utilisera la fonction `viewDidLoad` lorsqu'on voudra détecter un changement de mode au chargement d'une page.

On utilisera la seconde fonction lorsqu'on voudra détecter un changement de mode à n'importe quel moment.

Désactiver le mode sombre

On peut désactiver le mode sombre ou le mode clair sur une page de notre application en forçant le mode souhaité dans le code comme ci dessous:

```
self.overrideUserInterfaceStyle = .dark
```

On peut aussi désactiver le mode sombre ou clair pour toute l'application en rajoutant dans le fichier `Infos.plist` la ligne ci dessous:

```
User Interface Style    ⇅    String    Light
```


SF Symbols



SF(San Francisco) Symbol est la police qu'Apple utilise pour toute sa communication

SF Symbols est un ensemble d'icônes, fournis par Apple, qu'on va directement pouvoir utiliser dans nos applications

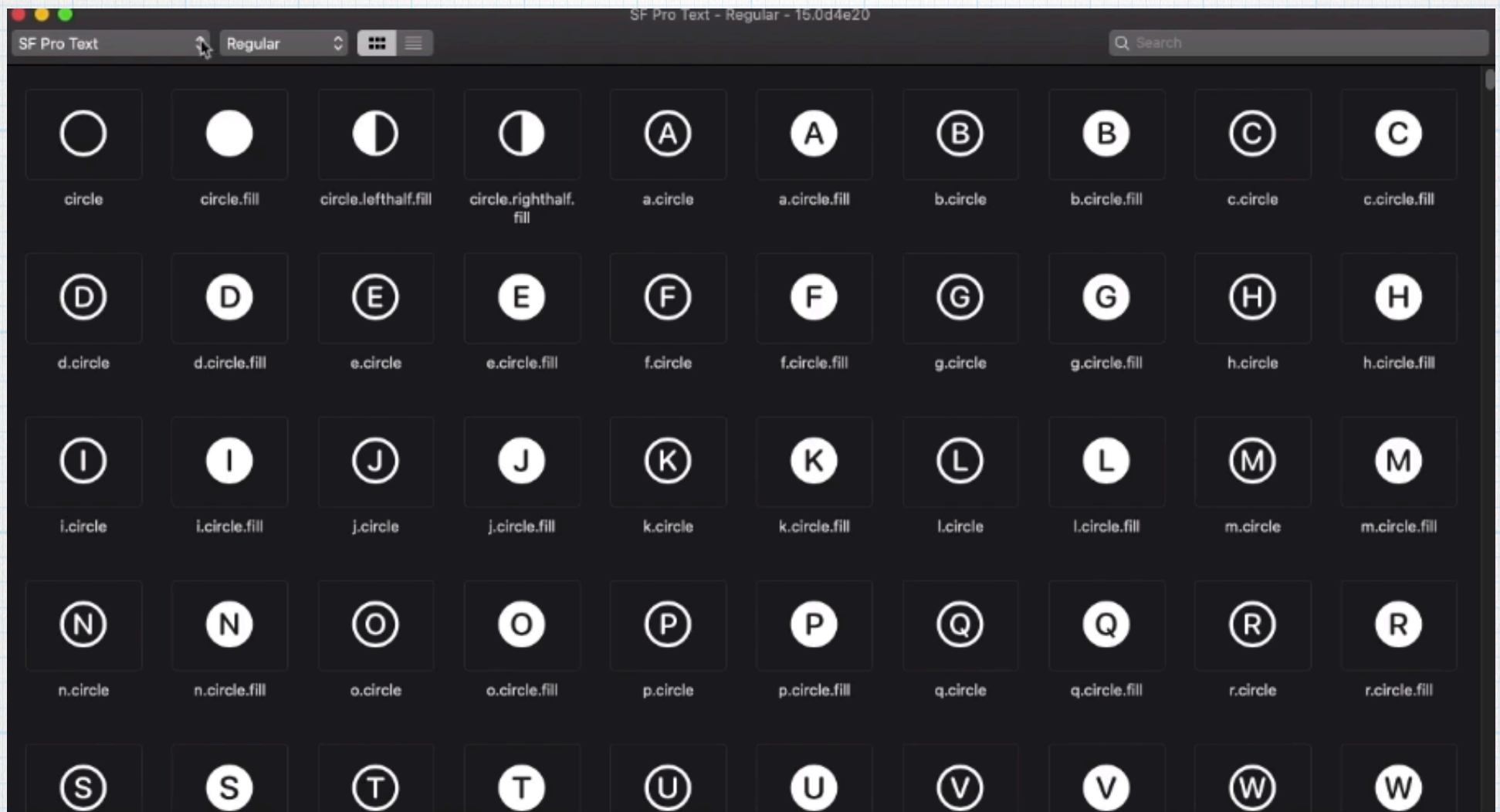
SF Symbol c'est :

- + de 1500 icônes
- des icônes personnalisables (tailles , épaisseur, couleur)
- des icônes utilisable directement dans le storyboard ou dans le code

Source : <https://developer.apple.com/design/human-interface-guidelines/sf-symbols/overview/>

SF Symbols

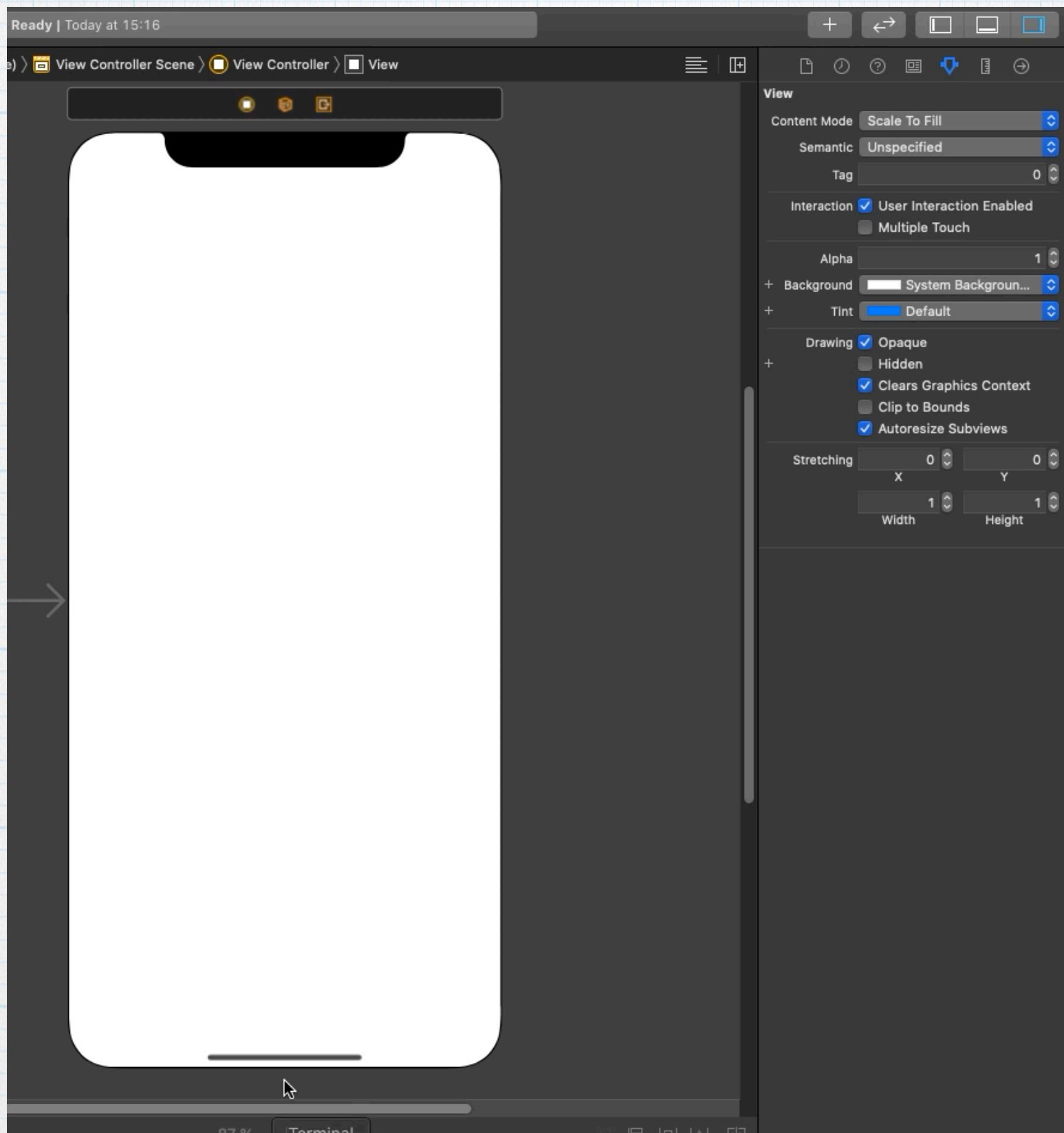
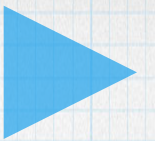
L'app SF Symbols



SF Symbols est téléchargeable ici: <https://developer.apple.com/design/downloads/SF-Symbols.dmg>

C'est très pratique pour naviguer entre les différentes icônes proposées. Pour les plus expérimentés on peut aussi exporter des icônes pour les modifier. Mais attention certaines icônes ne sont utilisables que pour un domaine spécifique (par exemple le nuage n'est utilisable que pour le cloud). En dehors de ces icônes on peut les utiliser comme on le souhaite.

SF Symbols depuis le storyboard



Dans le storyboard (donc si on ne se sert pas de SwiftUI) on peut directement ajouter les logo de SF Symbols depuis la propriété image (concerne les imageView et les buttons

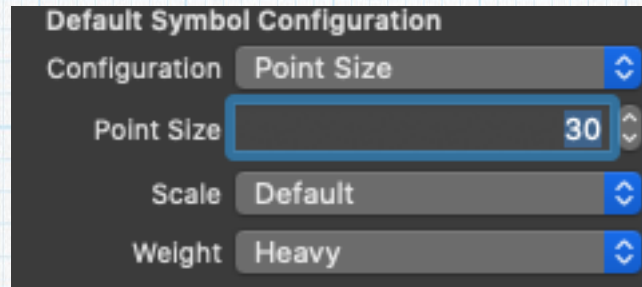
Pour ajouter ou modifier l'image d'un bouton directement dans le code :

```
trashButton.setImage(UIImage(systemName: "trash"), for: .normal)
```


SF Symbols

Personnaliser l'affichage

On peut choisir la police du symbole directement dans le storyboard avec le panneau Default Symbol Configuration



On peut modifier la police du symbole dans le code, en créant une constante contenant la configuration voulu et en l'appliquant aux endroits voulus, c est valable pour les boutons et les images.

```
let configurationDuSymbole = UIImage.SymbolConfiguration(pointSize: 30.0, weight: .heavy, scale: .default)
```

```
trashButton.setPreferredSymbolConfiguration(configurationDuSymbole, forImageIn: .normal)
```

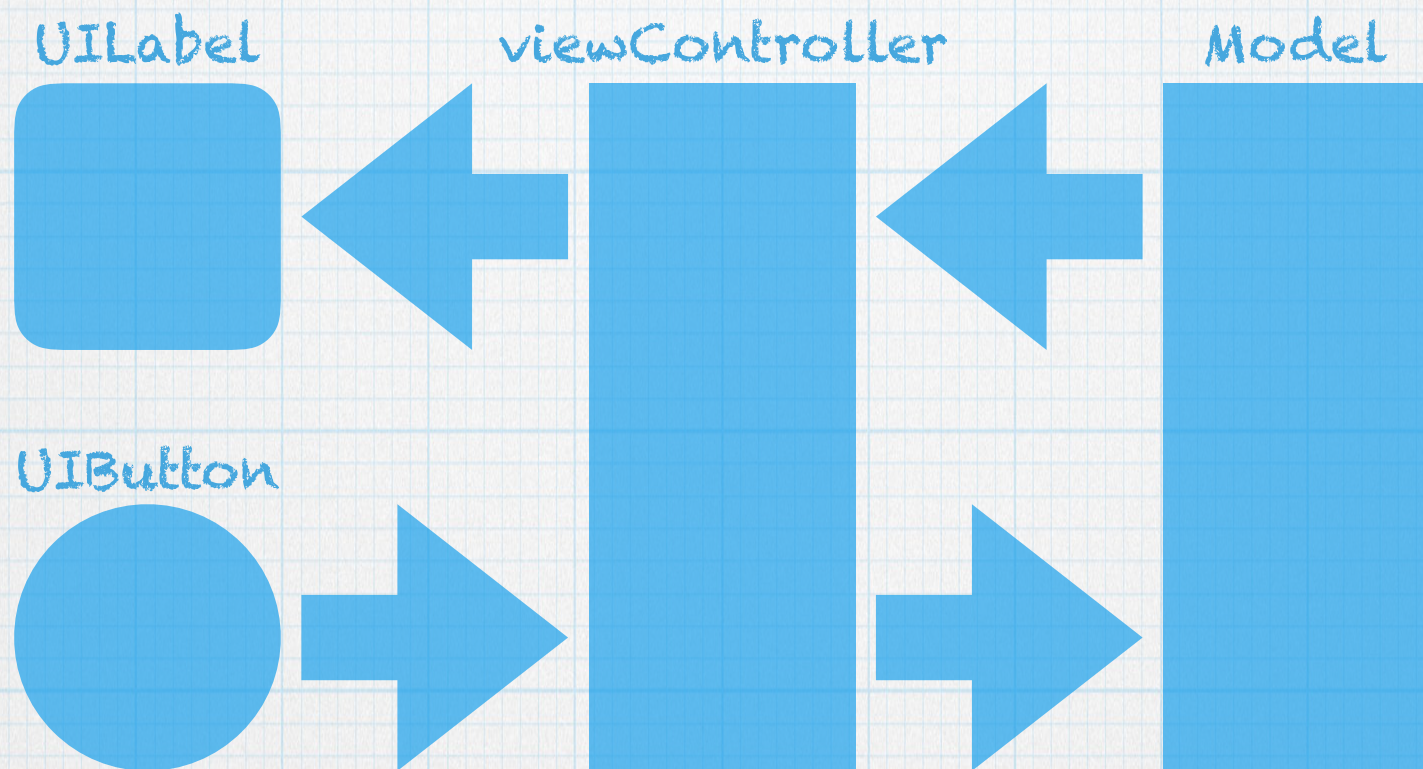
Pour conclure, SF Symboles est très pratique car les icônes sont directement utilisable dans Xcode. On peut les utiliser dans le storyboard ou/et dans le code. Finalement cela permet d'améliorer l'interface de nos applications.

SwiftUI

SwiftUI est le nouveau framework graphique proposé par Apple.

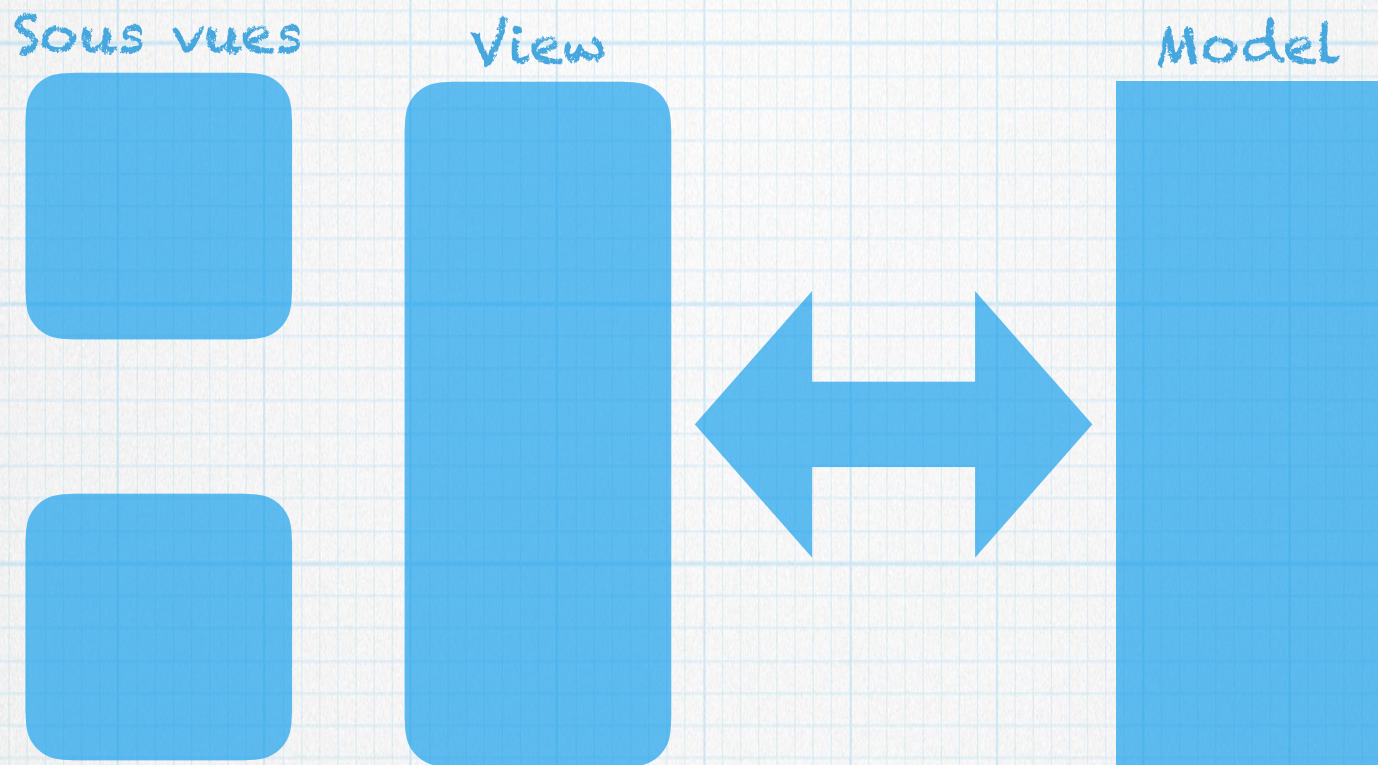
- > Nouveau système de mise en page
- > Multiplateforme (appelWatch, appleTV, iPhone etc)
- > Disparition du storyboard et du viewController
- > Simplification du code
- > Nouvelle gestion des données affichées

Fonctionnement de UIKit



L'information met un léger temps pour se déplacer ce qui entraîne une légère désynchronisation (une différence pendant un court moment entre les données affichées et les données réelles)

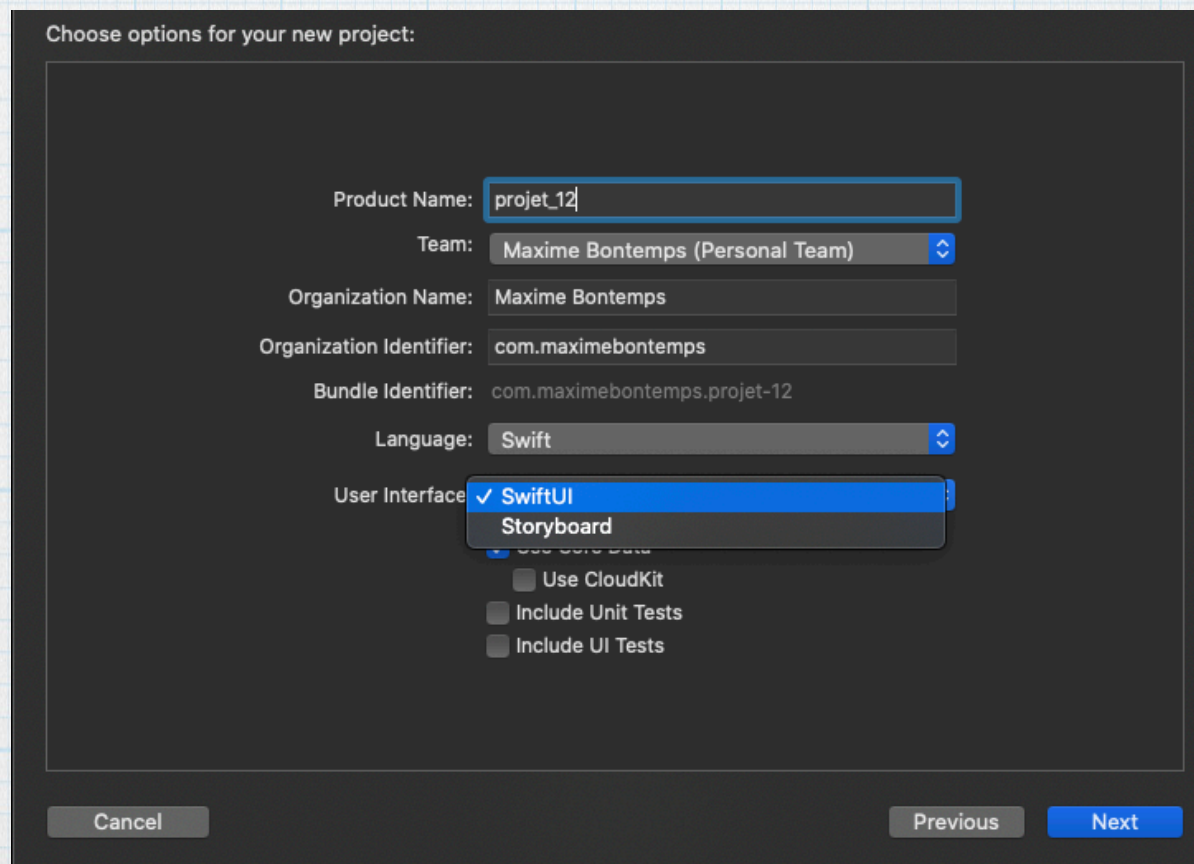
Fonctionnement de SwiftUI

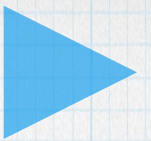


Ici les vues se mettent à jour toute seule si les données changent.

Utiliser Xcode avec SwiftUI


Lors de la création d'un projet, on peut cocher la case use swiftUI pour pouvoir l'utiliser

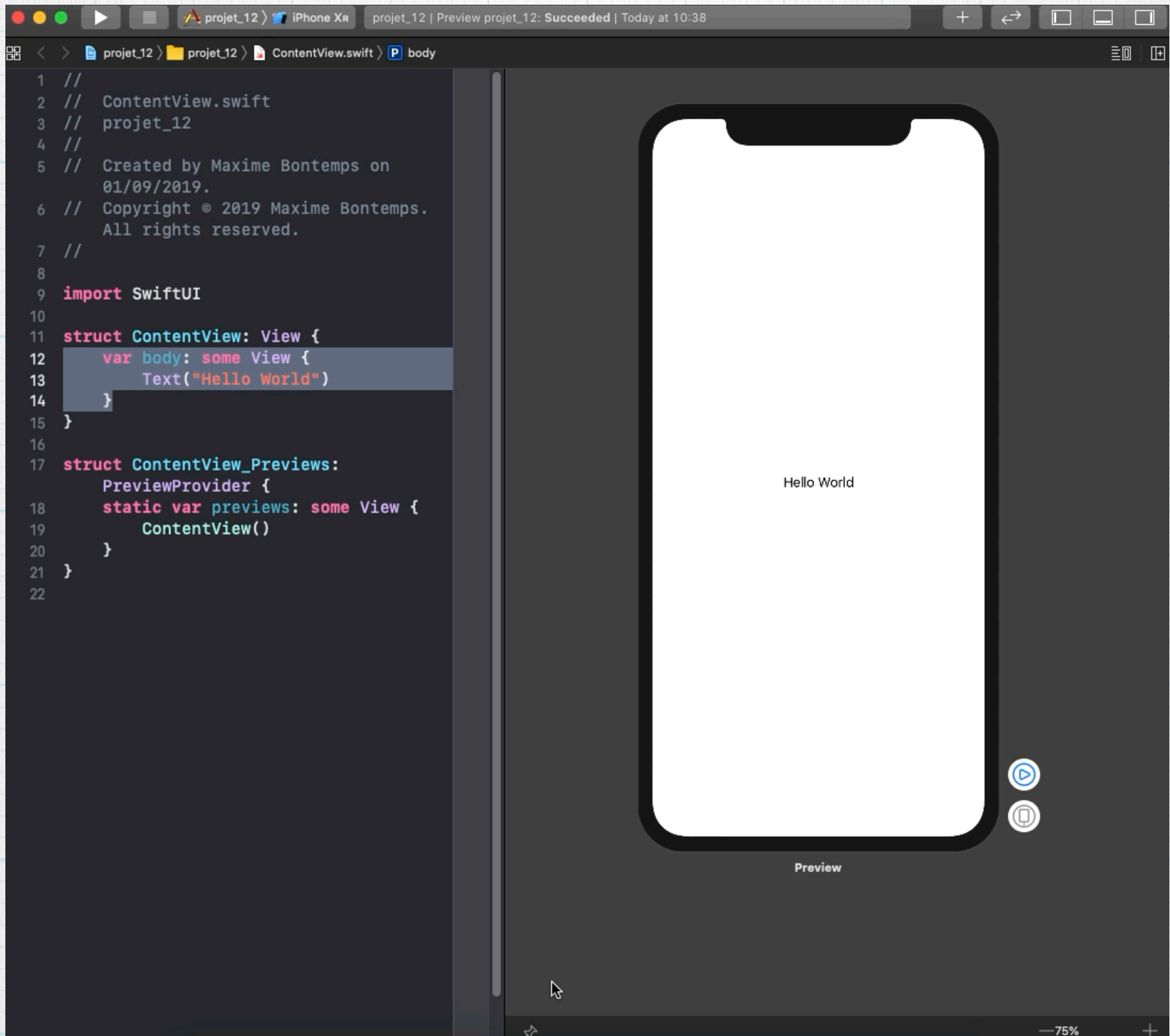


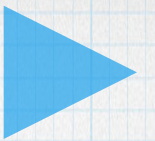


Par défaut une vue est créée. On peut facilement l'afficher dans le Canvas à l'aide du bouton : 


En modifiant un objet, il se modifie directement dans le canvas.

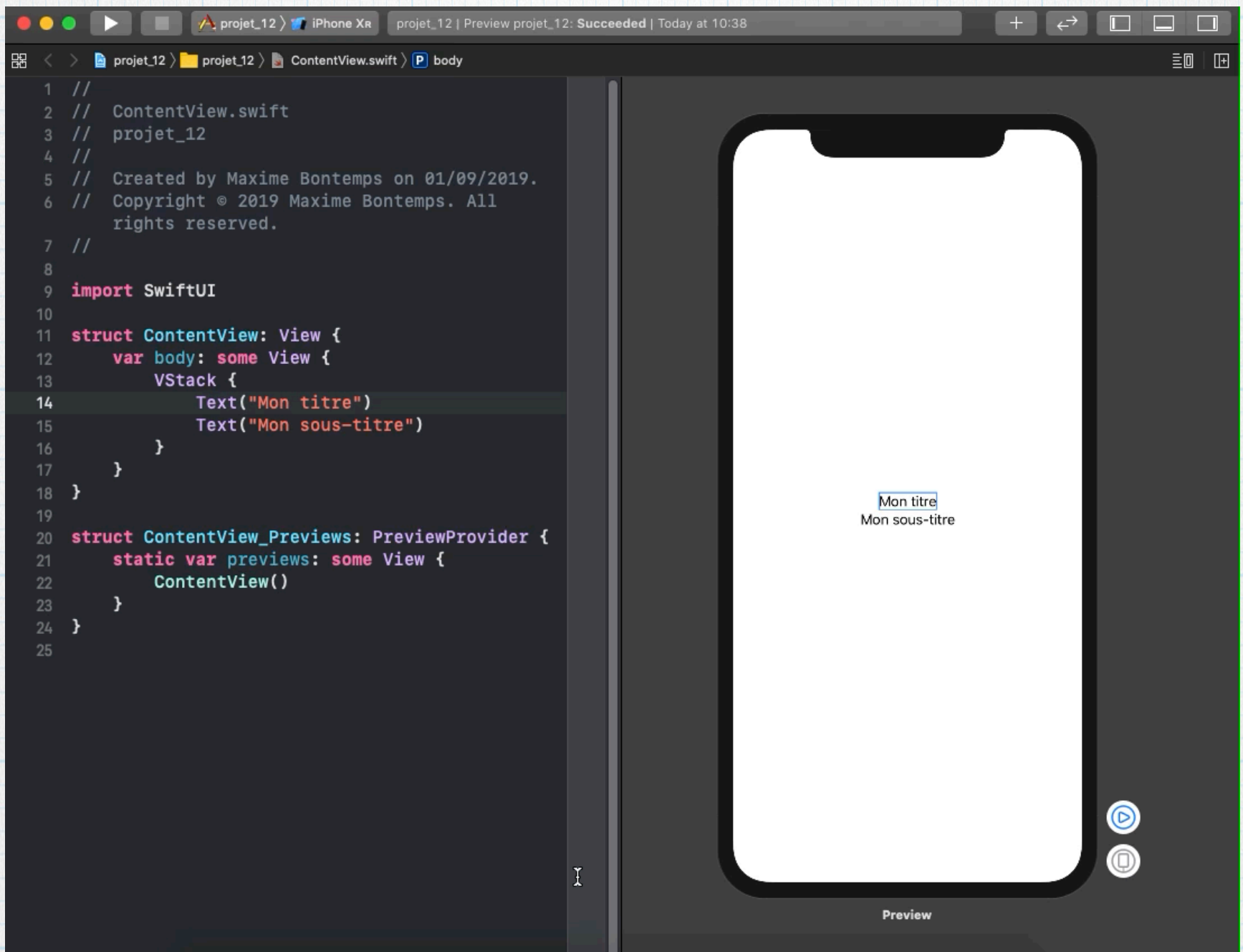
Le bouton :  nous permet d'ajouter des composants à notre vue rapidement, on pourra alors choisir son placement.





Les modificateurs

On va très facilement pouvoir modifier les paramètres d'un objet. En appuyant sur l'onglet  on pourra accéder à tous les modificateurs et en choisir un. On peut aussi directement affecter des paramètres à un objet dans le code.

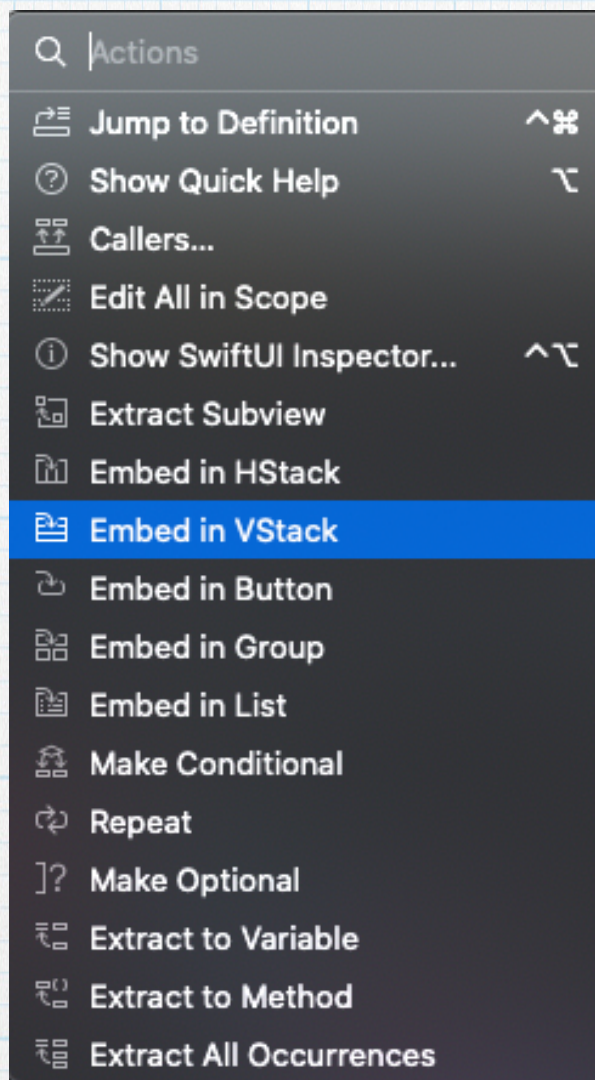


VStacks, HStacks, ZStacks

Le corps de notre vue ne peut avoir qu'une seule vue racine. Il faut donc lui ajouter des sous vues. Les stacks sont des piles qui regroupent les sous vues. Il en existe 3 types différents :

- VStack : assemble les vues verticalement
- HStack : assemble les vues Horizontalement
- ZStack : superpose les vues

En effectuant un cmd + click sur un élément on va pouvoir directement le mettre dans une Stack, où on peut l'écrire nous même



Spacer

Un spacer est un composant graphique qui permet de répartir l'espace vide à l'endroit où l'on souhaite.

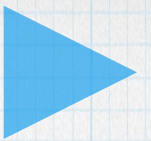
```

1 // ContentView.swift
2 // projet_12
3 //
4 // Created by Maxime Bontemps on 01/09/2019.
5 // Copyright © 2019 Maxime Bontemps. All
6 // rights reserved.
7 //
8
9 import SwiftUI
10
11 struct ContentView: View {
12     var body: some View {
13         VStack {
14             Text("Mon titre")
15                 .font(.title)
16             Spacer()
17             HStack {
18                 Text("Mon ")
19                     .font(.subheadline)
20                     .italic()
21                 Spacer()
22                 Text("sous-titre")
23                     .font(.subheadline)
24                     .italic()
25             }
26         }
27     }
28 }
29
30 struct ContentView_Previews: PreviewProvider {
31     static var previews: some View {
32         ContentView()
33     }
34 }
35

```



Dans cet exemple, il y a une `VStack` comprenant deux éléments : un texte et une `HStack`. En ajoutant un `spacer` entre les deux, Xcode va répartir tout l'espace disponible entre les deux (verticalement puisqu'il s'agit d'une `VStack`). Dans la `HStack` il y a deux textes, en ajoutant un `spacer` entre Xcode va répartir l'espace disponible entre eux (horizontalement).



Créer un composant

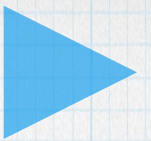
```
1 //
2 // ContentView.swift
3 // projet_12
4 //
5 // Created by Maxime Bontemps on 01/09/2019.
6 // Copyright © 2019 Maxime Bontemps. All rights reserved.
7 //
8
9 import SwiftUI
10
11 struct ContentView: View {
12     var body: some View {
13         VStack {
14             Text("Cocktails")
15                 .font(.title)
16
17             HStack {
18                 Image("gintonic")
19                     .resizable()
20                     .aspectRatio(contentMode: .fill)
21                     .frame(width: 50, height: 50)
22                 Text("Gin Tonic")
23
24                 Spacer()
25             }
26             Spacer()
27         }
28     }
29 }
30
31
32 struct ContentView_Previews: PreviewProvider {
33     static var previews: some View {
34         ContentView()
35     }
36 }
37
38
39 |
```



Dans cet exemple, On va créer un composant (avec cmd + click) qui contient toute la HStack.

On va pouvoir s'en resservir tout simplement et cela nous évite d'avoir des vues avec beaucoup de lignes de code. C'est un petit peu comme faire une fonction, ça permet d'y voir plus clair et on peut réutiliser ce composant autant de fois que l'on veut.

Il est conseillé d'enregistrer ces composants dans des nouveaux fichiers.



Naviguer entre deux écrans

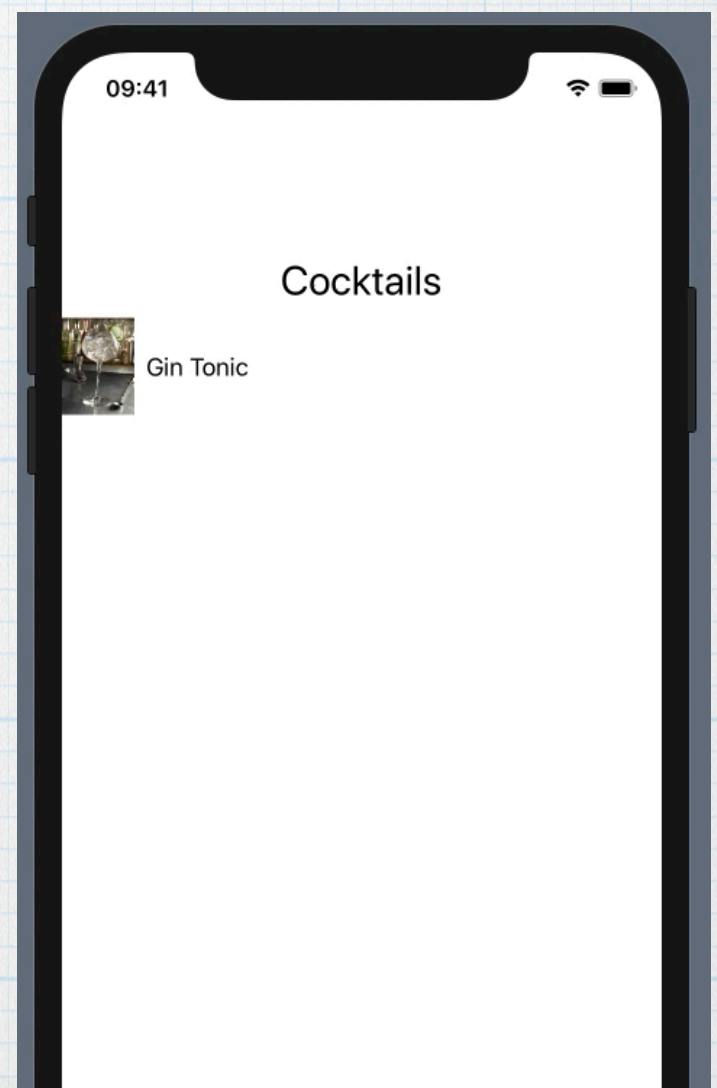
```
import SwiftUI

struct ContentView: View {
    var body: some View {
        NavigationView{
            VStack {
                Text("Cocktails")
                    .font(.title)
                NavigationLink(destination: CocktailDetails()){
                    cocktailsRow()
                }.buttonStyle(PlainButtonStyle())
                Spacer()
            }
        }
    }
}

struct ContentView_Previews: PreviewProvider {
    static var previews: some View {
        ContentView()
    }
}
```

Pour commencer j'ai créé une nouvelle vue nommée `CocktailDetails`. Ensuite j'ai créé un `NavigationLink` qui a pour destination cette vue et j'y ai mis la vue `cocktailRow` (si on appuie sur `cocktailRow` on va naviguer vers `CocktailsDetails`).

Enfin j'ai modifié sa propriété `.buttonStyle` afin d'avoir un affichage correct (par défaut tout ce qui se trouve dans un `navigationLink` est bleu)



SwiftUI

Sources

- apple
- purpleGiraffe
- Hacking with Swift
- appcoda
- différentes chaînes YouTube : Brian Advent, CodeWithChris, Lets Build That App

Pour aller plus loin

- tutoriaux swiftUI d'apple : <https://developer.apple.com/tutorials/swiftui/creating-and-combining-views>
- tutoriel raywenderlich : <https://www.raywenderlich.com/3715234-swiftui-getting-started#toc-anchor-001>