Projet 4: Développez Instagrid

Introduction

Le 4ème projet a pour but de continuer le développement d'une application dont le cahier des charges et le design sont déjà établis, cette application est Instagrid.

Instagrid est un service qui permet de combiner plusieurs photos ensembles avant de pouvoir partager sa création avec son entourage. Il est disponible sur iPhone (à partir d'iOS 11.0) et accessible depuis les modes portrait et paysage.

Vous pouvez retrouver le projet sur mon espace GitHub via ce lien (https://github.com/ClaireRimel/Projet-4-Instagrid)

Dans ce document, je vous présenterai l'architecture MVC, celle de l'application et pour terminer, je vous commenterai le bonus.

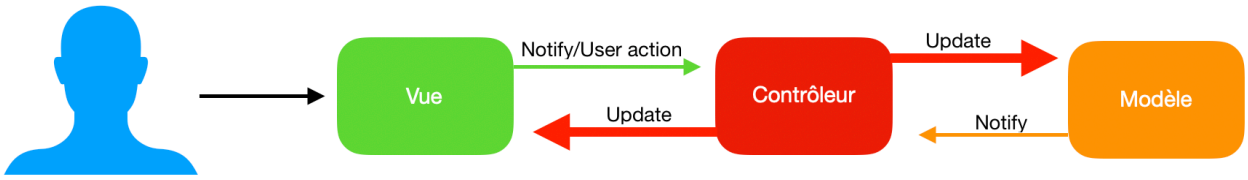
1- Architecture MVC

L'architecture MVC (Modèle Vue Contrôleur), concerne l'architecture globale d'un projet, il classe les objets en fonction de leurs rôles.

Modèle: il contient les données de l'application et définit la logique permettant de les manipuler.

Vue: il gère l'affichage. C'est ce que l'utilisateur voit de l'application, et d'où il pourra modifier les données du Modèle

Contrôleur: il sert d'intermédiaire entre les objets de vue et les objets de modèle, on y implémente la logique servant à réagir aux actions de l'utilisateur.



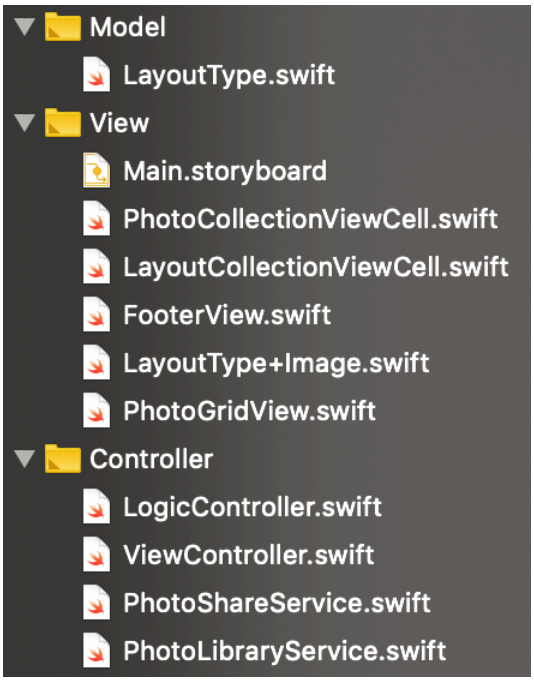
Le modèle MVC favorise la clarté du code et facilite la maintenance de l'application. Effectivement, il est rendu plus facile de localiser la partie de code à modifier, et permet par exemple, à une équipe back-end et front-end de travailler indépendamment et simultanément sur un même projet.

Cependant les applications répondent à des caractéristiques différentes, il se peut que MVC ne satisfasse pas les besoins des développeurs. C'est pourquoi, il existe différentes architectures telles que MVVM (Modèle, Vue, Vue-Modèle), MVP (Model, View, Presenter) et VIPER (View, Interactor, Presenter, Entity, Router).

2- L’architecture de l’application

Pour le développement d’Instagrid, j’ai choisi l’architecture MVC, car elle est mise en avant dans les cours.

Voici comment se présente l’architecture de mon projet :



Modèle

LayoutType.swift:
Nous définissons un type d’énumération, dont les cas décrivent toutes les dispositions de photos possibles offertes par l’application. De par sa nature, il nous permet de créer librement de nouvelles dispositions personnalisées en plus de celles définies dans la spécification du projet.

View

Main.storyboard:
Dans ce fichier, j’ai intégré les composants de l’application et les paramètres permettant à l’interface d’être responsive.

PhotoCollectionViewCell.swift:
PhotoCollectionViewCell est une sous classe de UICollectionViewCell, il représente une image dans la grille de photos affichée à l’utilisateur, et permet de définir le “content mode” des images de la photoImageView.

LayoutCollectionViewCell.swift:
LayoutCollectionViewCell est une sous classe de UICollectionViewCell, il représente un cas LayoutType, en affichant l'image du layout correspondant. Il affiche également une image au dessus du layout sélectionné par l'utilisateur.

FooterView.Swift:
C'est une sous classe de UIView, le FooterView permet de présenter les différents layoutTypes, Il adapte le sens du scroll permettant de présenter les layouts.

LayoutType+Image.swift:
Ce fichier permet d'introduit dans le dossier View, une extension de LayoutType. Cette extension est en charge d'attribuer une image aux différents cas de LayoutType.

PhotoGridView.swift:
Sous classe de UIView, c'est dans cette partie que nous contrôlons l'affichage du PhotoGrid.
On l'initialise avec le cas "oneTopTwoBottom", et instaure la "croix" en image par défaut.
On indique qu'il recevra une image de type UIImage.
PhotoGridView déléguera l'information qu'une cellule a été touchée.
Et l'extension du UICollectionViewDelegateFlowLayout, défini les bordures du PhotoGrid.

Contrôleur

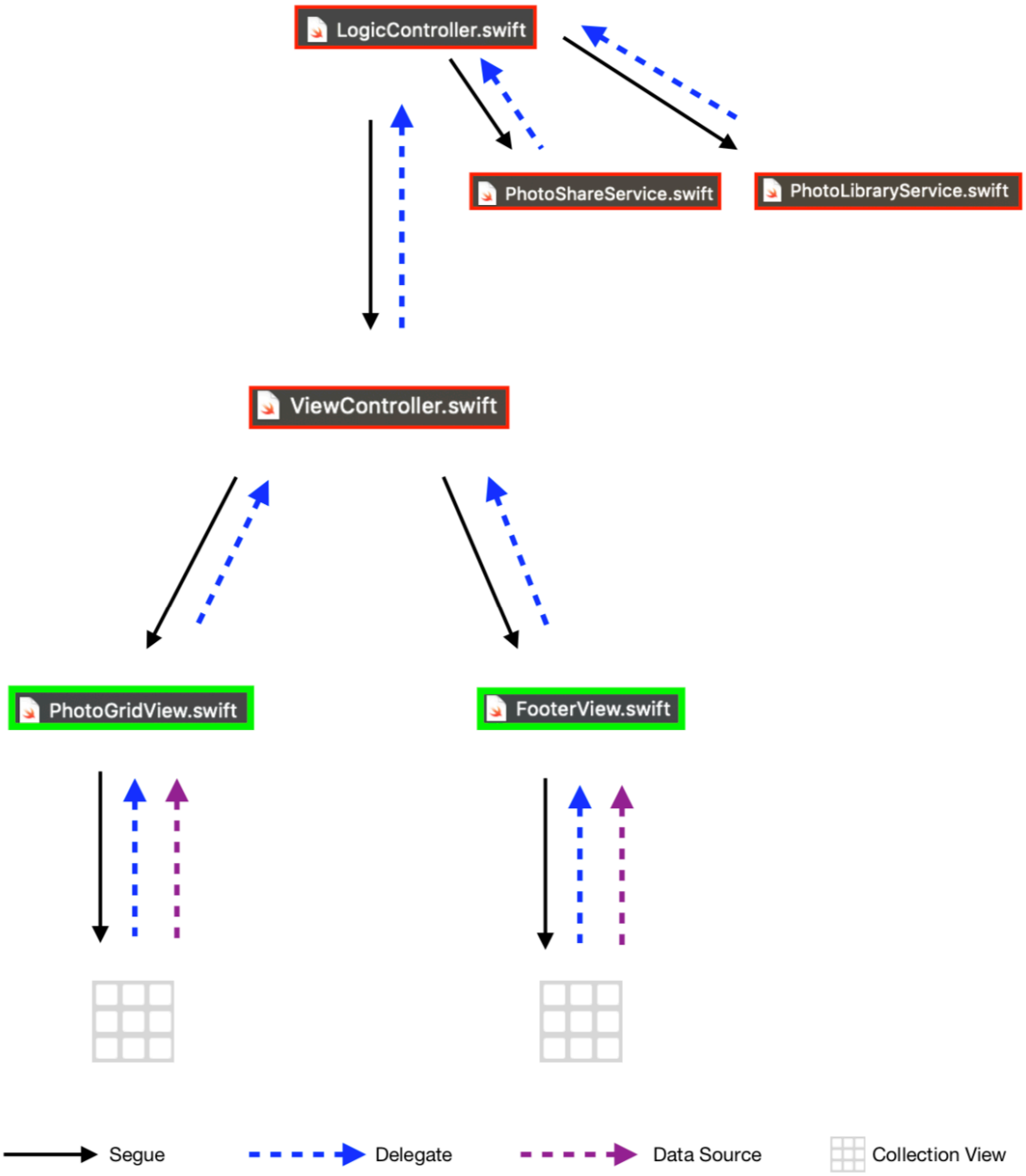
LogicController.swift:
A pour rôle, de faire la liaison entre le ViewController et les services (PhotoShare et PhotoLibrary)

ViewController.swift:
Détecte les changements d'orientation d'écran, afin d'y adapter le sens du swipe.
Détecte le swipe grâce au UISwipeGestureRecognizer.
Réalise l'animation.
Fait apparaître le message de type UIAlert, lorsque l'accès à la bibliothèque est refusé.
Il reçoit les delegates de PhotoGridView et FooterView.

PhotoShareService.swift:
Crée un screenshoot du PhotoGridView.
Fait apparaître les options de partage grâce à l'UIActivityViewController.

PhotoLibraryService.swift:
Initialise la source d'où, nous voulons récupèrer une image.
Contrôle les statuts d'autorisations d'accès à la bibliothèque d'images.
Si le statut d'autorisation est initialisé en "Autorisé", alors on affiche la bibliothèque, sinon on fait appel à la fonction "photoLibraryServiceAuthorizationDenied".
Indique l'image choisie par l'utilisateur grâce à l'UIImagePickerControllerDelegate

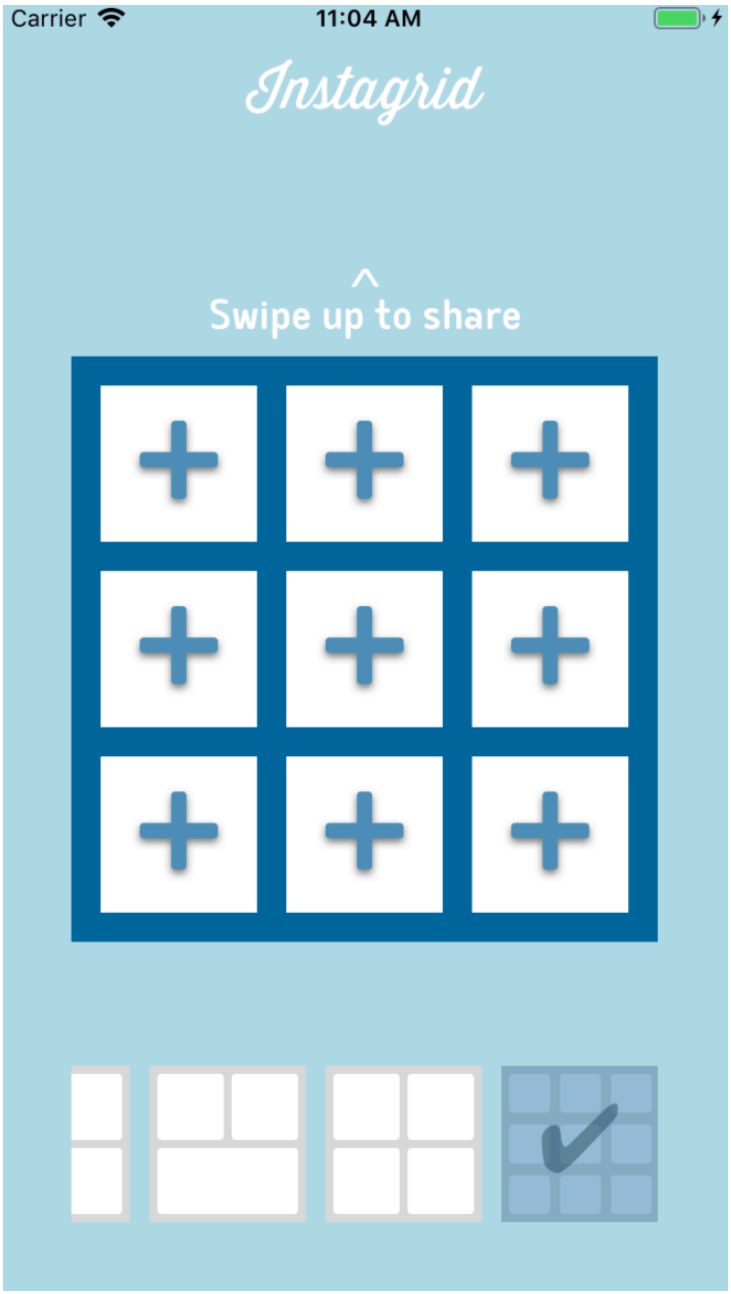
Voici comment le schéma de communication de l'application:



3 - Le Bonus

Le bonus choisi est le layout supplémentaire, grâce aux collections views, il était simple de choisir cette option, car avec l'énumération du LayoutType.swift, la possibilité d'ajouter des sections et des rangées, est infinie.

C'est dans le but de vous montrer cette possibilité, que j'ai choisi de créer un layout supplémentaire, avec une troisième section et une troisième rangée.



Conclusion

Ce projet est incroyable en terme de connaissance à acquérir, j'ai pris mon temps afin de comprendre comment le développer.

L'architecture MVC, m'a beaucoup aidé dans l'organisation, car elle m'a permise d'être attentive à chaque fonctionnalité de l'application et ainsi de me poser les bonnes questions.

Les collections views, les delegates ont été les deux notions importantes lors de mon apprentissage, puisque ceux sont des éléments très puissants et complexes.

j'ai aussi beaucoup appris sur les possibilités de gestures avec les UIGestureRecognizer, les contrôles d'accès pour la bibliothèque de photo avec l'UIImagePickerController et les possibilités de partage grâce à l'UIActivityViewController

Les concepts avec lesquels je me suis retrouvée en grande difficulté, ont été les animations et la réalisation des bordures du photoGrid, pour y faire face j'ai reçu l'aide et les conseils de mon mentor d'OpenClassroom ainsi que ceux de mon compagnon qui est lui même iOS developper.

C'est un application complète qui demande à être attentif sur de nombreux points, tels que l'adaptation des différentes tailles écrans et orientations, le rendu de l'image partagée, l'effet de l'animation du swipe et bien d'autres.

Sources

Access Photo Library:
https://medium.com/@deepakrajmurugesan/swift-access-ios-camera-photo-library-video-and-file-from-user-device-6a7fd66beca2
https://hackernoon.com/swift-access-ios-camera-and-photo-library-dc1dbe0cdd76

Auto Layout:
https://www.raywenderlich.com/7478-beginning-auto-layout
https://www.raywenderlich.com/4260-mastering-auto-layout
https://openclassrooms.com/fr/courses/4570776-creez-une-application-responsive-pour-ios

CaseIterable Protocol:
https://developer.apple.com/documentation/swift/caseiterable

Closure:
http://fuckingclosuresyntax.com
https://learnappmaking.com/closures-swift-how-to/

CollectionView:
https://www.raywenderlich.com/6308-beginning-collection-views

MVC
https://openclassrooms.com/fr/courses/4504796-developpez-une-application-iphone-avec-le-modele-mvc

Open Settings:
https://stackoverflow.com/questions/28152526/how-do-i-open-phone-settings-when-a-button-is-clicked#28152624

PHPhotoLibrary:
https://developer.apple.com/documentation/photokit/phphotolibrary

Render a UIView:
https://www.hackingwithswift.com/example-code/media/how-to-render-a-uiview-to-a-uiimage

UIGestureReconizer
https://developer.apple.com/documentation/uikit/uigesturerecognizer
https://www.raywenderlich.com/9225-gesture-recognizers-in-ios

UIGestureReconizerDelegate
https://developer.apple.com/documentation/uikit/uigesturerecognizerdelegate