



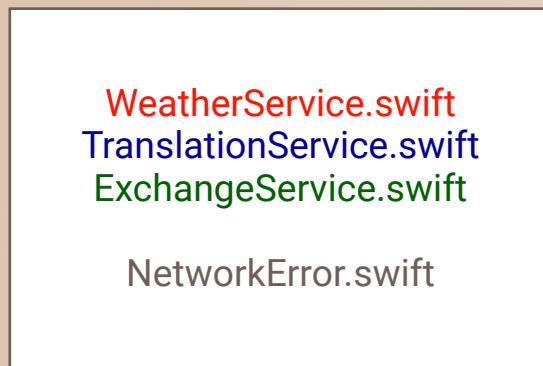
Projet présenté par Anthony Scott
Date de la soutenance : 29/07/20

PRÉSENTATION DU PROJET

- projet d'application iOS, centré sur les appels réseau
- créé en Swift 5.1 (Xcode 11.5)
- supporté par toutes les tailles d'iPhone, en mode Portrait
- en anglais
- conforme au patron de conception MVC
- disponible sur github.com/anthonymfscott/Baluchon



NETWORK



json decoder

MODÈLE

`Weather.swift`
`Translation.swift`
`Exchange.swift`

CONTROLEUR

`WeatherViewController.swift`
`TranslationViewController.swift`
`ExchangeViewController.swift`

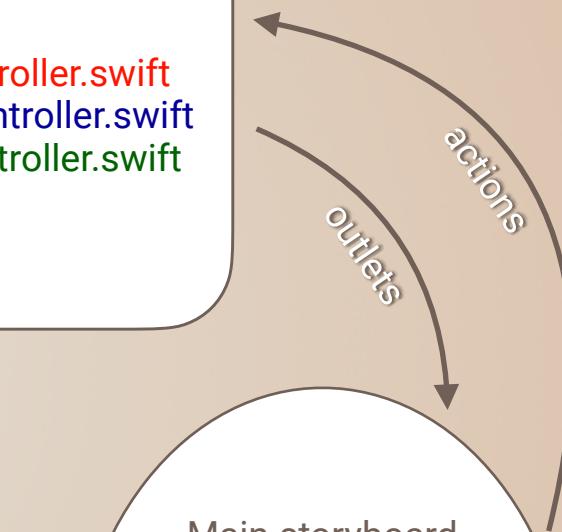
callbacks

VUE

`Main.storyboard`
`BaluchonView.swift`
`BannerView.swift`

↓

`WeatherView.swift`
`TranslationView.swift`
`ExchangeView.swift`





MODÈLE & NETWORK

Le modèle du projet est composé des fichiers **Weather**, **Translation** et **Exchange**. Il s'agit de structures conformes au protocole *Decodable*, permettant la récupération de données de fichiers JSON.

La structure **Weather**, en particulier, utilise l'enum *CodingKeys* ainsi qu'une initialisation personnalisée pour rendre l'accès aux propriétés plus lisible.

Les fichiers qui font appel aux services réseau ont été rassemblés dans un dossier nommé **Network**.

Chaque service a son propre fichier. En cela, **WeatherService**, **TranslationService** et **ExchangeService** sont très semblables. Ils sont initialisés avec le pattern singleton pour limiter l'instanciation à un seul objet. Ils utilisent *URLSession* pour lancer les requêtes réseau et en récupérer les données via une fermeture. Celle-ci utilise le générique *Result* en paramètre, et fait appel quand il le faut à une *Error* personnalisée, **NetworkError**.



Tout l'aspect visuel de l'application a été pensé et créé avec l'*Interface Builder*.

WeatherView, **TranslationView** et **ExchangeView** sont des vues personnalisées utilisant des *outlets* reliés au storyboard. L'utilisation de propriétés observées permet d'accéder plus efficacement aux propriétés spécifiques des *UILabel*, *UITextField* et *UITextView* qui nous intéressent dans ce projet. Ces vues héritent elles-mêmes de **BannerView**, qui fait appel à une initialisation personnalisée, permettant de régler d'un coup les aspects communs de leur design (*shadow*, *cornerRadius*).

BaluchonView, autre vue personnalisée, comprend les réglages du baluchon central. Elle utilise le même type d'initialisation que **BannerView**. Pour le reste des fonctionnalités, voir le document annexe « Bonus » du projet.



Encore une fois, *WeatherViewController*, *TranslationViewController* et *ExchangeViewController* fonctionnent selon le même principe général.

Ils prennent en propriétés les vues personnalisées dont on a parlé précédemment.

La méthode *viewDidLoad* initialise les contrôleurs avec les données à afficher lors de leur apparition.

L'appel réseau se fait chaque fois au moment où l'utilisateur appuie sur le baluchon central. Les données reçues par le modèle sont traitées au moyen d'une fermeture ; en cas de succès, l'UI est mis à jour selon les besoins de chaque contrôleur et en cas d'erreur, une alerte est présentée à l'utilisateur.

La méthode *toggleLoadingState* permet de gérer l'affichage de l'indicator view, en prenant en compte l'asynchronicité de la réponse et en bloquant les champs de texte concernés. De la même manière, *dismissKeyboard* gère l'assignation du first responder pour rendre l'utilisation de l'application plus confortable.

EXTENSIONS & UTILITIES



UIViewControllerExtension

implémente une méthode qui permet d'afficher une alerte commune aux 3 contrôleurs de l'application.

UIImageViewExtension implémente une méthode *downloaded* qui comprend un appel réseau supplémentaire, pour récupérer les icônes météo de l'API d'OpenWeatherMap.

FloatExtension ajoute deux méthodes très courtes pour faciliter l'arrondissement des données et convertir un Float en Int.



Constants, Network, Strings et Images

sont des énumérations qui rassemblent les données les plus importantes du projet, afin d'y accéder plus facilement en cas de modification ultérieure.

Les clés API sont contenues dans un fichier plist dont l'accès est géré par une méthode appelée depuis le Modèle (***Network***).



Les tests de cette application fonctionnent avec la création d'un double et d'une série de classes fichiers afin d'imiter les appels réseau.

FakeresponseData récupère les fausses données des fichiers JSON **Weather**, **Translation** et **Exchange**, qui respectent rigoureusement la structure des réponses API utilisées dans le projet. Il gère aussi les cas de données non valides, les status codes et les erreurs.

URLSessionFake et **URLSessionDataTaskFake** permettent une initialisation personnalisée des classes dont leurs noms s'inspirent.

Les faux appels réseau ainsi créés permettent des tests efficaces, qui sont gérés dans les fichiers **WeatherServiceTestCase**, **TranslationServiceTestCase** et **ExchangeServiceTestCase**.

AMÉLIORATIONS POSSIBLES

utilisation de
URLComponents
et *QueryItems*

ajout d'une toolbar
au-dessus des
claviers numériques
(*inputAccessoryViews*)

ajout d'autres langues
et devises avec *UIPickerView*
pour une utilisation plus large