



# Weather

Une application de météo utilisant l'API OpenWeather





# Sommaire

- 1) Simulation de l'application
- 2) Présentation du pattern de conception
- 3) Démarche utilisée
- 4) La logique des modules
- 5) Le diagramme d'architecture
- 6) Présentation du code
- 7) Compétences maîtrisées



## SIMULATION DE L'APPLICATION



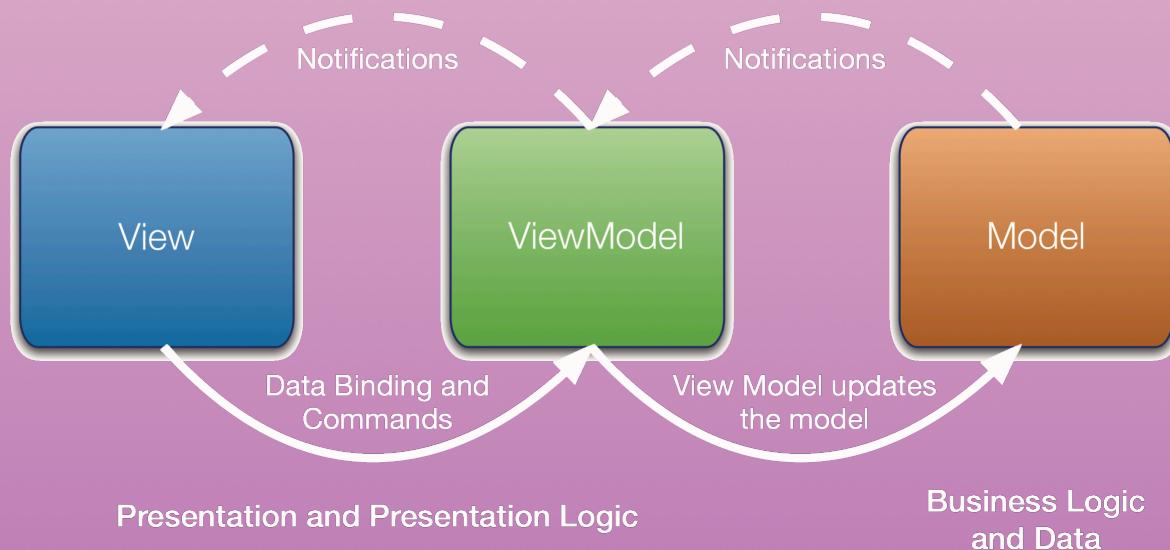


# QU'EST-CE QUE LE MODÈLE MVVM-C ?

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 connaît pas les données.





# Comment la logique communique avec la vue ?

**Chaque vue d'écran est construite à l'aide d'un ViewController et d'un ViewModel.**

- Le ViewController existe uniquement pour contrôler la vue, il est clair, simple et ne connaît pas les données. Le ViewController s'abonnent aux variables réactives du ViewModel par la fonction «bind ()», appelée dans «viewDidLoad ()».
- Le ViewModel encapsule toute la logique de chaque vue d'écran qui n'a pas besoin d'être dans le ViewController, divisée en deux parties:
  - inputs: La vue s'abonne aux variables réactives puis envoie la première entrée qui est souvent le «viewDidLoad ()»: la connexion d'entrée principale pour les variables réactives.
  - outputs: Ce sont les réceptacles d'événements des inputs correspondants.



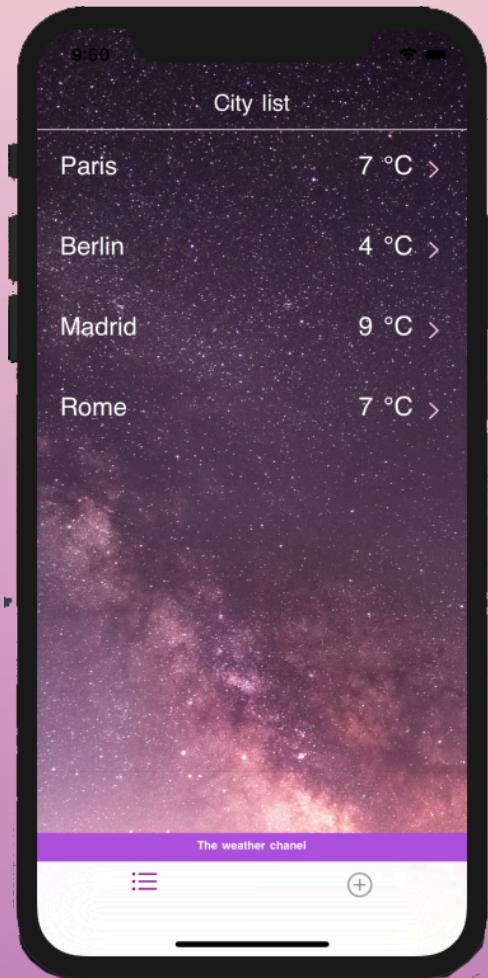
# LA DÉMARCHE UTILISÉE

- 1 - Creation de deux modules : “CityList” comprenant une vue detail “DetailDay” et “SelectCity” utilisant le Coordinator et le storyboard de chaque vue.
- 2 - Implementation de « UITabBar » avec 2 items:
  - . weather
  - . selectCity
- 3 - Creation du network avec un “HTTPClient”, un “HTTPEngine” et un “Token”
- 4 - Creation de la structure du model dans WeatherItem.swift en ajoutant les attributs de l’entity WeatherObject dans Core data.
- 5 – Écriture du WeatherRepository.swift
- 6 - Injection de Core data dans le context et le context dans l’AppDelegate.
- 7 - Unit tests
- 8 - UI tests



# Inputs / Outputs of each view

## CityListView



```
// MARK: - Output

var visibleWeatherItems: (([WeatherItem]) -> Void)?

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

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

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

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

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

// MARK: - Input

func viewDidLoad() {

}

func didSelectCity(at index: Int) {

}

func didPressDeleteCity(at index: Int) {
```



# Inputs / Outputs of each view

## WeekView



```
// MARK: - Outputs

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

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

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

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

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

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

// MARK: - Inputs

func viewDidLoad() {

}

func didSelectWeatherDay(at index: Int) {
```



# Inputs / Outputs of each view

## DetailDayView



```
// MARK: - Outputs

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

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

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

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

// MARK: - Inputs

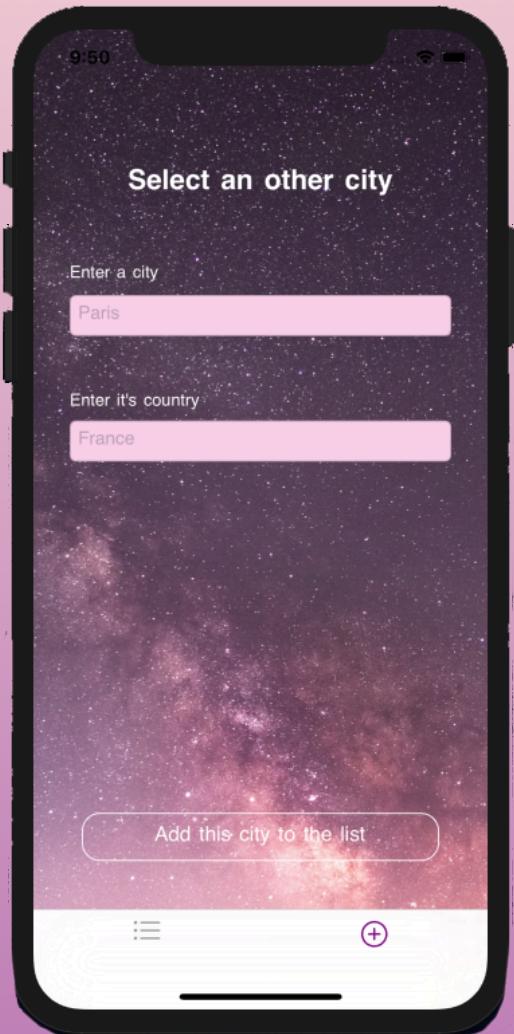
func viewDidLoad() {

}
```



# Inputs / Outputs of each view

## SelectCityView



```
// MARK: - Outputs

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

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

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

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

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

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

// MARK: - Inputs

func viewDidLoad() {

}

func addbuttonNormalState() {

}

func didPressAddCity(nameCity: String, country: String) {
```



## a) L'API OpenWeather

L'URL utilisée de l'API OpenWeather reçoit des données de météo de 5 jours toutes les 3 heures.

La structure du modèle «WeatherItem» a été créée pour afficher l'heure, la température, l'image de l'icône et plus, la température min / max, la pression, l'humidité, le ressenti et la description du ciel.



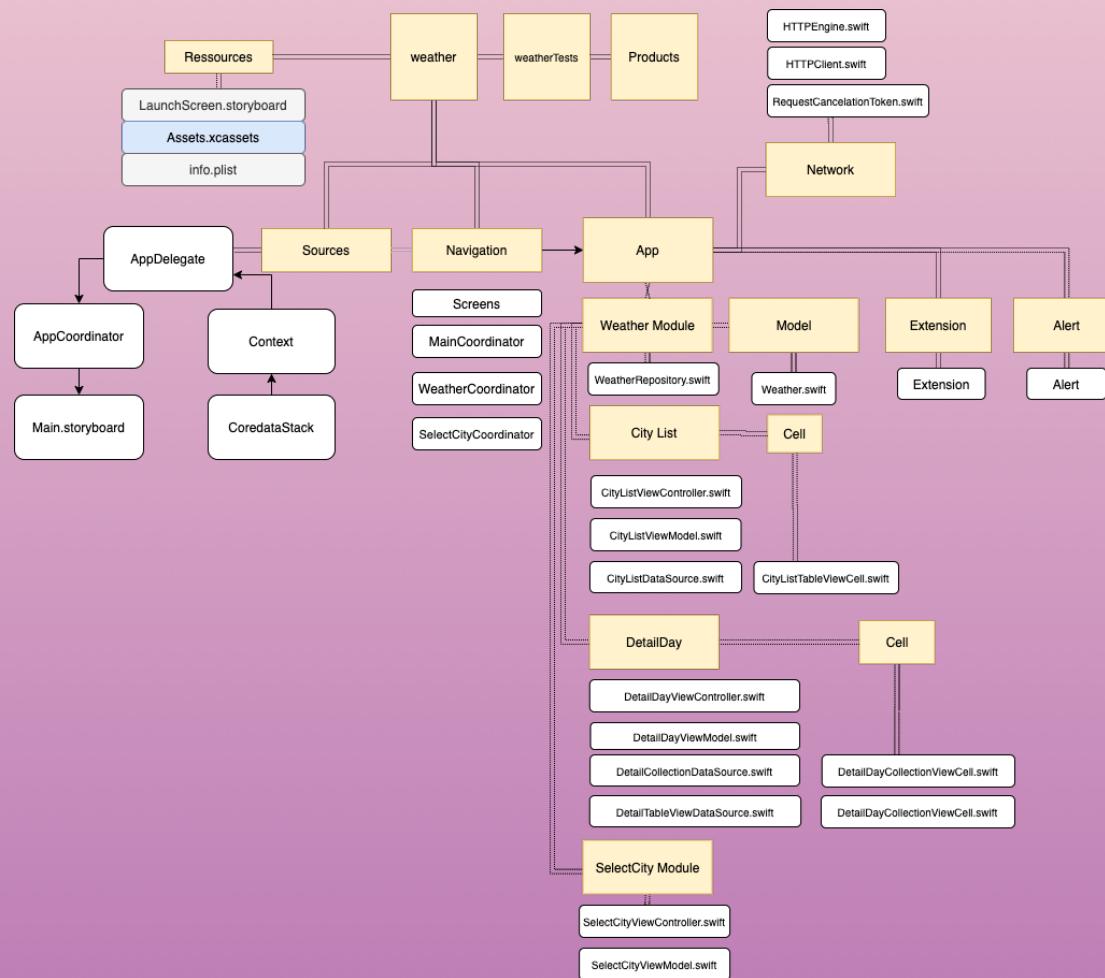
## b) Qu'est-ce que Core Data ?

Core Data est une base de données locale, dotée d'une API orientée objet.

Dans cette application, nous avions besoin de créer les objets:  
CityObject et WeatherObject dans le fichier «weather.xcdatamodeld» et d'ajouter  
leurs attributs.



# LE DIAGRAMME D'ARCHITECTURE





# PRÉSENTATION DU CODE





# COMPÉTENCES MAITRISÉES

**Perfectionnement de :**

- **MVVM avec Coordinator**
- **Requêtes réseaux avec une couche network et repository**
- **Affichage de tabBar Item sans storyboard**
- **TableView et CollectionView**
- **Persistence de données avec Core data**
- **Unit Tests avec utilisation de Mock**
- **UI Tests avec Protocol**



Merci pour votre attention