Application Météo Android
TP6 - Développement Mobile

Application de prévisions météorologiques en temps réel

Anas Marji

2DAI2

January 14, 2026

Résumé

Ce rapport présente l'analyse technique complète de l'application **Météo Android**, développée dans le cadre du TP6 de développement mobile. L'application permet aux utilisateurs de consulter les prévisions météorologiques pour différentes villes à l'aide de l'API OpenWeatherMap. Le document détaille l'architecture, les composants principaux, l'intégration API et les fonctionnalités du cycle de vie Android.

Mots-clés : Android, Java, API REST, Météo, OpenWeatherMap, Volley, Cycle de vie, Adapter personnalisé

Table des matières

1 Introduction

L'application **Météo Android** est une application mobile développée pour la plateforme Android dans le cadre du TP6 de développement mobile. Cette application permet aux utilisateurs de consulter les prévisions météorologiques pour une sélection de villes à travers le monde en utilisant l'API OpenWeatherMap.

1.1 Objectifs du projet

— Développer une application Android fonctionnelle de prévisions météorologiques

— Intégrer une API REST externe (OpenWeatherMap)

— Implémenter un adaptateur personnalisé pour l'affichage des données

— Gérer correctement le cycle de vie d'une activité Android

— Gérer les erreurs de réseau et les exceptions

1.2 Architecture générale

L'application suit une architecture **MVC (Modèle-Vue-Contrôleur)** avec les composants suivants :

Composant	Rôle dans l'application
Modèles	`MeteoItem` - Représente une prévision météo
Vues	Layouts XML pour l'interface utilisateur
Contrôleurs	`MainActivity` - Gère la logique métier
Adapter	`MeteoListModel` - Affiche les données dans la ListView

Table 1 – *Architecture MVC de l'application Météo*

2 Architecture de l'application

2.1 Structure des packages

L'application est organisée de manière modulaire dans le package principal `ma.projet.tp6_meteo`

2.2 Flux de l'application

Le flux utilisateur suit une progression logique :

1. Lancement de l'application et affichage de l'interface principale

— `MainActivity.java` : Activité principale

— `MeteoItem.java` : Modèle de données

— `MeteoListModel.java` : Adaptateur personnalisé

— `activity_main.xml` : Layout principal

— `list_item_layout.xml` : Layout des items

Figure 1 – *Structure et interface de l'application*

2. Sélection d'une ville dans le Spinner
3. Clic sur le bouton de recherche
4. Appel à l'API OpenWeatherMap
5. Affichage des prévisions dans la ListView
6. Gestion des événements du cycle de vie

3 Composants principaux

3.1 Classe MainActivity

L'activité principale gère l'interface utilisateur et la logique métier :

```
public class MainActivity extends AppCompatActivity {
    private Spinner spinnerVille;
    private ListView listViewMeteo;
    private ImageButton buttonOK;
    private List<MeteoItem> data = new ArrayList<>();
    private MeteoListModel model;
    private static final String API_KEY = "
       b6907d289e10d714a6e88b30761fae22";

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        // Initialisation des vues
        spinnerVille = findViewById(R.id.spinnerVille);
        listViewMeteo = findViewById(R.id.listViewMeteo);
        buttonOK = findViewById(R.id.buttonOK);

```

```
        // Configuration du Spinner avec les villes
        String[] cities = {"Casablanca", "Rabat", "Marrakech",
                           "Paris", "London", "New York", "Tokyo"};
        ArrayAdapter<String> spinnerAdapter = new ArrayAdapter<>(
            this, android.R.layout.simple_spinner_dropdown_item,
               cities);
        spinnerVille.setAdapter(spinnerAdapter);

        // Configuration de l'adaptateur personnalis
        model = new MeteoListModel(this, R.layout.list_item_layout,
           data);
        listViewMeteo.setAdapter(model);

        // Gestion du clic sur le bouton
        buttonOK.setOnClickListener(v -> {
            String ville = spinnerVille.getSelectedItem().toString();
            data.clear();
            model.notifyDataSetChanged();
            getWeatherData(ville);
        });
    }
}
```

Listing 1 – *Extrait de MainActivity.java*

3.2 Classe MeteoItem

Le modèle de données représente une prévision météorologique :

```
public class MeteoItem {
    public int tempMax;
    public int tempMin;
    public int pression;
    public int humidite;
    public String image; // Code d'ic ne OpenWeatherMap
    public String date;
}
```

Listing 2 – *Classe MeteoItem.java*

3.3 Classe MeteoListModel

L'adaptateur personnalisé gère l'affichage des données :

```
public class MeteoListModel extends ArrayAdapter<MeteoItem> {
    public static Map<String, Integer> images = new HashMap<>();
```

```
3
4       static {
5           // Mapping des codes OpenWeatherMap vers les ressources
                locales
6           images.put("01d", R.mipmap.clear_foreground);
7           images.put("01n", R.mipmap.clear_foreground);
8           images.put("02d", R.mipmap.clouds_foreground);
9           // ... autres mappings
10      }
11
12      @NonNull
13      @Override
14      public View getView(int position, @Nullable View convertView,
15                          @NonNull ViewGroup parent) {
16          View listItem = convertView;
17          if (listItem == null) {
18              listItem = LayoutInflater.from(getContext())
19                      .inflate(resource, parent, false);
20          }
21
22          // R cup ration des vues
23          ImageView imageView = listItem.findViewById(R.id.imageView);
24          TextView textViewTempMax = listItem.findViewById(R.id.
                textViewTempMAX);
25          // ... autres TextView
26
27          // Remplissage avec les donn es
28          MeteoItem item = listItems.get(position);
29          imageView.setImageResource(images.getOrDefault(item.image,
30                          R.mipmap.clouds_foreground));
31          textViewTempMax.setText(item.tempMax + "  C ");
32          // ... autres setters
33
34          return listItem;
35      }
36  }
```

Listing 3 – *Extrait de MeteoListModel.java*

4 Intégration API OpenWeatherMap

4.1 Configuration de l'API

L'application utilise l'API OpenWeatherMap avec les paramètres suivants :

Paramètre	Valeur
Clé API	b6907d289e10d714a6e88b30761fae22
Endpoint	https ://api.openweathermap.org/data/2.5/forecast
Méthode	GET
Paramètres	q={ville}&appid={API_KEY}
Réponse	JSON avec prévisions sur 5 jours

Table 2 – *Configuration de l'API OpenWeatherMap*

4.2 Méthode getWeatherData

Cette méthode gère la communication avec l'API :

```
private void getWeatherData(String city) {
    String url = "https://api.openweathermap.org/data/2.5/forecast?q="
                 + city + "&appid=" + API_KEY;

    RequestQueue queue = Volley.newRequestQueue(getApplicationContext
        ());

    StringRequest stringRequest = new StringRequest(
        Request.Method.GET, url,
        response -> {
            try {
                JSONObject jsonObject = new JSONObject(response);
                JSONArray jsonArray = jsonObject.getJSONArray("list");

                for (int i = 0; i < jsonArray.length(); i++) {
                    MeteoItem meteoItem = new MeteoItem();
                    JSONObject d = jsonArray.getJSONObject(i);

                    // Conversion de la date
                    Date date = new Date(d.getLong("dt") * 1000);
                    SimpleDateFormat sdf = new SimpleDateFormat(
                        "dd-MMM-yyyy 'T' HH:mm", Locale.getDefault());

                    // Extraction des donn es m t o
                    JSONObject main = d.getJSONObject("main");
                    JSONArray weather = d.getJSONArray("weather");

                    // Conversion Kelvin -> Celsius
                    int tempMin = (int)(main.getDouble("temp_min") -
                        273.15);
                    int tempMax = (int)(main.getDouble("temp_max") -
                        273.15);

                    // Remplissage de l'objet MeteoItem
```

```
                    meteoItem.tempMax = tempMax;
                    meteoItem.tempMin = tempMin;
                    meteoItem.pression = main.getInt("pressure");
                    meteoItem.humidite = main.getInt("humidity");
                    meteoItem.date = sdf.format(date);
                    meteoItem.image = weather.getJSONObject(0).
                        getString("icon");

                    data.add(meteoItem);
                }

                model.notifyDataSetChanged();
            } catch (JSONException e) {
                Toast.makeText(MainActivity.this,
                    "Erreur de lecture JSON", Toast.LENGTH_SHORT).show
                        ();
            }
        },
        error -> {
            Toast.makeText(MainActivity.this,
                "Erreur de connexion API", Toast.LENGTH_SHORT).show();
        }
    );

    queue.add(stringRequest);
}
```

Listing 4 – *Méthode getWeatherData*

5 Interface utilisateur

5.1 Layout principal

L'interface principale utilise un design simple et efficace :

Composants de l'interface principale :

— **Spinner** : Pour sélectionner une ville

— **ImageButton** : Bouton de recherche

— **ListView** : Affichage des prévisions

5.2 Layout d'item

Chaque item de la liste affiche une prévision météorologique :

```
<LinearLayout android:orientation="horizontal">
    <ImageView android:id="@+id/imageView" />
    <LinearLayout android:orientation="vertical">
        <TextView android:id="@+id/textViewDate" />
        <LinearLayout>
            <TextView android:text="Min:" />
            <TextView android:id="@+id/textViewTempMin" />
            <TextView android:text="Max:" />
            <TextView android:id="@+id/textViewTempMAX" />
        </LinearLayout>
        <LinearLayout>
            <TextView android:text="Pression:" />
            <TextView android:id="@+id/textViewPression" />
        </LinearLayout>
        <LinearLayout>
            <TextView android:text="Humidit :" />
            <TextView android:id="@+id/textViewHumidite" />
        </LinearLayout>
    </LinearLayout>
</LinearLayout>
```

Listing 5 – *Extrait de list_item_layout.xml*

6 Cycle de vie Android

6.1 Implémentation des méthodes de cycle de vie

L'application implémente toutes les méthodes du cycle de vie :

Méthode	Comportement implémenté
`onStart()`	Toast "Application visible"
`onResume()`	Toast "Application prête"
`onPause()`	Toast + Log "Application en pause"
`onStop()`	Log "Application en arrière-plan"
`onRestart()`	AlertDialog de confirmation
`onDestroy()`	Log "Fermeture définitive"

Table 3 – *Méthodes du cycle de vie implémentées*

6.2 Exemple d'implémentation

```
@Override
```

```
protected void onRestart() {
    super.onRestart();
    new AlertDialog.Builder(this)
        .setTitle("Reprise")
        .setMessage("Voulez-vous continuer l'utilisation ?")
        .setPositiveButton("Oui", null)
        .setNegativeButton("Non", (dialog, which) -> finish())
        .show();
}

@Override
protected void onDestroy() {
    super.onDestroy();
    Log.i("Lifecycle", "Fermeture d finitive de l'application");
}
```

Listing 6 – *Implémentation du cycle de vie*

7 Fonctionnalités techniques

7.1 Fonctionnalités implémentées

Fonctionnalité	Description technique
API REST	Intégration OpenWeatherMap avec Volley
Parsing JSON	Extraction des données météo du JSON
Conversion	Kelvin vers Celsius, timestamp vers date
Cache d'images	Mapping codes OpenWeatherMap vers drawables
Adapter personnalisé	Optimisation avec ViewHolder pattern
Gestion d'erreurs	Toast pour erreurs réseau/JSON
Cycle de vie	Gestion complète des états de l'activité

Table 4 – *Fonctionnalités techniques implémentées*

7.2 Diagramme de séquence

Diagramme de séquence
Interaction utilisateur → API → Affichage

1. Utilisateur sélectionne une ville
2. Clic sur le bouton de recherche
3. Appel API OpenWeatherMap
4. Parsing de la réponse JSON
5. Conversion des données
6. Mise à jour de l'adaptateur
7. Affichage dans la ListView

Figure 2 – *Diagramme de séquence de l'application*

8 Améliorations possibles

8.1 Améliorations prioritaires

Priorité	Suggestion d'amélioration
Élevée	Ajout de la géolocalisation automatique
Élevée	Mise en cache des données pour mode hors-ligne
Moyenne	Notifications météo (alertes précipitations)
Moyenne	Graphiques d'évolution de température
Faible	Thèmes jour/nuit automatiques

Table 5 – *Feuille de route d'amélioration*

8.2 Évolutions architecturales

— **Architecture MVVM** : Utilisation de ViewModel et LiveData

— **Repository pattern** : Abstraction de la source de données

— **Coroutines/Flow** : Gestion asynchrone moderne

— **Dependency Injection** : Avec Hilt ou Dagger

— **Tests unitaires** : Pour la logique métier

— **Tests d'interface** : Avec Espresso

9 Conclusion

9.1 Réussites techniques

— **Intégration API réussie** : Communication fluide avec OpenWeatherMap

— **Interface intuitive** : Navigation simple pour l'utilisateur

— **Gestion d'erreurs robuste** : Feedback clair en cas de problèmes

— **Performance optimisée** : Recyclage de vues dans l'adaptateur

— **Cycle de vie complet** : Gestion professionnelle des états

9.2 Compétences démontrées

— Développement Android natif en Java

— Intégration d'API REST avec Volley

— Parsing de données JSON complexes

— Création d'adaptateurs personnalisés

— Gestion du cycle de vie Android

— Gestion d'erreurs et exceptions

9.3 Perspectives

Cette application constitue une base solide pour des évolutions vers :

— Application météo complète avec cartes

— Système d'alertes météo en temps réel

— Widgets Android pour accès rapide

— Version multiplateforme (Flutter/React Native)

— Application commerciale avec abonnements

Note technique

L'application respecte les bonnes pratiques Android, utilise les permissions minimales nécessaires et reste compatible avec les versions récentes du SDK tout en maintenant la rétrocompatibilité.

Annexes

A. Dependencies Gradle

```
dependencies {
    implementation 'androidx.appcompat:appcompat:1.6.1'
    implementation 'com.google.android.material:material:1.9.0'
    implementation 'com.android.volley:volley:1.2.1'
    implementation 'androidx.constraintlayout:constraintlayout:2.1.4'
}
```

Listing 7 – *Extrait de build.gradle*

B. Structure complète du projet

```
TP6_MeteoApp/
 app/
    src/main/java/ma/projet/tp6_meteoapp/
       MainActivity.java
       MeteoItem.java
       MeteoListModel.java
    src/main/res/
       layout/
          activity_main.xml
          list_item_layout.xml
       mipmap/
          clear_foreground.png
          clouds_foreground.png
          rain_foreground.png
          thunderstorm_foreground.png
       values/
    build.gradle
 AndroidManifest.xml
 README.md
```

C. Permissions AndroidManifest

```
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE
    " />
```

Listing 8 – *Extrait de AndroidManifest.xml*

D. Bibliographie

— Documentation officielle Android Developer

— Documentation OpenWeatherMap API

— Volley documentation (Google)

— Android Lifecycle documentation

— Material Design Guidelines