Lien du projet GitHub : https://github.com/atalantes-hh/Instagrid

Cahier des charges du projet

❖ *Contexte :*

Une agence de design a besoin d'un développeur iOS pour concevoir l'application de son client InstaApps. L'agence a créé le design de l'application Instagrid.

Celle-ci fournit le design de l'application au format Adobe XD cela comprend notamment le design de l'icône, les polices de l'application et sa mise en page aux vues portrait et paysage.
L'application permet de combiner plusieurs photos dans un format carré et de partager le résultat avec ses amis via ses applications préférées à l'aide d'un Swipe de l'utilisateur.

Il faudra aussi respecter le panel des couleurs qui a été défini dans Adobe XD.

❖ *Contraintes techniques :*

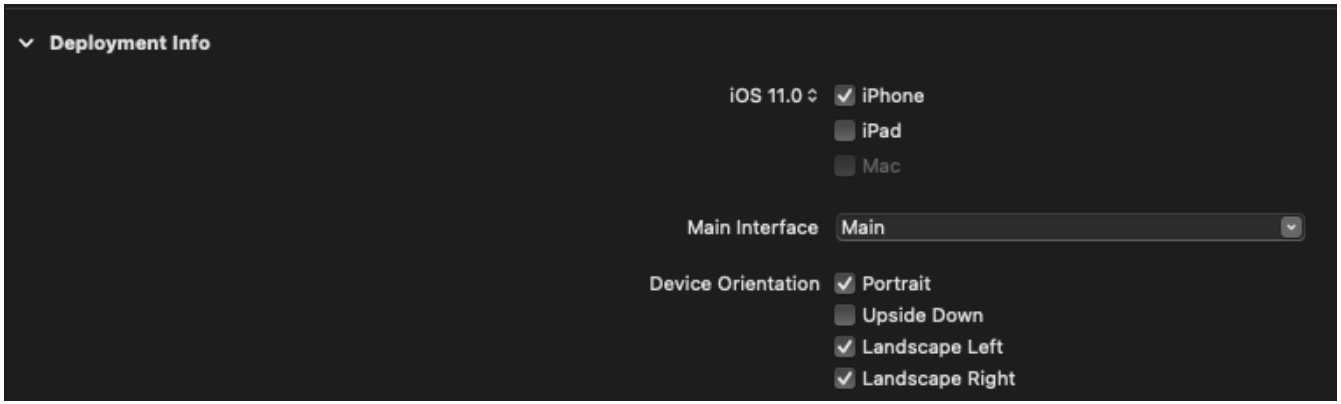
Le client final veut le respect des contraintes suivantes :

- Le langage utilisé doit être Swift 4 ou supérieur :

  ✓ Utilisation de la version Swift 5.3

- L'application doit être disponible à partir d'iOS 11.0
- L'application n'a pas à être disponible sur iPad
- L'application supporte l'orientation Portrait et Paysage

  ✓ Dans **General** / **Deployment Info** cela a été spécifié :

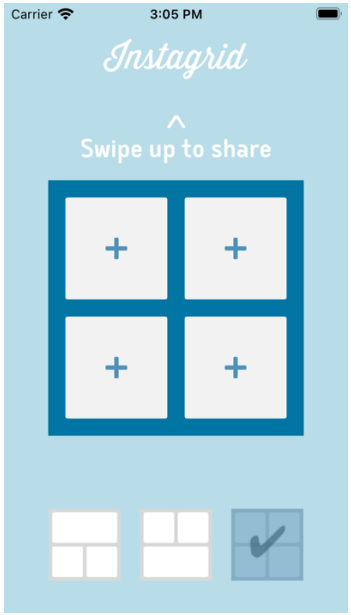


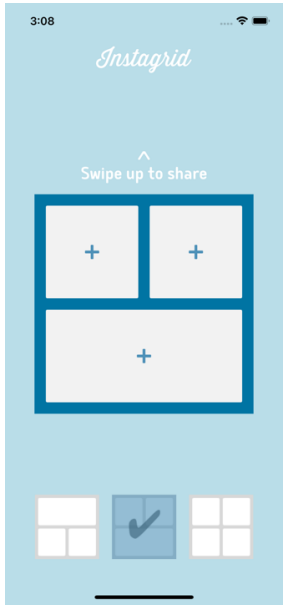
- L'application est supportée par toutes les tailles d'iPhone (de l'iPhone SE à l'iPhone XS Max)

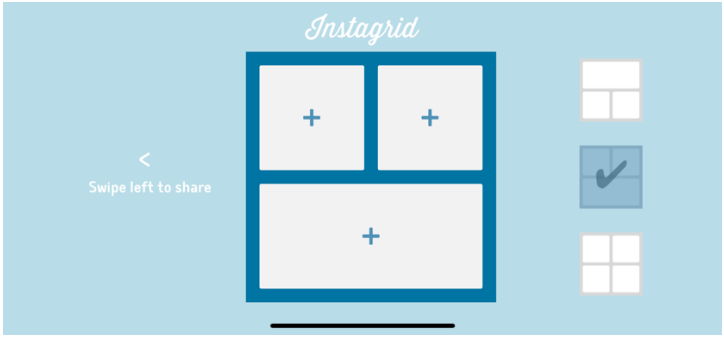
  ✓ Plus précisément et pour tenir compte des derniers modèles de l'iPhone SE (4'') à l'iPhone 12 Pro Max (6.7'')

  Soit un panel de 8 tailles.

  Ci-dessous divers exemples en mode Portrait et Paysage :







Architecture de l'application

L'application s'appuie sur le design pattern MVC à l'exception que nous n'avons pas de « Model » ici.

La raison est simple nous n'avons pas de données brutes, si nous avions un accès à une base de données ou des accès réseau cela aurait été implanté dans le Model or ici ce n'est pas le cas.

La vue c'est notre interface avec laquelle l'utilisateur va interagir.

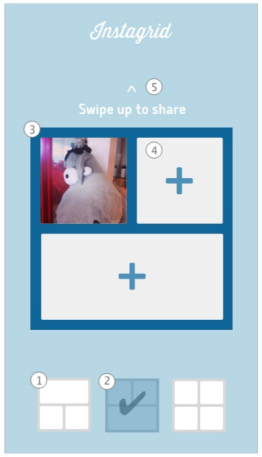
Elle va contenir les UIVIew, UILabel, tout ce qui concerne les éléments graphiques.

Le contrôleur lui apporte les modifications à la vue, par exemple le changement des photos, ou le comportement de l'application au changement d'orientation.
Exemple : Le Swipe

Dans le cas d'un MVC complet il va gérer l'interaction entre la vue et le modèle. C'est lui qui permet la communication entre la Vue et le Model. Lorsque les données changent, il va recevoir un événement du Modèle et envoyer un événement à la Vue.

Le Modèle

La vue



Le Layout 2 est sélectionné : @IBOutlet layout2 ②
On a un changement du Layout 1 ① vers le 2 et l'image du bouton passe en selected @IBAction selectedLayout2. ② Ce qui qui change la disposition de la Grid @IBOutlet gridPicture ③ via activeDisplay

La case topRight ④ est cliqué ce qui actionne le IBAction newPictureTopRight
Le contrôleur active loadPicture qui va charger imagePickerController

La vue renvoie l'information pour la sélection ou l'absence d'image
Le contrôleur assigne ou non l'image si l'utilisateur a quitté

Lorsque la grille est complète, l'utilisateur va réaliser un Swipe ⑤ pour la partager
Si celle-ci est incomplète alors il y aura une alerte pour l'inviter à compléter la grille. Dans le cas contraire l'animation se produit et la vue est transformée en UIImage ③ via gridImage et l'activity Controller se déclenche

La vue renvoie l'information de ce qui a été effectué par l'utilisateur image envoyée ou quittée
Le contrôleur exécute le reverseGridAnimation et la vue revient à sa place d'origine

Le contrôleur

- @IBOutlet layout1
- @IBOutlet layout2

- @IBAction selectedLayout2
- activeDisplay

\# viewDidLoad

- @IBOutlet gridPicture
- @IBAction newPictureTopRight

- loadPicture
\# imagePickerController

- @objc disposition

- @objc didSwipe
- checkPicture
- gridAnimation
- share
- gridImage

- reverseGridAnimation

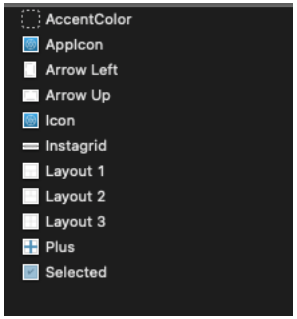
Glossaire

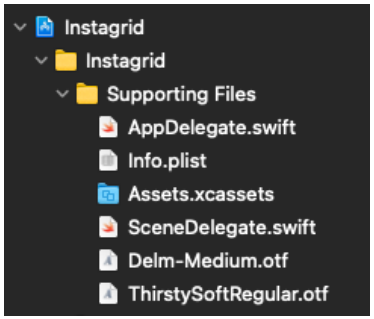
- : Private
\# : Internal
→ Vue vers Contrôleur
← Contrôleur vers vue

❖ *Design du projet :*

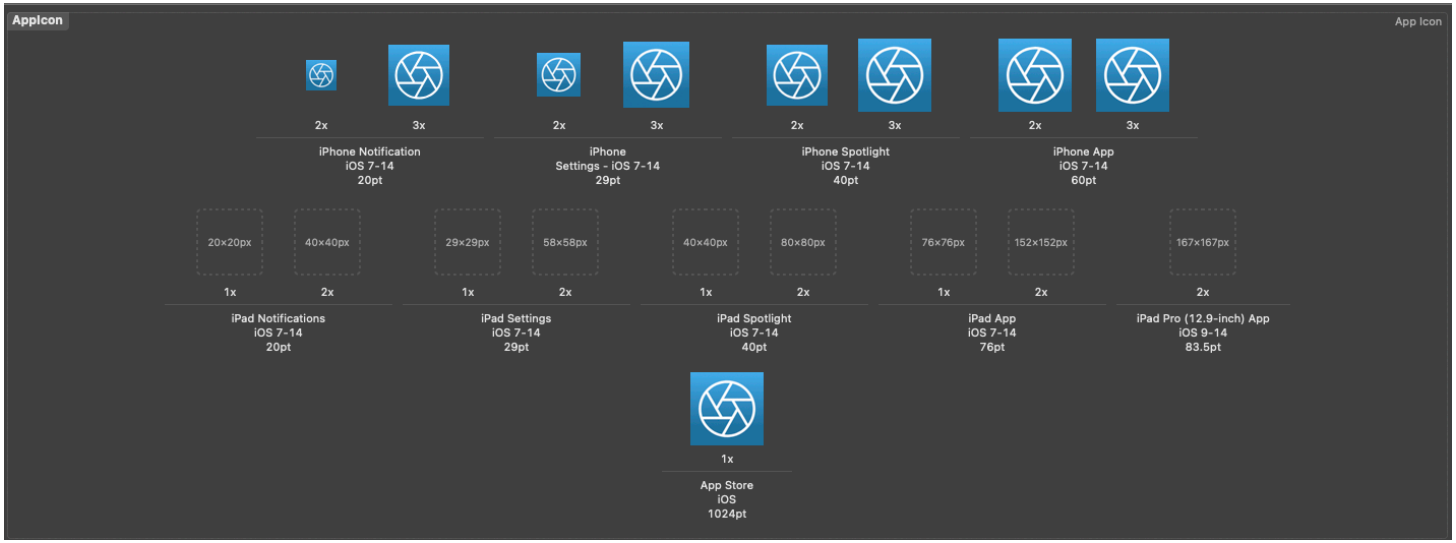
L'implémentation du projet passe par le respect du design demandé.

Pour cela on a donc utilisé les différents visuels fournis et qui ont été placés dans le dossier Assets.xcassets qui est prévu pour cet usage, les Polices sont dans le dossier Supporting Files :

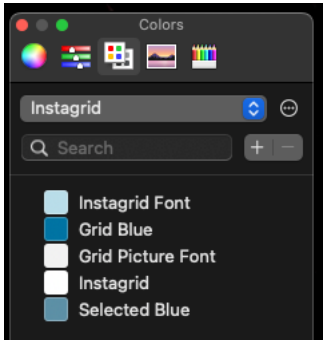




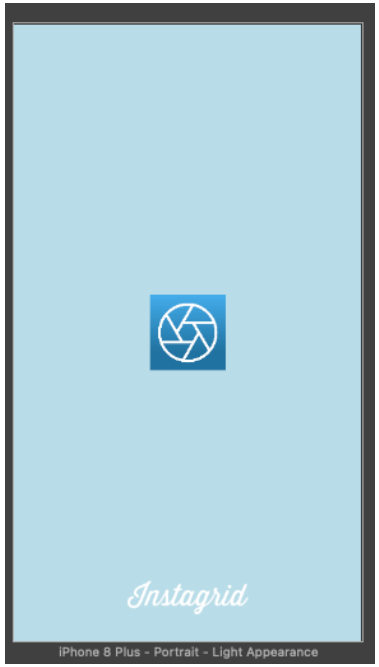
Par la même occasion on a ajouté les tailles AppIcon manquantes pour l'iPhone.



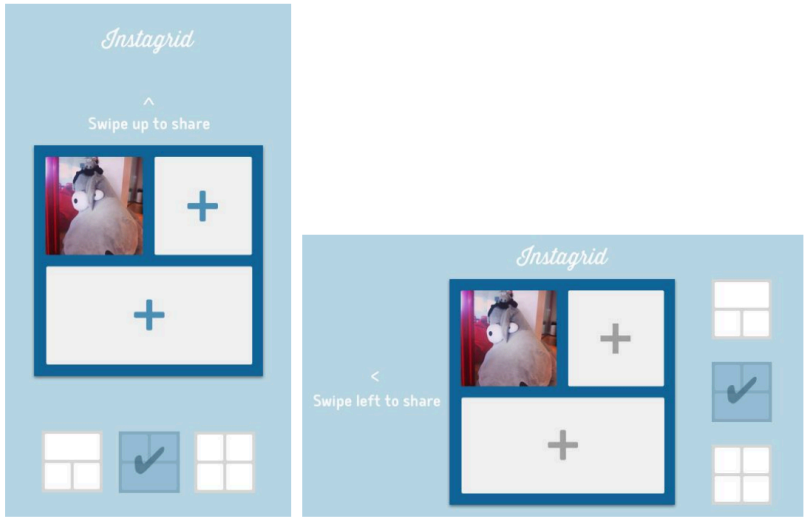
Le panel de couleur utilisé respecte le code couleur de l'application.



Celui-ci a permis la construction de la page de Lancement (LaunchScreen) qui reprend l'icône et le nom de l'application

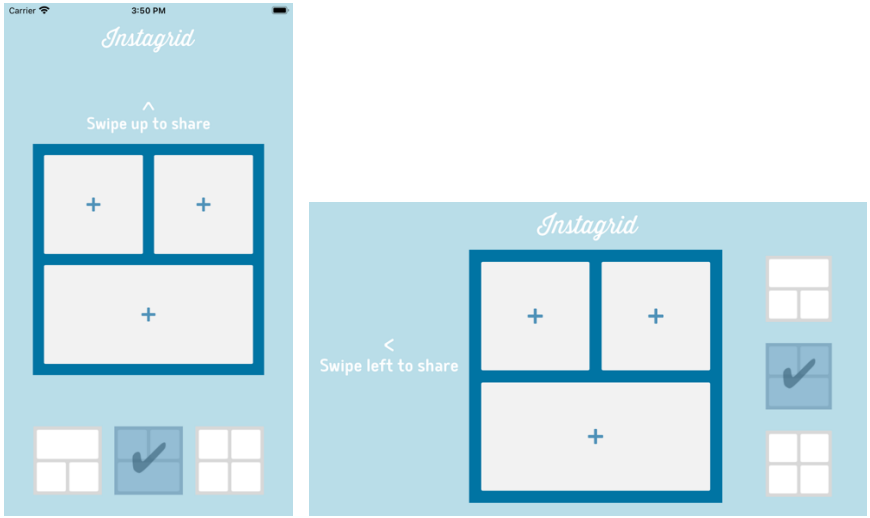


Concernant la page de l'application, le design demandé était le suivant :



Vue portrait et vue paysage

Le résultat est donc celui-ci :



❖ *Code du projet :*

Chaque élément important de l'application est lié à son équivalent en code Swift.

On utilise des IBOutlet pour créer une connexion entre l'objet de la vue et le contrôleur.

On a donc une liste pour les différents objets :

```
// MARK: - @IBOutlet

//  Buttons for layout
@IBOutlet private weak var layout1: UIButton!
@IBOutlet private weak var layout2: UIButton!
@IBOutlet private weak var layout3: UIButton!

// All views & Buttons for Grid and Subviews
@IBOutlet private weak var gridPicture: UIView!
@IBOutlet private weak var topLeftView: UIButton!
@IBOutlet private weak var topRightView: UIButton!
@IBOutlet private weak var bottomLeftView: UIButton!
@IBOutlet private weak var bottomRightView: UIButton!

// Main Title
@IBOutlet private weak var instagridLabel: UILabel!

// Share Display : Label & Arrow
@IBOutlet private weak var swipeLabel: UILabel!
@IBOutlet private weak var arrow: UIImageView!
```

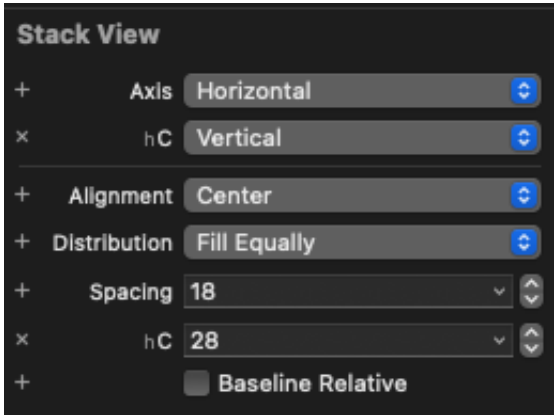
- *Le Layout :*

Le premier élément qui va permettre de choisir la disposition est lié au sélecteur de disposition qu'on nomme le Layout.

Dans la Vue, les boutons sont inclus dans une UIStackView. Celle-ci va contenir les 3 boutons, la StackView nous donne un côté modulable comme on le verra sur la partie Grid.

Exemple : si un jour on décide de rajouter un 4ème Layout on pourra le faire facilement

On garde les mêmes critères de disposition grâce aux paramètres suivants :
Alignement pour la manière dont la StackView va occuper l'espace par à son Axe ici Center pour Centrer
Distribution pour définir de quelle manière on effectue l'arrangement des vues, ici Fill Equally pour Remplir de manière égale.
Et enfin Spacing pour l'espace entre les différents boutons.



Côté contrôleur on a utilisé une énumération des dispositions et une fonction pour effectuer les réglages sur chaque Layout.

```swift
// Enumeration for the layout Display buttons
enum Display {
    case layoutT, layoutReverseT, fourSquare
}
```

La fonction va à la fois permettre de marquer le Layout comme actif en ajoutant l'image Selected mais aussi de définir le comportement des autres Layout et la disposition de la Grid via ses vues.

```swift
// Display Layout Selection and Apply to Grid
private func currentLayout(_ layoutDisplay: Display) {
    switch layoutDisplay {
    case .layoutT:
        layout1.setImage(UIImage(named: "Selected"), for: .normal)
        layout2.setImage(nil, for: .normal)
        layout3.setImage(nil, for: .normal)
        topRightView.isHidden = true
        topLeftView.isHidden = false
        bottomRightView.isHidden = false
        bottomLeftView.isHidden = false
    case .layoutReverseT:
        layout2.setImage(UIImage(named: "Selected"), for: .normal)
        layout1.setImage(nil, for: .normal)
        layout3.setImage(nil, for: .normal)
        bottomRightView.isHidden = true
        topRightView.isHidden = false
        topLeftView.isHidden = false
        bottomLeftView.isHidden = false
    case .fourSquare:
        layout3.setImage(UIImage(named: "Selected"), for: .normal)
        layout1.setImage(nil, for: .normal)
        layout2.setImage(nil, for: .normal)
        topRightView.isHidden = false
        topLeftView.isHidden = false
        bottomRightView.isHidden = false
        bottomLeftView.isHidden = false
    }
}
```

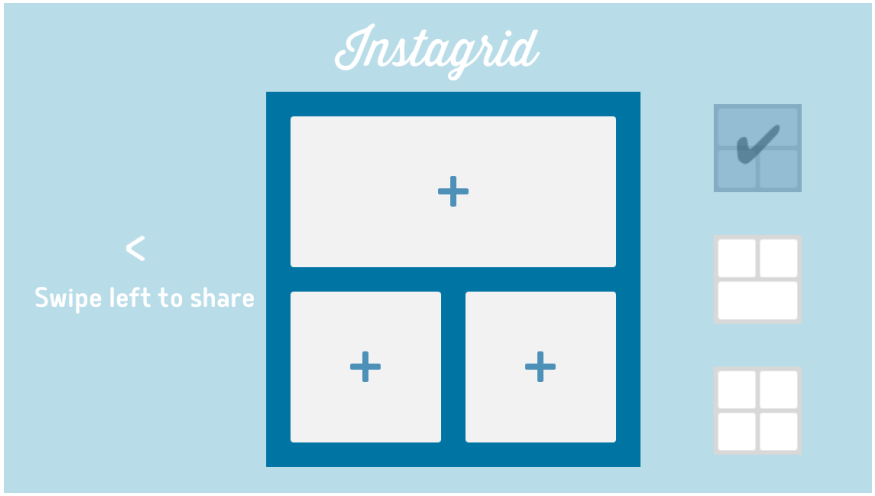
Pour ce faire ils sont reliés avec des IBAction, qui eux permettent la connexion entre l'élément de l'interface et le code qui lui est associé et sont activé via un sender (qui va initialiser la requête)

```
// @IBAction Buttons for selected Layout
@IBAction private func selectedLayout1(_ sender: Any) {
    activeDisplay = .layoutT
}

@IBAction private func selectedLayout2(_ sender: Any) {
    activeDisplay = .layoutReverseT
}

@IBAction private func selectedLayout3(_ sender: Any) {
    activeDisplay = .fourSquare
}
```

Soit un résultat de ce type ici pour le Layout3 :



- *La Grid :*

La Grid repose elle aussi sur des StackView, on a un ensemble de 3 StackView. On a une StackView pour les vues du haut et une autre pour les vues du bas qui sont en axe horizontal et la StackView principale qui va les englober est en mode Vertical.

Chaque vue de la Grid est présentée par un Plus qui disparait lors de l'ajout d'une image. Les emplacements ont été conçus comme des UIButtons pour plus de simplicité dont on vient remplacer l'image lors de l'accès à la photothèque.

```
// Access and choose image in Photo Library
private func loadPicture() {
    if UIImagePickerController.isSourceTypeAvailable(.photoLibrary) {
        let imagePickerController = UIImagePickerController()
        imagePickerController.delegate = self
        imagePickerController.sourceType = .photoLibrary
        self.present(imagePickerController, animated: true, completion: nil)
    }
}
```

Ici lors de l'application de la nouvelle image (newPicture) celle-ci va remplacer l'image du bouton et être disposé de manière adéquate avec le scaleAspectFill.

```
// Picking image with loadPicture and apply to choosen view
func imagePickerController(_ picker: UIImagePickerController,
                           didFinishPickingMediaWithInfo info: [UIImagePickerController.InfoKey: Any] ) {
    let newPicture = info[UIImagePickerController.InfoKey.originalImage] as? UIImage
    switch isView {
    case .topLeftView:
        topLeftView.setImage(newPicture, for: .normal)
        topLeftView.imageView?.contentMode = .scaleAspectFill
        topLeftView.imageView?.layer.cornerRadius = 2
```

Si l'utilisateur clique de nouveau sur la nouvelle image, il peut choisir une nouvelle image.

Et lors du changement de disposition, l'image va s'adapter convenablement à la vue grâce au scaleAspectFill.

Là aussi nous avons utilisé des IBAction pour chaque Vue :

- *Extrait :*

```
// @IBAction to Add pictures to each view
@IBAction private func newPictureTopLeft() {
    loadPicture()
    isView = .topLeftView
}

@IBAction private func newPictureTopRight() {
    loadPicture()
    isView = .topRightView
}
```

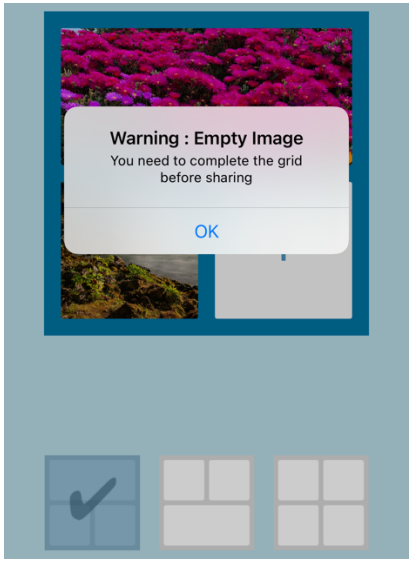
! Afin d'éviter l'envoie d'une grille incomplète, on a une fonction qui vient vérifier si les images ont bien été mise dans la grille selon la disposition qui est active à ce moment là.

```
// Image Completion Checking if view is Hidden ignore PopUp Alert
private func checkPicture() -> Bool {
    // checking image data if pngData is not equal to Plus image the function is ignore
    guard let plusImage = UIImage(named: "Plus")?.pngData() else { return false }

    if topLeftView.isHidden == false, topLeftView.image(for: .normal)?.pngData() == plusImage {
        showAlertPopUp()
        return false
    }
```

Dans un premier temps on vient vérifier les données de l'image à l'aide de Guard si celles-ci sont différentes de celles de l'image Plus qui est définie, on sort de la fonction car l'image a bien été remplacée.

Si la vue n'est pas inactive on vient le vérifier aussi et on envoie une alerte à l'utilisateur pour le prévenir que la grille n'est pas complète, dans le cas contraire on sort de la fonction.



Ce message d'alerte apparait uniquement lors du Swipe de l'utilisateur

- *Le Swipe :*

Au moment du Swipe l'application vérifie donc si les conditions du Check Picture sont respectées si c'est le cas l'animation se déclenche et l'action Share par la même occasion.

```swift
// Swipe when action recognized & Add Methods to checkEmptyPicture
@objc private func didSwipe(_ sender: UISwipeGestureRecognizer) {
    guard checkPicture() else { return}
    gridAnimation()
    if sender.state == .recognized {
        share()
    }
}
```

On a un @objec qui est un héritage du langage Objective – C.

La fonction share est composée de 2 actions, tout d'abord nous avons un appel à la fonction gridImage qui nous permet la conversion d'une UIView en UIImage.

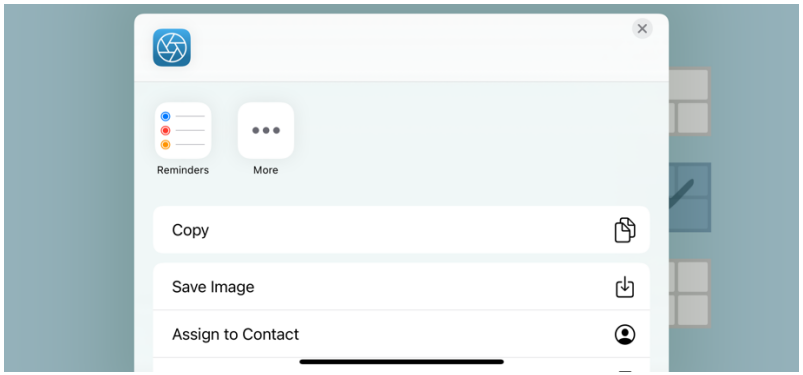
Puis ensuite on présente le ActivityViewController pour le partage de l'image que l'on vient de créer.

```swift
// Conversion UIView to UIImage
private func gridImage(with view: UIView) -> UIImage {
    let renderer = UIGraphicsImageRenderer(size: view.bounds.size)
    let image = renderer.image { _ in
        view.drawHierarchy(in: view.bounds, afterScreenUpdates: true)
    }
    return image
}

// Share picture after Conversion with gridImage
private func share() {
    let imageToShare = gridImage(with: gridPicture)
    let activityViewController = UIActivityViewController(activityItems: [imageToShare], applicationActivities: nil)
    activityViewController.popoverPresentationController?.sourceView = self.view
    self.present(activityViewController, animated: true, completion: nil)
    activityViewController.completionWithItemsHandler = {(_, _, _, _) in
        self.reverseGridAnimation()
    }
}
```

! Dans le cas où l'utilisateur venait à retourner à la Grid pour quelque raison que ce soit on a le completionWithItemHandler qui vient répondre à ce problème et effectuer une action inverse, pour le retour à la grille de départ.

Exemple de l' ActivityViewController :



- *Les Animations :*

On a procédé à l'ajout des animations lors du Swipe.
Une animation lors de l'envoi de la Grid pour les modes Landscape ou Portrait dont l'orientation sera différente lors du Swipe.
L'animation va déplacer la vue vers le haut ou le côté à l'aide du Swipe, en même temps que celle-ci sera réduite dans sa dimension pour donner un effet plus visuel.

```swift
// Lauch animation when Share to each mode of orientation
private func gridAnimation() {
    // Reduction of the grid
    let minusGrid = CGAffineTransform(scaleX: 0.4, y: 0.4)
    if UIDevice.current.orientation == .landscapeRight || UIDevice.current.orientation == .landscapeLeft {
        // Combine minusGrid and swipe translation
        let swipeLeftAnimation = CGAffineTransform(translationX: -self.view.frame.width, y: 0)
        UIView.animate(withDuration: 0.5, delay: 0, options: [], animations: {
            self.gridPicture.transform = minusGrid.concatenating(swipeLeftAnimation)
        })
        arrow.isHidden = true
        swipeLabel.isHidden = true
    } else {
        let swipeUpAnimation = CGAffineTransform(translationX: 0, y: -self.view.frame.height)
        UIView.animate(withDuration: 0.5, delay: 0, options: [], animations: {
            self.gridPicture.transform = minusGrid.concatenating(swipeUpAnimation)
        })
        arrow.isHidden = true
        swipeLabel.isHidden = true
    }
}
```

Et la seconde animation se produit lors du retour à la Grid après l'envoi ou lors du Dismissed de l'utilisateur.
Notre UIViewActivity qui va donc permettre un retour de la grille à sa position de départ grâce à la propriété .identity

```swift
//  Reverse Animation when end Share
private func reverseGridAnimation() {
    UIView.animate(withDuration: 0.3, delay: 0, animations: {
        self.gridPicture.transform = .identity
    })
    arrow.isHidden = false
    swipeLabel.isHidden = false
}
```

➢ *Exemples sur les 8 tailles d'écrans :*