To implement an Action Bar in Android using a `Toolbar`, we can go step by step from setting up the `XML` layout to handling user interactions in the Kotlin (`.kt`) file. We will walk through each part in detail, explaining the code along the way.

**Step-by-Step Guide to Implementing an Action Bar with Toolbar**

**1. Set up your project**

Make sure you have set up an Android project in Android Studio. If you don't have one, follow these steps:

1. Open Android Studio.

2. Select New Project → Empty views Activity.

3. Name the project (e.g., `ActionBarDemo`) and choose Kotlin as the language.

4. Click Finish.

This will generate a basic project structure with the following important files:

**MainActivity.kt** (the main activity for Kotlin code)

**activity\_main.xml** (the layout file for your UI)

**2. Add the Toolbar in XML Layout**

The first step is to define the `Toolbar` in the `activity\_main.xml` layout file.

Open `res/layout/activity\_main.xml` and add the following code:

```xml

<?xml version="1.0" encoding="utf-8"?>

<androidx.coordinatorlayout.widget.CoordinatorLayout

xmlns:android="http://schemas.android.com/apk/res/android"

android:layout\_width="match\_parent"

android:layout\_height="match\_parent">

<!-- Toolbar setup -->

<androidx.appcompat.widget.Toolbar

android:id="@+id/toolbar"

android:layout\_width="match\_parent"

android:layout\_height="?attr/actionBarSize"

android:background="?attr/colorPrimary"

android:theme="@style/ThemeOverlay.AppCompat.Dark.ActionBar"

android:elevation="4dp" />

<!-- Main content below the Toolbar -->

<TextView

android:id="@+id/textView"

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:text="Hello World!"

android:layout\_gravity="center" />

</androidx.coordinatorlayout.widget.CoordinatorLayout>

```

**Explanation:**

**CoordinatorLayout:** A flexible layout for arranging views, such as `Toolbar` and main content, allowing views like `Snackbar` to overlap. It helps in positioning views in layers.

**Toolbar:**

**`id`:** Unique identifier for the Toolbar so we can reference it in the Kotlin code.

`layout\_width` and `layout\_height`: Set the toolbar to take up the full width and use the default height.

`background`: Assigns the toolbar a background color (`colorPrimary` from your theme).

`theme`: Applies the dark theme overlay to ensure action icons (like search) are visible.

`elevation`: Adds a shadow effect to the toolbar to distinguish it from the rest of the UI.

**TextView:** A simple text view placed below the toolbar as the main content.

**3. Set up Toolbar in Kotlin**

Next, we need to make the toolbar act as the Action Bar. This is done in the `MainActivity.kt` file.

Open `MainActivity.kt` and modify the code as follows:

```kotlin

package com.example.actionbar  
  
import android.graphics.Color  
import android.os.Bundle  
import android.text.SpannableString  
import android.text.style.ForegroundColorSpan  
import android.view.Menu  
import android.widget.Toast  
import androidx.appcompat.app.AppCompatActivity  
import androidx.appcompat.widget.Toolbar  
  
class MainActivity : AppCompatActivity() {  
  
 override fun onCreate(savedInstanceState: Bundle?) {  
 super.onCreate(savedInstanceState)  
 setContentView(R.layout.*activity\_main*)  
  
 val toolbar: Toolbar = findViewById(R.id.*toolbar*)  
  
 *// Set the toolbar as the ActionBar* setSupportActionBar(toolbar)  
  
 *// Optional: Set a title (if not set in the theme)  
 supportActionBar*?.*title* = "My Action Bar"  
 }  
  
 *// Inflate the menu items (if any) in the Action Bar* override fun onCreateOptionsMenu(menu: android.view.Menu?): Boolean {  
 *menuInflater*.inflate(R.menu.*menu\_main*, menu)  
 val settingsItem = menu?.findItem(R.id.*action\_settings*)  
 val spannableTitle = SpannableString("Settings")  
 spannableTitle.setSpan(ForegroundColorSpan(Color.*RED*), 0, spannableTitle.length, 0)  
 settingsItem?.*title* = spannableTitle  
 return true  
 }  
  
 *// Handle item clicks from the Action Bar* override fun onOptionsItemSelected(item: android.view.MenuItem): Boolean {  
 return when (item.*itemId*) {  
 R.id.*action\_search* -> {  
 *// Handle search action* Toast.makeText(this, "Search selected", Toast.*LENGTH\_SHORT*).show()  
 true  
 }  
 R.id.*action\_settings* -> {  
 *// Handle settings action* Toast.makeText(this, "Settings selected", Toast.*LENGTH\_SHORT*).show()  
 true  
 }  
 else -> super.onOptionsItemSelected(item)  
 }  
 }  
}

```

**Explanation:**

setSupportActionBar(toolbar):

* This method is used to set the toolbar as the activity’s action bar. The toolbar is a UI element that can function as the app's top bar (the action bar).
* You need to have a Toolbar view in your layout (usually an XML file) that you refer to in your code as toolbar.

supportActionBar?.title = "My Action Bar":

* This optional line sets the title of the action bar (if it’s not already set through the app's theme). The title displayed will be "My Action Bar".
* supportActionBar refers to the action bar that is set up by setSupportActionBar. Using ?., it checks if the action bar is not null before setting the title.

onCreateOptionsMenu(menu: android.view.Menu?):

* This function is overridden to inflate the menu options defined in the XML file into the action bar.
* menuInflater.inflate(R.menu.main\_menu, menu):
  + R.menu.main\_menu refers to a main\_menu.xml file located in the res/menu directory, which defines the menu items (like "Search" and "Settings").
  + The menuInflater takes the XML menu file and populates it into the action bar.
* return true indicates that the menu should be displayed in the action bar.

onOptionsItemSelected(item: android.view.MenuItem):

* This method is called when a user selects an item from the action bar menu.
* when (item.itemId) checks the ID of the selected menu item to determine which one was clicked.
* The IDs (R.id.action\_search and R.id.action\_settings) correspond to the menu items defined in the main\_menu.xml file.
* For each case:
  + If the action\_search item is clicked, the true indicates that the event is handled. You would place logic here to handle the "Search" action.
  + If the action\_settings item is clicked, the true indicates that the event is handled. Logic for handling "Settings" goes here.
* If neither is selected, the else case calls super.onOptionsItemSelected(item) to pass control back to the parent class for handling.

**1. val settingsItem = menu?.findItem(R.id.action\_settings)**

This line retrieves the menu item with the ID R.id.action\_settings from the menu object.

* menu?.findItem(R.id.action\_settings) returns the MenuItem object corresponding to the "Settings" item.
* The ? (safe call operator) ensures that if menu is null, the code won't throw an error. If the menu is null, settingsItem will also be null.

**2. val spannableTitle = SpannableString("Settings")**

* A SpannableString is used to apply styling (such as colors, fonts, or text effects) to portions of text.
* This line creates a SpannableString object with the text "Settings". This string will later be styled with a ForegroundColorSpan to change its color.

**3. spannableTitle.setSpan(ForegroundColorSpan(Color.RED), 0, spannableTitle.length, 0)**

This line applies the red color to the entire text of the SpannableString:

* ForegroundColorSpan(Color.RED): This span changes the text color to **red**. Color.RED is a constant in the Android framework that represents the color red (#FF0000).
* The setSpan method applies this span to a portion of the string.
  + The first argument is the span (in this case, the ForegroundColorSpan for red).
  + The second argument (0) is the start index (inclusive) of the portion of text that will be styled (start of the string).
  + The third argument (spannableTitle.length) is the end index (exclusive), which here is the entire length of the string.
  + The fourth argument (0) is the flags, often set to 0 for default behavior.

**4. settingsItem?.title = spannableTitle**

This line sets the modified SpannableString as the title of the settingsItem:

* settingsItem?.title: If settingsItem is not null, the code assigns a new title to the menu item. The title property of MenuItem sets the displayed text for the menu item.
* spannableTitle: The SpannableString with the red text is now assigned as the title. Since it's a SpannableString with a color span, the text will be displayed in red when rendered in the menu.

**Summary:**

* The code finds the "Settings" item in the menu.
* It creates a SpannableString containing the word "Settings."
* It applies a red color span to the entire text.
* It then sets the modified, colored title back to the menu item, so it will be displayed in red in the action bar or options menu.

**4. Create a Menu for the Action Bar**

To add buttons or actions (like search or settings) to the Action Bar, we need to create a `menu` resource.

**Create the `menu` XML file:**

1. Right-click on the `res` folder.

2. Select **New → Android Resource Directory**.

3. Choose **menu** as the resource type, and click OK.

4. **Right-click** on the new `menu` folder and choose **New → Menu resource file.**

5. Name it `**main\_menu.xml**`.

**Open `res/menu/main\_menu.xml` and add the following code:**

```xml

<menu xmlns:android="http://schemas.android.com/apk/res/android">

<item

android:id="@+id/action\_search"

android:title="Search"

android:icon="@android:drawable/ic\_menu\_search"

android:showAsAction="ifRoom|collapseActionView"

android:actionViewClass="android.widget.SearchView" />

<item

android:id="@+id/action\_settings"

android:title="Settings"

android:showAsAction="never" />

</menu>

```

**Explanation:**

**action\_search:** This is a search item with a search icon (`ic\_menu\_search`), which collapses into a `SearchView` when clicked.

**action\_settings:** This is a settings option that only appears in the overflow menu (because of `showAsAction="never"`).

**5. Customizing the Toolbar’s Appearance (Optional)**

You can customize the toolbar further by modifying its appearance in the `styles.xml` file.

Open `res/values/styles.xml` and add the following:

```xml

**<resources>**

**<!-- Base application theme -->**

**<style name="Theme.ActionBarDemo" parent="Theme.AppCompat.Light.DarkActionBar">**

**<item name="colorPrimary">#6200EE</item>**

**<item name="colorPrimaryDark">#3700B3</item>**

**<item name="colorAccent">#03DAC5</item>**

**</style>**

**<!-- Toolbar title text appearance -->**

**<style name="ToolbarTitleText">**

**<item name="android:textColor">#FFFFFF</item>**

**<item name="android:textSize">20sp</item>**

**</style>**

**</resources>**

```

**Create a custom text appearance for the menu item**

Define a custom style in your res/values/styles.xml:

<resources>

<style name="MenuTextStyle" parent="TextAppearance.AppCompat.Menu">

<item name="android:textColor">#FF0000</item> <!-- Red color -->

</style>

</resources>

**6. Run the App**

- Now you can run the app on an emulator or physical device.

- You should see the toolbar at the top with the title "My Action Bar".

- Clicking the three dots (overflow menu) should display the `Settings` item, and if there's space, the search icon will appear in the action bar.

Now, your Action Bar with the `Toolbar` is complete! You can customize it further by adding more functionality or design elements.