CountOnMe

Présentation

Lauriane Haydari
2019

RAPPEL DU CONTEXTE :

Je suis lead développeur dans une entreprise de solutions bureautiques. Mon manager me remonte qu'il y a un client en attente de la livraison de son application CountOnMe.
Une calculatrice.

Améliorations :

- Le design et le responsive en portrait uniquement
- L'architecture du projet
- Les tests unitaires
- Il manque la division et la multiplication.

Bonus :

- Ajout d'un bouton Clear et Delete
- Ajout d'un bouton Comma
- Priorités de calcul
- Réduction des nombres à virgules à 3 chiffres après la virgule

QU’EST-CE QUE LE MODÈLE MVC ?

Le principe MVC est un patron de conception célèbre et très utilisé qui signifie :
Model View Controller.

- Le Modèle gère la logique du programme, c’est le cerveau de l’application.
- La Vue se concentre sur l'affichage. Elle ne fait pas de calcul et se contente de récupérer des variables pour savoir ce qu'elle doit afficher. C'est ce que l'utilisateur voit, c'est l'interface de l’application.
- Le Contrôleur récupère les informations du modèle et les affiche dans la vue. C'est en quelque sorte l'intermédiaire entre le modèle et la vue.

QU’EST-CE QUE LE MODÈLE MVVM?

Le modèle MVVM, qui veut dire Model-View-ViewModel, est un modèle de conception de l'interface utilisateur (UI). Il fait partie d'une grande famille de modèles connus par le préfixe MV, comme le MVC et Model View Presenter (MVP).

Chacun de ces modèles a pour but de séparer la logique métier de l'interface utilisateur afin de rendre les applications beaucoup plus faciles à maintenir et surtout à tester.

Pour ce projet il a été préférable de choisir le MVVM Réactif natif.

Réactif natif c’est à dire sans bibliothèque externe. Différent du MVVM RX par exemple.

Réactif => La vue est abonnée à des closures, lorsqu’un évènement est envoyé par la closure, la vue se met à jour intrinsèquement.
La vue s’abonne donc aux variables réactives puis envoie le premier input qui est le **viewdidload :** l’input de connexion pour les variables réactives.

À chaque input envoyé, les outputs auxquels le ViewController est abonné, vont envoyer les données.

Le ViewController est claire, simple, il ne connait pas les données, il contrôle que le flot de data va bien là ou il faut quand il faut.

EXPLICATION DE LA DÉMARCHE UTILISÉE

1. D’après le code Legacy fourni, difficilement réutilisable et qui fonctionne de façon incomplète.

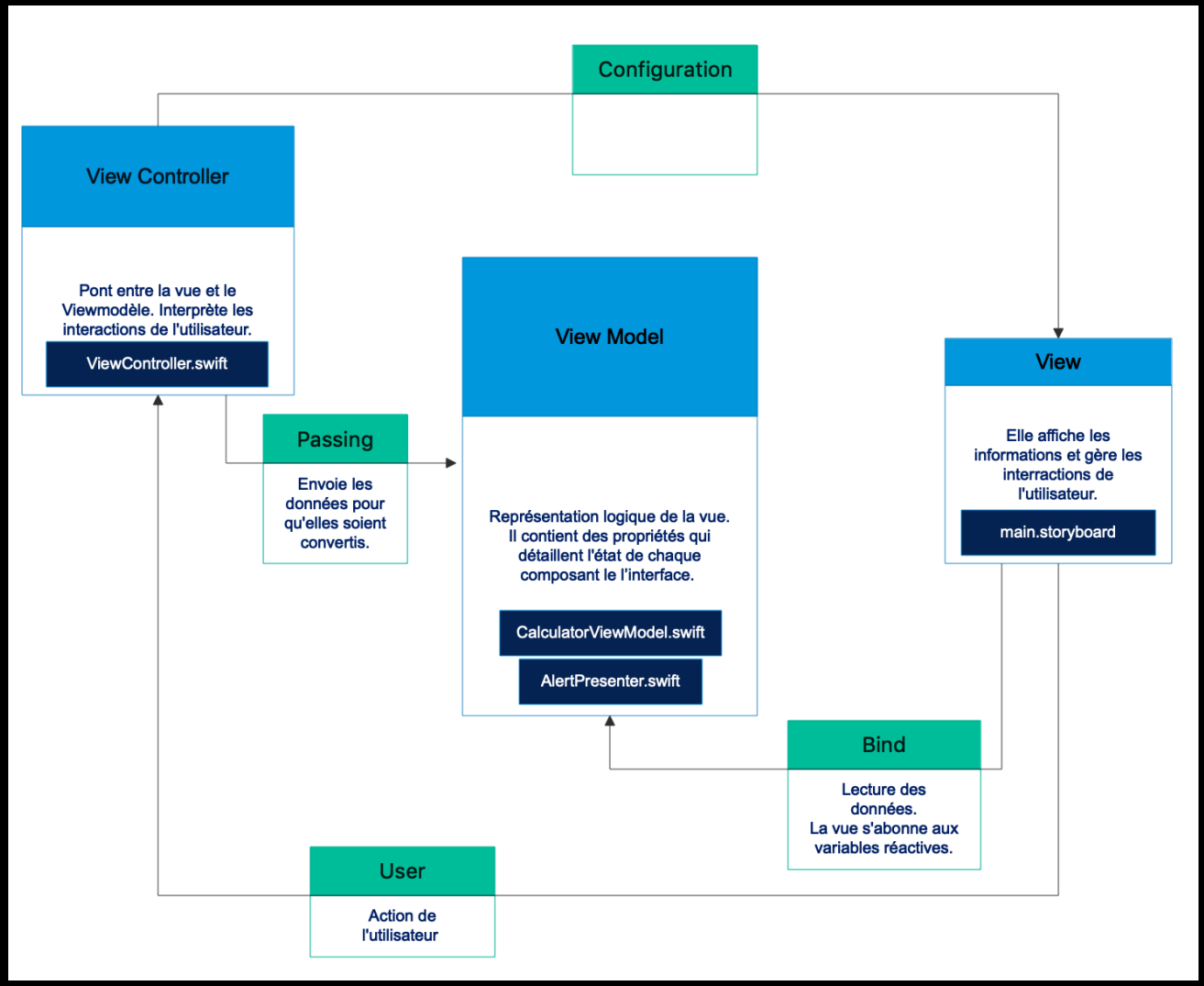
2. Je ne me risque pas à toucher le code tant que je ne l’ai pas testé. Je développe donc une interface qui est le ViewModel et qui sera le réceptacle des évènements en entrée et des évènements en sortie.

3. J’écris la suite des tests de mon ViewModel, testant les comportements attendus. Une fois que mon ViewModel est écris et qu’il est complètement testé.

4. Je déporte la logique du ViewController dans mon ViewModel en toute sécurité.

SIMULATION DE L’APPLICATION

LE DIAGRAMME MVVM RÉACTIF NATIF DE L'APPLICATION



PRÉSENTATION DU CODE

PRÉSENTATION DU CODE


```
// Outputs

var displayedText: ((String) -> Void)?
var nextScreen: ((Screen) -> Void)?
```

```
// Inputs

func viewDidLoad()
func didPressEqualButton()
func didPressOperand(operand: Int)
func didPressOperator(at index: Int)
func didPressClear()
func didPressDelete()
```

QUESTIONS ?