PAYAN Mickaël

Projet 4 : Instagrid, une application de montage photo.

- ## Orientation

```
28      private var phoneOrientation: UIDeviceOrientation {
29          get {
30              return UIDevice.current.orientation
31          }
32      }
33
34      override func viewWillLayoutSubviews() {
35          if phoneOrientation.isPortrait == true {
36              swipeToShareLabel.text = "Swipe up to share"
37              arrowSwipeToShareLabel.transform = .identity
38              swipeGesture.direction = .up
39          } else {
40              swipeToShareLabel.text = "Swipe left to share"
41              arrowSwipeToShareLabel.transform = CGAffineTransform(rotationAngle:
                    -CGFloat.pi / 2)
42              swipeGesture.direction = .left
43          }
44      }
```

Afin de définir l'orientation de l'iPhone, un propriété « phoneOrientation » de type « UIDeviceOrientation » (l.28), implémenter d'une propriété calculée « get » (l.29) qui va retourner l'orientation (.orientation) actuelle (.current) de l'appareil (UIDevice, l.30).

De plus, un override de la méthode « viewWillLayoutSubviews » (l.34) vient ajuster la position des sous-vues lorsque que les limites d'une vue changent. Celle-ci est implémenter d'une condition, si l'orientation du téléphone est en mode portrait et est strictement vraie (phoneOrientation.isPortrait == true, l.35), alors le texte est/devient « Swipe up to share » (l.36). La flèche de direction du swipe revient à sa position initiale (vers le haut ^) grâce à la propriété « .identity » de type « CGAffineTransform » (l.37). Enfin, orienter la direction du « swipeGesture » de type « UISwipeGestureRecognizer » (l.38) et qui va reconnaître le balayage de l'utilisateur.

Pourquoi utiliser un « .identity » ? Car lorsque que la condition passe dans le « else », la flèche va réaliser une rotation négative de -90° (soit vers la gauche <, l.41), afin de s'adapter à l'orientation paysage de l'iPhone.

Le texte est lui aussi est changé « Swipe left to share » (l.40) ainsi que la direction du « swipeGesture » (l.42).

- ## Grille centrale et dispositions

Pour faire un rappel du cahier des charges, les photos sont organisées selon une disposition que l'utilisateur peut choisir. De plus il était demandé que lorsqu'on tape sur l'une d'entre-elles :

- La précédente disposition sélectionnée n'est plus marquée comme sélectionnée.
- La sélection tapée est marquée comme sélectionnée.
- La grille centrale s'adapte en fonction de la nouvelle disposition.

```
20      /// Top left button = gridViewButtons[0] - Top right button = gridViewButtons[1] - Bottom left button =
            gridViewButtons[2] - Bottom right button = gridViewButtons[3]
        @IBOutlet var gridViewButtons: [UIButton]!
22
23      /// The three buttons to change the selection appearance of the "GridView". First button on the left =
            updateGridButtons[0], second button in the middle = updateGridButtons[1] and the third on the right =
            updateGridButtons[2]. The same positions for the property updateGridButtonIsSelectedImageView.
        @IBOutlet var updateGridButtons: [UIButton]!
        @IBOutlet var updateGridButtonIsSelectedImageView: [UIImageView]!

        @IBAction private func changeAppareanceGridView(_ sender: UIButton) {
48          displayTheSelectedButton()
49          gridViewButtons.forEach { $0.isHidden = false }
50          switch sender {
51          case updateGridButtons[0]:
52              gridViewButtons[1].isHidden = true
53          case updateGridButtons[1]:
54              gridViewButtons[3].isHidden = true
55          default:
56              break
57          }
58      }
59
60      private func displayTheSelectedButton() {
61          updateGridButtonIsSelectedImageView.forEach { $0.isHidden = true }
62          if updateGridButtons[0].isTouchInside {
63              updateGridButtonIsSelectedImageView[0].isHidden = false
64          } else if updateGridButtons[1].isTouchInside {
65              updateGridButtonIsSelectedImageView[1].isHidden = false
66          } else if updateGridButtons[2].isTouchInside {
67              updateGridButtonIsSelectedImageView[2].isHidden = false
68          }
69      }
```

La fonction « displayTheSelectedButton » permet de répondre au choix des dispositions. Un IBOutlet Collection « updateGridButtons » de type « UIButton » contient les trois buttons et un autre IBOutlet Collection « updateGridButtonIsSelectedImageView » du type « UIImageView » contient les images marquant la sélection. Afin d'éviter les répétitions « .isHidden = true », une boucle « forEach » (l.61) est utilisée. Par la suite, une condition : si le premier bouton est touché à l'intérieur alors l'image affichant la sélection du bouton n'est pas cachée (…[ ].isHidden = false, l.63, l.65, l.67).

Enfin, IBAction fonction « changeAppareanceGridView », comme son nom l'indique va changer l'apparence de la vue centrale. Tout d'abord, l'appelle de la fonction « displayTheSelectedButton » (l.48) pour gérer les dispositions. De nouveau un « forEach » pour éviter les répétitions dans le code. Et pour finir, une instruction « switch », dans le cas où le premier bouton de sélection des dispositions est sélectionnée (IBOutlet Collection « gridViewButtons », le bouton de la grille correspondant à la disposition est caché (…[ ].isHidden = true, l.52, l.54).

- **Ajout de photos dans la grille centrale**

En tapant sur un bouton plus, l'utilisateur a accès à sa photothèque et peut choisir une des photos de son téléphone. Une fois choisie, celle-ci vient prendre la place de la case correspondant au bouton plus (+) tapé.

En référence au cahier des charges, la photo doit être centrée, les proportions sont maintenues et prendre tout l'espace possible (pas de « blanc »). Si l'utilisateur clique sur la photo dans la grille, sa photothèque s'ouvre de nouveau et peut permuter les photos.

```
72      @IBAction private func selectButtonInGridToChangeImage(_ sender: UIButton) {
73          buttonSelected = sender
74          let picker = UIImagePickerController()
75          picker.allowsEditing = true
76          picker.delegate = self
77          picker.sourceType = .photoLibrary;
78          present(picker, animated: true, completion: nil)
79      }
80
81      func imagePickerController(_ picker: UIImagePickerController,
            didFinishPickingMediaWithInfo info: [UIImagePickerController.InfoKey : Any]) {
82          let image = info[UIImagePickerController.InfoKey.originalImage] as? UIImage
83          self.buttonSelected?.setImage(image, for: .normal)
84          self.buttonSelected?.isSelected = true
85          self.dismiss(animated: true, completion: nil)
        }
```

IBAction fonction « selectButtonInGridToChangeImage » permet à l'utilisateur d'avoir accès à sa photothèque. Premièrement, faire référence au bouton sélectionné (buttonSelected = sender, l.72) qui sera le bouton « + ». Une constante « picker » qui est une instance de « UIImagePickerController », ce qui permet d'avoir accès aux propriétés et méthodes de celle-ci.

- picker.allowsEditing = true (l.74), permet à l'utilisateur de recadrer l'image.
- picker.delegate = self (l.75) , permet de se conformer aux protocoles.
- picker.sourceType = .photoLibrary (l.76), permet d'afficher la photothèque.

Tant qu'à la fonction « imagePickerController » va permettre le contrôle de sélection d'images. Tout d'abord, une constante « image » (l.81) pour extraire l'image du dictionnaire passé en paramètre. Nous ne savons pas que cette valeur existe en tant que UIImage, elle est optionnelle, d'où l'utilisation d'une méthode de conversion tel que « as ? ».

- self.buttonSelected?.setImage(…) (l.82), permet d'accéder aux réglages de l'image du bouton.
- self.buttonSelected?.isSelected = true (l.83), l'image est sélectionnée.
- Self.dismiss(animated: , completion:) (l.84), rejet du contrôleur de vue.

- **Swipe to share**

L'utilisateur peut partager la création qu'il vient de réaliser. Il peut réaliser un glisser vers le haut (en mode portrait) ou vers la gauche (en mode paysage).
Pour cela il est vérifié si la grille est complète, soit qu'il ne manque aucune photo.

```
87      private func checkIfTheGridViewIsComplete() -> Bool {
88          for button in gridViewButtons where !button.isHidden && !button.isSelected {
89              alertPopUp()
90              return false
91          }
92          return true
93      }
94
95      private func alertPopUp() {
96          let alert = UIAlertController(title: "Grid incomplete", message: "Select all
                images for swipe up and share.", preferredStyle: .alert)
97          alert.addAction(UIAlertAction(title: "OK", style: .default, handler: nil))
98          self.present(alert, animated: true)
```

La fonction « checkIfTheGridViewIsComplete » (l.87) qui retourne un booléen, va contrôler si la grille est complète à l'aide d'une boucle « for in ». Celle-ci va venir parcourir le tableau « gridViewButtons » et vérifier si les boutons ne sont pas cachés ni sélectionnés. Si ce n'est pas le cas, la fonction « alertPopUp » (l.95) sera appelée (l.89). Elle affichera un message d'alerte expliquant que la grille est incomplète et qu'il faut sélectionner toutes les images pour glisser et partager. Ensuite, il sera retourné faux (l.90). Dans le cas où la condition est respectée, que les boutons sont cachés et sélectionnés alors il sera retourné vrai (l.92). L'animation pourra être lancée.

La vue principale sera glissée vers le haut ou vers la gauche (selon l'orientation de l'Iphone) jusqu'à la faire disparaître de l'écran.

```
100
101     @IBAction private func swipeAction(_ sender: UISwipeGestureRecognizer) {
102         guard checkIfTheGridViewIsComplete() else { return }
103         switch sender.direction {
104         case .up:
105             self.animatedGridView(x: 0, y: -1000)
106         case .left:
107             self.animatedGridView(x: -1000, y: 0)
108         default:
109             break
110         }
111         shareCompleteGridView()
112     }
113
114     private func animatedGridView(x: CGFloat, y: CGFloat) {
115         UIView.animate(withDuration: 0.6, animations: {
116             self.gridView.transform = CGAffineTransform(translationX: x, y: y)
117         })
118     }
```

Tout d'abord, la fonction « animatedGridView » (l.114) avec pour paramètre d'entrée x, y du type CGFloat (nombres décimaux), permet de réaliser l'animation, avec une durée de 6 secondes (l.115) et une transformation du geste sur la grille centrale en translation x et y (l.116).

Afin de finaliser ce geste, cette IBAction fonction « swipeAction » va d'abord vérifier à l'aide d'un « guard » si la grille centrale remplie bien les conditions sinon retour à la fonction (l.102). Après, une instruction « switch », dans le cas où la direction du geste serait vers le haut ou vers la gauche, la fonction « animatedGridView » fera glisser la vue centrale à -1000 sur l'axe des x ou y.

Une fois l'animation terminée, la fonction « shareCompletGridView » (l.128) est appelée (l.111), implémentée à l'aide d'une instance UIActivityController (l.130) s'affiche et permet à l'utilisateur de choisir son application préférée pour partager sa création.

```
128     private func shareCompleteGridView() {
129         let sharingImage = viewToImage(with: gridView)
130         let activityViewController = UIActivityViewController(activityItems:
                [sharingImage], applicationActivities: nil)
131         present(activityViewController, animated: true)
132         activityViewController.completionWithItemsHandler = { (_, _, _, _) in
133             UIView.animate(withDuration: 0.6, animations: {
134                 self.gridView.transform = .identity
135             })
136         }
137     }
```

Une fois terminée et la grille partagée, l'animation s'inverse avec une durée de 6 secondes et la grille centrale a repris sa position initiale (l.134), prête à être utilisé de nouveau.