Structure MVC dans l'application « Baluchon »

A la lecture de l'énoncé, il apparaissait notablement que le projet était constitué de trois parties.

1. Taux de change

Dans la page taux de change, on peut insérer un montant dans votre monnaie locale et voir le résultat en dollar **($)**. Rien de bien sorcier a priori !

Pour obtenir le taux de change, vous utiliserez l'API fixer.io, actualisé chaque jour. Il vous faudra donc obtenir le taux de change au minimum une fois par jour pour être sûr d'afficher le bon montant en dollar à vos utilisateurs.

2. Traduction

Dans la page traduction, l'utilisateur peut écrire la phrase de son choix en **français** et obtenir sa traduction en **anglais** *of course !*

Pour cela, vous utiliserez l'API de Google Translate. Contrairement à la précédente, cette API nécessite une clé que vous obtiendrez en suivant les étapes expliquées dans la documentation.

3. La météo

Dans la page météo, vous afficherez les informations météo de New York et de la ville de votre choix (là ou vous habitez).

Pour chaque ville, vous afficherez les conditions actuelles en utilisant l'API OpenWeat hermap en précisant notamment :

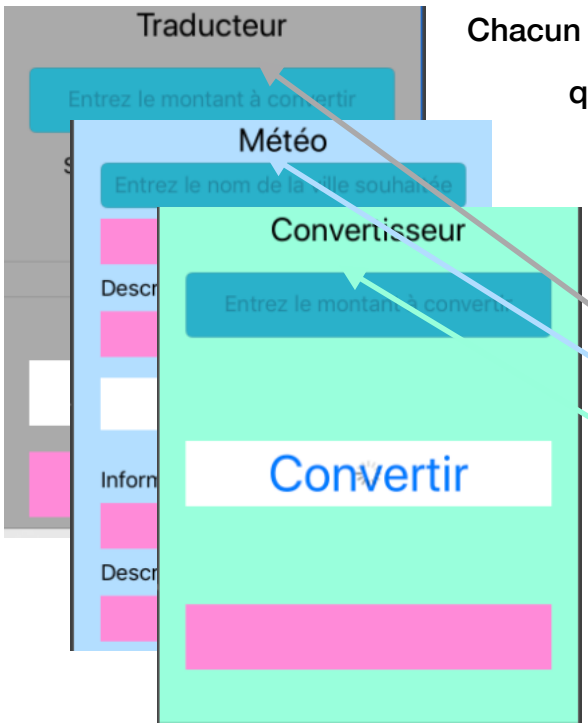
- La température

LE TAUX DE CHANGE,

LA TRADUCTION

LA MÉTÉO DANS DEUX VILLES DISTINCTS DONT L'UNE DEVAIT ÊTRE NEW-YORK.

Ces trois secteurs indépendants devaient pouvoir être accessible à tout moment alors, j'ai opté pour l'installation d'une tab Bar.

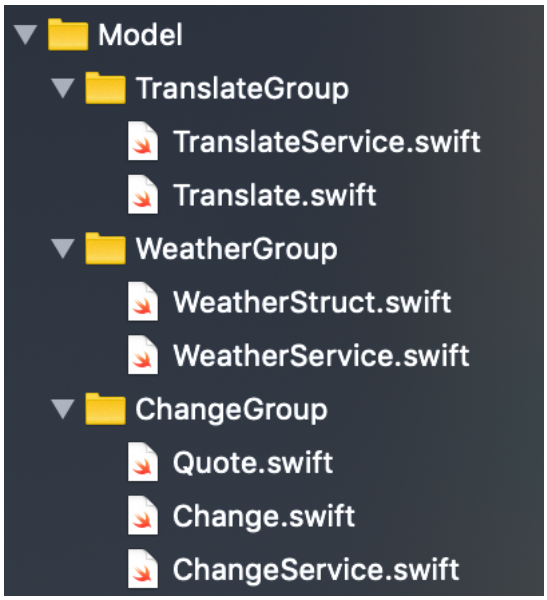


Chacun d'eux devait avoir une vue

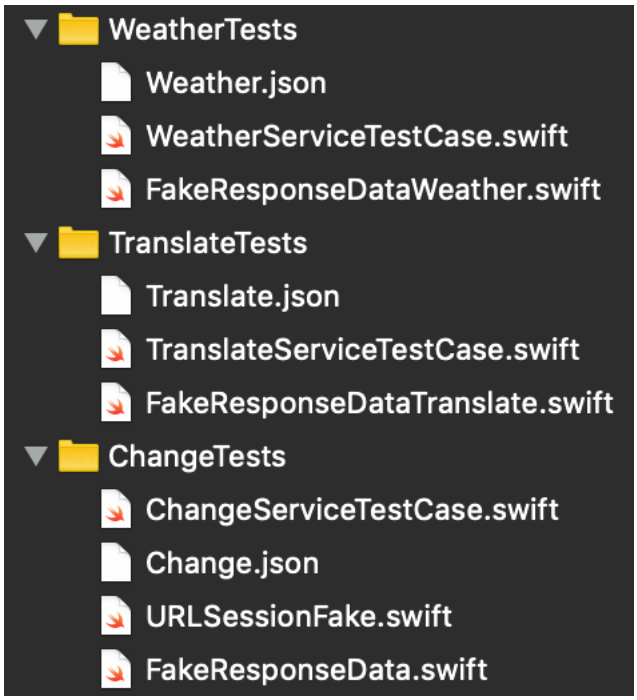
qui serait gérée par « ViewController »

lui même assisté d'un modèle dédié.

Au regard que toutes ces activités allaient avoir des accès réseaux et devaient recueillir des données sous la forme Json, il devenait nécessaire de compléter le model d'une structure d'accueil.



Alors, dans un souci d'ordonnancement, il apparaissait utile de créer des Groupes de Models que j'ai nommé TranslateGroup, WeatherGroup et ChangeGroup.



Ce constat fut le même pour organiser le groupe dédié aux tests.

En effet, ceux-ci devaient se doter d'un fichier Json de simulation de données en plus des retours fictifs d'informations réseau et des fichiers listant les tests proprement dit.

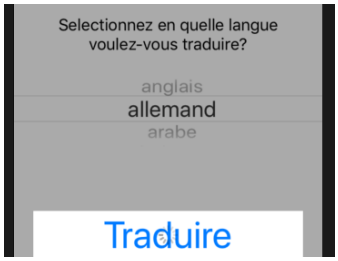
La factorisation m'a conduit à créer un autre ViewController nommé que j'ai nommé « ExtentViewController.swift ». Identifié comme une extension de classe UIViewController et attendant un paramètre message, j'ai pu l'utiliser pour informer l'utilisateur de l'absence de saisie.

```
import UIKit

extension UIViewController {

    func presentAlert(message: String) {
        let alertVC = UIAlertController(title: "Error", message: message,
            preferredStyle: .alert)
        alertVC.addAction(UIAlertAction(title: "OK", style: .cancel, handler: nil))
        present(alertVC, animated: true, completion: nil)
    }
}
```

Le bonus dans l'application « Baluchon »

J'ai retenu l'idée de proposer une traduction multilingue.

Pour cela j'ai installé au centre de la vue un pickerView dont le contenu était des langues au choix



J'ai étoffé la structure Translate d'un dictionnaire « languages » contenant en clés la langue en String et en valeur un String qui contenait l'extension attendu par l'API.
Jai créé un tableau « languagesPossible » contenant les même langues alimentant le pickerView. Ainsi il devenait possible de traduire la saisie en quinze langues.

```
let languages = ["anglais" : "en", "allemand" : "de", "arabe" : "ar", "bulgare" :
    "bg", "chinois" : "zh", "danois" : "da", "espagnol" : "es", "français" : "fr",
    "grec" : "el", "italien" : "it", "japonais" : "ja", "lituanien": "lt","portugais"
    : "pt", "russe" : "ru", "suedois" : "sv" ]


let languagesPossible = ["anglais", "allemand", "arabe", "bulgare", "chinois",
    "danois", "espagnol", "français", "grec", "italien", "japonais", "lituanien",
    "portugais", "russe", "suedois"]
```

Pour des raisons de visibilité, j'ai regroupé tout ce qui gère le pickerView dans une extension de la classe ViewControllerTranslate au sein de cette même classe.

```
extension ViewControllerTranslate: UIPickerViewDelegate, UIPickerViewDataSource {
    func numberOfComponents(in pickerView: UIPickerView) -> Int {
        //delimiting the size of the pickerview
        return 1
    }

    func pickerView(_ pickerView: UIPickerView, numberOfRowsInComponent component:
        Int) -> Int {
        //power of the pickerview by the controller
        return languagesPossible.count
    }

    func pickerView(_ pickerView: UIPickerView, titleForRow row: Int, forComponent
        component: Int) -> String? {
        return languagesPossible[row]
    }
}
```