Baluchon P9

Notions à voir et revoir :

La frame c'est ce que tu vois dans ta vue parente

La bounds c'est la taille réelle de ta vue

L ‘appDelegate démarre l’application avec la fonction start() qui présente la mainView.

La responsabilité de ma classe Screens c’est de passer le viewmodel

Screens utilise le storyboard main et instancie un viewController dans le storyboard.

La responsabilité du viewModel c’est de parler au coordinator mais pas au viewController

Mon viewController s’abonne aux variables réactive de mon viewModel dans la func bind et dans la func viewDidLoad du viewController

HTTPEngine c’est une classe créée pour effectuer des URL request

HTTPClient c’est une couche qui elle contient une instance d’un engine donc elle va appeler les request de l’Engine et l’engine va lui renvoyer de (data, response, error)

Le rôle du client c’est de décoder cette data du format Json (ou autre) vers un format objet.

Donc le client transforme mon Json en objet.

L’objet est donc renvoyé dans le callback.

Le callback de la fonction request de mon HTTPclient est injecté dans la fonction decodeJson. Au moment où engine.send reçoit une (data, response, error).

Dans ma fonction decodeJSON<T> mon callback est passé en paramètre et sera appelé uniquement si ma fonction request a réussie à recevoir la data. Mon objet sera donc renvoyé.

C’est ensuite le Repository correspondant à chaque vue qui va effectuer les request. Il ne connaît pas l’URL.

Le Token intervient dans la requete du HTTPClient

Le Token c’est une classe qui a une variable réactive qui est appelée au deinit() de l’objet.

L’HTTPEngine utilise un Token, si la variable réactive du Token est appelée, (willDealocate), la task s’annule. La variable réactive est appelé quand l’objet est détruit, donc dans le deinit()

Quand le repository est détruit l’Engine annule la task, le repository est détruit quand sa vue n’est plus affiché.

Explication du fonctionnement de l’architecture :

La variable réactive est appelée quand l’objet est détruit.

Le Repository crée la requête

Le contexte est une classe qui permet d’encapsuler des propriétés. Il est injecté dans mon appDelegate, qui va être injecté dans l’appCoordinator, qui va être injecté dans le main Coordinator car on y a crée une instance de Screens.

Dans la classe Screens lorsque je crée un createCurrencyViewController j’ai besoin de créer un repository. Et j’ai besoin de la propriété client de mon contexte.

Dans le contexte on peut y créer la classe Theme qui contient tout les couleurs.

On est capable d’afficher le label des rates quand le pickerView didselect