IUT Lyon 1
l'excellence technologique

DEPARTEMENT INFORMATIQUE
92 Bd Niels Bohr
69100 Villeurbanne

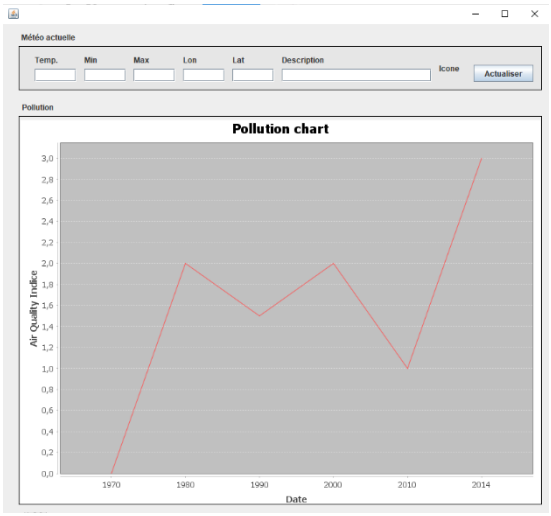
TP5 – Weather Station

R3.04 QUALITE DE DEVELOPPEMENT

I. INTRODUCTION

L'objectif de ce TP est la découverte de nouveaux modules permettant de récupérer des données depuis une API, manipuler ces données aux format JSON et de les stocker dans une base de données SQLite3. Pour cela, on utilisera des Managers et les Design Patterns vus dans les précédents TP, notamment MVC et Observer.

II. PRESENTATION DU PROJET



Le sujet du TP consiste à afficher, sous forme graphique, diverses informations obtenues via l'API *OpenWeatherMAP* (https://openweathermap.org/api).

Le projet de base est constitué de 2 vues, avec un bouton permet d'actualiser les données :

- Les données Météo (température, localisation, etc.)
- Les données de pollution (Indice de Qualité de l'air, concentration de certains polluants, etc.)

L'API est assez complète, mais tout n'est pas gratuit… Nous allons utiliser essentiellement les *Current Weather Data* (https://openweathermap.org/current) pour les informations météo, et Air Pollution API concept (https://openweathermap.org/api/air-pollution) pour les prévisions sur la pollution de l'air.

Création de compte : Si ce n'est pas déjà fait : créer un compte sur https://home.openweathermap.org et activer votre compte pour avoir accès à une clé API.

Mots-clés : NetWork Manager, DB Manager, JSON

III. TRAVAIL PRELIMINAIRE

Relire le code et tracer le diagramme de classe correspondant en détectant l'ensemble des Design Pattern existant (MVC, Callback, …), des spécificités de mise en place des threads et de chercher les endroits de connexion à l'API.

IV. TRAVAIL A REALISER : METEO

IV.1 BULLETIN METEO ET REQUETE HTTPS

La vue supérieure permet l'affichage des données Météo.

Le Modèle est géré par la classe WeatherReport, dans lesquelles on va stocker les données. Au départ, ces informations sont vides ou nulles. Elles seront récupérées via l'API pour être stockées dans les différents champs.

Pour accéder à OpenWeatherMap, il faut utiliser un objet de la classe HttpClient pour émettre une requête GET avec un objet HttpRequest.

L'url sera stockée dans un objet de type URI, par exemple (changer les *xxxx* avec votre *clé API*):
https://api.openweathermap.org/data/2.5/weather?q=lyon,fr&units=metric&lang=fr&appid=xxxx

```
URI url = URI.create(requestUrl);

// create a client
var client = HttpClient.newHttpClient();

// create a request
var request = HttpRequest.newBuilder(url)
    .header("accept", "application/json")
    .build();

// use the client to send the request
var response = client.send(request, HttpResponse.BodyHandlers.ofString());
```

Il est aussi possible de tester l'état de la réponse :

```
if (response.statusCode() == HttpURLConnection.HTTP_OK) {
    String responseText = response.body();

}
```

IV.2 DOCUMENT JSON

Ensuite, il faudra traiter les données au format JSON. Le document JSON contiendra des éléments structurels avec des ensembles de paires « nom » (alias « clé ») / « valeur » : "id": "file"

Ces mêmes éléments représentent trois types de données :

1. des objets : { ... } ;
2. des tableaux : [...] ;
3. des valeurs génériques de type tableau, objet, booléen, nombre, chaîne de caractères ou null (valeur vide).

Voir l'exemple de format JSON en annexe.

On commence par récupérer l'Objet JSON contenant la réponse

```
JSONObject jsonObj = new JSONObject(textJSON);
```

On peut accéder aux différents champs ainsi, par exemple pour la valeur du champ "temp" contenu dans l'objet "main" :

```
JSONObject mainJSON = jsonObj.getJSONObject("main");
double temp = mainJSON.getDouble("temp");
```

et pour les tableaux :

```
JSONArray weather = (JSONArray) jsonObj.getJSONArray("weather");
for (int i=0; i < list.length() && i< 40; i+=5) {
     System.out.println(list.getJSONObject(i));

}
```

IV.3 MISE A JOUR : MODELE ET VUE

Ajoutez une méthode pour mettre à jour le Modèle (la classe WeatherReport) depuis la méthode de traitement de la réponse à la requête (onWorkDone(JSONObjects objects)). Nous utilisons ici un *DP Callback* pour avertir le modèle à se mettre à jour quand le traitement a été réalisé par un thread.

Connecter le bouton « Actualiser » pour demander la mise à jour des données et donc de la vue par la mise en place du DP Observer. Dans ce cas, c'est les classes *ConnexionManager* et *ConnexionThread* qui feront office de *Controller*. Ce sont elles qui seront chargées de gérer les erreurs et de faire les contrôles.

V. TRAVAIL A REALISER : POLLUTION

On va maintenant traiter les données de prévision de la pollution, notamment l'indice de qualité de l'air.

V.1 MISE A JOUR : MODELE ET VUE

Créer une nouvelle méthode dans le *ConnexionManager* **loadpollution**() sur le même modèle que le bulletin Météo. Elle sera connectée sur le même bouton « Actualiser », qui lancera donc les 2 requêtes en simultanée.

Les requêtes se faisant en asynchrone, rajoutez un 2ème ConnexionThread. Et faites une nouvelle requête, cette fois sur l'adresse suivante :

https://api.openweathermap.org/data/2.5/air_pollution/forecast?lat=46.0398&lon=5.4133&units=metric&lang=fr&appid=XXXX

Dans la réponse lors de l'appel du Callback, vous allez maintenant recevoir une liste de prévision (la requête précédente contenait aussi une liste, mais avec un seul élément, les informations sur la météo actuelle). Ici, vous allez avoir des prévisions par heure pour les X prochaines heures.

Récupérez les données *aqi* (Air Quality Indice) pour chaque *dt* (DateTime) de la liste. Les *dt* sont en secondes, au format Unix, UTC. Il faudra convertir au format *Date* en n'oubliant pas de passer les données en millisecondes (x1000) pour pouvoir ensuite les formater en fonction de vos envies avec *SimpleDateFormat*.

```
long epoch = ((long)dt)*1000; // to long and  millisecond
Date date = new Date(epoch);

String pattern = "MM-dd-yyyy ";
SimpleDateFormat simpleDateFormat = new SimpleDateFormat(pattern);
String dateString = simpleDateFormat.format(date);
```

V.2 Enregistrement BD

On va maintenant enregistrer ces données dans une **BD SQlite**. Pour cela, on va utiliser la classe QsqlDatabase. Le projet de base fournit une classe DBManager qui va vous aider à manipuler les données.

Dans le constructeur, on initialise la base en créant une connexion avec l'url locale de la BD :

```
Connection con;
con = DriverManager.getConnection(url);
```

puis on crée les Tables (ici, une seule avec 2 champs, dt et aqi), avec une requête préparée :

```
String query = "CREATE TABLE IF NOT EXISTS pollution(id INTEGER PRIMARY KEY,
dt INTEGER, aqi INTEGER);";
try (Statement statement = con.createStatement();) {
    int number = statement.executeUpdate(query);
}
catch (SQLException e) {
    System.out.println("Error : "+ e.getMessage());
}
```

Étudiez aussi les autres méthodes de la classe DBManager, notamment bool **addData**(int dt, int aqi) qui vous sera utile pour ajouter des entrées dans la Table.

Un exemple d'utilisation est proposé en commentaire dans le main() avec des liens d'utilisation de la BD.

V.3 MISE A JOUR : VUE LINE

La classe *ViewPollution* permet un affichage en ligne des valeurs. Dans le projet de base, l'initialisation du Widget est fournie, avec affichage de données fictives depuis une *QlineSeries*… Elle fonctionne comme les autres *ChartPanel*. Ce qui change ici est l'axe des X, qui représente des *Date*, qu'on affiche seulement en *heures:minutes*.

Ajouter la récupération des valeurs depuis la BD, et le DP Observer pour mettre à jour les données à chaque actualisation.

VI. AMELIORATIONS

Vous pouvez explorer les différentes possibilités offertes par ce logiciel en améliorant divers points :

- Offrir la possibilité de changer la localisation des données via l'interface.
- Afficher d'autres informations Météo.
- Tracer les courbes pour les concentrations de polluants (taux de CO, NO3, SO2, etc...) dans la Vue Pollution.
- Choisir l'intervalle de temps pour les données pollution (et aller chercher sur OpenWeatherMap les données manquantes dans la BD, si besoin).
- Modifier les Charts (changer de **thème**, ajouter des **animations,** montrer les valeurs au **survol du graphique**, etc.).

VII. ANNEXE : REPONSE DE L'API


```
{
 "coord": {
  "lon": 10.99,
  "lat": 44.34
 },
 "weather": [
  {
   "id": 501,
   "main": "Rain",
   "description": "moderate rain",
   "icon": "10d"
  }
 ],
 "base": "stations",
 "main": {
  "temp": 298.48,
  "feels_like": 298.74,
  "temp_min": 297.56,
  "temp_max": 300.05,
  "pressure": 1015,
  "humidity": 64,
  "sea_level": 1015,
  "grnd_level": 933
 },
 "visibility": 10000,
 "wind": {
  "speed": 0.62,
  "deg": 349,
  "gust": 1.18
 },
 "rain": {
  "1h": 3.16
 },
 "clouds": {
  "all": 100
 },
 "dt": 1661870592,
 "sys": {
  "type": 2,
  "id": 2075663,
  "country": "IT",
  "sunrise": 1661834187,
  "sunset": 1661882248
 },
 "timezone": 7200,
 "id": 3163858,
 "name": "Zocca",
 "cod": 200
}
```