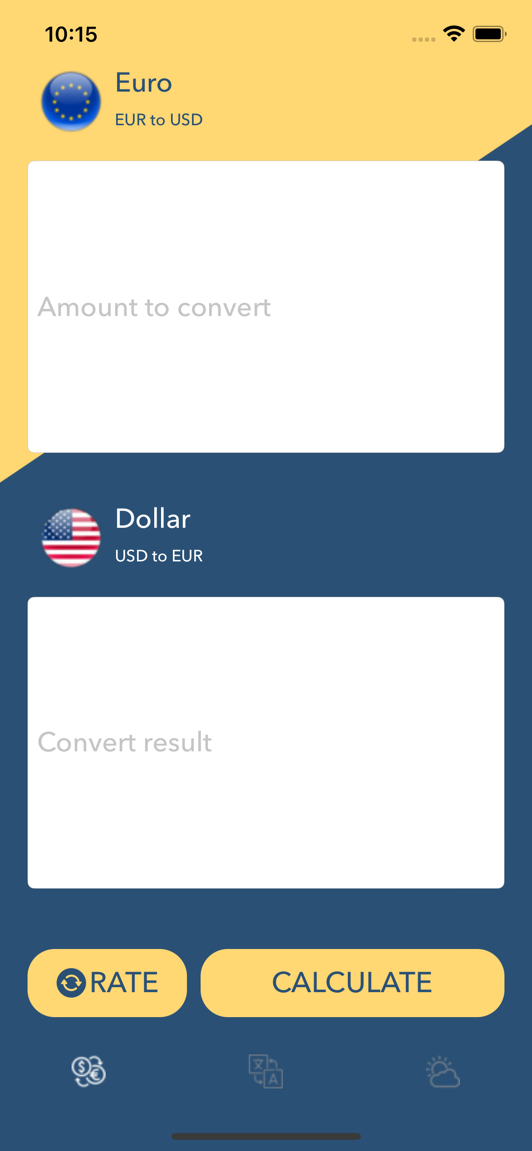
Contexte

|  |  |
| --- | --- |
| Projet 9 | Rapport  Présentation de la structure MVC et de leur communication dans l’application « Le Baluchon »  Mickaël HORN  Etudiant sur le parcours “Développeur d’applications iOS” - OpenClassrooms |

Avant toute chose, rappelons à quelle application nous avons à faire.

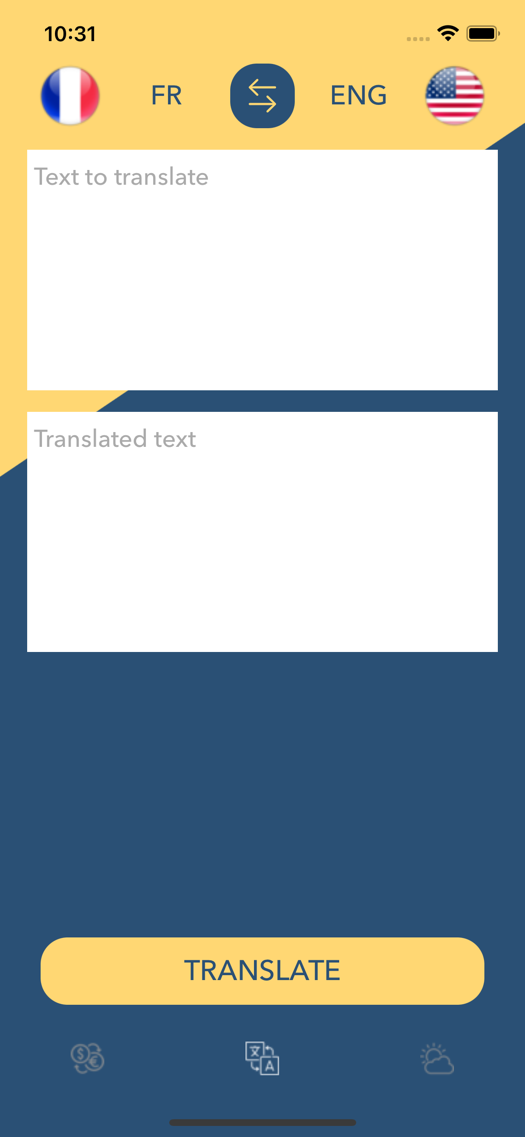
Le Baluchon est une application mobile iOS, codée en SWIFT proposant trois fonctionnalités.

1. **Taux de change**



Affichage du taux de change EUR/USD, qui sera utilisé lorsque l’utilisateur choisira un montant en euro et qu’il pressera le bouton « CALCULATE ». Il se verra alors afficher la conversion en dollars. Bien entendu, il pourra, à l’aide du bouton « RATE », obtenir une nouvelle fois le taux de change le plus récent.

1. **Traduction**



A partir d’une phrase de son choix (en français ou en anglais), l’utilisateur pourra presser le bouton « TRANSLATE » afin d’obtenir sa traduction.

Notez la présence du bouton composé de flèches en haut de l’écran, permettant d’échange la langue source et celle de destination, afin d’obtenir la traduction dans l’autre sens.

1. **Météo**

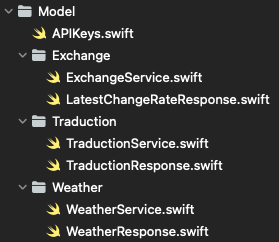


Pour les villes de Paris et New York, l’écran présentera la météo de ces deux villes, avec la possibilité de rafraîchir les données à l’aide du bouton « WEATHER ».

# Communications des MVC entre eux

Dans cette partie, je vais présenter la structure MVC de l’application et nous verrons comment ils communiquent entre eux.

## Les Modèles



Le dossier « Model » contient l’intégralité des modèles de l’application.

Le premier fichier, « APIKeys.swift », comme son nom l’indique, contient les clés API pour faire nos appels réseaux.

Ce fichier sensible a été mis dans notre gitignore afin de le garder secret.

Ensuite, chaque fonctionnalité a un dossier associé comprenant ses différents modèles.

Les fichiers « Service » contiennent les appels API alors que ceux « Response » disposent de des structures des fichier JSON récupérés lors des appels, afin de pouvoir les traiter.

### Exchange

Tout d’abord, voici les trois fichiers gérants les appels API, un fichier pour chaque fonctionnalité.

#### ExchangeService.swift

Une image contenant texte

Description générée automatiquement

#### TraductionService.swift

Une image contenant texte

Description générée automatiquement

#### WeatherService.swift

Une image contenant texte

Description générée automatiquement

Les fichiers étant très ressemblants, je vais les détailler ensemble.

L’utilisation d’un Singleton est justifiée pour gérer les requêtes concurrentes.

La variable « rate » est utilisée pour stocker le taux de change actuel, afin d’éviter de faire trop d’appels, et d’épuiser le quota disponible.

« baseUrl » contient l’URL de l’API, « task » lancera l’appel et « exchangeSession », combiné avec sa méthode dataTask, permettrons de rendre cet appel possible.

L’initialiseur, avec en paramètre un URLSession, pour les tests unitaires.

La fonction getLatestChangeRate() a pour rôle de lancer l’appel réseau.

getCompleteRequest() sera utilisé au sein même de getLatestChangeRate() afin de construire l’URL, c’est-à-dire lui ajouter l’Endpoint et les paramètres nécessaires.

La fonction getWeatherInformations() est utilisée pour récupérer la description du temps, ainsi que l’icône correspondante, afin d’afficher le tout dans la vue.

Intéressons-nous maintenant aux fichiers de type « Response ».

#### LatestChangeRateResponse.swift

Une image contenant texte

Description générée automatiquement

#### TraductionResponse.swift

Une image contenant texte, écran, capture d’écran

Description générée automatiquement

#### WeatherResponse

Une image contenant texte, mur, écran

Description générée automatiquementUne image contenant texte

Description générée automatiquement

Ils contiennent la structure reprenant le squelette du JSON que l’on recevra via l’appel API.

Nous devons le schématiser de la sorte afin de faire correspondre le JSON reçu avec le type d la variable de notre code dans laquelle on va le stocker.

## Les Vues

Une image contenant texte

Description générée automatiquement

Rien de particulier, le Launchscreen que vous avez pu observer en page de garde, ainsi que le Storyboard, dans lequel sont affichés les différents écrans de la partie « Contexte » de ce document.

## Les Contrôleurs

Une image contenant texte

Description générée automatiquement

Même logique ici, un contrôleur par fonctionnalité.

Notez la présence d’un fichier « Extension.swift », qui contient la fonction presentAlert(), commune à tous les contrôleurs pour la gestion des erreurs.