app/src/main/AndroidManifest.xml 1

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

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

    <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>
        <activity android:name=".DetailActivity" />
```

SUGGESTION

Pro tip: you can add the `android:parentActivityName` attribute and set it to the MainActivity to automatically implement proper "up" navigation via app bar on DetailActivity 😄

```xml
    </application>

</manifest>
```

app/src/main/res/layout/activity_detail.xml

```xml
<?xml version="1.0" encoding="utf-8"?>
<ScrollView xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
```

AWESOME

Good job setting up your ScrollView for your activity_detail.xml file. By providing a mechanism to scroll your app up and down, you can help your users view all of the content even if there was too much information to be shown or User in using small screen device. Well done! 👏

```xml
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_margin="10dp"
    android:fillViewport="true"
    tools:context="com.udacity.sandwichclub.DetailActivity">


    <android.support.constraint.ConstraintLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content">


        <ImageView
            android:id="@+id/image_iv"
            android:layout_width="match_parent"
            android:layout_height="200dp"
```

SUGGESTION

you should define XML dimension constants in `dimens.xml` then refer them in layout XML using `@dimen/constant_name` or just simply put your cursor on 200dp and press Alt+enter and see the magic.

```xml
            android:layout_marginBottom="15dp"
            android:layout_marginTop="15dp"
            android:adjustViewBounds="true"
            android:contentDescription="@string/sandwich_picture_content_description"
            android:scaleType="centerCrop" />


        <TextView
```

SUGGESTION

To make it align left just remove the constraint `app:layout_constraintEnd_toEndOf="parent"` and your Also know As label will perfectly align left. It is aligning in the center because it has not that much data to show and you have set toStartOf and toEndOf constraint both. so just remove `toEndOf` and your label will be aligned left perfectly.

app/src/main/java/com/udacity/sandwichclub/DetailActivity.java

```
    }


    private void initializeLocators() {

        ingredientsIv = findViewById(R.id.image_iv);
```

SUGGESTION

To learn more,
one thing you could check is a package called “ButterKnife”. “Butterknife”
is a lightweight library that you could use to inject views into Android components in an easier way, which, can make your life as a developer much easier.

For example, you could create views using “ButterKnife” like this:

```
    @BindView(R.id.step_number)
    TextView stepNumber;
    @BindView(R.id.short_description)
    TextView shortDescription;
    @BindView(R.id.description)
    TextView description;
    @BindView(R.tutorial_button)
    TextView tutorialButton;
```

you do not need to call `findViewById()` for any view.
More to read:
http://jakewharton.github.io/butterknife/
https://code.tutsplus.com/tutorials/quick-tip-using-butter-knife-to-inject-views-on-android--cms-23542

```java

    private void setValues(Sandwich sandwich) {
        Picasso.with(this)
                .load(sandwich.getImage())
```

SUGGESTION

To learn more, here, you can also try to use `error()` and `placeholder()` provided by Picasso to avoid the potential crash due to empty or null image URL values. Before the error placeholder is shown, Picasso will retry your request for three times.

You could try to use these two methods as shown in the sample code below (from Picasso documentation):

```java
Picasso.with(context)
    .load(url)
    .placeholder(R.drawable.user_placeholder)
    .error(R.drawable.user_placeholder_error)
    .into(imageView);
```

Using Picasso without error() might not cause any problem. However, when it comes to some other APIs (unfortunately, Spotify is one of them), the chance of fighting against some strange values could get higher.
So that's why I strongly suggest you to use these two methods in your future projects.

Try it yourself! :)

```java
                .into(ingredientsIv);

        setTitle(sandwich.getMainName());

        List<String> alsoKnownAs = sandwich.getAlsoKnownAs();
        String akaNames = TextUtils.join(", ", alsoKnownAs);
        alsoKnownTv.setText(getString(R.string.detail_also_known_as_label)
                .concat(" ")
                .concat(akaNames));
        if (akaNames.isEmpty()) {
            alsoKnownTv.setText("");
        }
        String placeOfOrigin = sandwich.getPlaceOfOrigin();
        originTv.setText(placeOfOrigin);
```

AWESOME

All sandwich details are shown in a sensible manner.

```java
        if (placeOfOrigin.isEmpty()) {
            originLabelTv.setText("");
            originTv.setText("");
        }

        List<String> ingredients = sandwich.getIngredients();
        ingredientsTv.setText(TextUtils.join(", ", ingredients));
```

AWESOME

TextUtils class is specially designed for operation on String. Good work 👏

app/src/main/java/com/udacity/sandwichclub/utils/JsonUtils.java

```java
package com.udacity.sandwichclub.utils;

import com.udacity.sandwichclub.model.Sandwich;

import org.json.JSONArray;
import org.json.JSONException;
import org.json.JSONObject;

import java.util.ArrayList;
import java.util.List;

public class JsonUtils {

    public static Sandwich parseSandwichJson(String json) {
        try {
            JSONObject jsonObject = new JSONObject(json);
            JSONObject jsonObjectName = jsonObject.getJSONObject("name");
```

SUGGESTION

1 JSON, Intents, etc. make use of Strings as keys. These keys should be converted to constants. There are several benefits of such an approach.

1. Avoid typographical errors. When it comes to keys hello and Hello are two different keys.
2. Avoid duplication of code.
3. Easy to refer.
4. Can be used across the app.
   public static final String JSON_INTENT_KEY = "some_json_key";

```java
            String mainName = jsonObjectName.getString("mainName");
```

SUGGESTION

I would suggest using `optString()` over `getString()`. It will make sure that you are returned an empty String when no value is returned from the API. Please see the reference doc for `optString()`. same with `optJSONArray()` and `optJSONObject()`.

```
            JSONArray alsoKnownAsJSONArray = jsonObjectName.getJSONArray("alsoKnownAs");
            List<String> alsoKnownAs = JSONArray2List(alsoKnownAsJSONArray);

            String placeOfOrigin = jsonObject.getString("placeOfOrigin");
            String description = jsonObject.getString("description");

            String image = jsonObject.getString("image");
            JSONArray ingredientsJSONArray = jsonObject.getJSONArray("ingredients");
            List<String> ingredients = JSONArray2List(ingredientsJSONArray);
            return new Sandwich(mainName, alsoKnownAs, placeOfOrigin, description, image, ingredients);
```

AWESOME

Data is completely parsed without using any 3rd party library 👍 Good work.

```
        } catch (JSONException e) {
            e.printStackTrace();
        }
        return null;
    }

    private static List<String> JSONArray2List(JSONArray JSONarray) {
```

AWESOME

Declaring Separate method for the same type of work is good programming practice.