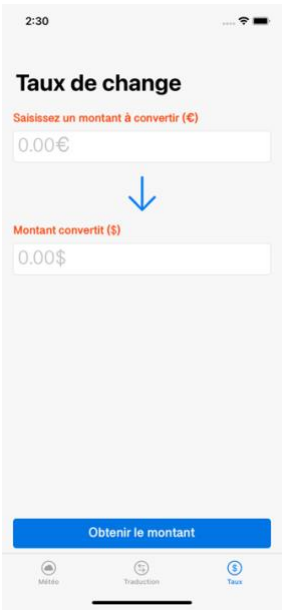
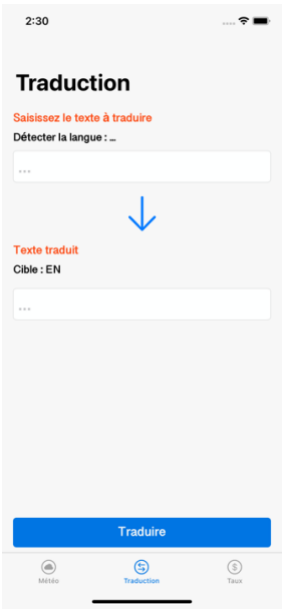
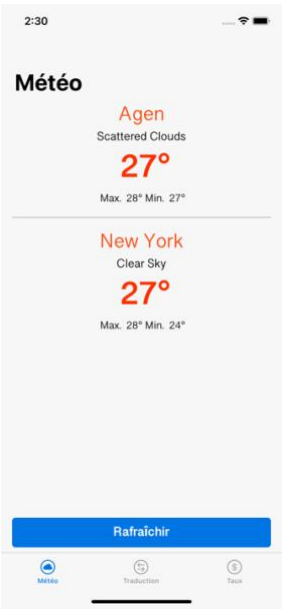
Projet 9 : Réalisez une application de voyage

INTRODUCTION

L'application se nomme « Le Baluchon », celle-ci est composée de trois pages :







- **Taux de change :** permet d'acquérir le taux de change entre le dollar et la monnaie locale.
  Pour obtenir le taux de change, il doit y être utiliser l'API fixer.io, actualisé chaque jour. Il 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

- **Traduction :** traduire depuis le français vers l'anglais.
  Pour cela, il y sera utilisé l'API de Google Translate.

- **Météo :** afficher les informations météo de New York et de la ville choisie.
  Pour chaque ville, il est obligatoire d'afficher les conditions actuelles en utilisant l'API OpenWeathermap en précisant la température ainsi que la description des conditions (nuageux, ensoleillé, etc.).

Pour naviguer entre les pages, il faudra utiliser une barre d'onglet ("*tab bar*"), chaque onglet correspondant à une des trois pages décrites précédemment.
Lors des appels au réseaux dans l'application, si une erreur a lieu, je dois la présenter sous forme de pop-up en utilisant la classe UIAlertController.

Le projet doit respecter le modèle MVC, l'application doit être responsive (s'adapte à toutes les tailles d'écran) et en mode portrait.

API = (Application Programming Interface) permet de rendre disponibles les données ou les fonctionnalités d'une application existante afin que d'autres applications les utilisent. Le client envoie une requête pour obtenir une information et le serveur envoie une réponse contenant les données demandées si cela est possible.

LES APPELS RESEAU

Tout d'abord, pour obtenir les données, il faut pouvoir les décoder afin de pouvoir les utiliser.

```swift
final class Networking {

    enum APIError: Error {
        case error(String)
    }

    func getJSON<T: Decodable>(urlString: String,
                               dataDecodingStrategy: JSONDecoder.DataDecodingStrategy = .deferredToData,
                               keyDecodingStrategy: JSONDecoder.KeyDecodingStrategy = .useDefaultKeys,
                               completion handler: @escaping ((Result<T, APIError>) -> Void)) {
        guard let string = urlString.addingPercentEncoding(withAllowedCharacters: .urlQueryAllowed),
              let url = URL(string: string) else {
            handler(.failure(.error(NSLocalizedString("Error: Invalid URL", comment: ""))))
            return
        }
        let request = URLRequest(url: url)
        URLSession.shared.dataTask(with: request) { data, _, error in
            if let error = error {
                handler(.failure(.error(NSLocalizedString(error.localizedDescription, comment: ""))))
            }
            guard let data = data else {
                handler(.failure(.error(NSLocalizedString("Error: Data is corrupted", comment: ""))))
                return
            }
            let decoder = JSONDecoder()
            decoder.dataDecodingStrategy = dataDecodingStrategy
            decoder.keyDecodingStrategy = keyDecodingStrategy
            do {
                let decodedData = try decoder.decode(T.self, from: data)
                handler(.success(decodedData))
                return
            } catch let decodingError {
                handler(.failure(APIError.error("Error: \(decodingError.localizedDescription)")))
                return
            }
        }.resume()
    }
}
```

Pour cela nous avons besoin de :

- l'URL, identifie l'API. J'utilise l'initialisation de la classe avec un String. Elle renvoie un optionnel qui est déballé avec guard let.

- Result<T, APIError>, est une énumération où il y a deux cas, succès ou échec. Dans le premier cas est renvoyé le modèle de réponse générique. Dans le second cas, le modèle d'erreur sera retourné.

- l'URLRequest, spécifie la requête : l'URL (en paramètre), la méthode HTTP (GET, POST, etc..) n'est pas spécifié.

- l'URLSession, permet de lancer des requêtes, soit de télécharger des données depuis et vers des points de terminaison indiqués par l'URL.

    - .shared, partage la session pour récupérer le contenu d'une URL

- .dataTask, stocke la tâche et admet dans sa fermeture trois paramètres : data, response, error qui vont permettre de gérer la réponse.

- JSONDecoder, comporte une méthode .decode qui convertit le format JSON vers un dictionnaire Swift et permet donc de lire les données reçues de l'appel réseau.

  - Le mot clé **try** : la fonction .decode peut engendrer une erreur. Il est nécessaire de le placer avant pour la gérer. Doit être placer dans le corps de do.
  - Instruction do/catch : dans la partie do, la fonction va tenter d'être exécuter. Si jamais elle renvoie une erreur, la partie catch va la gérer.

- .resume, lance l'appel.

LE SERVICE

```swift
final class ExchangeRateService {

    private let apiService: Networking
    private let endPoint = "http://data.fixer.io/api/latest?"

    init(apiService: Networking = .init()) {
        self.apiService = apiService
    }

    func getCurrency(_ completion: @escaping (Result<Currency, Networking.APIError>) -> Void) {
        apiService.getJSON(urlString: "\(endPoint)access_key=\(APIConfiguration.exchangeRateKey)") { (result: Result<Currency, Networking.APIError>) in
            switch result {
            case .success(let currency):
                completion(.success(currency))
            case .failure(let apiError):
                completion(.failure(apiError))
            }
        }
    }
}
```

Le service est une couche intermédiaire qui permet de contenir le point final avec son modèle de demande et de réponse. Le fait d'isoler le service est de pouvoir le tester plus facilement, de limiter la refractorisation et d'obtenir un code maintenable.

OBTENIR LES DONNÉES

L'obtention du taux de change sera prise pour exemple

```swift
final class ExchangeRateViewController: UIViewController {

    @IBOutlet private weak var valueToConvertTextField: UITextField!
    @IBOutlet private weak var valueConvertedTextField: UITextField!
    @IBOutlet private weak var convertButton: ButtonSettings!
    @IBOutlet private weak var activityIndicator: UIActivityIndicatorView!
    private let exchangeRate = ExchangeRateService()
```

```swift
private extension ExchangeRateViewController {
    //    to get the currency using the network call
    func getCurrency(_ value: String) {
        exchangeRate.getCurrency { [weak self] result in
            guard let self = self else {
                return
            }
            DispatchQueue.main.async {
                self.activityIndicator.stopAnimating()
                switch result {
                case .success(let currency):
                    guard let valueToConvertText = self.valueToConvertTextField.text,
                          let doubleToConvert = Double(valueToConvertText) else { return }
                    let valueToMultiply = Double(currency.rates.usd)
                    let operation = doubleToConvert * valueToMultiply
                    self.valueConvertedTextField.text = String(operation.toTruncatedStringWithTwoDigits())
                case .failure(let apiError):
                    self.alertPopUp()
                    print(apiError)
                }
            }
        }
    }

    //    to get convertion when button tapped
    @IBAction func didTapConvertButton(_ sender: Any) {
        dismissKeyboard()
        activityIndicator.startAnimating()
        guard let valueToConvertText = self.valueToConvertTextField.text else { return }
        getCurrency(valueToConvertText)
    }

    @objc func dismissKeyboard() {
        view.endEditing(true)
    }
}
```

La fonction getCurrency permet de lancer l'appel réseau et de charger les données dans la vue.

- weak, empêche que chaque objet est une référence vers l'autre, des fuites mémoires (retain cycle). Sans, la RAM de l'appareil arriverait à saturation et l'application crash.

- DispatchQueue.main.async, permet d'exécuter le bloc de code sans passer par la file d'attente, et donc limiter les blocages de l'application.

- switch result va gérer deux cas :

    - succès qui prend pour paramètre d'entrée une constante currency de type Currency (modèle de données du taux de change) et où il sera passé toutes les données que l'on veut charger dans l'application.
      Dans un premier temps, je déballe mes optionnels à l'aide guard let. Le premier est le UITextField.text. Le second est ce même UITextField mais qui a changé de type à l'aide du constructeur Double.

      Ensuite, je viens récupérer les données du taux de change que je convertis en Double comme précédemment.
      Enfin, je crée une constante qui va contenir mon opération de conversion entre ma monnaie actuelle et la monnaie ciblée. Le résultat obtenu sera affiché dans le UITextField.

- échec, est le cas où les données sont inexistantes ou erronées, alors une alertPopUp sera affiché afin que l'utilisateur en soit informé.

Pour finir, la fonction didTapConvertButton permet d'obtenir ou recharger le taux de change. Lorsque le bouton est cliqué :

- dismissKeyboard, le clavier n'est plus visible
- activitiyIndicator.startAnimating, lance une animation qui indique qu'une tâche est en cours de traitement. Récupérer les données du taux de change prend du temps, alors un indicateur d'activité est déclenché pour informer l'utilisateur que l'application est toujours en fonctionnement et qu'elle n'est pas gelée.
- getCurrency, les données à charger.

EXTENSION

```swift
import Foundation

private let numberFormatter = NumberFormatter()

//    to delete numbers after the comma
extension Double {
    func toTruncatedString() -> String {
        let number = NSNumber(value: self)
        numberFormatter.minimumFractionDigits = 0
        numberFormatter.maximumFractionDigits = 0
        return String(numberFormatter.string(from: number) ?? "\(self)")
    }

    func toTruncatedStringWithTwoDigits() -> String {
        let number = NSNumber(value: self)
        numberFormatter.minimumFractionDigits = 0
        numberFormatter.maximumFractionDigits = 2
        return String(numberFormatter.string(from: number) ?? "\(self)")
    }
}

```

L'extension Double contient deux fonctions plus ou moins similaires. Les deux ont pour objectif de supprimer ou réduire les chiffres après la virgule.

Pour exemple : 147,19543

Dans la première fonction, il est réduit à 147. Il est utilisé pour la météo pour obtenir un nombre entier.
Dans la deuxième fonction, il est réduit et arrondis à 147,20. Il est utilisé pour le taux de change, ou la valeur a besoin d'être plus précise que pour la météo par exemple.

```swift
extension UIViewController {
    func alertPopUp() {
        let alert = UIAlertController(title: "Erreur",
                                      message: "Désolé, le service est actuellement indisponible.",
                                      preferredStyle: .alert)
        alert.addAction(UIAlertAction(title: "OK",
                                      style: .default,
                                      handler: nil))
        self.present(alert, animated: true)
    }
}

```

L’extension UIViewController contient la fonction alertPopUp. C’est elle-même qui gère le cas d’échec.