DÉPARTEMENT INFORMATIQUE
71 rue Peter Fink
01000 Bourg-en-Bresse
04 74 45 50 59

TP5 – Weather/Pollution Report

R3.04 QUALITÉ DE DÉVELOPPEMENT

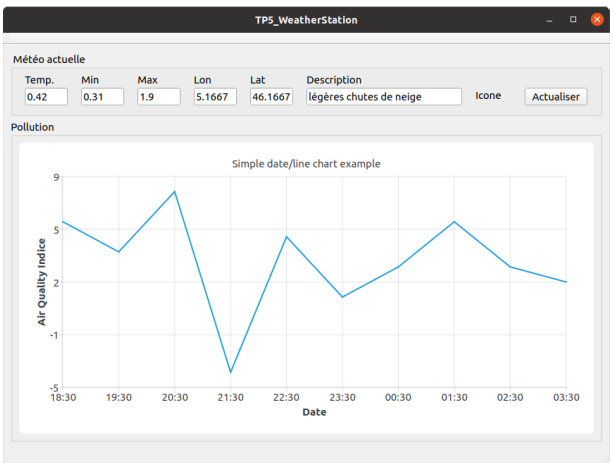
Adrien Peytavie – Fabrice Jaillet
adrien.peytavie@univ-lyon1.fr – fabrice.jaillet@univ-lyon1.fr

Table des matières

I INTRODUCTION

L'objectif de ce TP est la découverte de nouveaux composants Qt, 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 Manager et les Design Patterns vus dans les précédents TP, notamment MVC et Observer.

II PRÉSENTATION 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.

Mots-clés : NetWork Manager, DB Manager, Json

III TRAVAIL À RÉALISER: MÉTÉO

III.1 *BULLETIN MÉTÉO ET REQUÊTE 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 QNetworkAccessManager pour émettre une requête GET avec un objet QNetworkRequest.

L'url sera stocké dans un objet de type Qurl.

```
netmanager = new QNetworkAccessManager(this);
connect(netmanager, SIGNAL(finished(QNetworkReply*)),
        this, SLOT(weatherReplyFinished(QNetworkReply*)));

void TP5_WeatherStation::weatherRequest() {
{
    // Pour tester les entêtes HTTP
       QString URL = "https://api.openweathermap.org/data/2.5/weather?q=bourg-en-
bresse,fr&units=metric&lang=fr&appid=xxxx";
    QUrl url(URL);
    QNetworkRequest request;
    request->setUrl(url);
    //--header 'Accept: application/json'
    request->setRawHeader("Accept", "application/json");
    qDebug() << Q_FUNC_INFO << request->url();
    netmanager->get(request);
}
```

On obtiendra la réponse à la requête par le signal finished(QNetworkReply*) qui passera l'adresse d'un objet QNetworkReply. On accédera au contenu de la réponse en appelant la méthode readAll(). Une fois la requête traitée, ne pas oublier de demander la destruction de reply et du manager (deleteLater() de Qt).

```
void TP5_WeatherStation::weatherReplyFinished(QNetworkReply *reply)
{
    QByteArray datas = reply->readAll();
    QString infos(datas);
    //...
    reply->deleteLater();
}

// dans destructeur
netmanager->deleteLater();
```

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

```
if (reply->error() != QNetworkReply::NoError)
    {
       //Network Error
        qDebug() << reply->error() << "=>" << reply->errorString();
    }
     else if ( reply->attribute(QNetworkRequest::HttpStatusCodeAttribute).toInt() ==
200)
    { // do something good here } else { // failed to connect to API }
```

III.2 *Document JSon*

Ensuite, il faudra traiter les données au format JSon. Le document Json contiendra des éléments structurels : 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).

Exemple de format JSON en annexe.

On commence par récupérer l'Objet Json contenant la réponse

```
        QJsonDocument jsonResponse = QJsonDocument::fromJson(infos);
        QJsonObject jsonObj = jsonResponse.object();
```

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

```
        QJsonObject mainObj = jsonObj["main"].toObject();
        temp = mainObj["temp"].toDouble();
```

et pour les tableaux :

```
    QJsonArray weatherArray = jsonObj["weather"].toArray();
    for (auto w:weatherArray) {
        qDebug() << w.toObject() ... // Pour tester le type ou les valeurs
    }
```

III.3 *Mise à jour : modèle 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 (weatherReplyFinished).

Connecter le bouton « Actualiser » à weatherRequest() et mettre en place le DP Observer pour mettre la vue à jour (ViewReport).

Dans ce cas, ce sont les méthodes weatherRequest() et weatherReplyRequest() qui font office de Controller. C'est elles qui seront chargées de gérer les erreurs et de faire les contrôles, par exemple sur la validité des données JSon.

IV TRAVAIL À RÉALISER: POLLUTION

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

IV.1 *MISE À JOUR : MODÈLE ET VUE*

Créer une nouvelle méthode void TP5_WeatherStation::**pollutionRequest**() sur le même modèle que pour 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 QnetworkAccessManager. Et faites une nouvelle requête, cette fois sur "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 (void TP5_WeatherStation::**pollutionReplyFinished**(QNetworkReply* reply)), 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. Pas de Panique, Qt sait les convertir en format intelligible :

```
QdateTime localTime QdateTime::fromSecsSinceEpoch(dt); // s to local
qint64 msdt = localTime.toMSecsSinceEpoch(); // local to ms
```

IV.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 :

```
    sqldb = QSqlDatabase::addDatabase("QSQLITE");
    sqldb.setDatabaseName(path);

    if (!sqldb.open()) { ... }
```

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

```
    QSqlQuery query;
    query.prepare("CREATE TABLE pollution(id INTEGER PRIMARY KEY, dt INTEGER, aqi
INTEGER);");

    if (!query.exec()) { . . . }
```

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

Un exemple d'utilisation est proposé dans le main().

IV.3 *Mise à Jour : Vue Line*

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

```
    axisX = new QDateTimeAxis;
    axisX->setTickCount(10);
    axisX->setFormat("HH:mm");
```

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

Attention ! Bien penser à recalculer axisX->setRange(xmin,xmax) et axisY→setRange(ymin,ymax) avant de faire le widget→repaint() !

V POUR LES PLUS AVANCÉS

Vous pouvez explorer les possibilités offertes par les Charts (changer de **thème**, ajouter des **animations,** montrer les valeurs au **survol du graphique**, etc.).

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)

Offrir la possibilité de changer la localisation des données via l'interface.

VI ANNEXE : RÉPONSE 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
}
```