Développement Natif

Présentation des animations SwiftUI

PEISSEL Thomas

“ Qu’est-ce qu’une animation SwiftUI ? ”

Une animation est une transition fluide entre deux états visuels. Dans SwiftUI, elle peut apporter dynamisme et clarté à une interface utilisateur.

Pourquoi utiliser SwiftUI pour les animations ?

- Intégration simplifiée avec les vues.
- Gestion automatique des transitions grâce au **binding des états**.
- API intuitive pour créer des animations personnalisées.

Concept clé : le cycle de mise à jour

1. **Détection des changements d'état.**
2. **Recalcul de la vue via le body.**
3. **Interpolation des valeurs (si une animation est définie).**
4. **Rendu fluide.**

Corps d'une animation

Composants principaux d'une animation :

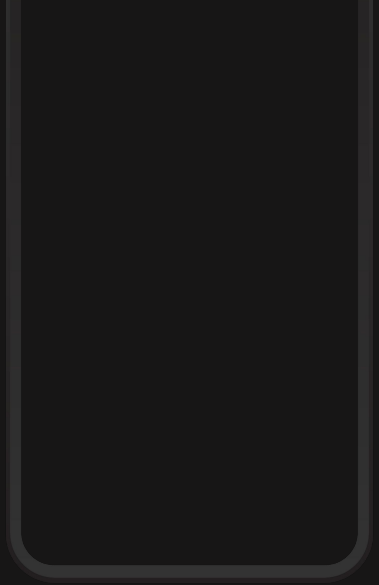
- **Propriété animable :** une donnée qui peut être interpolée (ex. : `scaleEffect`, `opacity`).
- **Animation :** le "comment" de la transition (ex. : durée, courbe, ressort).
- **Transaction :** le contexte où l'animation se produit.

Fonctionnement :

1. Un événement (ex. : un clic) modifie une propriété d'état.
2. SwiftUI détecte le changement et ouvre une **transaction**.
3. Si une animation est définie, SwiftUI interpole les valeurs de l'ancienne à la nouvelle.

```swift
struct ExampleView: View {
    @State private var isScaled = false

    var body: some View {
        Image(systemName: "star.fill")
            .scaleEffect(isScaled ? 1.5 : 1.0)
            .onTapGesture {
                withAnimation(.easeInOut(duration: 0.5)) {
                    isScaled.toggle()
                }
            }
    }
}
```



```
withAnimation(.spring(response: 0.6, dampingFraction: 0.8)) {
    isScaled.toggle()
}
```

Types d'animations

1. Animations prédéfinies :

- **Courbes de temps :** contrôle de la vitesse d'animation.
  - `easeIn`, `easeOut`, `easeInOut`, `linear`.
- **Ressorts :** animations naturelles avec rebonds.
  - Paramètres : durée perçue, bounciness (rebond).
  - Exemple : `.spring(response: 0.5, dampingFraction: 0.7)`.

2. Animations personnalisées :

- Utilisation de `CustomAnimation` pour des comportements uniques.

Propriétés animables et protocoles avancés

Propriétés animables :

- Une propriété est "animable" si elle implémente le protocole `Animatable`.
- SwiftUI anime automatiquement les types comme `CGFloat`, `CGPoint`, `CGSize`, etc.

Protocole `Animatable`:
Permet de définir une animation sur mesure en contrôlant la manière dont une propriété est interpolée.

Exemple :

```
struct CustomView: Animatable {
    var animatableData: CGFloat

    // ...
}
```

Cas d'utilisation :

- Création d'animations complexes (exemple : suivre une courbe).
- Exemples réels : animations circulaires ou transformations 3D.

Exemple pratique : Avatar animé

Animer la mise à jour d'un avatar lors d'un clic.

```swift
struct AvatarView: View {
    @State private var isSelected = false

    var body: some View {
        VStack {
            Image(systemName: "person.circle")
                .resizable()
                .frame(width: 100, height: 100)
                .scaleEffect(isSelected ? 1.2 : 1.0)
                .onTapGesture {
                    withAnimation(.spring()) {
                        isSelected.toggle()
                    }
                }
            Text(isSelected ? "Sélectionné" : "Non sélectionné")
        }
    }
}
```

Présentation :

1. **`scaleEffect` :** Modifie la taille de l'image.
2. **`withAnimation` :** Ajoute une animation avec un ressort.
3. **Gestion d'état :** La variable `isSelected` contrôle la transition.

Résultat :

Cela permet d’obtenir une animation fluide qui donne du dynamisme à l'interface.

MERCI

PEISSEL Thomas