Dev mobile swift (cours 1)

Sommaire :

[Dev mobile swift (cours 1) 1](#_Toc130370279)

[M : pour Modèle 2](#_Toc130370280)

[V : pour Vue 2](#_Toc130370281)

[C : pour Contrôleur 2](#_Toc130370282)

[Créer un projet avec Xcode 2](#_Toc130370283)

[Fichier ViewControlller.swift 3](#_Toc130370284)

[Le fichier AppDelegate.swift 4](#_Toc130370285)

[Création d’une interface (Vue) 4](#_Toc130370286)

[Interaction entre la vue et le contrôleur 5](#_Toc130370287)

[Comment réaliser une action 6](#_Toc130370288)

[Ecriture du contrôleur 6](#_Toc130370289)

[Pour le CC : 6](#_Toc130370290)

[Création d’une table View pour afficher des données (cours 2) 6](#_Toc130370291)

[Le table view 7](#_Toc130370292)

[Configuration de l’objet UITableViewCell 8](#_Toc130370293)

[Ecriture de UITableViewController 8](#_Toc130370294)

[Suite de l’écriture du contrôleur TCVMessage 9](#_Toc130370295)

[Dans le cas du style Subtile : 9](#_Toc130370296)

L’archtecture MVC appartient à la famille des Patrons de conception

Ils décrivent des solutions standards face a des problèmes de conception qui se sont déjà produits par le passé

Patrons de conception = capitalisation d’expérience

Exemple :

En php on avait vu fetch() comme patron de conception

L’architecture MVC sépare la logique du code de l’application en 3 parties :

M : pour Modèle

Objectif : encapsule la logique métier et extrait les données brutes provenant de la BD, de fichiers textes.

V : pour Vue

Objectif : afficher et récupérer les données

C’est la couche ou l’on trouve les composants ergonomiques

C : pour Contrôleur

Objectif : gérer toute la logique applicative. Il pilote la vue, en réalité aux actions de l’utilisateur, demande des données au modèle. C’est le chef d’orchestre de l’application.

Model view controller

User 🡪(actions) views 🡪 control 🡪 data

🡨(data) 🡪

On utilise l’utilitaire (IDE)

Application Xcode

Créer un projet avec Xcode

Lancer l’application Xcode puis choisir l’option « create new pXcode project »

Dans le choix du template 🡪 IOS 🡪 (volet droit APP) puis cliquer sur next.

Sélectionner l’option (du nouveau projet) interface : storyboard

Language : Swift

Décocher le reste en dessous

Ensuite ont choisi le répertoire du projet

Les fichiers :

**AppDelegate.swift** : fichier source qui mémorise les méthodes appelées à différents moments du cycle de vie des applications. Il gère le cycle de vie des applications.

**ViewController.swift** : fichier source permettant de contrôler la vue par défaut de l’application. A ce stade du projet, ce fichier est le contrôleur de l’application.

**Main.storyboard** : le fichier d’interface pouvant contenir plusieurs écrans liés entre eux. Il décrit les différentes vues de l’application.

Assets.Xassets : répertoire destiné à stocker des éléments graphiques tels que des images, icones etc…

LancerScree.storyboard : autre fichier d’interface qui permet de dessiner l’écran a afficher, au démarrage de l’application

Info.plist : fichier CML contenant les finformations …

Fichier ViewControlller.swift

* Code du fichier ViewControlleur.swift

On créé une classe en swift avec un héritage **UIViewController**

Avec override func viewDidLoad(){

Super.viewDidLoad()

}

Chaque vue est associé son controlleur

Rappel : override = redéfinir

Le fichier AppDelegate.swift

Cette classe gère le cycle de vie de l’application. Elle utilise le protocole UIApplicationDelegate lui permettre si besoin d’implémenter les méthodes quand :

* L’application :
  + vient de s’ouvrir
  + Est passé en arrière-plan
  + Sur le point de se fermer
  + N’est plus active

**C’est dans ce fichier que l’on va définir les variables globales**

(Visible dans tous les autres contrôleurs)

* La classe AppDelegate permet d’avoir accès à des variables globales (accessibles partout dans le code) grâce à la classe UIApplication.
* UIApplication est la classe qui représente notre application et elle utilise le patron de conception singleton. Il est donc toujours possible d’en récupérer l’instance grâce à la propriété **UIApplication.shared**
* Pour accéder a notre AppDelegate dans le code de l’application, il faudra écrire : **UIApplication.shared.delegate**

Création d’une interface (Vue)

* L’interface Builder est un éditeur permettant de créé et modifier des vues en WYSIWYG (**w**hat **y**ou **s**ee **i**s **w**hat **y**ou **g**et)
* Il est accessible grâce a la sélection du fichier Main.storyboard
* A partir de la bibliothèque d’objets fournie par le framework, il est possible pas un cliquer/glicer d’ajouter des éléments graphiques sur la vue.

Il faudra à la fois modifier l’interface que l’on nous donne pour changer les attributs des boutons (dans l’inspecteur d’attributs) et il y a une partie a coder.

Interaction entre la vue et le contrôleur

* Les interactions entre la vue et le contrôleur se font dans le framework grâce à la notion de prises et d’actions :
  + Les prises également appelées **Outlets** sont des liens permettant de référencer un élément graphique de la vue
  + Mes actions permettent de lancer des méthodes du contrôleur à la suite d’action de l’utilisateur.

Comment réaliser une prise ?

1. Afficher le code du contrôleur grâce à l’éditeur d’assistant

Le menu est une icône ressemblant à aligner à gauche de Word (pour le texte)

1. Création d’une prise :
   1. Par un **ctrl clic** (ou à partir du menu contextuel), déplacer la souris vers le code du contrôleur. Un trait bleu symbolise la création de la prise.
   2. Une annotation **@IBOutlet** avec le nom de la prise est alors automatiquement créée

Rappel : nom\_var : typed\_donné

Est une variable pointant sur un type de type : typed\_donné

Pour vérifier si on a aucune erreur : on fait cliquer droit sur la partie édition d’interface.

On est obligé de supprimer une prise et la recréer pour pouvoir avoir un nom différent on ne peut pas juste modifier le code !!!

Comment réaliser une action

* La création d’une action se déroule de la même manière

1. Afficher le code du contrôleur grâce a l’éditeur d’assistant :
2. Création d’une action
   1. Par ctrl + clic (ou a partir du menu contextuel), déplacer la souris

Vers le code du contrôleur. Un trait bleu…

Une annotation @quelquechose

Pour voir toutes les prises : aller voir l’inspecteur de connexion (l’icône est une sphère à côté de l’équerre en haut à droite de la fenêtre)

Ecriture du contrôleur

Le point d’entrée de l’application est spécifié dans MainStory par une flèche.

C’est le contrôleur associé à cette vue qui sera exécuté au lancement de l’application et plus précisément sa méthode viewDidLoad()

Pour le CC :

* Sur papier (50% note finale)
* + Un projet à réaliser (50%)

Création d’une table View pour afficher des données (cours 2)

(Cours 2 de dev mobile)

Le table view

* C’est un composent qui permet d’afficher une liste d’élément, organisées en section sans limite de taille.
* Une infinité de cellules ne sera pas créée, uniquement celles qui seront vues a l’écran, les autres seront réutilisées. Quand on fait défiler l’écran la première cellule disparait et va être. Quand on fait défiler l’écran la première cellule disparaitre et va être

Dans la taible view. Il y a :

* Une section
  + Des cellules
  + Etc…
* Une autre section
  + Des cellules
* La table view ne gère toute la présentation de leurs données pas les données elles-mêmes

- En fait la vue délègue toutes les prises de décision au contrôleur d’où le nom de mécanisme de délégation.

- le contrôleur devient ainsi « délégué » celui à qui on délègue la responsabilité. Quand la vue aura besoin de quelque chose cela sera c

Au contrôleur de répondre à son besoin

* La clase UITableView qui se situent dans la couche Vue qui va afficher les données
* Et c’est la classe UITableViewController qui pilotera la vue et qui configurera la liste complète.
* Le contrôleur issu de la classe UITableViewController, utilisera deux protocoles pour fonctionner :
  + UITableViewDataSource pour foirnir les données à la table view
  + UITableViewController
* Un protocole en swift se rapproche de la notion d’interface dans les autres langagesde programmation.
* Une interface peut être désignée comme un contrat : toute classe qui souhaite utiliser un protocole s’engage à implémenter les méthodes dont les signatures ont étés définies dans le protocole.

Exemple :

UITableViewDataSource fournit à la table View impose au contrôleur de coder 2 méthodes :

* L’une qui retourne le nombre de sections :

Override func numberofsection(etc…){

Return 1 // pour 1 section

}

* L4une qui retourne le nombre de lignes :

Override func tabView(etc…){

Return 0 // pour le nombre de lignes

}

* Une autre classe est généralement nécessaire pour afficher une liste d’éléments :
  + La classe UITableViewCell qui va permettre de configurer un élément de la table UITableView

C’est la seule fois ou le UITableViewController le fait automatiquement

Configuration de l’objet UITableViewCell

* La solution la plus simple :

Sélectionner la cellule :

Choisir dans l’inspection d’attributs :

subtile (automatiquement le titre et le sous-titre)

Il faut absolument donner un identifiant a la cellule.

* Solution moins simple :

On choisit custom ou on ne verra pas les emplacements des titres et des sous titres

* Il faut ensuite déposer les labels
* Et déposer les prises

Ecriture de UITableViewController

3 méthodes au minimum sont a coder pour afficher les données dans la table view

Au minimum :

Override func numberOFSections(etc…)->Int{

…

}

Override func tableView(etc…){

…

}

Override func tableVieww){

…

}

indexPath est une structure : contenant nb de lignes et nb de colonnes

pour identifier les coordonnées d’une cellule la tabvle view

Suite de l’écriture du contrôleur TCVMessage

* Implémentation de la méthode qui permet de donner corps à une cellule pour un index donné

Override func tabView(){

…

Var lemessage : Message

Let unecellule : …tableView.dequereusableCell(etc..) as ! MessageCell

}

Dans le cas du style Subtile :

cell.testLabel ?.text

Cell.detailTextLabel?.text

* C’est la méthode “dequeuReusableCell” qui instancie une cellule pour un index donné. Ne pas oublier le cast
* Pour des soucis de performance, la méthode

**POO dev mobile :**

Title

Machin et section (c’est le header de la page)

Faire un TableViewControler customisé :

* + on créé un projet
  + on supprime ViewControler
  + On créer un nouveau fichier avec l’onglet fichier en haut a gauche
  + On créer un template Cocoa Touch Class
  + On créer tout ça dans le bon répertoire qui est demandé

# end page