HAI912I : Programmation Mobile Avancée et IoT
Flutter

Le but de ces exercices et de vous familiariser avec les Future, Asynk, FutureBuilder, ainsi que la gestion des state en utilisant les Provider, et les Bloc (Business Logic Component).

Ainsi, vous serez amenés à organiser votre projet par package de responsabilité comme ça a été expliqué dans le cours : package presentation (avec des sous package : animations, pages, screens, widgets), package data (avec des sous package : dataproviders, models, repositories), et package business_logic (avec des sous package : blocs, cubits)

Application 1

Avec l'application « Questions/Réponses », vous avez créé un Quiz sur une thématique qui vous intéresse. Vous avez ainsi utilisé un widget de type « StatefulWidget » qui regroupe une image, une question ainsi qu'un ensemble de boutons.

La figure suivante montre à quoi devait ressembler l'application que vous aviez à créer :

Exercice 1 : Utilisation du Povider

Le premier exercice consiste à utiliser les « Povider » pour la gestion des états plus tôt que l'utilisation du setState.

Pour ce faire, voici quelques indications :

- Ajouter provider : ^6.0.1 dans le « pubspec.yml »
- Des classes que vous pouvez utiliser : ChangeNotifier, ChangeNotifierProvider, Consumer, Provider.of<ClassModel>(context, listen: false).method();
- Notifiez un changement en utilisant la méthode notifyListeners();
- Déclarez un ChangeNotifier comme suit : ChangeNotifierProvider(create: (context) => CartModel(), child: const MyApp(),)

Exercice 2 : Utilisation du Bloc/Cubit

Le deuxième exercice consiste à utiliser les Bloc/Cubit pour la gestion des états plus tôt que l'utilisation du setState.

Application 2

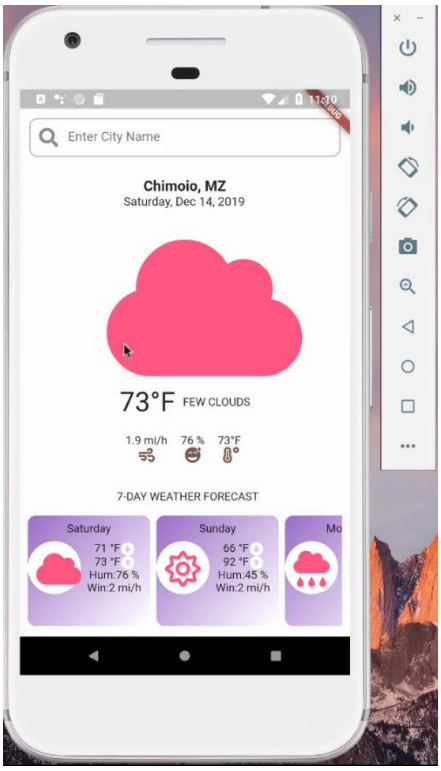
Exercice 1 : création d'application Météo

L'objectif de cet exercice est de vous familiariser avec les blocs/cubits, FutureBuilder, les constructeurs nommés « fromJson », etc.

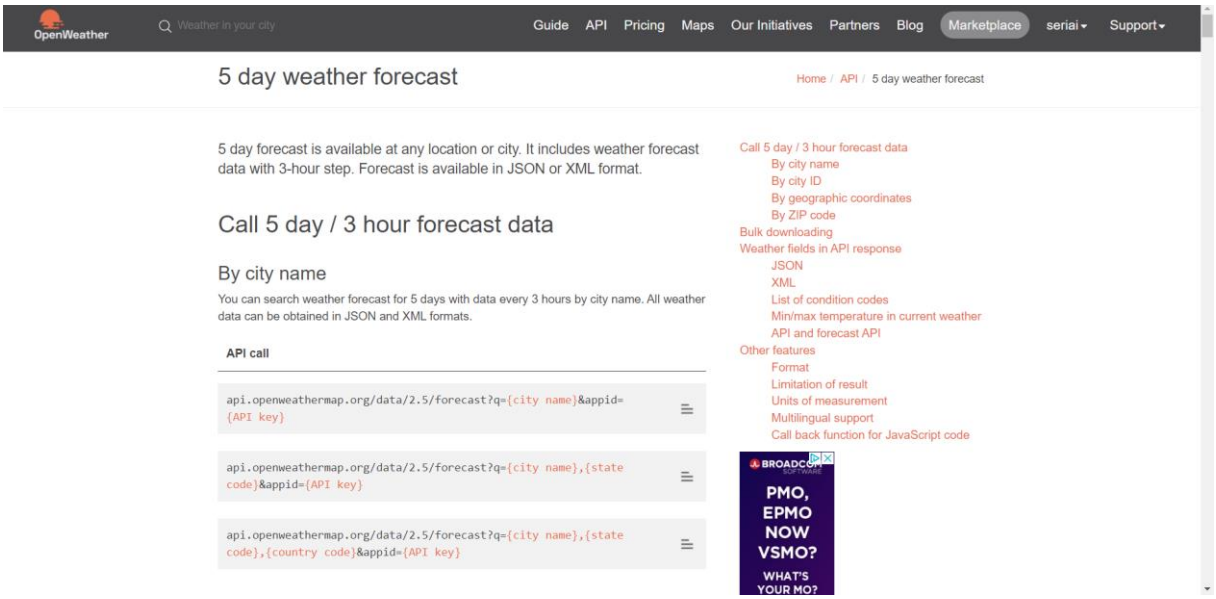
Ainsi, Avec l'application « Météo », vous allez être amenés à créer une application Flutter qui permet d'avoir la météo pour une ville donnée. Vous allez ainsi devoir créer un widget de type « StatefulWidget » qui regroupe trois parties :

- Une partie avec un « TextField »,
- Une partie avec le nom de la ville, la date, une icône ainsi que les informations concernant la température, la précipitation, et l'humidité
- Et une partie avec les prévisions pour au moins 3 ou 4jours

La figure suivante montre à quoi doit ressembler l'application que vous allez créer :



Avant de commencer le développement de votre application, il vous faut créer un compte sur « openweathermap.org », puis créer une clé d'api pour que vous puissiez utiliser les « endpoints rest » exposés par « openweathermap » (regardez la figure suivante)



Voici quelques indications pour vous aider à développer votre application :

- Commencez par créer une classe « WeatherModel» qui représente le modèle des donnés récupérés via l'api. Pour ça : utiliser le site

https://javiercbk.github.io/json_to_dart/ afin de transformer les données brutes en classe Dart comme suit :

- Créez une classe qui encapsule les appels réseau comme suit :

```dart
class Network {
  Future<WeatherForecastModel> getWeatherForecast({required String cityName}) async{


    var finalUrl ="https://api.openweathermap.org/data/2.5/forecast?q="+cityName+"&appid="+Util.appId;

    final response = await get(Uri.parse(finalUrl));
    print("URL: ${Uri.encodeFull(finalUrl)}");

    if (response.statusCode == 200) {
      // we get the actual mapped model ( dart object )
      print("weather data: ${response.body}");
        return WeatherForecastModel.fromJson(json.decode(response.body));
    }else {
      throw Exception("Error getting weather forecast");
    }
```

- Créez vos interfaces graphiques et n'oubliez surtout pas l'utilisation de « FutureBuilder ».
- Pour formater la date : utilisez la classe « DateFormat » du « package » « intl » : https://api.flutter.dev/flutter/intl/DateFormat-class.html

- Voici un ensemble de « package » dont vous pouvez avoir besoin dans votre application :

```
http: ^0.13.4
intl: ^0.17.0
font_awesome_flutter: ^9.1.0
```

- Vous pouvez utiliser cette méthode utilitaire afin d'afficher un icone adaptée à la description de la météo.

```
Widget getWeatherIcon({required String weatherDescription, required Color color, required double size}) {

  switch(weatherDescription) {
    case "Clear":
       { return Icon(FontAwesomeIcons.sun, color: color, size: size,); }
       break;
    case "Clouds":
      { return Icon(FontAwesomeIcons.cloud, color: color, size: size); }
      break;
    case "Rain":
      { return Icon(FontAwesomeIcons.cloudRain,color: color,size: size,);}
      break;
    case "Snow":
      {return Icon(FontAwesomeIcons.snowman,color: color,size: size,);}
      break;
    default: {return Icon(FontAwesomeIcons.sun,color: color,size: size,);}
    break;

  }
}
```