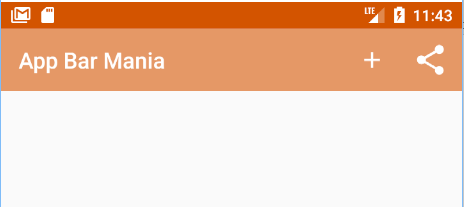
Design Support Library

We will start with the app (*AppBarMania\_START.zip*) developed during the “*AppBar*” lesson in week 6.



We’ll change the *AppBarMania* app so that it uses tab navigation.

Edit *strings.xml* and change the value of *app\_name* from “App Bar Mania” to “Economy Supermarket”, as shown below:

01. strings.xml.txt

<string name="app\_name">Economy Supermarket</string>

We will create fragments for the different options. When the user clicks on one of the tabs, the fragment for that option will be displayed:

Economy Supermarket

Home Bakery Produce Seafood

Economy Supermarket

Top Fragment

Bakery

Produce

Seafood

TopFragment

BakeryFragment

ProduceFragment

SeafoodFragment

What are we going to do?

1. Create the fragments
2. Enable swipe navigation between the fragments (
3. Add a tab layout

### Create TopFragment.

02. TopFragment.txt

File >> New… >> Fragment >> Fragment (Blank)

Name the fragment “*TopFragment*”. Replace code in *TopFragment.java* with the code below:

public class TopFragment extends android.support.v4.app.Fragment {

@Override

public View onCreateView(LayoutInflater inflater, ViewGroup container, Bundle savedInstanceState) {

return inflater.inflate(R.layout.fragment\_top, container, false);

}

}

Add the following to *strings.xml:*

<string name="title\_top">Top Fragment</string>

03. strings.xml.txt

Update *fragment-top.xml* as follows:

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

xmlns:tools="http://schemas.android.com/tools"

android:layout\_width="match\_parent"

android:layout\_height="match\_parent"

android:orientation="vertical"

tools:context="ca.bcit.appbarmania.TopFragment">

<TextView

android:layout\_width="match\_parent"

04. fragment-top.xml.txt

android:layout\_height="match\_parent"

android:text="@string/title\_top" />

</LinearLayout>

### Create BakeryFragment

File >> New… >> Fragment >> Fragment (Blank)

Name the fragment “BakeryFragment” and uncheck the option to create a layout. Add following string-array to *strings.xml*:

05. strings.xml.txt

<string-array name="bakery\_items">

<item>Croissant</item>

<item>Muffin</item>

<item>Cookie</item>

</string-array>

Change code in *BakeryFragment.java* so that it is a *ListFragment* as shown below:

06. BakeryFragment.txt

public class BakeryFragment extends android.support.v4.app.ListFragment {

@Override

public View onCreateView(LayoutInflater inflater, ViewGroup container, Bundle savedInstanceState) {

ArrayAdapter<String> adapter = new ArrayAdapter<>(

inflater.getContext(),

android.R.layout.simple\_list\_item\_1,

getResources().getStringArray(R.array.bakery\_items));

setListAdapter(adapter);

return super.onCreateView(inflater, container, savedInstanceState);

}

}

### Create ProduceFragment

Just like *BakeryFragment*, create a fragment named *ProduceFragment* without a layout. Add s string-array to *strings.xml* as shown below:

07. strings.xml.txt

<string-array name="produce\_items">

<item>Carrot</item>

<item>Cucumber</item>

<item>Cabbage</item>

</string-array>

Change code in *ProduceFragment.java* so that it is a *ListFragment* and have the list get populated with produce items as shown below:

08. ProduceFragment.txt

public class ProduceFragment extends android.support.v4.app.ListFragment {

@Override

public View onCreateView(LayoutInflater inflater, ViewGroup container, Bundle savedInstanceState) {

ArrayAdapter<String> adapter = new ArrayAdapter<>(

inflater.getContext(),

android.R.layout.simple\_list\_item\_1,

getResources().getStringArray(R.array.produce\_items));

setListAdapter(adapter);

return super.onCreateView(inflater, container, savedInstanceState);

}

}

### Create SeafoodFragment

Similar to *BakeryFragment* and *ProduceFragment*, create a fragment named *SeafoodFragment* without a layout. Add s string-array to *strings.xml* as shown below:

09. strings.xml.txt

<string-array name="seafood\_items">

<item>Crab</item>

<item>Shirimp</item>

<item>Salmon</item>

</string-array>

Change code in *SeafoodFragment.java* so that it is a *ListFragment* and have the list get populated with seafood items as shown below:

10. SeafoodFragment.txt

public class SeafoodFragment extends android.support.v4.app.ListFragment {

@Override

public View onCreateView(LayoutInflater inflater, ViewGroup container, Bundle savedInstanceState) {

ArrayAdapter<String> adapter = new ArrayAdapter<>(

inflater.getContext(),

android.R.layout.simple\_list\_item\_1,

getResources().getStringArray(R.array.seafood\_items));

setListAdapter(adapter);

return super.onCreateView(inflater, container, savedInstanceState);

}

}

The *ViewPager* is used to swipe through the different fragments. Make the following changes to activity\_main.xml:

1. Add a *ViewPager* to *activity\_main.xml*.
2. delete the “Hello World” *TextView*.
3. Change the layout to *LinearLayout* with *vertical* orientation

## Add ViewPager

11. activity\_main.xml.txt

Your *activity\_main.xml* will look like the following:

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

<LinearLayout 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"

android:orientation="vertical"

tools:context="ca.bcit.appbarmania.MainActivity">

<include layout="@layout/toolbar\_main" android:id="@+id/toolbar" />

<android.support.v4.view.ViewPager

android:id="@+id/pager"

android:layout\_width="match\_parent"

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

</LinearLayout>

Tell the *ViewPages* about its pages with a fragment pager adapter. Inside *MainActivity.java*, create an inner class named *SectionsPageAdapter* and add to it the code below:

12. SectionsPageAdapter.txt

public class SectionsPageAdapter extends FragmentPagerAdapter{

public SectionsPageAdapter(FragmentManager fm) { super(fm); }

@Override

public int getCount() { return 4; }

@Override

public Fragment getItem(int position) {

switch (position) {

case 0:

return new TopFragment();

case 1:

return new BakeryFragment();

case 2:

return new ProduceFragment();

case 3:

return new SeafoodFragment();

}

return null;

}

}

Attach the fragment pager adapter to the *ViewPager*. This is done by adding the following code to the bottom of *onCreate()* method in *MainActivity.java*:

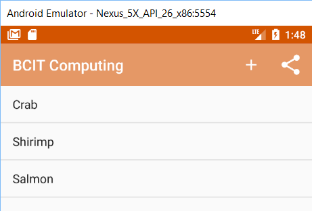
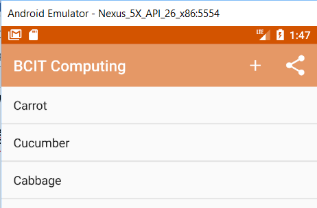
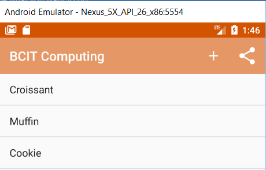
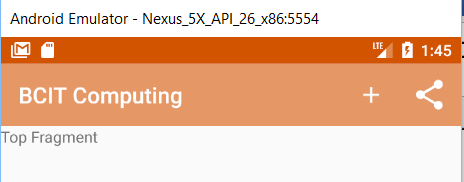
13. onCreate.txt

// Attach the SectionsPageAdapter to the ViewPager  
SectionsPageAdapter pagerAdapter = new SectionsPageAdapter(getSupportFragmentManager());

[](https://www.bing.com/images/search?view=detailV2&ccid=g6nYHgjm&id=9DCD800E35FF6162FE4C44ABC6C70A5783B46357&thid=OIP.g6nYHgjml5I7LyjheiVEGwElEs&q=run+logo&simid=608050058236200411&selectedIndex=67)ViewPager pager = (ViewPager) findViewById(R.id.pager);

pager.setAdapter(pagerAdapter);

Run your app.



## Add tab navigation to MainActivity

Tabs are added using two components: *TabLayout* and *AppBarLayout*.

We will make the following changes to *activity\_main.xml*:

1. Add “Android Design Support Library” to your project
2. Add a *TabLayout* control above the *ViewPager* control
3. Contain the *<include...>* and *TabLayout* controls inside an *AppBarLayout* control.

### Add “Android Design Support Library” to project

File >> Project Structure…

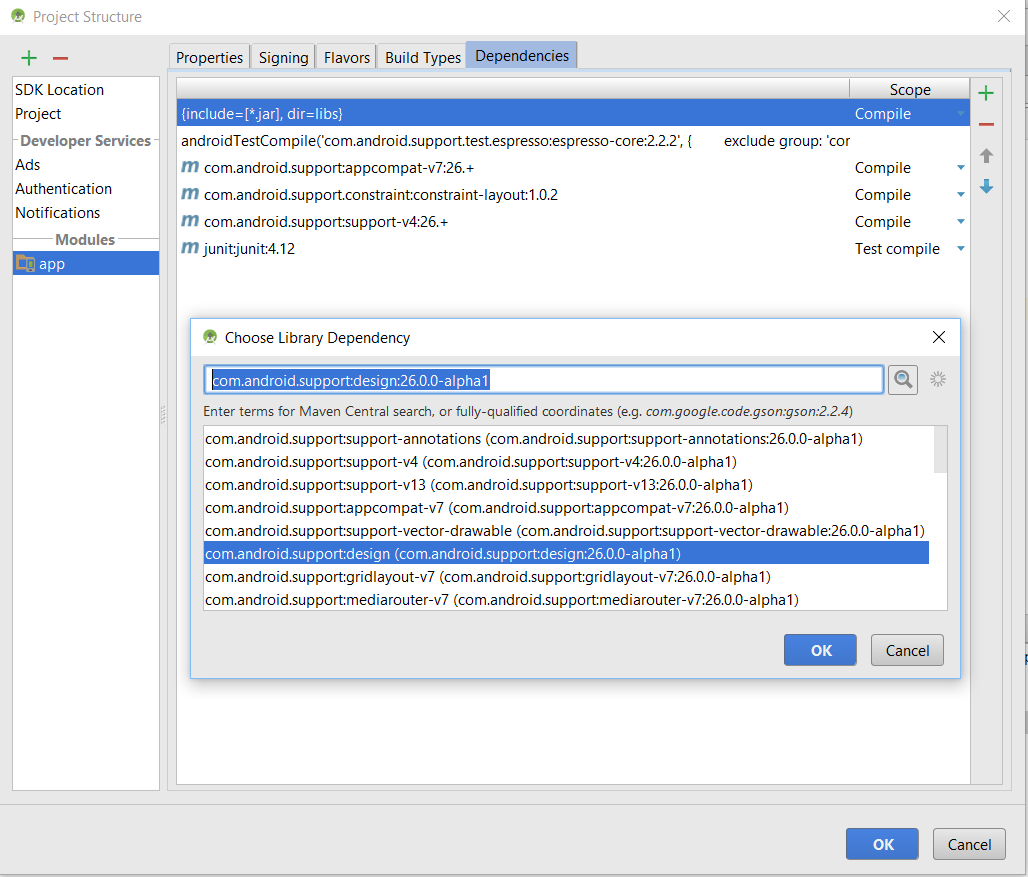
On left side select app

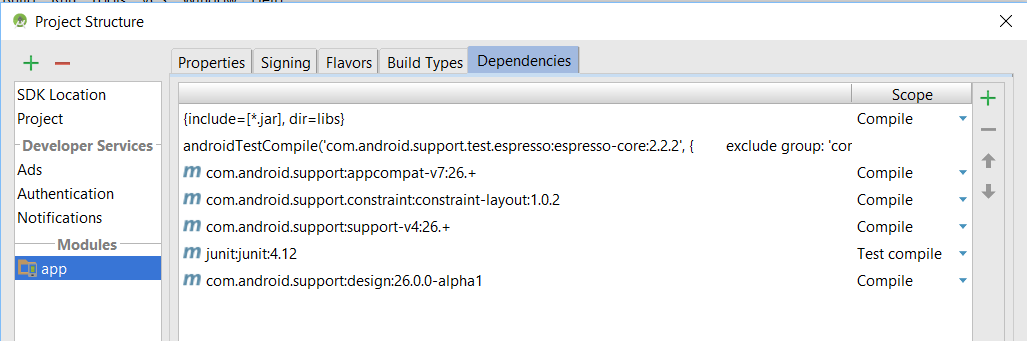
Select Dependencies tab

Click on + at top right corner

Select *Library Dependency*

Choose *com.android.support:design:26.0.0-alpha1* from the list:





Add the following to strings.xml:

14. strings.xml.txt

<string name="home\_tab">Home</string>

<string name="bakery\_tab">Bakery</string>

<string name="produce\_tab">Produce</string>

<string name="seafood\_tab">Seafood</string>

Change your activity\_main.xml so that it looks like this:

15. activity\_main.xml.txt

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

<LinearLayout 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"

android:orientation="vertical"

tools:context="ca.bcit.appbarmania.MainActivity">

<android.support.design.widget.AppBarLayout

android:layout\_width="match\_parent"

android:layout\_height="wrap\_content"

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

<include layout="@layout/toolbar\_main" android:id="@+id/toolbar" />

<android.support.design.widget.TabLayout

android:id="@+id/tabs"

android:layout\_width="match\_parent"

android:layout\_height="wrap\_content" />

</android.support.design.widget.AppBarLayout>

<android.support.v4.view.ViewPager

android:id="@+id/pager"

android:layout\_width="match\_parent"

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

</LinearLayout>

To add text to each of the tabs, it is necessary to implement the *getPageTitle()* method in the *MainActivity.java* inner class *SectionsPageAdapter* as shown below:

16. getPageTitle.txt

@Override

public CharSequence getPageTitle(int position) {

switch (position) {

case 0:

return getResources().getText(R.string.home\_tab);

case 1:

return getResources().getText(R.string.bakery\_tab);

case 2:

return getResources().getText(R.string.produce\_tab);

case 3:

return getResources().getText(R.string.seafood\_tab);

}

return null;

}

Finally, we need to attach the *ViewPager* to the *TabLayout*. Add the following code to the bottom of *onCreate()* method in *MainActivity.java*:

17. onCreate.txt

// Attach the ViewPager to the TabLayout

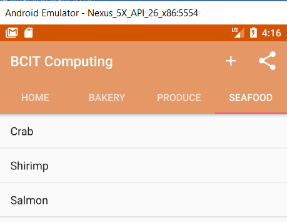
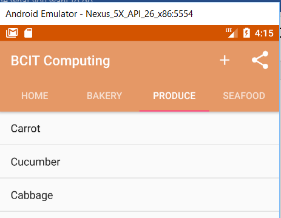
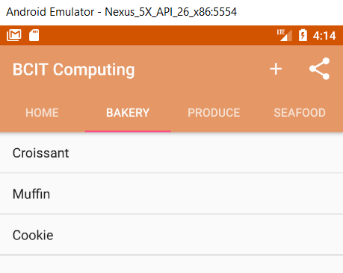
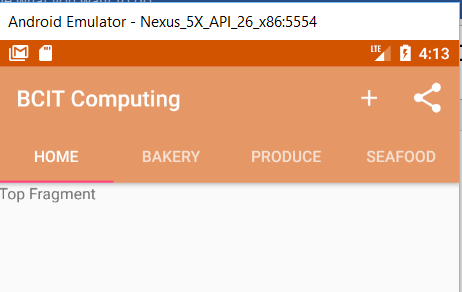
TabLayout tabLayout = (TabLayout) findViewById(R.id.tabs);

tabLayout.setupWithViewPager(pager);

Test the application.

You can swipe through the fragments and navigate to each fragment by clicking on the appropriate tab.

[](https://www.bing.com/images/search?view=detailV2&ccid=g6nYHgjm&id=9DCD800E35FF6162FE4C44ABC6C70A5783B46357&thid=OIP.g6nYHgjml5I7LyjheiVEGwElEs&q=run+logo&simid=608050058236200411&selectedIndex=67)



# Implementing Material Design

Material design was introduced with Lollipop to give a consistent look across all Android apps. With material design, you can do the following:

1. Floating action bars (FABs)
2. Snackbars
3. Animate toolbars
4. Navigation drawer layout

We will do the following:

1. Enable *MainActivity* toolbar to scroll
2. Add a collapsing toolbar
3. Add a FAB
4. Make FAB display a *Snackbar*

### Enable MainActivity toolbar to scroll

We must change *MainActivity* to enable the toolbar to scroll. We need to use *CoordinatorLayout* to coordinate animations between views. In *activity\_main.xml*:

1. Replace the *LinearLayout* with *CoordinatorLayout*
2. Delete *LinearLayout’s* vertical orientation

In *activity\_main.xml*, delete the *orientation* attribute of *LinearLayout*. Replace the *LinearLayout* container with the following *CoordinatorLayout* container:

18. CoordinatorLayout.txt

<android.support.design.widget.CoordinatorLayout

Next, we need to:

1. Mark the view that the user will scroll
2. Tell the toolbar to respond to it.

Add the following namespace and attribute to the toolbar defined in in *toolbar\_main.xml*:

19. toolbar\_main.xml.txt

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

app:layout\_scrollFlags="scroll|enterAlways"

Next, add the following to *ViewPager* attribute in *activity\_main.xml* to indicate the view whose content that you expect the user to scroll:

20. layout\_behavior.txt

app:layout\_behavior="@string/appbar\_scrolling\_view\_behavior"

### Add scrollable content to TopFragment

21. strings.xml.txt

Open *strings.xml* and add the following to it:

<string name="company\_name">BCIT</string>

<string name="bcit\_image">BCIT image</string>

<string name="bcit\_bby\_campus">BCIT Burnaby Campus</string>

<string name="home\_text">

The British Columbia Institute of Technology (also referred to as BCIT), is a public polytechnic institution of higher education. The post-secondary institute has five campuses located in the Metro Vancouver region, with its main campus in Burnaby, British Columbia, Canada. There is also the Aerospace Technology Campus in Richmond, the Marine Campus in the City of North Vancouver, Downtown campus in Vancouver, and Annacis Island Campus in Delta. It is provincially chartered through legislation in the College and Institute Act. The school operates as a vocational and technical school, offering apprenticeships for the skilled trades and diplomas and degrees in vocational education for skilled technicians and workers in professions such as engineering, accountancy, business administration, nursing, medicine, architecture, and law.

BCIT was first established as the British Columbia Vocational School in 1960. When BCIT opened its Burnaby campus in 1964, initial enrollment was 498 students. As of 2013, enrollment has grown to 17,453 full-time students and 29,224 part-time students. Since its foundation, the institution has been home to over 125,000 alumni.

</string>

Next, copy *bcit.jpg* & *bby\_campus.jpg* from the snippets *images* folder and paste them in the *res/drawable-nodpi* directory in your project.

We will use the nested scroll view to scroll the contents of *TopFragment*. The *CoordinatorLayout* only listens for nested scroll events. The nested scroll view can only have one direct child.

The image needs a *FrameLayout*.

Below is the full code for *fragment\_top.xml*:

22. fragment\_top.xml.txt

<android.support.v4.widget.NestedScrollView xmlns:android="http://schemas.android.com/apk/res/android"

xmlns:tools="http://schemas.android.com/tools"

android:layout\_width="match\_parent"

android:layout\_height="match\_parent"

android:orientation="vertical"

tools:context="ca.bcit.appbarmania.TopFragment">

<FrameLayout

android:layout\_width="match\_parent"

android:layout\_height="wrap\_content">

<ImageView

android:layout\_width="match\_parent"

android:layout\_height="wrap\_content"

android:id="@+id/bcit\_image"

android:scaleType="centerCrop"

android:src="@drawable/bcit"

android:contentDescription="@string/bcit\_image"/>

<LinearLayout

android:layout\_width="match\_parent"

android:layout\_height="wrap\_content"

android:layout\_marginTop="40dp"

android:layout\_marginLeft="16dp"

android:layout\_marginRight="16dp"

android:padding="16dp"

android:background="#FFFFFF"

android:orientation="vertical">

<TextView

android:textSize="32sp"

android:layout\_width="match\_parent"

android:layout\_height="wrap\_content"

android:text="@string/company\_name" />

<TextView

android:layout\_width="match\_parent"

android:layout\_height="match\_parent"

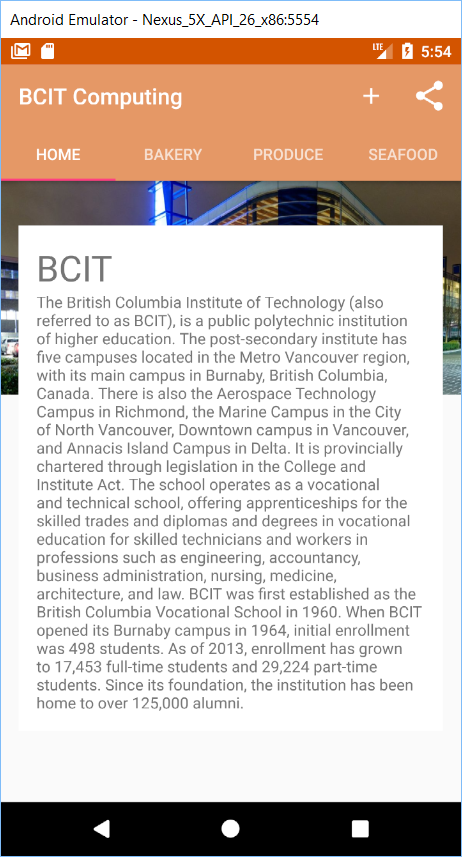
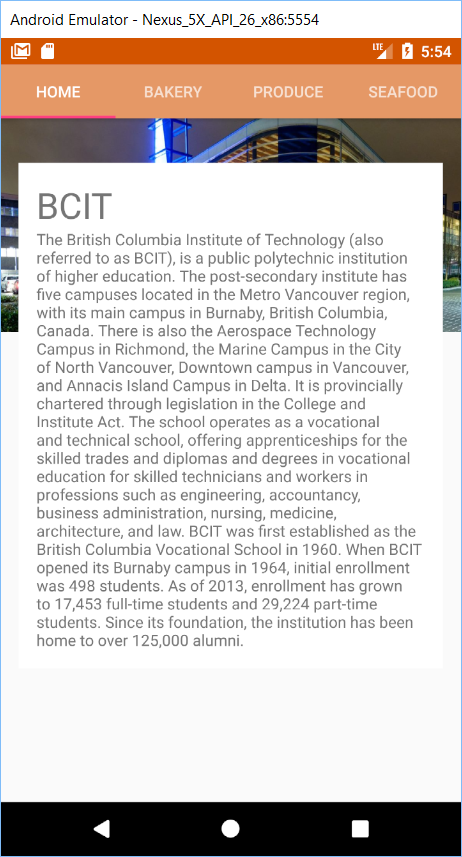
android:text="@string/home\_text" />

</LinearLayout>

</FrameLayout>

</android.support.v4.widget.NestedScrollView>

[](https://www.bing.com/images/search?view=detailV2&ccid=g6nYHgjm&id=9DCD800E35FF6162FE4C44ABC6C70A5783B46357&thid=OIP.g6nYHgjml5I7LyjheiVEGwElEs&q=run+logo&simid=608050058236200411&selectedIndex=67)Test the app to try it out. When you scroll down, the toolbar will disappear and it will appear again when you scroll in the opposite direction.

# Collapsing toolbar

A collapsing toolbar is one that starts large, shrinks when the user scrolls the screen content up, and grows again when the user scrolls the screen content back down. We will add a collapsing toolbar to the *InfoRequestActivity*.

23. strings.xml.txt

Add the following to *strings.xml*:

<string name="info\_request\_name\_hint">Please enter your name</string>

<string name="info\_request\_details\_hint">Please enter your order</string>

Add a collapsing toolbar to an app bar layout that’s included within the coordinator layout. The collapsing toolbar layout should contain the toolbar you want to collapse.

Replace *activity\_info\_request.xml* with the following:

24. activity\_info\_request.xml.txt

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

<android.support.design.widget.CoordinatorLayout

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="ca.bcit.appbarmania.InfoRequestActivity"

android:id="@+id/coordinator">

<android.support.design.widget.AppBarLayout

android:layout\_width="match\_parent"

android:layout\_height="300dp"

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

<android.support.design.widget.CollapsingToolbarLayout

android:layout\_width="match\_parent"

android:layout\_height="match\_parent"

app:layout\_scrollFlags="scroll|exitUntilCollapsed">

<android.support.v7.widget.Toolbar

android:id="@+id/toolbar"

android:layout\_width="match\_parent"

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

app:layout\_collapseMode="pin" />

</android.support.design.widget.CollapsingToolbarLayout>

</android.support.design.widget.AppBarLayout>

<android.support.v4.widget.NestedScrollView

android:layout\_width="match\_parent"

android:layout\_height="match\_parent"

app:layout\_behavior="@string/appbar\_scrolling\_view\_behavior">

<LinearLayout

android:layout\_width="match\_parent"

android:layout\_height="wrap\_content"

android:orientation="vertical"

android:padding="16dp">

<EditText

android:layout\_width="match\_parent"

android:layout\_height="wrap\_content"

android:hint="@string/info\_request\_name\_hint"/>

<EditText

android:layout\_width="match\_parent"

android:layout\_height="wrap\_content"

android:hint="@string/info\_request\_details\_hint"/>

</LinearLayout>

</android.support.v4.widget.NestedScrollView>

</android.support.design.widget.CoordinatorLayout>

# Explanations:

### Nested scroll view attributes:

It is necessary to tell the coordinator layout which view the user is expected to scroll. This is done by setting the nested scroll view’s *app:layout\_behavior* to *@string/appbar\_scrolling\_view\_behavior*.

### Collapsing toolbar layout attributes:

We want the collapsing toolbar layout to collapse until it’s the size of a standard toolbar. Therefore, we set the attribute to *scroll|exitUntilCollapsed*.

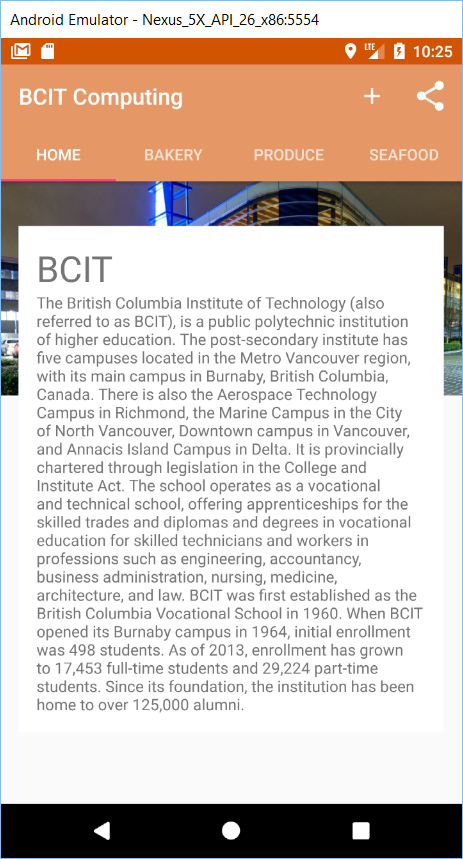
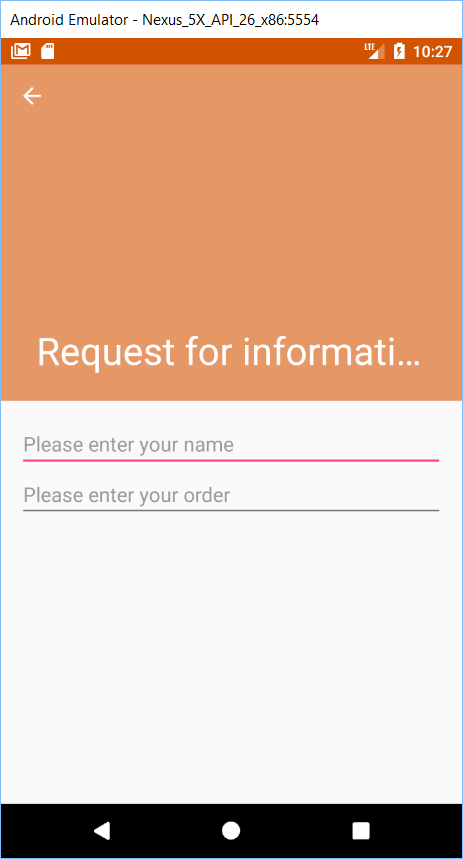
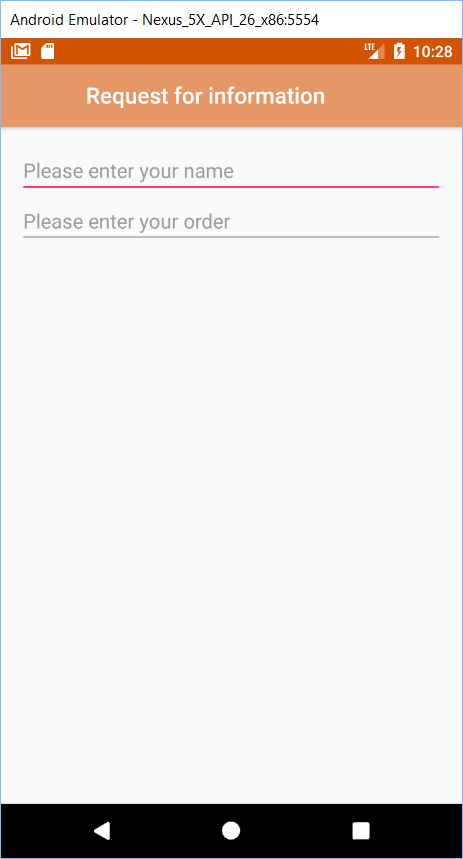
### App bar layouts attributes

We apply a theme for the *AppBarLayout* of *@style/ThemeOverlay.AppCompat.Dark.ActionBar* and set the of *300dp*.

### Toolbar attributes

Set the toolbar *layout\_collapseMode* to *pin* so that items on the toolbar do not scroll off the screen.

[](https://www.bing.com/images/search?view=detailV2&ccid=g6nYHgjm&id=9DCD800E35FF6162FE4C44ABC6C70A5783B46357&thid=OIP.g6nYHgjml5I7LyjheiVEGwElEs&q=run+logo&simid=608050058236200411&selectedIndex=67)Test the app by running it. It starts off fully expanded. As you scroll up, the toolbar collapses.

Let’s add an image to the collapsing toolbar. In *activity\_info\_request.xml,* just add an *ImageView* widget to the *CollapsingToolBarLayout*, specifying the *bby-campus.jpg* image. Here’s the code for the *ImageView* that you should place before the *Toolbar* in *activity\_info\_request.xml:*

25. ImageView.txt

<ImageView

android:layout\_width="match\_parent"

android:layout\_height="match\_parent"

android:scaleType="centerCrop"

android:src="@drawable/bby\_campus"

android:contentDescription="@string/bcit\_bby\_campus"

app:layout\_collapseMode="parallax"/>

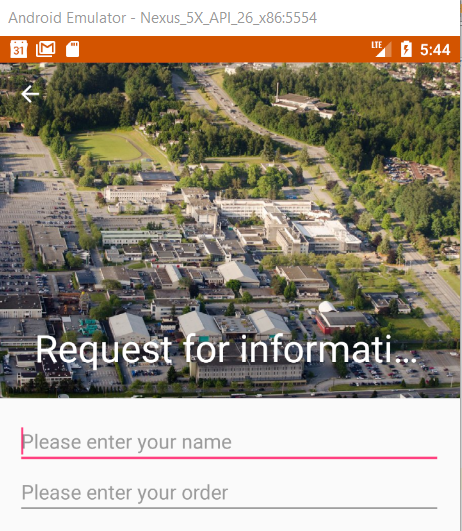
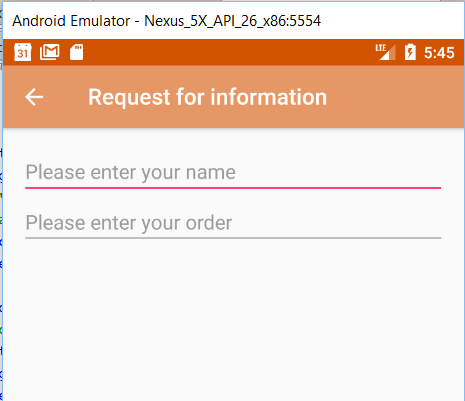
centerCrop >> crop image to fir inside *AppBarLayout*  
parallax >> adds parallax animation so image moves at different rate than scrollable content (optional)

To get the toolbar to revert to a plain background color when it is collapsed, add *contentScrim* attribute to *CollapsingToolbarLayout* as below:

26. contentScrim.txt

app:contentScrim="?attr/colorPrimary"

Test out the latest incarnation of your app and check that there is an image in the collapsing toolbar.

[](https://www.bing.com/images/search?view=detailV2&ccid=g6nYHgjm&id=9DCD800E35FF6162FE4C44ABC6C70A5783B46357&thid=OIP.g6nYHgjml5I7LyjheiVEGwElEs&q=run+logo&simid=608050058236200411&selectedIndex=67) 

# FABs and Snackbars

A FAB is s a floating action bar. It is a circled icon that floats above the UI.

A snackbar is like a toast except that you can interact with it. Unlike with a toast, you can add actions to a snackbar, such as an action to undo an operation.

We will add a FAB to *InfoRequestActivity*. When the user clicks on the FAB, we will display a snackbar that shows a message to the user.

Add the following to *activity\_info\_request.xml* just before the closing *CoordinatorLayout* tag:

27. activity\_info\_request.xml.txt

<android.support.design.widget.FloatingActionButton

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:layout\_gravity="end|bottom"

android:layout\_margin="16dp"

android:src="@drawable/ic\_done\_white\_24dp"

android:onClick="onClickDone"/>

Next, add the following *onClickDone()* method to *InfoRequestActivity.java*:

28. onClickDone.txt

public void onClickDone(View v) {

CharSequence text = "Your request for more information has been received.";

Snackbar snackbar = Snackbar.make(findViewById(R.id.coordinator), text, Snackbar.LENGTH\_LONG);

snackbar.setAction("Undo", new View.OnClickListener() {

@Override

public void onClick(View view) {

Toast t = Toast.makeText(InfoRequestActivity.this, "Undone!", Toast.LENGTH\_LONG);

t.show();

}

});

snackbar.show();

}

Test out your final product.

[](https://www.bing.com/images/search?view=detailV2&ccid=g6nYHgjm&id=9DCD800E35FF6162FE4C44ABC6C70A5783B46357&thid=OIP.g6nYHgjml5I7LyjheiVEGwElEs&q=run+logo&simid=608050058236200411&selectedIndex=67)