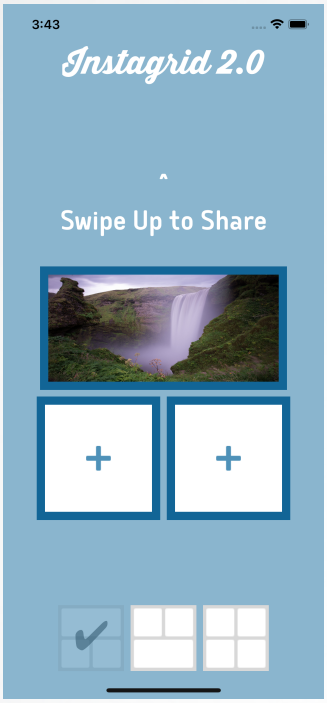
Instagrid

Instagrid 2.0 swiftUI

Présentation générale de l'application:

Instagrid est une Application iOS pour iPhone réalisée dans le cadre du parcours de développeur iOS **OpenClassRooms.**

Le présent document fait référence à la version SwiftUI de l'application développée dans un but d'apprentissage et dans le cadre du projet final du parcours.



Les boutons « + » permettent de charger des images depuis la photothèque de l'iPhone afin de générer une composition d'images personnelle.

Les 3 cases en bas (portrait) ou à droite (paysage) permettent de choisir une disposition.

Un swipe permet d'invoquer l'activity controller afin de partager le contenu crée.

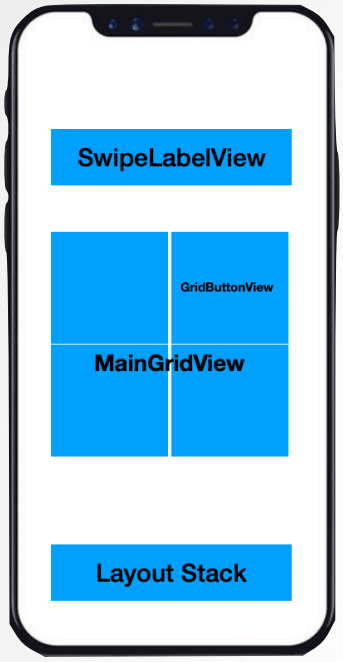
Une alerte se déclenche si l'utilisateur swipe alors qu'aucune photo n'a été chargée.

Le swipe doit etre initié depuis le texte en lui même « Swipe Up/ Left to Share »

L'application s'adapte au mode Paysage.

Architecture générale:

Dans la mesure où l'application ne dispose que d'un seul écran et de peu de logique à implémenter, l'architecture est simple et consiste en une vue (Content View) décomposée en sous vues, et dont les méthodes associées ont été écrites dans le même fichier que la vue pour plus de lisibilité compte tenu de la simplicité de l'application.



Content View:

```
VStack {
    Text("Instagrid 2.0")
        .foregroundColor(.white)
        .font(.custom("ThirstyScriptExtraBoldDemo", size: 40))
    Spacer()
    SwipeLabelView(sharingLayout: $showMainGrid)
    if showMainGrid {
        MainGridView()
            .transition(AnyTransition.move(edge: .top))
            .animation(.linear(duration: 1))
    }
    Spacer()
    HStack {
        LayoutStackView()
    }
```

Swipe Label View:

Le label reconnait le geste du swipe et déclenche la fonction shareLayout() qui va créer une image de type UIImage à partir de la grille, puis invoquer un UIActivityController à partir duquel l'utilisateur pourra partager ou exploiter l'image.

MainGridView:

Cette vue compose la grille principale avec des GridButton selon le layout choisi. Elle sert également de support pour la création de l'image à l'aide du ViewModifier .background.

GridButtonView:

Chaque bouton dispose d'une fonction loadImage() qui invoque le ImagePicker.
L'image se charge sur le bouton.

LayoutStackView:

Cette vue affiche les images des différents layouts disponibles.
La fonction switchLayout() modifie la Main Grid selon le layout choisi.
A noter que le changement de layout efface les éventuelles photos déjà chargées sur un bouton.

Notes relatives à SwiftUI:

Décomposition en sous vues:

Décomposer les vues en différentes sous vues est une bonne façon d'organiser son code en SwiftUI permettant de les réutiliser, de limiter la responsabilité de chacune, et ainsi de favoriser la scalabilité et le debogage.
Nous avons donc isolé les différents composants de notre écran pour les coder séparément et les assembler ensuite.
Ceci nous permet en l'occurence de réécrire notre code pour le mode paysage dans la mesure où la disposition des éléments selon l'orientation est un choix arbitraire.

Orientation du Device:

Pour gérer l'orientation de l'appareil, nous avons utilisé UIKit et la classe UIDevice.

```
UIDevice.current.orientation
```

Nous avons ensuite crée la classe OrientationInfo conforme au protocole ObservableObject, ce qui nous permet de publier la variable « orientation » et de la placer dans l'environnement, la rendant accessible à toutes nos sous vues.

```
@EnvironmentObject var device: OrientationInfo
```

Nous avons pu ainsi programmer l'affichage de chaque sous vue selon l'orientation.
Concernant notre vue mère (ContentView), et après avoir essayé différentes approches, nous avons opté pour la réécriture du code pour le mode Paysage, qui s'avère finalement la plus optimisée en terme de volume de code, mais aussi la plus lisible.

```
if device.orientation == .portrait {
} else if device.orientation == .landscape {
```

Image Picker:

Il n'existe pas à ce jour d'objet SwiftUI natif pour accéder au ImagePickerController tel qu'on le connait avec UIKit.
Selon la documentation Apple: « Use a UIViewControllerRepresentable instance to create and manage a UIViewController object in your SwiftUI interface ».

Nous avons donc crée une structure ImagePicker qui se conforme au protocole UIViewControllerRepresentable puis la classe Coordinator qui nous permet de faire le pont entre UIKit et SwiftUI et de récupérer l'image.

Rect Getter:

Cette structure nous permet de générer une View rectangulaire dont la taille est définie par GeometryReader.
Le ViewModifier .background nous permet de placer le rectangle derrière la MainGridView et ainsi de générer une image sur la base de celle ci.

```
.background(RectGetter(size: $imageSize))
```