Labsheet 5 – Friday 24th September 2021

Styles, Themes, and Action bars

Software Development for Portable Devices

Information about HD TA:

AASHITA DUTTA <h20201030130@hyderabad.bits-pilani.ac.in>

Breakout room link: https://meet.google.com/erq-uxqi-egv

Information about FD TA:

ARUNEEMA DESHMUKH <f20180568@hyderabad.bits-pilani.ac.in>

Breakout room link: https://meet.google.com/pvo-osiz-cuh

In this practical, you learn how to add an App Bar (also known as Action Bar) to an app. The App Bar provides a visual structure and interactive elements that are familiar to users. Using the app bar makes your app consistent with other Android apps, allowing users to quickly understand how to operate your app and have a great experience. The key functions of the app bar are as follows:

- A dedicated space for giving your app an identity and indicating the user's location in the app. For example, an email app might use the app bar to indicate whether the user is in their inbox or junk folder.
- Access to important actions in a predictable way, such as search or sharing content.
- Support for navigation and view switching (with tabs or drop-down lists).

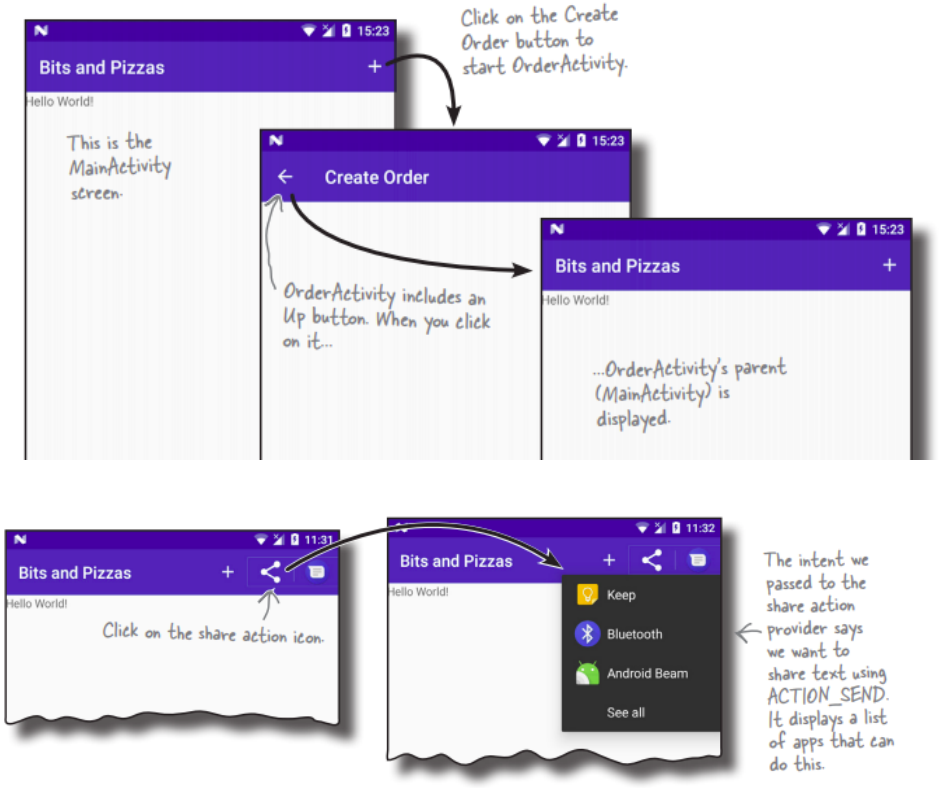
Two ways to implement an app bar

1. Some themes set up an ActionBar as an app bar by default
2. Using the Toolbar widget as an app bar. The Toolbar makes it easy to set up an app bar that works on the widest range of devices and gives you room to customize your app bar later on as your app develops.

We will create an App Bar both ways.

1. Create a project and add a basic app bar by applying a theme.
2. Replace the basic app bar with a toolbar.
3. Add an action to MainActivity's app bar to open another Activity.
4. Implement the Up (Left) button to get back to the MainActivity.
5. Add a share action provider.

App Screenshots: Bits and Pizzas



Task 1: Create a new project with a basic app bar- Bits and Pizzas

1. Select **Empty Activity** as Project Template. You can leave the name of the activity as Main Activity.
2. All applications that use the default theme provided by the Android (**app/src/main/res/values/themes.xml**) contain an ActionBar by default. However, developers can customize it in several ways depending upon their needs.
3. A **theme** is a style that's applied to an activity or application so that the app has a consistent look and feel. It controls such things as the color of the activity background and app bar, and the style of the text. https://developer.android.com/guide/topics/ui/look-and-feel/themes

Material Components themes

The following is the list of Material Components themes you can use to get the latest component styles and theme-level attributes.

- `Theme.MaterialComponents`
- `Theme.MaterialComponents.NoActionBar`
- `Theme.MaterialComponents.Light`
- `Theme.MaterialComponents.Light.NoActionBar`
- `Theme.MaterialComponents.Light.DarkActionBar`
- `Theme.MaterialComponents.DayNight`
- `Theme.MaterialComponents.DayNight.NoActionBar`
- `Theme.MaterialComponents.DayNight.DarkActionBar`

4. Apply a theme by hardcoding in AndroidManifest.xml: Update the android:theme `"@style/Theme.BitsAndPizzas"` attribute in the file to specify the name of the theme you want to use. This approach works well if you want to apply a basic theme without making any changes to it. You can see the themes that are available as they are suggested by the IDE while typing.

   Eg `android:theme="@style/Theme.MaterialComponents.Light.DarkActionBar">`

5. Apply a theme using a style: To customize a theme, we will start by creating a style resource.

themes.xml : Add style with name as AppTheme, and its parent as Theme.MaterialComponents.Light.DarkActionBar from where it will inherit its properties. By changing the color values, we override an existing theme's properties in the style resource file. The value can either be a hardcoded hexadecimal color value or a reference to a color resource.

```
<style name="AppTheme" parent="Theme.MaterialComponents.Light.DarkActionBar">
    <!-- Customize your theme here. -->
    <item name="colorPrimary">@color/colorPrimary</item>
    <item name="colorPrimaryDark">@color/colorPrimaryDark</item>
    <item name="colorAccent">@color/colorAccent</item>
</style>
```

colors.xml : Add the color resources

- colorPrimary refers to the "main color", it is the color of the app bar.
- colorPrimaryDark is the color of the status bar
- colorAccent refers to the color of any UI controls such as editable text views or checkboxes.

```
<color name="colorPrimary">#3F51B5</color>
<color name="colorPrimaryDark">#303F9F</color>
<color name="colorAccent">#FF4081</color>
```

AndroidManifest.xml : Update the android:theme `android:theme="@style/AppTheme">`

Additional Ref:
https://guides.codepath.com/android/developing-custom-themes
https://material.io/design/color/dark-theme.html https://material.io/develop/android/theming/dark
https://github.com/material-components/material-components-android/blob/master/docs/getting-started.md

Questions:

1. What is the difference between Theme.AppCompat and Theme.Material themes in Android?
2. How are styles different from themes? How do they change the application design?

Task 2: Replace the basic app bar with a toolbar

Behind the scenes, any activity that acquires an app bar via a theme uses the ActionBar class for its app bar. The most recent app bar features, however, have been added to the **Toolbar class** instead. We will be using MaterialToolbar that extends androidx.appcompat.widget.Toolbar (Toolbar) and behaves as MaterialComponent. For more info, refer App bars: top

A toolbar is a type of view that you add to your layout just as you would any other type of view, and this makes it much easier to position and control than a basic app bar.

1. Remove the existing app bar by applying a theme which does not include an App Bar. Change
   `<style name="AppTheme" parent="Theme.MaterialComponents.Light.DarkActionBar">`
   To
   `<style name="AppTheme" parent="Theme.MaterialComponents.Light.NoActionBar">`
2. To add a toolbar to the activity_main layout.

activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    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">
    <com.google.android.material.appbar.AppBarLayout
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        android:layout_width="match_parent"
        android:layout_height="wrap_content">
        <com.google.android.material.appbar.MaterialToolbar
        android:id="@+id/mtoolbar"
        android:layout_width="match_parent"
        android:layout_height="?attr/actionBarSize"
        android:background="?attr/colorPrimary"
        android:theme="@style/ThemeOverlay.MaterialComponents.Dark.ActionBar"/>
    </com.google.android.material.appbar.AppBarLayout>

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello World!"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

</androidx.constraintlayout.widget.ConstraintLayout>
```

Run the app

3. The above approach works if we want to add the toolbar to a single activity. If we want to add a toolbar to multiple activities, instead of changing the layout of each activity, we can define the toolbar in a separate layout and then include the toolbar layout in each activity.
   Create a new layout file (app/src/res/main/layout folder. Right-click → File menu and choose New → Layout resource file. When prompted, give the layout file a name of “toolbar_main” and then click on OK.)

Replace the existing code in toolbar_main with the Toolbar code (Cut from activity_main - Paste in toolbar_main)
Include tag is used to display one layout inside another.

toolbar_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<com.google.android.material.appbar.AppBarLayout
xmlns:android="http://schemas.android.com/apk/res/android"
   xmlns:app="http://schemas.android.com/apk/res-auto"
   app:layout_constraintStart_toStartOf="parent"
   app:layout_constraintTop_toTopOf="parent"
   app:layout_constraintEnd_toEndOf="parent"
   android:layout_width="match_parent"
   android:layout_height="wrap_content"
>
   <com.google.android.material.appbar.MaterialToolbar
       android:id = "@+id/mtoolbar"
       android:layout_width="match_parent"
       android:layout_height="?attr/actionBarSize"
       android:background="?attr/colorPrimary"
       android:theme="@style/ThemeOverlay.MaterialComponents.Dark.ActionBar"/>
</com.google.android.material.appbar.AppBarLayout>
```

activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
xmlns:android="http://schemas.android.com/apk/res/android"
   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">

   <include
       layout="@layout/toolbar_main"
       android:id="@+id/appbarlayout" />

   <TextView
       android:layout_width="wrap_content"
       android:layout_height="wrap_content"
       android:text="Hello World!"
       app:layout_constraintBottom_toBottomOf="parent"
       app:layout_constraintLeft_toLeftOf="parent"
       app:layout_constraintRight_toRightOf="parent"
       app:layout_constraintTop_toTopOf="parent" />

</androidx.constraintlayout.widget.ConstraintLayout>
```

4. We will set the toolbar as the main activity's app bar since the toolbar is currently just displayed and doesn't yet have any app bar functionality. Also, we would like the toolbar to display the title of the app in main activity.

To get the toolbar to behave like an app bar, we need to call the AppCompatActivity's **setSupportActionBar()** method in the activity's onCreate() method, which takes one parameter: the toolbar.

MainActivity.java

```
package com.example.bitsandpizzas;

import androidx.appcompat.app.AppCompatActivity;
import androidx.appcompat.widget.Toolbar;

import android.os.Bundle;

public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        Toolbar toolbar = findViewById(R.id.mtoolbar);
        setSupportActionBar(toolbar);
    }
}
```

Run the app

Questions:

1. What is the ?attr prefix?
2. What is a theme overlay?
3. What's the difference between app bar, action bar, and toolbar?

Task 3: Add actions to the app bar

We would now like to add actions to the app bar. These are buttons or text in the app bar that are clickable. We're going to add a "Create Order" button to the app bar to start a new activity, OrderActivity.

1. Create a new activity "OrderActivity" and name the layout "activity_order".
   Try it yourself: We want OrderActivity to display the same toolbar as MainActivity. Update the activity_order.xml layout to include the toolbar. Update OrderActivity so that it uses the toolbar we set up in the layout as its app bar.

activity_order.xml

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    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=".OrderActivity">

    <include
        layout="@layout/toolbar_main"
        android:id="@+id/appbarlayout" />

</androidx.constraintlayout.widget.ConstraintLayout>
```

2. Changing the text on AppBar: The app's label attribute in **AndroidManifest.xml** tells Android what text to display in the app bar (by default, the app name). When the OrderActivity is on, the toolbar should display the text "Create Order" instead of the app's name. To do so, create a string resource `<string name="create_order">Create Order</string>.` To override the app's label, add a new label attribute to OrderActivity's element to display the new text in AndroidManifest.xml.
```
<activity android:name=".OrderActivity" android:label="@string/create_order">
</activity>
```

3. Add resources for the action's icon (+) and text: The icon usually gets displayed if the action appears in the main area of the app bar. If the action doesn't fit in the main area, it's automatically moved to the app bar overflow, and the title appears instead. Icons has some standard icons that can be used. For this app , get the resources from bitsandpizzas (Copy Paste the folders to **app/src/main/res**). For the text add a string resource `<string name="create_order_title">Create Order</string>`

4. A **menu resource** file tells Android what actions that should appear on the app bar. An app can contain multiple menu resource files. For example, one can create a separate menu resource file for each set of actions; this is useful if we want different activities to display different actions on their app bars. We're going to create a new menu resource file called menu_main.xml in the folder app/src/main/res/menu.

   To create the menu resource file: Select the app/src/main/res folder, go to the File menu, and choose New. Then choose the option to create a new Android resource file. You'll be prompted for the name of the resource file and the type of resource. Give it the name "menu_main" and a resource type of "Menu", and make sure that the directory name is menu. When you click on OK, Android Studio will create the file for you and add it to the app/src/main/res/menu folder.

   The menu resource file has a <menu> element at its root. Every <item> element describes a separate action

menu_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto">
    <item android:id="@+id/action_create_order"
        android:title="@string/create_order_title"
        android:icon="@drawable/ic_add_white_24dp"
        android:orderInCategory="1"
        app:showAsAction="ifRoom" />
```

```
</menu>
```

5. Add the menu to the app bar with the **onCreateOptionsMenu()** method. This method runs when the app bar's menu gets created. It takes one parameter, a Menu object that's a Java representation of the menu resource file. Also, to react to action item clicks, we need to add **onOptionsItemSelected()** method.The onOptionsItemSelected() method runs whenever an action gets clicked. It takes one parameter, a MenuItem object that represents the action on the app bar that was clicked.

MainActivity.java

```
}package com.example.bitsandpizzas;

import androidx.appcompat.app.AppCompatActivity;
import androidx.appcompat.widget.Toolbar;

import android.content.Intent;
import android.os.Bundle;
import android.view.Menu;
import android.view.MenuItem;

public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        Toolbar toolbar = findViewById(R.id.mtoolbar);
        setSupportActionBar(toolbar);
    }

    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        // Inflate the menu; this adds items to the app bar.
        getMenuInflater().inflate(R.menu.menu_main, menu);
        return super.onCreateOptionsMenu(menu);
    }

    @Override
    public boolean onOptionsItemSelected(MenuItem item) {
        switch (item.getItemId()) {
            case R.id.action_create_order:
                //Code to run when the Create Order item is clicked
                Intent intent = new Intent(this, OrderActivity.class);
                startActivity(intent);
                return true;
            default:
                return super.onOptionsItemSelected(item);
        }
    }
}
```

Run the app

Questions:

1. What is OrderInCategory attribute and showAsAction attribute in menu_main.xml?
2. Menu resource file has <item> under <menu>. What does this item describe? Can you implement one more item under the menu to view order?
3. What do you mean by the ifRoom value that showAsAction has? What are the other values that showAsAction can take?

Task 4: Implement the Up (Left) button to return to the MainActivity

1. The Up button enables the user to navigate up a hierarchy of activities in the app. You declare this hierarchy in AndroidManifest.xml by specifying the parent of each activity. As an example, we want the user to be able to navigate from OrderActivity to MainActivity when they press the Up button, so this means that MainActivity is the parent of OrderActivity.

```
<activity android:name=".OrderActivity"
android:label="@string/create_order"
android:parentActivityName=".MainActivity">
</activity>
```

2. Next, to enable the Up button from within the activity code, we will need to get a reference to the app bar using the activity's **getSupportActionBar()** method. This returns an object of type ActionBar. We will then call the ActionBar **setDisplayHomeAsUpEnabled()** method, passing it a value of true.

OrderActivity.java

```
package com.example.bitsandpizzas;

import androidx.appcompat.app.ActionBar;
import androidx.appcompat.app.AppCompatActivity;
import androidx.appcompat.widget.Toolbar;

import android.os.Bundle;

public class OrderActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_order);
        Toolbar toolbar = findViewById(R.id.mtoolbar);
        setSupportActionBar(toolbar);
        ActionBar actionBar = getSupportActionBar();
        actionBar.setDisplayHomeAsUpEnabled(true);
    }
}
```

Run the app

Questions:

1. How is the **Up** button different from the **back** button on the mobile device?

Task 5: Add a share action provider ShareActionProvider

An action provider is an action that defines its own appearance and behavior. The share action provider defines its own icon, so we don't have to add it. When a user clicks on it, it provides a list of apps you can use to share content. It adds a separate icon for the most commonly used app the user chooses to share content with.

The activity creates an intent and passes it to the share action provider. When the user clicks on the share action, the share action uses the intent to present the user with a list of apps that can deal with the intent.

1. We will add a share action to the app bar by including it in the menu resource file. Add string resource `<string name="action_share">Share</string>`

menu_main.xml: Add item

```
<item android:id="@+id/action_share"
   android:title="@string/action_share"
   android:orderInCategory="2"
   app:showAsAction="ifRoom"
   app:actionProviderClass="androidx.appcompat.widget.ShareActionProvider" />
```

2. To get the share action to share content when it's clicked, we need to specify what to share in the activity code. This is done by passing the share action provider an intent using its **setShareIntent()** method.

```
package com.example.bitsandpizzas;

import androidx.appcompat.app.AppCompatActivity;
import androidx.appcompat.widget.ShareActionProvider;
import androidx.appcompat.widget.Toolbar;
import androidx.core.view.MenuItemCompat;

import android.content.Intent;
import android.os.Bundle;
import android.view.Menu;
import android.view.MenuItem;

public class MainActivity extends AppCompatActivity {

   private ShareActionProvider shareActionProvider;

   @Override
   protected void onCreate(Bundle savedInstanceState) {
       super.onCreate(savedInstanceState);
       setContentView(R.layout.activity_main);
       Toolbar toolbar = findViewById(R.id.mtoolbar);
       setSupportActionBar(toolbar);
   }

   @Override
   public boolean onCreateOptionsMenu(Menu menu) {
       // Inflate the menu; this adds items to the app bar.
       getMenuInflater().inflate(R.menu.menu_main, menu);
       MenuItem menuItem = menu.findItem(R.id.action_share);
       shareActionProvider =
               (ShareActionProvider)
MenuItemCompat.getActionProvider(menuItem);
       setShareActionIntent("Are you hungry?");
```

```
        return super.onCreateOptionsMenu(menu);
    }

    @Override
    public boolean onOptionsItemSelected(MenuItem item) {
        switch (item.getItemId()) {
            case R.id.action_create_order:
                //Code to run when the Create Order item is clicked
                Intent intent = new Intent(this, OrderActivity.class);
                startActivity(intent);
                return true;
            default:
                return super.onOptionsItemSelected(item);
        }
    }

    private void setShareActionIntent(String text) {
        Intent intent = new Intent(Intent.ACTION_SEND);
        intent.setType("text/plain");
        intent.putExtra(Intent.EXTRA_TEXT, text);
        shareActionProvider.setShareIntent(intent);
    }
}
```

Homework Assignment:

Extend the app for PVR cinemas from the last lab sheet and customize themes and styles consistent throughout the app and action bar having the following actions (refer to screenshot below):

Note: In left of app bar at (1) instead of navigation bar, you can implement back button to go up one page

- Navigation/Back - To go back up one page (from Detailed Activities) (1)
- Title - Title of the app bar (2)
- Search - Search for a movie (3)
- Like - Like the movie that you wishlisted (3)
- Menu - Having actions like book tickets, cancel tickets, edit, etc. (4)