Prep' etna

Programmation mobile: Android

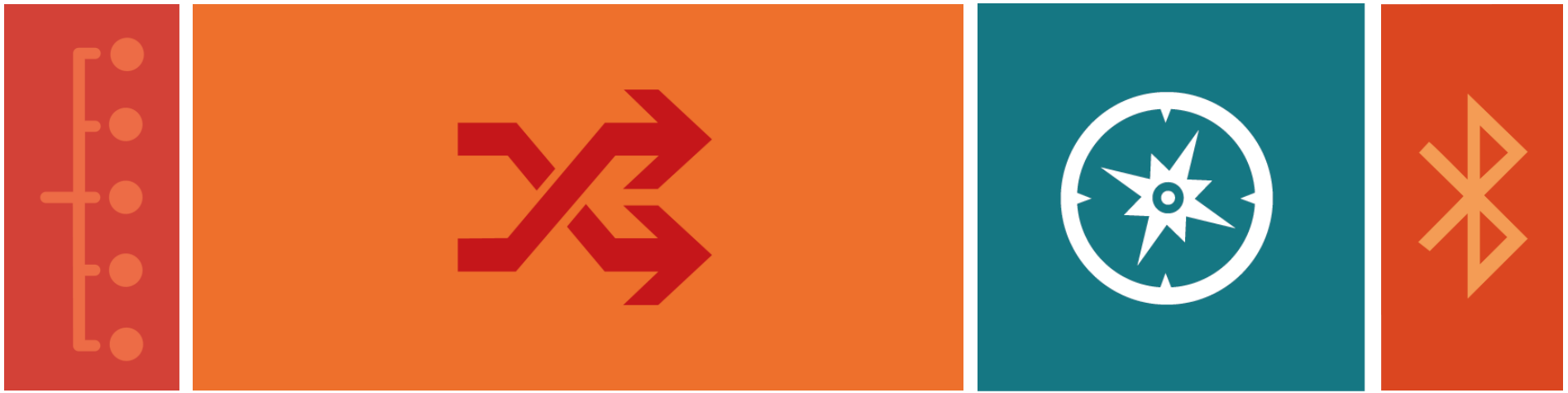
TIC-MOB2

Introduction

Grandes lignes du cours

- Création d'une simple application Android
- Connexion à l'API à l'aide de la bibliothèque « RetroFit »
- Envoie et réception des données
- Mise à jour de l'UI
- Géolocalisation
- Appareil photo

Prep'etna



Structure de projet, Activity, Layouts

Découverte d'un projet Android

AndroidManifest.xml

Description du fichier

- Fournit l'information sur la version du projet.
- Décrit les composantes de l'application: Activités, Services etc.
- Définit les permissions: Accès à l'internet, lecture des sms etc.

```xml
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="pgrl.mikhai.com.etnaticmob2">


    <uses-sdk android:minSdkVersion="17" />

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

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">
        <activity android:name=".MainActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>

</manifest>
```

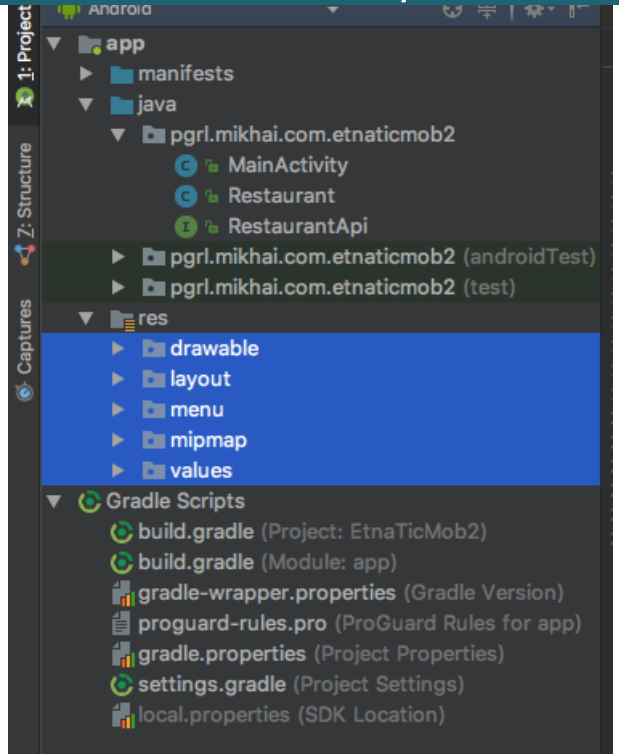
Ressources du projet

- Les ressources d'une application se trouvent dans le dossier « res ».
- Elles sont écrites en XML.

Types de ressources

Exemple de ressources

- Layout: vues attachées aux activités, aux fragments, etc.
- Drawable: contient les images (bitmap généralement).
- Mipmap: contient les icônes du lancement de l'application.
- Menu: élément permettant de définir les menus.
- Values: contient des valeurs de dimensions, tableau, couleurs, strings, etc.



Activité

Description

- Une application contient une ou plusieurs activités
- Une activité est associé à un écran de l'application
- Possède un cycle de vie

Activité

Exemple d'activité

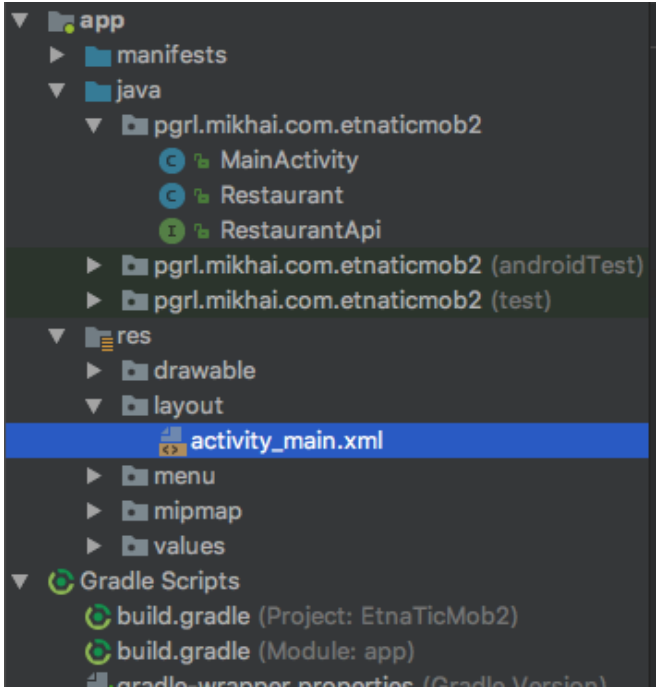
- La méthode « onCreate » est appelée au moment de la création de l'activité
- La vue « activity_main2 » est rattachée à cette activité.
- Au moment de la création de l'activité, la vue affichée à l'écran sera celle qui est définie dans le fichier « activity_main ».

```
public class Main2Activity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main2);
    }
}
```

Vue associée à l'activité

Layout de l'activité

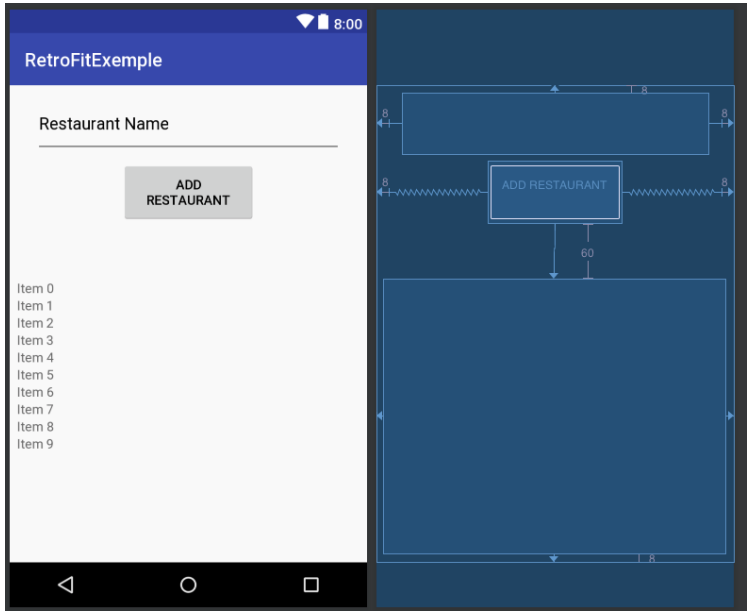
- **L'activité « MainActivity » est rattachée à un layout**



Vue associée à l'activité

activity_main.xml

Design



Text

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.constraint.ConstraintLayout xmlns:android="http://schemas.android.co
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">


    <Button
        android:id="@+id/buttonAddRestaurant"
        android:layout_width="131dp"
        android:layout_height="55dp"
        android:layout_marginBottom="8dp"
        android:layout_marginEnd="8dp"
        android:layout_marginStart="8dp"
        android:text="Add Restaurant"
        app:layout_constraintBottom_toTopOf="@+id/listView"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent" />

    <EditText
        android:id="@+id/et_restname"
        android:layout_width="257dp"
        android:layout_height="51dp"
        android:layout_marginBottom="8dp"
        android:layout_marginEnd="8dp"
        android:layout_marginStart="8dp"
        android:layout_marginTop="8dp"
        android:ems="10"
        android:text="Restaurant Name"
        app:layout_constraintBottom_toTopOf="@+id/buttonAddRestaurant"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintHorizontal_bias="0.504"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

    <ListView
        android:id="@+id/listView"
        android:layout_width="368dp"
        android:layout_height="371dp"
        android:layout_marginBottom="8dp"
        android:layout_marginEnd="8dp"
        android:layout_marginStart="8dp"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent" />

</android.support.constraint.ConstraintLayout>
```

Layout

Contenu du fichier xml

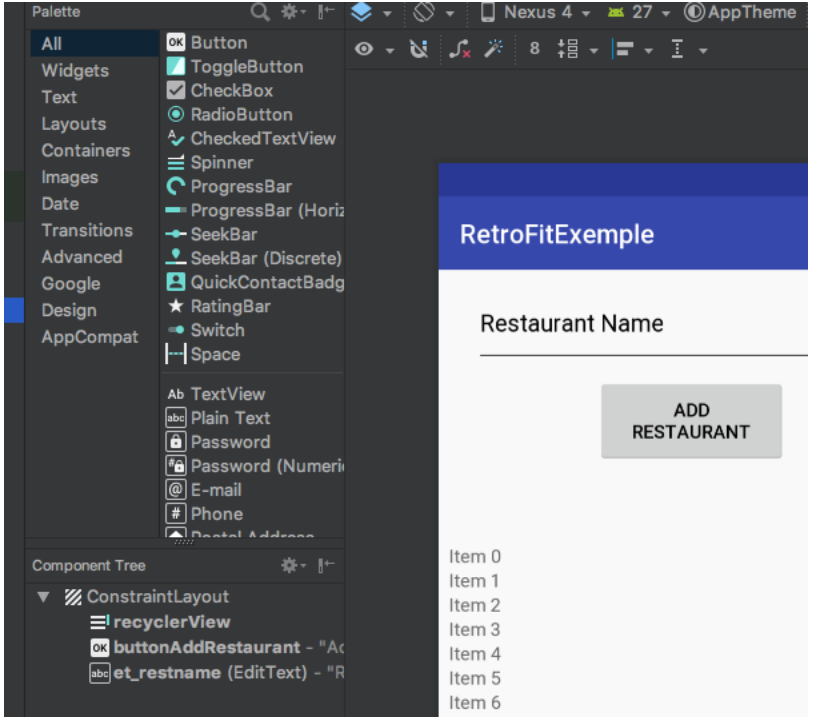
- **Le layout est composé de plusieurs éléments:**
  - Une RecyclerView (pour afficher une liste)
  - Un élément Button
  - EditText qui représente un champs pour saisir du texte

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.constraint.ConstraintLayout xmlns:android="http://schemas.android.co
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">


    <Button
        android:id="@+id/buttonAddRestaurant"
        android:layout_width="131dp"
        android:layout_height="55dp"
        android:layout_marginBottom="8dp"
        android:layout_marginEnd="8dp"
        android:layout_marginStart="8dp"
        android:text="Add Restaurant"
        app:layout_constraintBottom_toTopOf="@+id/listView"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent" />

    <EditText
        android:id="@+id/et_restname"
        android:layout_width="257dp"
        android:layout_height="51dp"
        android:layout_marginBottom="8dp"
        android:layout_marginEnd="8dp"
        android:layout_marginStart="8dp"
        android:layout_marginTop="8dp"
        android:ems="10"
        android:text="Restaurant Name"
        app:layout_constraintBottom_toTopOf="@+id/buttonAddRestaurant"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintHorizontal_bias="0.504"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

    <ListView
        android:id="@+id/listView"
        android:layout_width="368dp"
        android:layout_height="371dp"
        android:layout_marginBottom="8dp"
        android:layout_marginEnd="8dp"
        android:layout_marginStart="8dp"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent" />

</android.support.constraint.ConstraintLayout>
```

Editeur

Eléments disponibles

- **Vous avez la possibilité de créer vos layouts en utilisant l'éditeur graphique ou bien en rajoutant les éléments directement de le XML de la vue.**



Intent

Communication entre les composants

- **Afin de pouvoir lancer une nouvelle activité nous allons utiliser l'objet de la classe « Intent »**
- **Créer la communication entre composants d'une application.**

```java
public class Main2Activity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main2);

        /**
         * Recuperation du bouton depuis le layout
         */
        Button button = (Button) findViewById(R.id.button4);

        /**
         * Mise en place du listener
         */
        button.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                startActivity();
            }
        });
    }

    /**
     * Lance l'activité {@link MainActivity}
     */
    private void startActivity() {
        Intent intent = new Intent( packageContext: this, MainActivity.class);
        startActivity(intent);
    }
}
```

Intent

Envoyer des informations dans l'intent

- Nous pouvons communiquer des informations à travers l'intent et ajoutant les extras.
- Les extras peuvent être de différents types: String, long, int, etc.

```
/**
 * Lancement d'une nouvelle activité
 */
private void startActivity(String restaurantId) {
    Intent intent = new Intent( packageContext: this, Main2Activity.class);

    intent.putExtra( name: "restID",restaurantId);

    startActivity(intent);
}
```

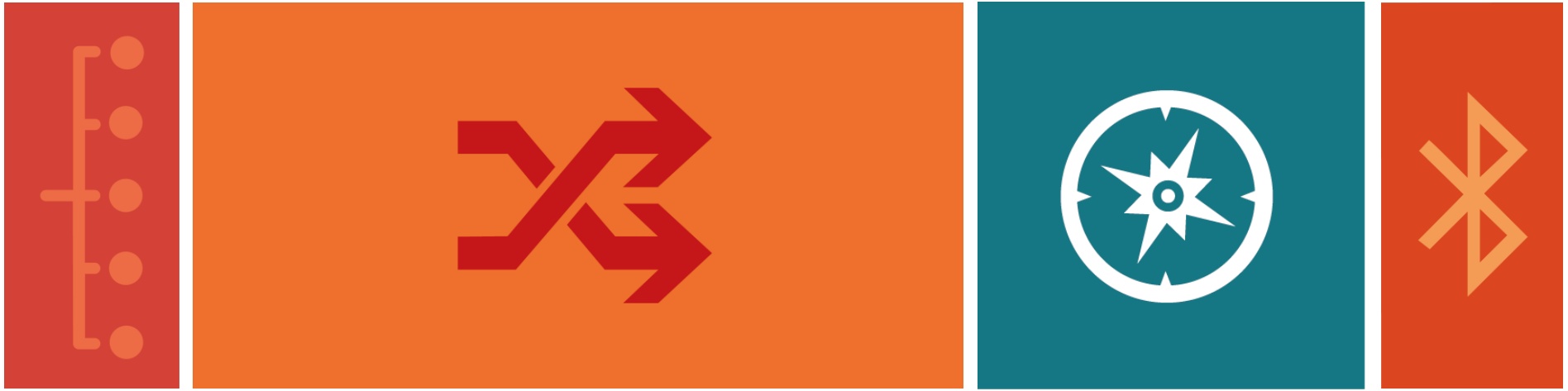
Intent

Récupérer les informations depuis l'intent

- **Une fois qu'on est dans la nouvelle activité, on peut les récupérer en appelant l'intent.**

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main2);


    String restaurantId = getIntent().getStringExtra( name: "restID");
```

Prep'etna



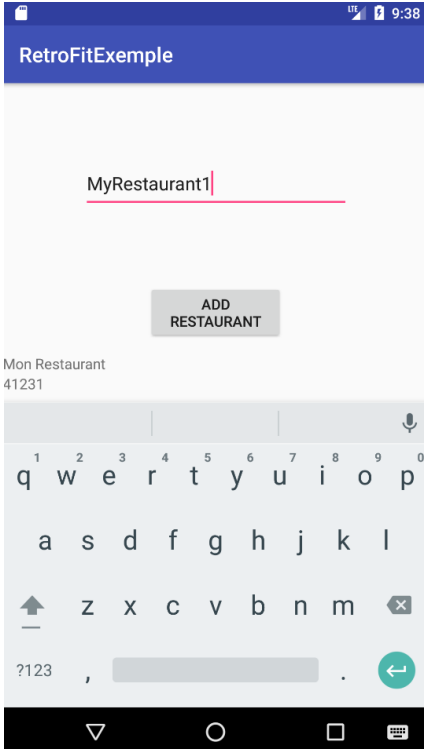
Traitement et affichage des données

Création d'application

Exemple d'application

Création d'une Application avec deux activités

- **Nous allons créer une application qui contient: un champs texte, un bouton et une liste.**



ListView

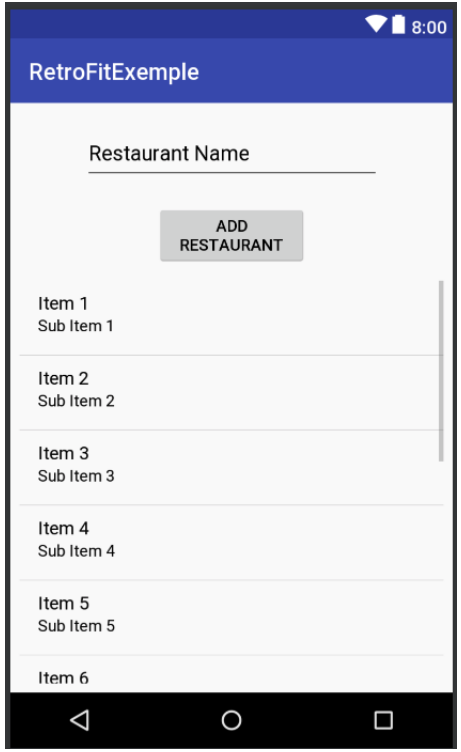
Liste d'éléments nécessaires

- ListView: élément permettant l'affichage des données
- Adapter: permet de lier les données avec la liste
- Liste de données: les entités contenant les informations

Layout de l'activité

activity_main.xml

- Ajouter les éléments (EditText, Button, ListView) au layout qui est rattaché à votre activité.
- Il faut donner un id à chaque élément de la vue afin de pouvoir le récupérer depuis l'activité.



Item View

Elément représentant une case de la liste

- **Créer un layout dans le dossier « layout » qui aura pour but d'afficher les cases de la « ListView ».**

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/a
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical">

    <TextView
        android:id="@+id/restaurant_name"
        android:layout_width="match_parent"
        android:layout_height="wrap_content" />

    <TextView
        android:id="@+id/retaurant_id"
        android:layout_width="match_parent"
        android:layout_height="wrap_content" />

</LinearLayout>
```

Modèle

- Créer une classe qui va représenter votre modèle.

```
public class Restaurant {

    private String id;

    private String name;


    public String getId() {
        return id;
    }

    public void setId(String id) {
        this.id = id;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }
}
```

BaseAdapter

Création de l'adapter

- Créer une classe qui étend la classe « BaseAdapter ».
- Permet de lier les données à la vue « ListView ».

```
public class MyListViewAdapter extends BaseAdapter {

    private Context context;
    private List<Restaurant> restaurantList;


    public MyListViewAdapter(Context context, List restaurantList) {
        this.context = context;
        this.restaurantList = restaurantList;
    }

    @Override
    public int getCount() {
        return restaurantList.size();
    }

    @Override
    public Object getItem(int position) {
        return restaurantList.get(position);
    }

    @Override
    public long getItemId(int position) {
        return position;
    }

    @Override
    public View getView(int position, View convertView, ViewGroup parent) {
        /**
         * A implémenter
         */
        return null;
    }
}
```

MyListViewAdapter

Implémentation de l'adapter

- **Définir les méthodes:**
  - getCount(): renvoi le nombre d'éléments que doit contenir la listView
  - getItem(): renvoi l'objet
  - getItemId(): renvoi l'id de l'objet
  - getView(): renvoi la vue (la case) à afficher dans la liste.

```
public class MyListViewAdapter extends BaseAdapter {

    private Context context;
    private List<Restaurant> restaurantList;


    public MyListViewAdapter(Context context, List restaurantList) {
        this.context = context;
        this.restaurantList = restaurantList;
    }

    @Override
    public int getCount() {
        return restaurantList.size();
    }

    @Override
    public Object getItem(int position) {
        return restaurantList.get(position);
    }

    @Override
    public long getItemId(int position) {
        return position;
    }

    @Override
    public View getView(int position, View convertView, ViewGroup parent) {
        /**
         * A implémenter
         */
        return null;
    }
}
```

MyListViewAdapter

Implémentation de la méthode getView

- Vérifions si la vue existe déjà (dans le cas où on réutilise la vue pour afficher une autre case).
- Récupération de l'objet depuis la liste des modèles.
- Récupération des textView de la vue pour les hydrater.
- On met le texte (nom et id) dans les textView.

```
@Override
public View getView(int position, View convertView, ViewGroup parent) {

    /**
     * Vérifie si la vue existe déjà (ryclage des vues)
     */
    if (convertView == null) {
        convertView = LayoutInflater.from(context)
                .inflate(R.layout.restaurant_row, parent,  attachToRoot: false);
    }

    /**
     * Récupération de l'objet en fonction de la position
     */
    Restaurant restaurant = (Restaurant) getItem(position);

    /**
     * Récupération des vues par leurs ids
     */
    TextView textViewRestaurantName = convertView.findViewById(R.id.restaurant_name);
    TextView textViewRestaurantId = convertView.findViewById(R.id.retaurant_id);

    /**
     *  On set le texte dans les textView
     */
    textViewRestaurantName.setText(restaurant.getName());
    textViewRestaurantId.setText(restaurant.getId());


    return convertView;
}
```

Affichages des données

Exemple d'utilisation

- Créer une liste contenant les restaurants (ArrayList)
- Récupérer les vues afin de pouvoir les utiliser
- Créer un Adapter que nous avons défini
- Associer l'adapter à la ListView

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

    restaurants = new ArrayList<>();

    this.listView = (ListView) findViewById(R.id.listView);
    this.addRestaurantBtn = (Button) findViewById(R.id.buttonAddRestaurant);
    this.editText = (EditText) findViewById(R.id.et_restname);

    Restaurant restaurant = new Restaurant();
    restaurant.setName("Mon Restaurant");
    restaurant.setId("41231");
    restaurants.add(restaurant);


    this.myListViewAdapter = new MyListViewAdapter(getApplicationContext(), restaurants);
    this.listView.setAdapter(myListViewAdapter);
```

Mise à jour de la liste

notifyDataSetChanged()

- **Une fois que les données changent au niveau de la liste des modèles, appeler la méthode notifyDataSetChanged() afin que l'adapter mette à jour la ListView.**

```
addRestaurantBtn.setOnClickListener((v) → {

        Restaurant restaurant = new Restaurant();
        restaurant.setName(editText.getText().toString().trim());
        restaurant.setId(String.valueOf(View.generateViewId()));
        restaurants.add(restaurant);
        myListViewAdapter.notifyDataSetChanged();

});
```

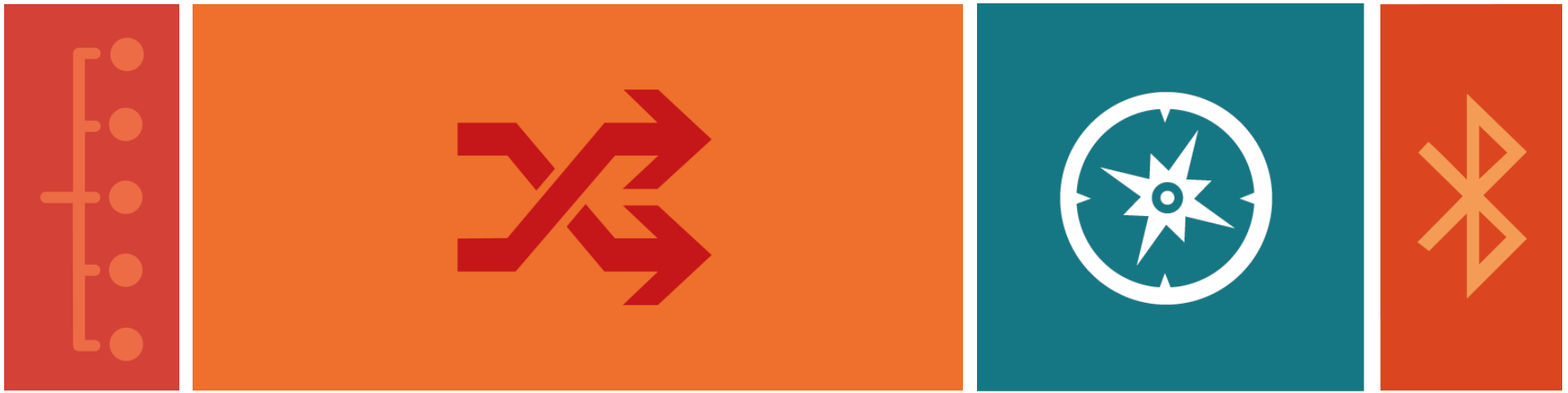
Clique sur un item de la liste

OnItemClickListener

- On peut mettre en place un « OnItemClickListener » afin de pouvoir identifier l'item cliqué dans la liste.

```
this.listView.setOnItemClickListener(new AdapterView.OnItemClickListener() {
    @Override
    public void onItemClick(AdapterView<?> parent, View view, int position, long id) {
        /**
         * Récupère le restaurant depuis la liste
         */
        Restaurant restaurant1 = restaurants.get(position);

        /**
         * Lance une activité pour afficher la fiche restaurant
         */
        startActivity(restaurant1.getId());
    }
});
```

Prep'etna



Mise en place du client REST

RetroFit

Dépendances

Build.gradle

- Afin de pouvoir utiliser la bibliothèque « RetroFit 2 », nous devons d'abord ajouter les dépendances nécessaires dans notre projet.
- Se trouvent dans build.gradle de l'application
- Synchroniser le projet une fois que les dépendances ont été ajoutées.

```
dependencies {

    compile 'com.squareup.retrofit2:retrofit:2.2.0'
    compile 'com.squareup.retrofit2:converter-gson:2.2.0'

    compile 'com.google.code.gson:gson:2.8.0'
```

Interface API

Exemple de méthodes

- Tout d'abord nous allons créer une interface qui va définir le comportement d'envoi des requêtes vers l'API.

```
public interface RestaurantApi {


    @GET("restaurants/getrestaurants")
    Call<List<Restaurant>> getRestaurants();


    @POST("restaurants/addrestaurant")
    Call<String> postRestaurant(@Body Restaurant restaurant);

}
```

Interface API

Exemple de méthodes

- La première requête méthode est associée à la ressource dont l'uri est « restaurants/getrestaurants ».
- Restaurant représente l'objet « restaurant » reçu dans la réponse du serveur.

```
public interface RestaurantApi {


    @GET("restaurants/getrestaurants")
    Call<List<Restaurant>> getRestaurants();


    @POST("restaurants/addrestaurant")
    Call<String> postRestaurant(@Body Restaurant restaurant);

}
```

Mise à jour du modèle

Objet sérializable

- Nous allons mettre à jour notre classe « Restaurant » afin de pouvoir l'utiliser dans les requêtes et dans les réponses de l'API.
- Dans le cas où le serveur renvoi un objet (JSON) dans la réponse, nous allons pouvoir le matcher directement en objet JAVA.

```
public class Restaurant{


    @SerializedName("name")
    @Expose
    private String name;


    @SerializedName("id")
    @Expose
    private String id;

    public String getName() { return name; }

    public void setName(String name) { this.name = name; }

    public String getId() {
        return id;
    }

    public void setId(String id) { this.id = id; }
}
```

Utilisation de l'inteface

Exemple d'appel de l'API

- Notre interface de communication avec les WebServices est maintenant déclarée.
- Nous allons l'associer à notre serveur local qui tourne sur la même machine.
- Ensuite nous allons envoyer quelques requêtes sur le serveur et lire ses réponses.

Connexion aux WebServices

Récupération de l'IP du Serveur

- Lancer vos WebServices sur votre machine
- Récupérer l'IP de votre machine

```
MacBook-Pro-de-Mikhail:bin mikhailpogorelov$ ifconfig
lo0: flags=8049<UP,LOOPBACK,RUNNING,MULTICAST> mtu 16384
        options=1203<RXCSUM,TXCSUM,TXSTATUS,SW_TIMESTAMP>
        inet 127.0.0.1 netmask 0xff000000
        inet6 ::1 prefixlen 128
        inet6 fe80::1%lo0 prefixlen 64 scopeid 0x1
        nd6 options=201<PERFORMNUD,DAD>
gif0: flags=8010<POINTOPOINT,MULTICAST> mtu 1280
stf0: flags=0<> mtu 1280
XHC20: flags=0<> mtu 0
en0: flags=8863<UP,BROADCAST,SMART,RUNNING,SIMPLEX,MULTICAST> mtu 1500
        ether 18:65:90:dc:07:bd
        inet6 fe80::4d3:cdd0:5fec:1a15%en0 prefixlen 64 secured scopeid 0x5
        inet 192.168.1.10 netmask 0xffffff00 broadcast 192.168.1.255
        inet6 2a01:e35:8aa8:5330:1c35:441b:21d3:759d prefixlen 64 autoconf secured
        inet6 2a01:e35:8aa8:5330:c4a8:c7d2:c6b9:6750 prefixlen 64 autoconf temporary
        nd6 options=201<PERFORMNUD,DAD>
        media: autoselect
        status: active
```

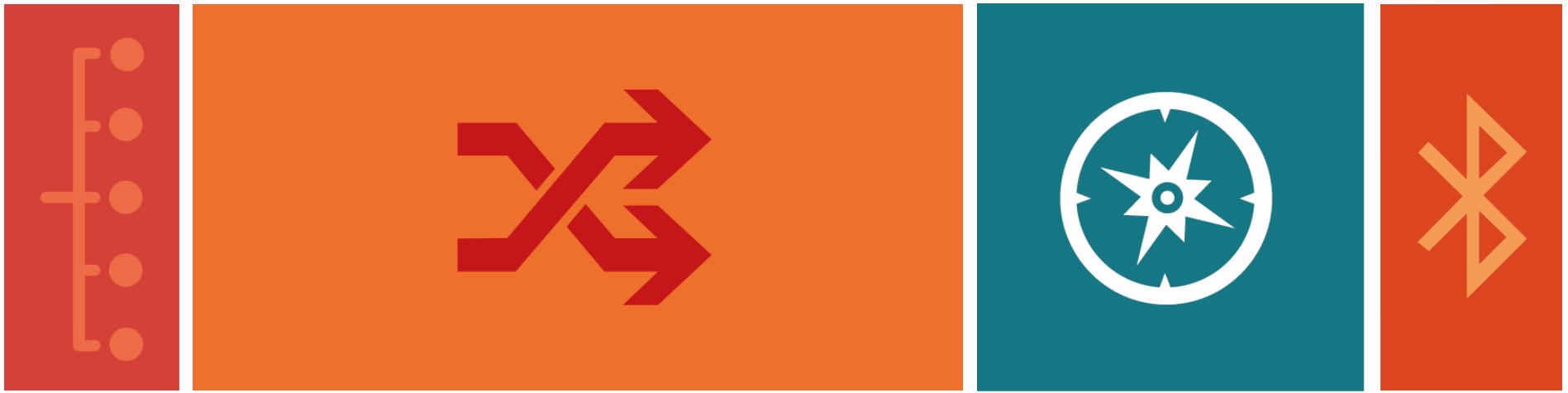
Connexion aux WebServices

Configuration de l'API côté client

- **Connecter votre Application à l'API**
- **Créer l'interface qu'on vient de déclarer**

```
/**
 * Configuration of retrofit
 */
private void configureRetrofit() {
    Gson gson = new GsonBuilder()
            .setLenient()
            .create();

    retrofit = new Retrofit.Builder().baseUrl("http://192.168.1.10:8080/restaurantsadvisor/")
            .addConverterFactory(GsonConverterFactory.create(gson))
            .build();


    restaurantApi = retrofit.create(RestaurantApi.class);
}
```

Prep' etna



Envoie des requêtes et lecture des réponses

RetroFit: Requêtes Asynchrones

Envoi des requêtes

Exemple d'appel des méthodes de l'API

- **Nous pouvons maintenant utiliser l'interface « restaurantApi » pour faire des requêtes sur notre API.**

Envoi de requête en POST

Exemple d'envoi de requête

- Dans cet exemple, on appelle la méthode « postRestaurant » dont le corps de la requête sera un objet de type « Restaurant » qui sera automatiquement envoyé en JSON.
- La méthode « enqueue » permet d'envoyer le requête en asynchrone.
- Les callBacks « onResponse » ou « onFailure » seront appelés s'il y a une réponse du serveur ou si un problème est survenu.

```
/**
 * Sends POST Request via API
 *
 * @param restaurant
 */
private void addRestaurant(Restaurant restaurant) {
    restaurantApi.postRestaurant(restaurant).enqueue(new Callback<String>() {
        @Override
        public void onResponse(Call<String> call, Response<String> response) {
            if (response.body() != null) {
                Log.d(TAG,  msg: "onResponse: " + response.body().toString());
            }
        }
        @Override
        public void onFailure(Call<String> call, Throwable t) {
            Log.e(TAG,  msg: "onFailure: " + t.getMessage().toString());
        }
    });
}
```

Envoi de requête en POST

Exemple d'envoi de requête

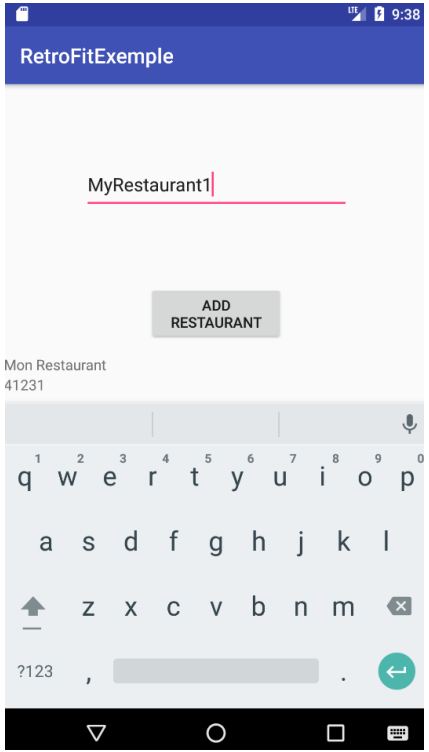
- Nous allons envoyer la requête sur le serveur.
- La requête est de type « POST »
- L'objet contenu dans le corps (body) de la requête est un objet « Restaurant ».

```
addRestaurantBtn.setOnClickListener((v) → {
        Restaurant restaurant = new Restaurant();
        restaurant.setName(editText.getText().toString().trim());

        /**
         * Envoi le l'objet sur le serveur
         */
        addRestaurant(restaurant);
});
```

Envoi de requête en POST

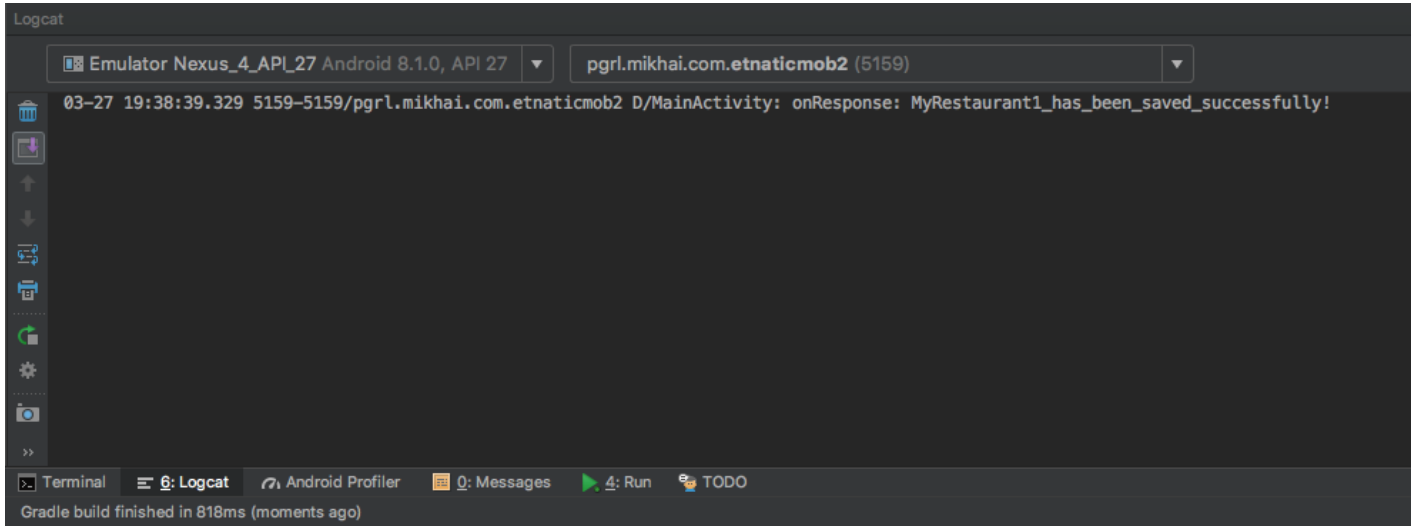
Exemple d'envoi de requête

- **Une fois la réponse des WebServices reçue, on l'affichage dans la console.**



Réponse du serveur

Type de réponse attendu: String



Envoi de requête en GET

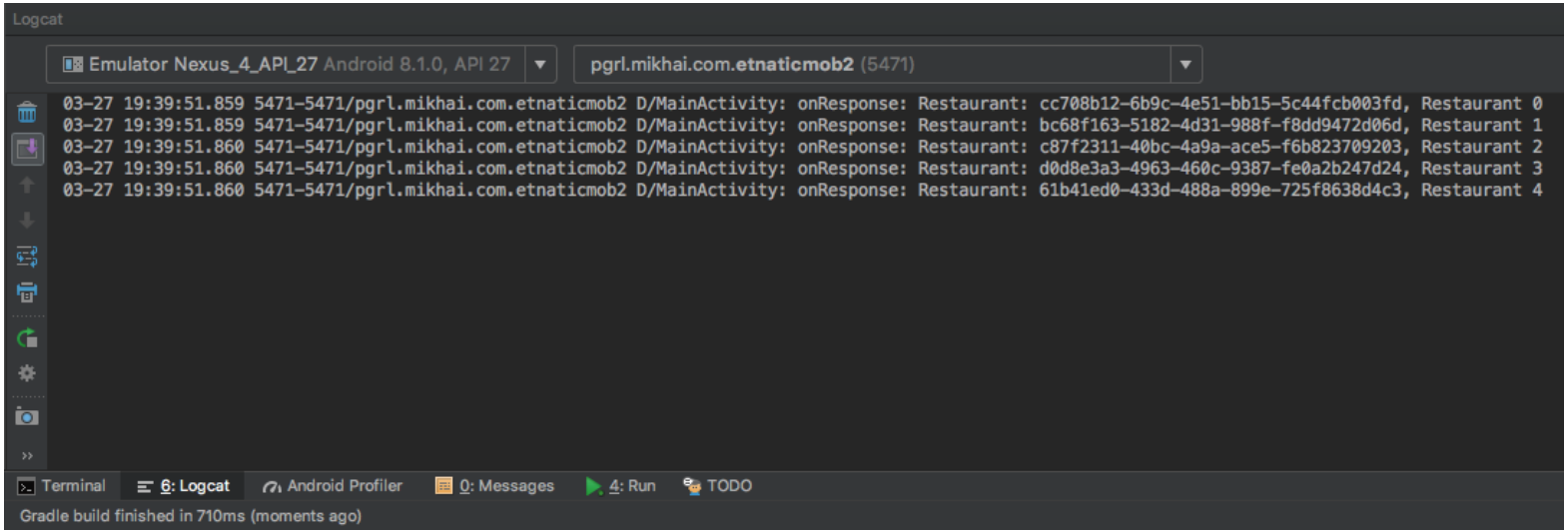
Exemple d'envoi de requête

- Dans cet exemple, on va envoyer une requête pour récupérer la liste des restaurants.
- La méthode « enqueue » permet d'envoyer le requête en asynchrone.
- Les callBacks « onResponse » ou « onFailure » seront appelés s'il y a une réponses du serveur ou si un problème est survenu.

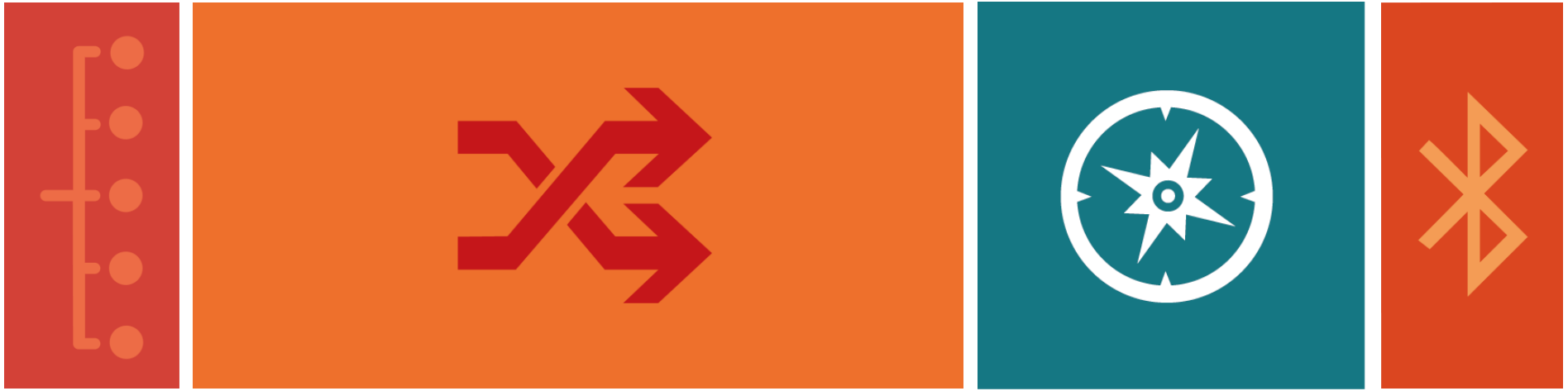
```
/**
 * Sends GET Request via API
 */
private void getRestaurantsVieAPI() {
    restaurantApi.getRestaurants().enqueue(new Callback<List<Restaurant>>() {
        @Override
        public void onResponse(Call<List<Restaurant>> call, Response<List<Restaurant>> response) {
            List<Restaurant> restaurants = response.body();
            if (restaurants != null) {
                for (Restaurant restaurant : restaurants) {
                    Log.d(TAG,  msg: "onResponse: Restaurant: id" + restaurant.getId()
                            + ", name" + restaurant.getName());
                }
            } else {
                Log.d(TAG,  msg: "onResponse: restaurants is empty: " + response.body().toString());
            }
        }

        @Override
        public void onFailure(Call<List<Restaurant>> call, Throwable t) {
            Log.e(TAG,  msg: "onFailure: " + t.getMessage());
        }
    });
}
```

Réponse du serveur

Type de réponse attendu: JSON



Prep'etna



ListView, lecture de réponses du serveur

Mise à jour de l'UI

Mise à jour de l'UI

Mise à jour des données dans la liste

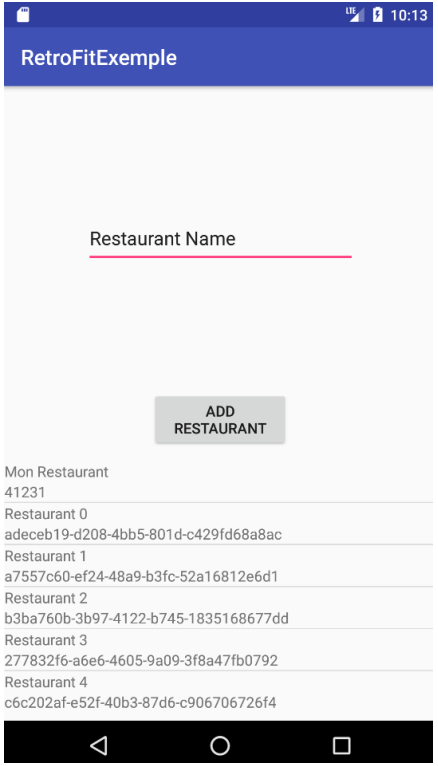
- Une fois qu'on a reçu la réponse de l'API
- Ajouter les restaurants reçus à la liste utilisée dans l'affichage
- Appeler la méthode « notifyDataSetChanged » de l'adapter pour le prévenir du changement des données au niveau de la liste.

```
/**
 * Sends GET Request via API
 */
private void getRestaurantsVieAPI() {
    restaurantApi.getRestaurants().enqueue(new Callback<List<Restaurant>>() {
        @Override
        public void onResponse(Call<List<Restaurant>> call, Response<List<Restaurant>> response) {
            List<Restaurant> restaurantsList = response.body();
            if (restaurantsList != null) {
                for (Restaurant restaurant : restaurantsList) {
                    restaurants.add(restaurant);
                }
                myListViewAdapter.notifyDataSetChanged();
            } else {
                Log.d(TAG, msg: "onResponse: restaurants is empty: " + response.body().toString());
            }
        }

        @Override
        public void onFailure(Call<List<Restaurant>> call, Throwable t) {
            Log.e(TAG, msg: "onFailure: " + t.getMessage());
        }
    });
}
```

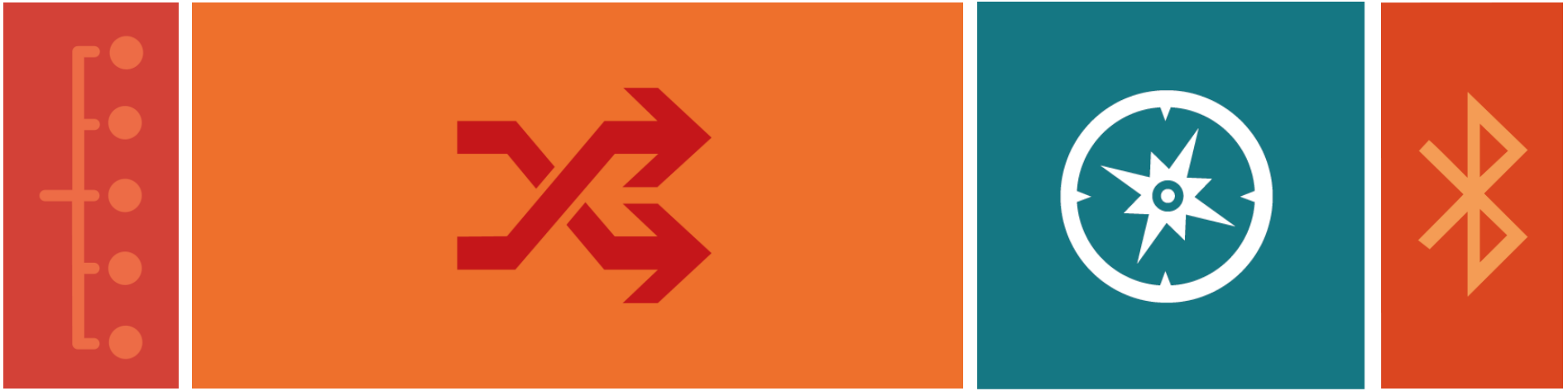
Mise à jour de l'UI

Mise à jour des données dans la liste

- On obtient le résultat suivant
- La liste est à jour
- Contient les restaurants reçus depuis les WebServices



Prep'etna



Activation et configuration de la géolocalisation, mise à jour de la position

Géolocalisation

Géolocalisation

Types de précision

- **Il existe deux niveaux de précision de localisation sur Android.**
  - **Fine location (GPS_PROVIDER): utilise le GPS**
  - **Coarse location (NETWORK_PROVIDER): utilise internet. Calcule la position en fonction des bornes wifi et des tours cellulaires.**

Géolocalisation

Permission: Localisation précise

- ACCESS_FINE_LOCATION nécessite l'utilisation du gps android.hardware.location.gps
- Déclarer l'utilisation de ce type de géolocalisation dans le fichier AndroidManifest.xml

```
<uses-permission
android:name="android.permission.ACCESS_FINE_LOCATIO
N"/>
<uses-feature
android:name="android.hardware.location.gps"/>
```

Géolocalisation

Permission: Localisation internet

- **ACCESS_COARSE_LOCATION nécessite l'utilisation du matériel android.hardware.location.network (moins précis).**
- **Déclarer l'utilisation de ce type de géolocalisation dans le fichier AndroidManifest.xml.**

```
<uses-permission
android:name="android.permission.ACCESS_COARSE_LOCAT
ION" />
    <uses-feature
android:name="android.hardware.location.network" />
```

Géolocalisation

Permissions: Inclure les deux permissions

- **Votre application peut utiliser les deux providers à la fois. Pour cela, il suffit de déclarer la localisation fine. La permission de la localisation ACCESS_FINE_LOCATION inclut la permission de la localisation ACCESS_COARSE_LOCATION.**

```
<uses-permission
android:name="android.permission.ACCESS_FINE_LO
      CATION"/>
<uses-feature
      android:name="android.hardware.location.gps"/
>
```

Géolocalisation

LocationManager

- **Le service qui s'occupe de la localisation sur Android s'appelle LocationManager.**

```
private LocationManager getLocationManager(){
   LocationManager locationManager = (LocationManager) getSystemService(Context.LOCATION_SERVICE);
   return locationManager;
}
```

Géolocalisation

Récupérer la dernière position connue

- **Il est possible de récupérer la dernière position connue depuis ce service en appelant la méthode « getLastKnownLocation ». Il suffit de passer le nom du provider en paramètre.**

```
private Location getLocation(){
   return locationManager.getLastKnownLocation(LocationManager.GPS_PROVIDER);
}
```

Géolocalisation

LocationListener

- **LocationListener permet d'être notifié sur la mise à jour de la position et sur le statut des providers.**

Géolocalisation

LocationListener

- **LocationListener contient quatre méthodes qui informent sur la position et sur le statut des providers:**
  - **onLocationChanged: appelée quand la position a été mise à jour.**
  - **onStatusChanged: appelée quand le statut d'un provider a changé (GPS ou Network).**
  - **onProviderEnabled: appelée quand un provider est disponible (GPS ou Network).**
  - **onProviderDisabled: appelée quand l'utilisateur a désactivé un provider (GPS ou Network).**

```
locationListener = new LocationListener() {
   @Override
   public void onLocationChanged(Location location) {

latitude.setText(String.valueOf(location.getLatitude()));

longitude.setText(String.valueOf(location.getLongitude()));
   }

   @Override
   public void onStatusChanged(String provider, int status,
Bundle extras) {

   }

   @Override
   public void onProviderEnabled(String provider) {

   }

   @Override
   public void onProviderDisabled(String provider) {

   }
};
```

Géolocalisation

Mise à jour de la position

- **Afin d'enregistrer le listener auprès du locationManager, utiliser la méthode « requestLocationUpdates » dans laquelle il faudra passer le nom du provider, la fréquence du rafraichissement (en ms), la distance minimum (en m) et le listener.**
- **La distance minimum est la distance qu'il faudra réaliser avant de recevoir la mise à jour de la position.**
- **L'appel de la méthode « requestLocationUpdates » nécessite la permission qu'il faut gérer explicitement dans le code.**

```
>_
locationManager.requestLocationUpdates(LocationManager.NETWORK_PROVIDER, 1000, 10f, locationListener);
locationManager.requestLocationUpdates(LocationManager.GPS_PROVIDER, 1000, 10f, locationListener);
```

Géolocalisation

Les permissions

- **Depuis Android 6.0, lorsqu'on souhaite utiliser des ressources ou des informations autres que celles qui se trouvent dans votre application, il faut demander explicitement les permissions.**

Géolocalisation

Vérifier les permissions

- La vérification des permissions sur une ressource se fait à l'aide de la méthode « checkSelfPermission ».
- La méthode renvoi PackageManager.PERMISSION_GRANTED si l'application a les permissions sur la ressource demandée, et PackageManager.PERMISSION_DENIED si l'application doit demander les permissions explicitement.

```
private void checkLocationPermission() {
   if (ContextCompat.checkSelfPermission(this, ACCESS_FINE_LOCATION) != PackageManager.PERMISSION_GRANTED)
{
        ActivityCompat.requestPermissions(this, new String[]{Manifest.permission.ACCESS_FINE_LOCATION},
                REQUEST_PERMISSION_LOCATION_CODE);
   } else {
      initLocationManager();
   }
}
```

Géolocalisation

Vérifier les permissions

- **Afin de demander les permissions sur une ressource, il faut utiliser la méthode « requestPermissions » qui prend en paramètre un context, les permissions à demander et le requestCode. Cette méthode affichera une Dialogue dans laquelle l'utilisateur devra soit accepter soit refuser la permission demandée.**

```
private void checkLocationPermission() {
   if (ContextCompat.checkSelfPermission(this, ACCESS_FINE_LOCATION) != PackageManager.PERMISSION_GRANTED)
{
        ActivityCompat.requestPermissions(this, new String[]{Manifest.permission.ACCESS_FINE_LOCATION},
                REQUEST_PERMISSION_LOCATION_CODE);
   } else {
      initLocationManager();
   }
}
```

Géolocalisation

Vérifier les permissions

- Afin de demander les permission sur une ressource, il faut utiliser la méthode « requestPermissions » qui prend en paramètre un context, les permissions à demander et le requestCode. Cette méthode affichera une Dialogue dans laquelle l'utilisateur devra soit accepter soit refuser la permission demandée.
- Une fois que l'utilisateur a accepté (ou a refusé) la demande de permission, la callback « onRequestPermissionsResult » sera appelée.
- Le requestCode sera utilisé dans la méthode « onRequestPermissionsResult » de votre activity.

```
@Override
public void onRequestPermissionsResult(int requestCode, @NonNull String[] permissions, @NonNull int[] grantResults)
{
   super.onRequestPermissionsResult(requestCode, permissions, grantResults);
    if (requestCode == REQUEST_PERMISSION_LOCATION_CODE) {
      if (grantResults.length != 0 && grantResults[0] == PackageManager.PERMISSION_GRANTED) {
         initLocationManager();
       }
   }
}
```

Géolocalisation

Redemander les permissions

- Dans le cas où l'utilisateur refuse les permissions une fois, on peut lui redemander ces permissions en appelant la méthode « *shouldShowRequestPermissionRationale* »
- Cette méthode renvoi « true » si l'utilisateur avait précédemment refusé de donner la permission, et "false" si l'utilisateur avait refusé la permission et avait coché la checkbox "Ne plus redemander la permission".

Géolocalisation

Exemple de vérification de permissions

```
private void checkLocationPermission() {
            // Verifie si on a les permissions
            if (ContextCompat.checkSelfPermission(this, ACCESS_FINE_LOCATION) !=
PackageManager.PERMISSION_GRANTED) {
                  if (ActivityCompat.shouldShowRequestPermissionRationale(this,
Manifest.permission.ACCESS_FINE_LOCATION)) {
                        // Afficher un message expliquant pourquoi l'utilisateur devrait donner les
                          permissions.
                        // On peut redemander la permission
                  } else {
                        // Demander la permission
                        ActivityCompat.requestPermissions(this, new
String[]{Manifest.permission.ACCESS_FINE_LOCATION}, REQUEST_PERMISSION_LOCATION_CODE);                }
            } else {
                  // La permission a déjà été accordée
                  initLocationManager();
            }
      }
```

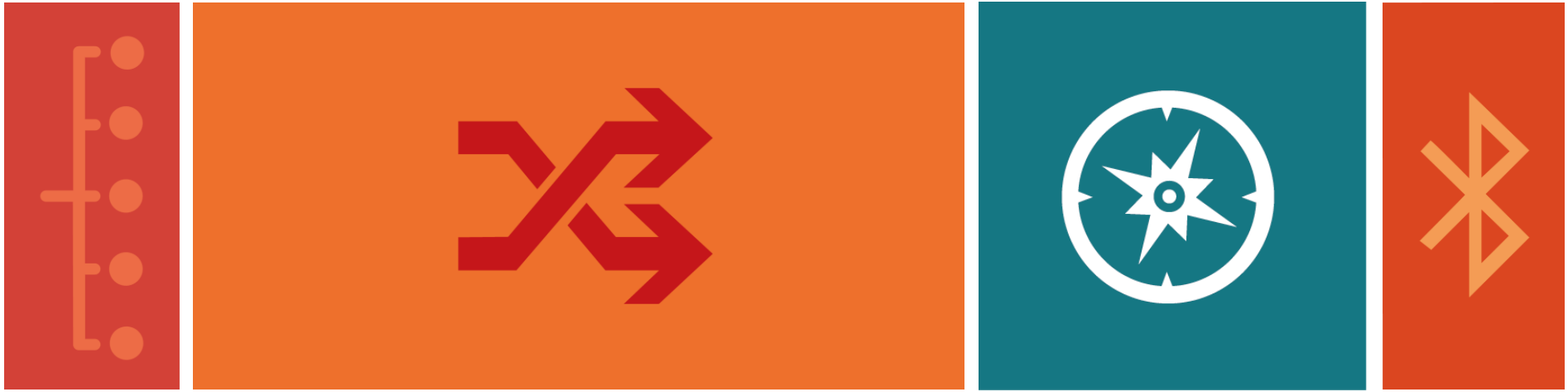
Géolocalisation

Se désenregistrer du service de la géolocalisation

- Quand l'application est en pause, il est préférable d'arrêter l'utilisation du service de la géolocalisation.
- Utiliser la méthode « removeUpdates » du LocationManager afin de ne plus être notifié sur les mises à jour du service.

```
@Override
protected void onPause() {
   super.onPause();
   if (locationManager != null &&
      locationListener != null) {

      locationManager.removeUpdates(locationLis
tener);
      locationManager = null;
      locationListener = null;
   }
}
```

Prep'etna



Prendre et enregistrer des photos

Appareil photo

Appareil photo

Utiliser l'application

- La prise des photos peut se faire par l'application de l'appareil photo par défaut.
- Pour faire cela il suffit lancer l'activity externe (correspondant à l'appareil photo) à votre application afin de faire la photo, puis récupérer l'information sur la photo dans la callback « onActivityResult ».

Appareil photo

Faire une photo et la sauvegarder dans l'appareil

- Utiliser l'intent correspondant à l'activity externe qui servira pour faire les photos
- Vérifier s'il y a bien une activity associée à l'appareil photo
- Créer le fichier où la photo sera sauvegardée
- Récupérer l'uri du fichier et la passer à la méthode « putExtra » de l'intent correspondant à l'appareil photo
- Lancer l'activity à l'aide de l'intent

```
private void startTakePictureIntent() {
    Intent intent = new
Intent(MediaStore.ACTION_IMAGE_CAPTURE);
    if (intent.resolveActivity(getPackageManager()) !=
null) {
        File photoFile = null;
        try {
            photoFile = createImageFile();
        } catch (IOException e) {
            Log.e(TAG, "startTakePictureIntent: " +
e.getMessage());
        }

        if (photoFile != null) {
            Uri photoURI = FileProvider.getUriForFile(this,
                    "com.example.android.fileprovider",
                    photoFile);
            intent.putExtra(MediaStore.EXTRA_OUTPUT,
photoURI);
            startActivityForResult(intent,
REQUEST_IMAGE_CAPTURE);
        }
    }
}
```

Appareil photo

Déclaration du FileProvider

- Déclarer le FileProvider dans le fichier AndroidManifest.xml
- L'attribut « android:authorities » doit être le même que le paramètre que vous avez passé à la méthode « getUriForFile() ».
- Le fichier contenant les paths (file_paths) doit être configuré dans les méta-données du FileProvider.

```
<provider
      android:name="android.support.v4.content.FileProvider"
      android:authorities="com.example.android.fileprovider"
      android:exported="false"
      android:grantUriPermissions="true">
      <meta-data
          android:name="android.support.FILE_PROVIDER_PATHS »
          android:resource="@xml/file_paths">
     </meta-data>
</provider>
```

Appareil photo

Création du fichier file_paths.xml

- Le fichier file_paths.xml doit se trouver dans le dossier de ressources « xml ».
- Le path (my_images) indiqué dans ce fichier correspond au path qui est retourné par la méthode « getExternalFilesDir » avec le paramètre « Environment.DIRECTORY_PICTURES ».
- Ce dossier reste privé à votre application. Aucune autre application n'y aura accès.
- Lorsque vous supprimez l'application, ce dossier sera automatiquement supprimé.

```
<?xml version="1.0" encoding="utf-8"?>
<paths xmlns:android="http://schemas.android.com/apk/res/android">
    <external-path name="my_images" path="Android/data/com.example.package.name/files/Pictures" />
</paths>
```

Appareil photo

Récupération de la photo

- **Une fois que la photo a été prise par l'appareil photo, la méthode « onActivityResult » est appelée.**
- **Afin de s'assurer que la callback a bien été appelée par l'activity qui s'occupait de la prise de photo, il suffit de vérfier le requestCode qui a été passé en paramètre lorsqu'on lançait l'activity et le resultCode.**
- **Une fois que ces paramètres sont vérifiés, on peut directement utiliser la photo à l'aide du path correspondant au fichier (de l'image) qui a été créé avant que la photo ne soit prise.**

```
@Override
protected void onActivityResult(int requestCode, int resultCode, @Nullable Intent data) {
    if (requestCode == REQUEST_IMAGE_CAPTURE && resultCode == RESULT_OK) {
        // La librarie Glide est utilisée pour charger les images
        Glide.with(this).load(currentPhotoPath).into(imageView);
    }
}
```