Baluchon

Présentation du projet 9
Openclassrooms
-
Mentor : Bertrand Bloc'h
Élève: Lauriane Haydari

Baluchon

Le Baluchon est une application de trois pages :

1. Comparez la météo locale Américaine et celle de votre bon vieux chez vous.

2. Obtenez le taux de change entre le dollar et votre monnaie actuelle

3. Traduisez depuis votre langue favorite (le français) vers l'anglais

Bonus :

- Ajout d'autres villes français et américaines

- Ajout de la livre sterling et du Yen japonais ainsi que de la possibilité de changer la devise initiale en plus de la devise de destination

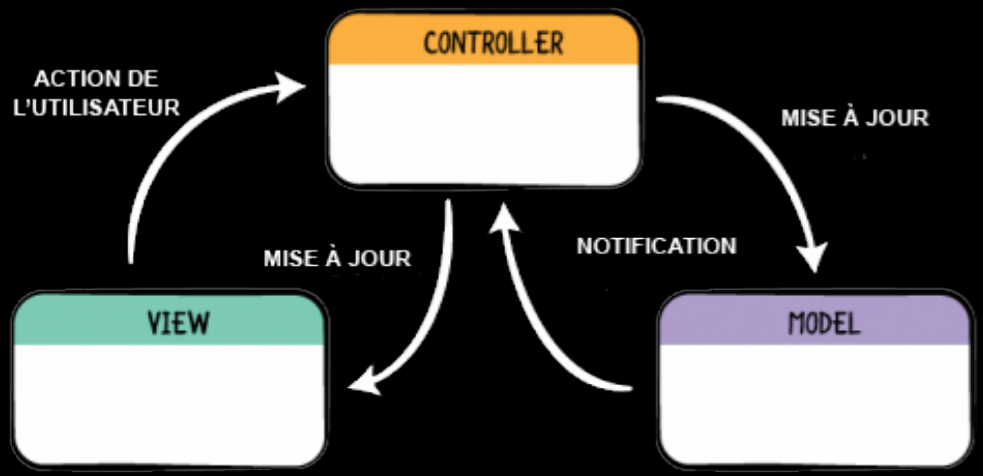
- Ajout d'autres langues de traduction ainsi que la possibilité de changer la langue initiale à traduire

SIMULATION DE L’APPLICATION

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
- La Vue : se concentre sur l’affichage
- Le Contrôleur : récupère les informations du modèle et les affiche dans la vue.

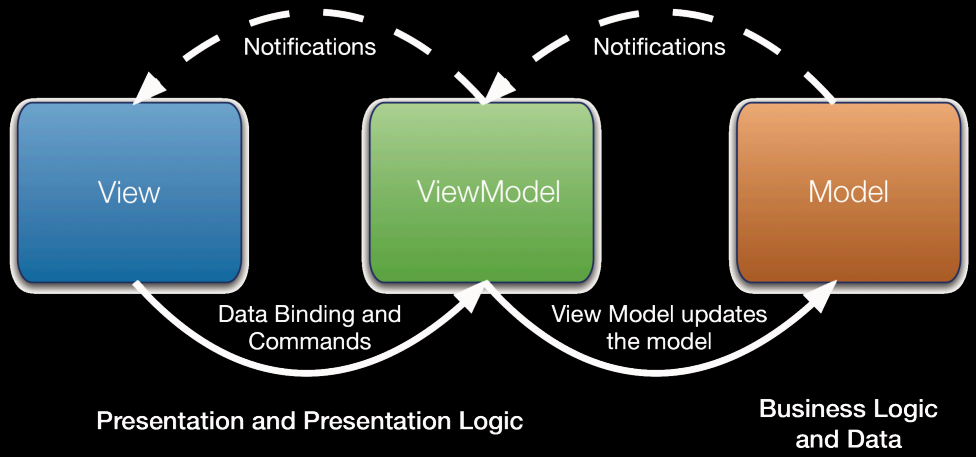


QU'EST-CE QUE LE MODÈLE MVVM ?

Le Model-View-ViewModel, est un modèle d'architecture.

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

- Les inputs : les événements
- Les outputs : receptacles des événements des inputs correspondants.
- Le ViewController : contrôle la vue, il est claire, simple, il ne connait pas les données.



LA DÉMARCHE UTILISÉE

1 - Création des trois modules Weather / Converter / Translator en MVVM utilisant un Coordinator par navigation d'écran

2- Implémentation de UI TabBar avec 3 items :

.weather

.converter

.translator

3- Écriture du network et des repository par module => requêtes réseaux

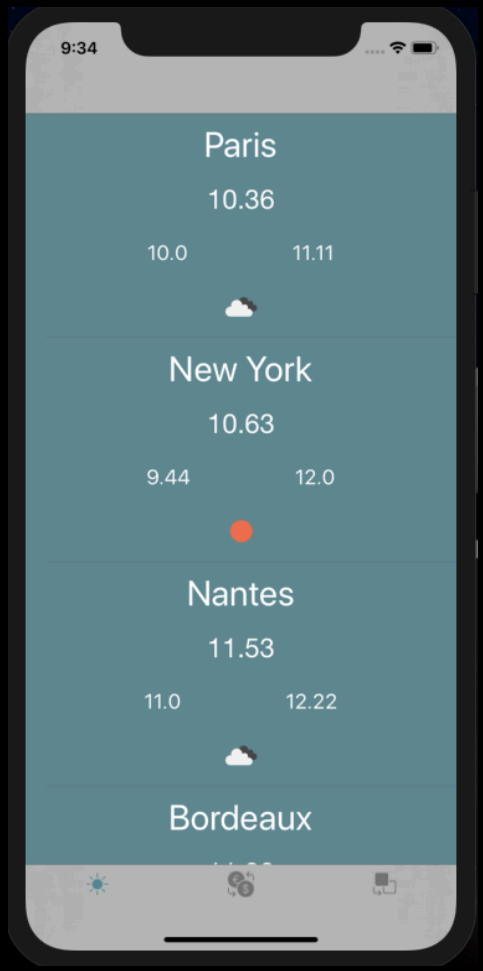
4- Création du modèle de chaque module

5- Unit et UI Tests

LE PLAN DE LA LOGIQUE DE CHAQUE MODULE

WeatherViewModel

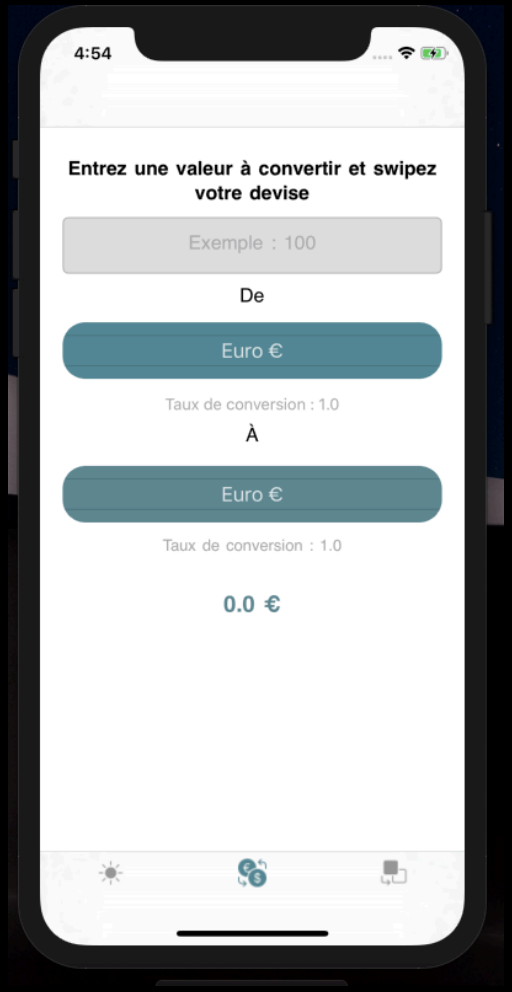
```swift
// MARK: - Outputs

var items: (([Item]) -> Void)?

var nextScreen: ((NextScreen) -> Void)?

var activityIndicatorIsLoading: ((Bool) -> Void)?


// MARK: - Inputs

func viewDidLoad() {

}
```



LE PLAN DE LA LOGIQUE DE CHAQUE MODULE

ConverterViewModel

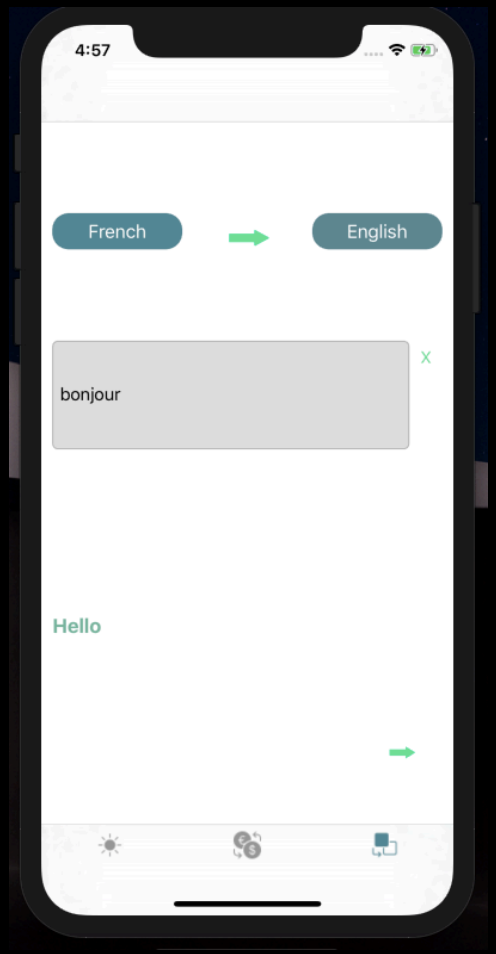
```
// MARK: - Outputs

var requestLanguageTextButton: ((String) -> Void)?

var resultLanguageTextButton: ((String) -> Void)?

var requestTextField: ((String) -> Void)?

var resultText: ((String) -> Void)?

var nextScreen: ((NextScreen) -> Void)?

// MARK: - Inputs

func viewDidLoad() {
}

func didTapRequestTextField(text : String?) {
}

func didSelectLanguageType(for type: LanguageType) {
}

func didPressClearButton() {
}

func didPressTranslatButton(for requestText: String) {
}
```



LE PLAN DE LA LOGIQUE DE CHAQUE MODULE

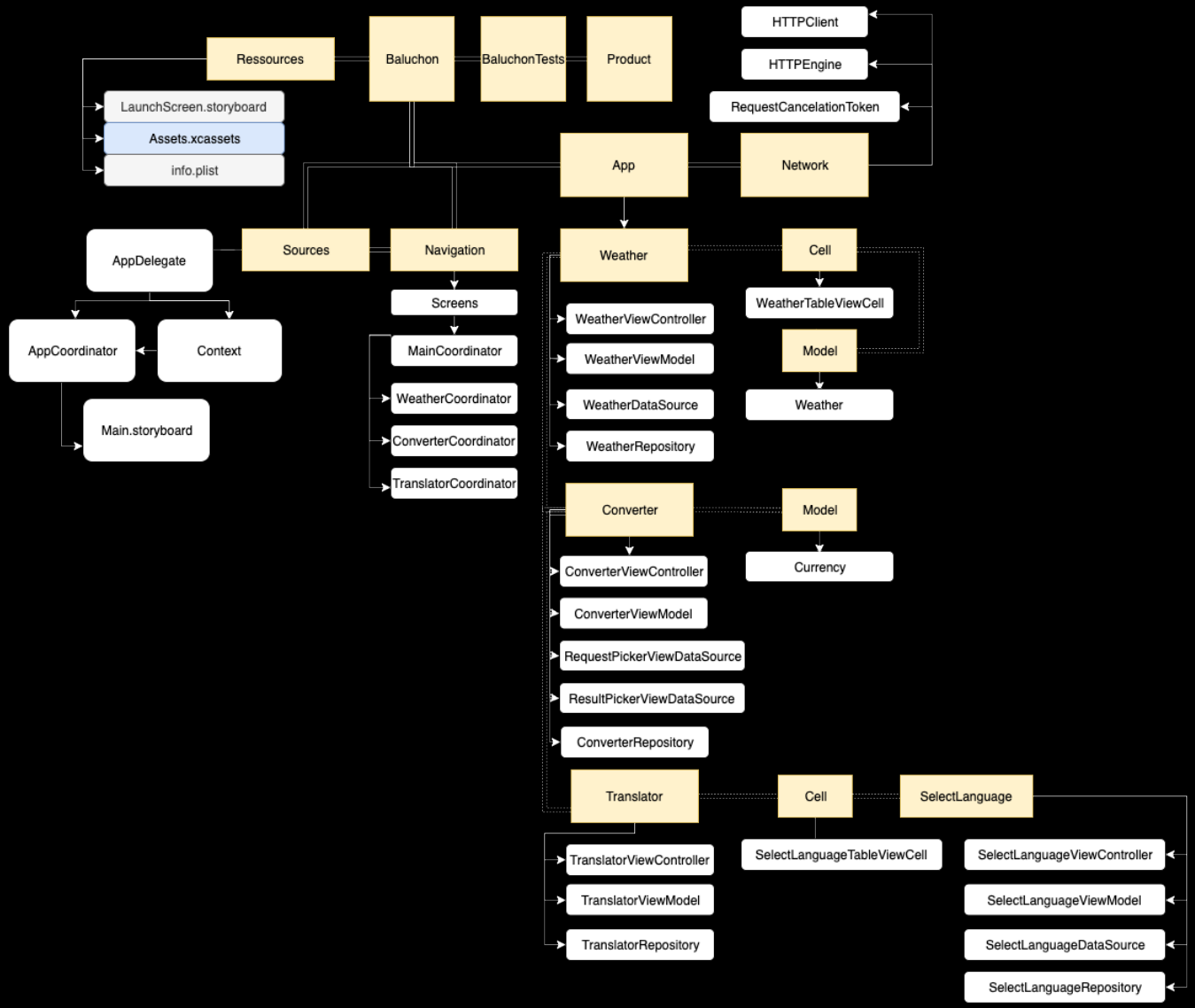
TranslatorViewModel

```
// MARK: - Outputs

var visibleItems: (([ItemLabel]) -> Void)?

var nextScreen: ((NextScreen) -> Void)?

// MARK: - Inputs

func viewDidLoad() {
}
func didSelectItem(at index: Int) {
}
```



LE DIAGRAMME MVVM RÉACTIF NATIF DE L'APPLICATION

- SOURCES
- NETWORK
- NAVIGATION



PRÉSENTATION DU CODE

COMPÉTENCES APPRISENT

- MVVM avec Coordinator
- Requêtes réseaux avec une couche network comportant un client, un engine et un repository
- Affichage de tabBar Item sans storyboard
- PickerView avec Datasource et Delegate en MVVM
- TableView et MVVM
- Unit Tests avec utilisation de Mock

QUESTIONS ?