Buse Çarık 01/10/18

INTERRA R&D Material Design

Material Design

Tab Layout

https://github.com/InterraMaterialDesign/TabLayout

**INDEX**

1. **INTRODUCTION….….….…….….……….………….……………………….……….3**
   1. **Purpose of this document…………………………………………………………...3**
   2. **Overview………………………….……….…….…………………………………...3**
2. **WHAT IS TABLAYOUT…...………...…………………………………………………4**
   1. **Types…………………………………………………………………………………4**
   2. **Tabs………………………………………………………………………………….5**
   3. **Attributes of Tab Layout…………………………………………………………...7**
   4. **Material Styles………………………………………………………………………9**
3. **VIEWPAGER………………………………………….……….………………………11**
4. **IMPLEMENTATION………………….………………………………………………13**
5. **REQUIREMENTS FOR IMPLEMENTATION…….….………….………………...15**
6. **REFERENCES…………………………………….……………………………………16**

**INTRODUCTION**

**PURPOSE OF THIS DOCUMENT**

The aim of this project is analyzing tabs, tab layout, implementation and their attributes and styles.

**OVERVIEW**

The project that is described in this document, includes tab layout and tab’s description, viewholder concept, styles, where and when it is used and how it should be designed according to Material Design.

**Tab Layout**

Tab Layout is introduced as one of the material design components. It is part of the ***Design Support Library***. The main function of this component is providing a horizontal layout for displaying tabs; it is a container for tab items. Also, it manages the swipe gestures, scrolling behavior, animations and alignment. Tabs can be scrolled horizontally and can be created unlimited number in the Tab Layout. A Tab Layout is usually anchored to the ***Toolbar***, but it can be used with other components like ***Top App Bar***, ***BottomSheet*** or ***Material Card***. One of the advantages of the Material Tab Layout is that it is easy to customize. When the screen is scrolled, the Tab Layout can be fixed into top of the screen if it is associated with the Top App Bar. Or it can be scrolled out of the screen. But it should not be scrolled behind to the Top App Bar. If it is used with Top App Bar, they should be both fixed on the screen or scrolled out of the screen.

Tabs are used with ***Fragments***. The content views of the Tab Items are Fragments which are usually handled with ***ViewPager***. With the content views, they manage the area underneath them. To navigate between fragments, Tab Items can be tapped or scrolled horizontally. Also, the swipe gestures which is also handled by ViewPager on the content views enable to navigate between tabs. The swipe movements can be changed according to the type of the Tabs. The swipeable items should not be placed into the Tab’s content view because undesired component can be swiped.

**Types**

There are two types of tabs which are ***scrollable*** and ***fixed***. The type of the Tab Layout can be set in XML with ***app:tabMode=”fixed/scrollable”*** attribute. Or it can be arranged dynamically with ***setTabMode(int mode)*** method (two options: ***MODE\_FIXED*** or ***MODE\_SCROLLABLE***). And the Tab Layout’s can be found with ***getTabMode()*** method.

**Fixed Tabs**

In that type, all Tabs can be visible on the screen at the same time with a fixed width. The layout is not scrollable. The width of each Tab is determined by dividing the number of Tabs to the width of the screen. If the number of Tab Item is limited, this type can be preferred. It is ideal for the quick switch between related contents like transportation options in a map app. To navigate between Tabs, user can tap to the Tab or swipe the content view horizontally. The text should not be truncated. The number of Tab in that type should be at most four. By default, Tab Layout’s mode is fixed.

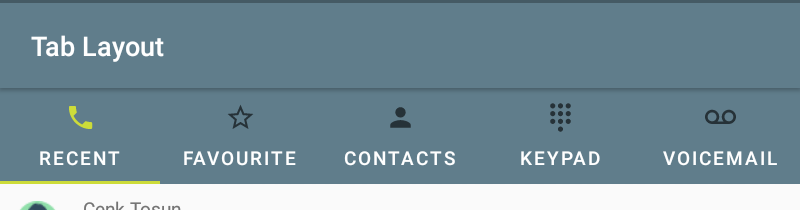
**Scrollable Tabs**

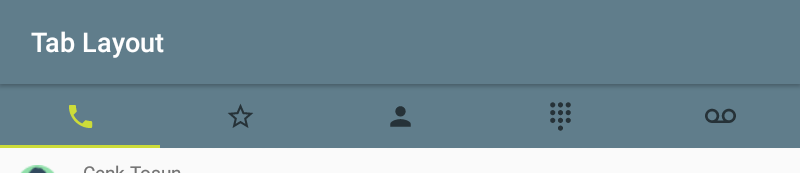
The Tab Layout can be scrollable. Tab Items do not have fixed width. If the number of Tabs does not fit to the screen, this type should be preferred. It is also suitable for longer text labels. Width of each Tabs is determined according to the length of the its text label. To navigate between Tabs, the Tab Layout can be scrolled horizontally, Tabs can be tapped, or content views can be swiped horizontally.

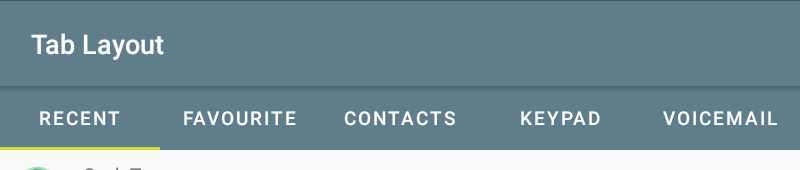
**Tabs**

Tabs are used for organizing and navigating at the same hierarchical level between related contents. Each Tab should consist of different contents. They can be used to display different type of music or sections of the news. They can be observed in the *Whatsapp* and *Google Play Store*.

A Tab can consist of a text label and an icon. There can be just a text or just an icon. All Tabs should be in the same row and each of them should merge on the same subject. The text should be short and concise. It should also be a single line. In a Tab Layout, all Tabs should be at the same anatomy. If a Tab consists of only on icon, others should also consist of just icons.







To add a Tab Item into the Tab Layout, there are two options:

If the number of Tab Item that are placed into the layout is known, they are defined directly into the layout in the XML file.

<android.support.design.widget.TabLayout

android:id="@+id/tabLayout"

android:layout\_width="match\_parent"

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

<android.support.design.widget.TabItem

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:icon="@drawable/ic\_call\_black\_24dp"

android:text="@string/tab\_item" />

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

Or they can be added to the Tab Layout dynamically with ***addTab(TabLayout.Tab tab)*** method. It adds to the end of list and the first added Tab become the selected Tab. This function can take 2 more parameters. One of them is ***addTab(Tab tab, int position, boolean setSelected)***. With the setSelected parameter, Tab’s state can be arranged as selected/unselected. If true is sent, the Tab will become the selected. Position sets the position of the new tab.

The new Tab object is created with ***newTab()*** method. Also, text and an icon can be set to the new Tab dynamically with ***setText(int resId)/ setText(CharSequence text)*** and ***setIcon(int resId)/ setIcon(Drawable icon)*** methods.

The position of the current Tab object can be learnt with ***getPosition()***. And the Tab can be selected dynamically later with ***select()*** and learn if it is selected or not with ***isSelected()***.

tabLayout.addTab(tabLayout.newTab().setText("Tab"));

**Attributes of Tab Layout**

The position of the Tab Layout can be set with ***app:tabGravity=””*** attributes. If fill option is chosen, the layout covers the all width of the screen. If the center is chosen, the layout is aligned to the center of the screen. Also, ***setTabGravity(int gravity)*** and ***getTabGravity()*** are for managing the alignment of the Tabs dynamically.

The width of each Tab can be set with ***app:tabMinWidth*** and ***app:tabMaxWidth*** attributes. These attributes determine the lower and upper bound of the width of the Tabs.

The background color of the Tabs is changed with ***app:tabBackground=””***.

The icon of the Tabs’ color also is changed with ***app:tabIconTint=””*** and dynamically can change with ***setTabIconTint (ColorStateList iconTint)*** and ***setTabIconTintResource (int iconTintResourceId)***.

***app:tabInlineLabel=”true/false”*** determines that the text is in the same line with icon or is underneath of the icon. Also, can be set dynamically with ***setInlineLabelResource (int inlineResourceId)***.

The text of the Tabs can be customized with ***app:tabTextAppearance=””*** attribute.

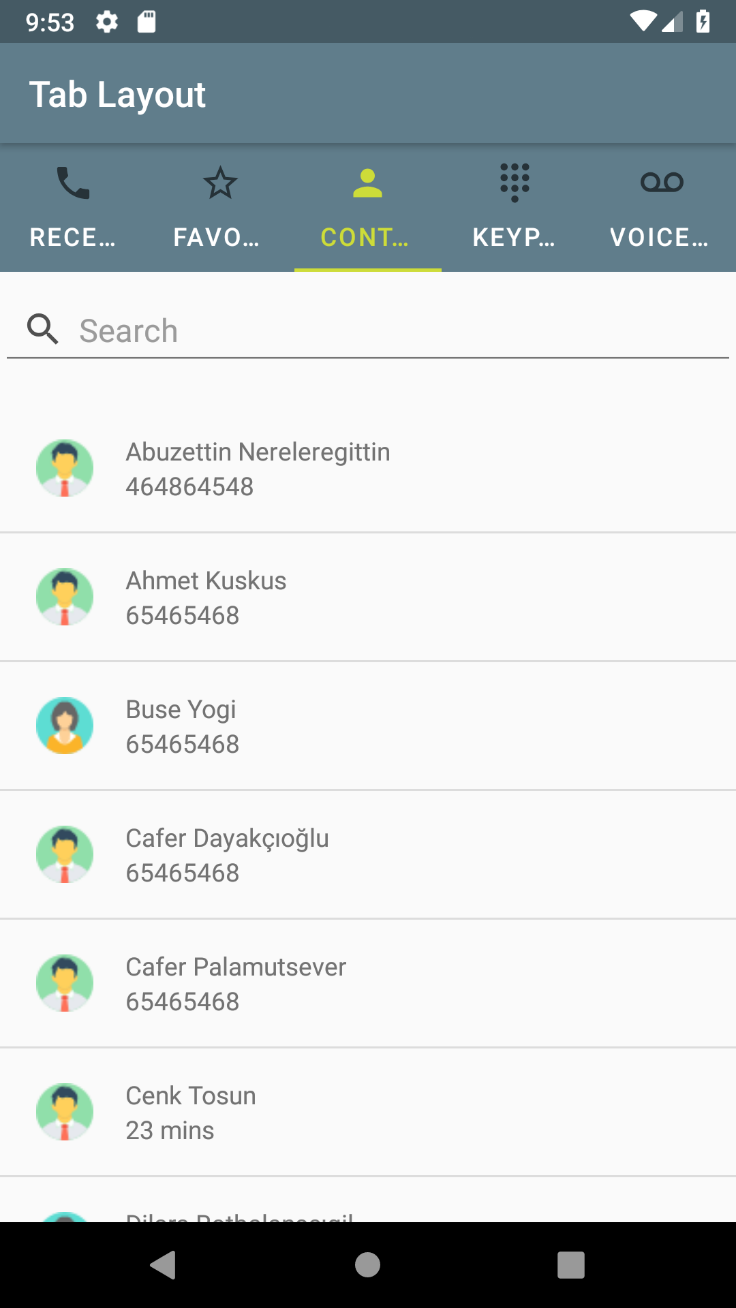
The color of the text is also changed with ***app:tabTextColor*** and ***app:tabSelectedTextColor*** attributes. First one changes the unselected Tabs’ color; second one changes the selected Tab’s color. These texts color is also changed dynamically with ***setTabTextColors(int normalColor, int selectedColor)*** and ***getTabTextColors()*** (returns both states’ colors).

The position of the Tabs can be arranged with attributes: ***app:tabContentStart***, ***app:tabPaddingStart***, ***app:tabPaddingTop***, ***app:tabPaddingEnd***, ***app:tabPaddingBottom***, ***app:tabPadding***.

The ripple effect is part of the default Material styles. The ripple color can be changed with ***app:tabRippleColor***. To changed dynamically, use ***setTabRippleColor(ColorStateList color)*** or ***setTabRippleColorResource(int tabRippleColorResourceId)***.

One of the features of the Tab Layout is Tab Indicator. The state of it indicates that which Tab is displayed. If the indicator is active, its text’s and icon’s color become the ***?attr/colorAccent***. The color of the indicator when it is active, can be customized with ***app:tabIndicatorColor***. Dynamically: ***setSelectedTabIndicatorColor(int color)***. The placement of the indicator can be changed with ***app:tabIndicatorGravity***. Dynamically: ***setSelectedTabIndicatorGravity(int indicatorGravity)***. The indicator’s width can be full width of the TabItem or as the size of the text label. These can be set with ***app:tabIndicatorFullWidth***. Dynamically: ***setTabIndicatorFullWidth(boolean tabIndicatorFullWidth)***. The thickness of the indicator is arranged with ***app:tabIndicatorHeight***. Dynamically: ***setSelectedTabIndicatorHeight(int height)***. All appearance of the indicator can be customized

with ***app:tabIndicator***. Dynamically: ***setSelectedTabIndicator(int tabSelectedIndicatorResourceId)***.



Custom Tab Layout Design

<android.support.design.widget.TabLayout

android:id="@+id/tabLayout"

android:layout\_width="match\_parent"

android:layout\_height="wrap\_content"

app:tabBackground="@color/colorPrimary"

app:tabSelectedTextColor="@color/colorAccent"

app:tabGravity="fill"

app:tabTextColor="#ffffff">

Other methods of the TabLayout class:

***getSelectedTabPosition()***: this function returns the current position of the Tab.

***getTab(int index)***: this returns the Tab object at that index.

***getTabCount()***: this returns the number of the Tabs in the Tab Layout.

***removeAllTabs()***: this removes all the Tabs in the Tab Layout.

***removeTab(Tab tab)***: this removes the specific tab from the Tab Layout. If the removed Tab is selected one, it becomes automatically deselected.

***removeTabAt(int position)***: this removes the Tab in that position.

***setUnboundedRipple(boolean unboundedRipple)***: this sets the ripple effect is bounded with the Tab Item’s size or not.

Listeners:

***TabLayout.OnTabSelectedListener***: this callback is invoked when the Tab’s selection is changed. It has three methods which are ***onTabReselected(T tab)***, ***onTabSelected(T tab)*** and ***onTabUnselected(T tab)***.

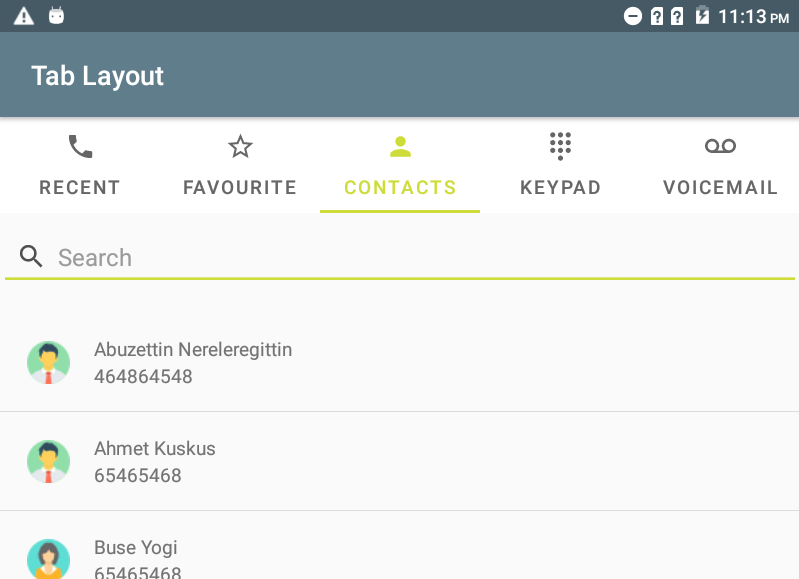
***TabLayout.TabLayoutOnPageChangeListener*** and ***TabLayout.ViewPagerOnTabSelectedListener*** classes keep the position of the Tabs in sync. TabLayoutOnPageChangeListener has three methods which are ***onPageScrollStateChanged(int state)***, ***onPageScrolled(int position, float positionOffset, int positionOffsetPixels)*** and ***onPageSelected(int position)***. ViewPagerOnTabSelectedListener also has three methods that are ***onTabReselected(TabLayout.Tab tab)***, ***onTabSelected(TabLayout.Tab tab)*** and ***onTabUnselected(TabLayout.Tab tab)***.

**Material Styles**

The default Material Design theme of the Tab Layout is applied, if the theme of the application is one of the updated Material Theme (Theme.MaterialComponents). if the application’s theme is not the Material Theme, the updated Material style can be applied with style attribute. There are three Material style which are ***Updated Material Style*** and ***Colored Material Style***.

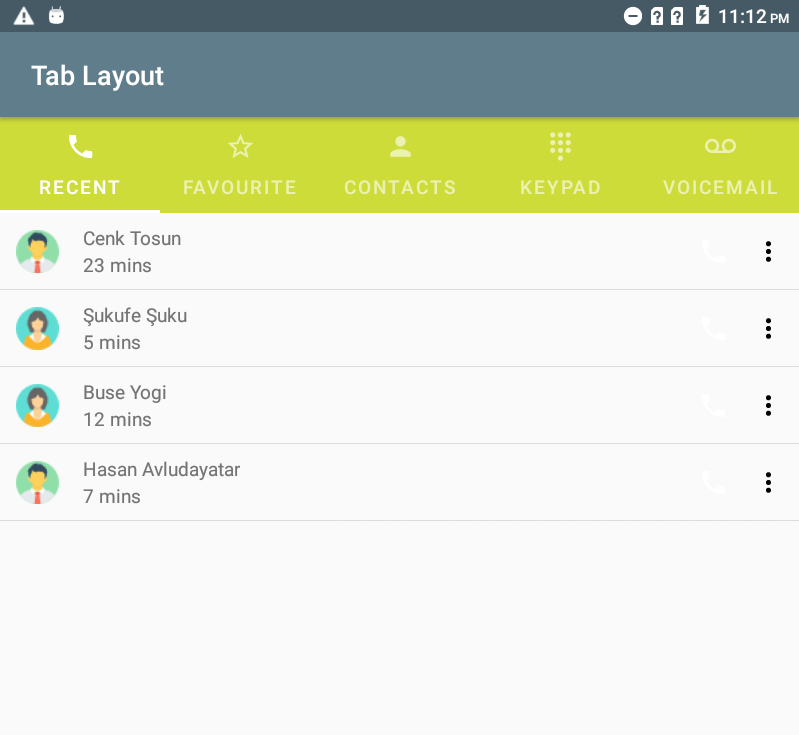
In Updated Material Style, there are updated icon and label tints, ripple color and ripple shape. The background is white and active tab and indicator’s color is colorAccent. It is applied with:

style="@style/Widget.MaterialComponents.TabLayout"



In Colored Material Style, the background color becomes colorAccent and the active Tab and indicator color become white. It is applied with:

style="@style/Widget.MaterialComponents.TabLayout.Colored"



**ViewPager**

Content views of the Tabs are held in the ViewPager. ViewPager is a layout manager that enables to swipe content pages left and right. ***PagerAdapter*** must be provided to the ViewPager to create pages that displays the content views. It is added in version 22.1.0 and parts of the com.android.support:viewpager:28.0.0-alpha1 library. ViewPagers are usually used with Fragments to create and control the life cycle of each page properly. It has two standard adapters that are ***FragmentPageAdapter*** and ***FragmentStatePageAdapter***. These allow Fragments to be used with the ViewPager.

**FragmentPagerAdapter**

It is extended from PagerAdapter. Each page is represented as Fragment. It keeps the page in the layout manager if the page can accessible by the user. This adapter is ideal for the fixed and limited number fragments like fixed type of Tab Layout. Even the page is not visible on the screen, fragments that are visited by the user are held in the memory. This causes too much memory to be wasted if the number of pages is larger. FragmentStatePagerAdapter should be preferred is the number is higher. In the implementation of the adapter, ***getItem(int)*** and ***getCount()*** must be implemented.

getItem(int position) returns the fragment in that position. getCount() returns the number of fragments.

**FragmentStatePagerAdapter**

This is also extended from PagerAdapter. It uses fragments to control each page. Saving and restoring of the state of the fragments are handled by the FragmentStatePagerAdapter. This adapter has more advantage in case of many pages because when the page is not visible on the screen, the fragment is destroyed; it keeps only the saved state of the fragment. Therefore, it uses less memory when compares with the FragmentPagerAdapter. Also, navigation between pages becomes more smoothly. ***getItem(int)*** and ***getCount()*** methods must be implemented.

A ViewPager can be created either in the XML file or dynamically.

<android.support.v4.view.ViewPager

android:layout\_width="match\_parent"

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

<android.support.design.widget.TabLayout

android:id="@+id/tabLayout"

android:layout\_width="match\_parent"

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

</android.support.v4.view.ViewPager>

The adapter that is created from one of the above pager adapters or custom adapter is set to the ViewPager with ***setAdapter()*** method.

**Implementation**

When creating a Tab Layout, first a Tab Layout and ViewPager should be defined. Then, adapter of the ViewPager should be implemented. The adapter must extend from FragmentPagerAdapter or FragmentStatePagerAdapter. The list of fragments and the titles of that fragments can be stored in ArrayLists. In the adapter, getItem(int i) and getCount() functions must be implemented. Other function that should be implemented is ***getPageTitle(int position)***. This method returns the title of the page at that position. If the tabs consist of only icons which means that they do not have titles, this function returns null. The default implementation returns null. Also, to add fragments to the adapter, a function can be implemented like ***addFragment(Fragment fragment, String title)*** in the example.

public class CallPagerAdapter extends FragmentStatePagerAdapter {

private final List<Fragment> mFragmentList = new ArrayList<>();

private final List<String> mFragmentTitleList = new ArrayList<>();

public CallPagerAdapter(FragmentManager manager) {

super(manager);

}

@Override

public Fragment getItem(int i) {

return mFragmentList.get(i);

}

public void addFragment(Fragment fragment, String title) {

mFragmentList.add(fragment);

mFragmentTitleList.add(title);

}

@Override

public int getCount() {

return mFragmentList.size();

}

@Nullable

@Override

public CharSequence getPageTitle(int position) {

return mFragmentTitleList.get(position);

}

}

If the Tabs do not contain text, only consist of only an icon; getPageTitle(int position) is implemented like this:

@Override

public CharSequence getPageTitle(int position) {

return null;

}

After creating the adapter, each fragment of the tabs should be implemented. Then, all fragments are added to the adapter with addFragment().

PagerAdapter pagerAdapter = new pagerAdapter(getSupportFragmentManager());

pagerAdapter.addFragment(new Fragment1(), "Fragment1");

pagerAdapter.addFragment(new Fragment2(), "Fragment2");

After that the adapter should set to the ViewPager with setAdapter(pagerAdapter) method. Tab Layout and ViewPager are connected with ***setupWithViewPager(ViewPager)*** method. according to the titles on the ViewPager, each tab item in the Tab Layout will be generated automatically.

viewPager.setAdapter(callPagerAdapter);

tabLayout.setupWithViewPager(viewPager);

The icons on the tab (if there is) must also set dynamically.

tabLayout.getTabAt(0).setIcon(R.drawable.ic\_call\_black\_24dp);

tabLayout.getTabAt(1).setIcon(R.drawable.ic\_star\_border\_black\_24dp);

tabLayout.getTabAt(2).setIcon(R.drawable.ic\_person\_black\_24dp);

**Requirements for Implementation**

Tab layouts are part of android design library. To use tab layout, you must add the following requirements to the build.gradle file in the project:

dependencies {  
    implementation ' com.android.support:appcompat-v7:28.0.0 -rc02'  
 implementation 'com.android.support: design:28.0.0 -rc02’ }

Also, the Tab Layout and Tabs are adding to the XML file as the following:

<android.support.design.widget.TabLayout

android:id="@+id/tabLayout"

android:layout\_width="match\_parent"

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

<android.support.design.widget.TabItem

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:icon="@drawable/ic\_call\_black\_24dp"

android:text="@string/recents" />

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

**References**

Android Developers. (n.d.). Retrieved June 27, 2018, from https://developer.android.com/

Android Material Design Tabs with TabLayout. (2018, January 21). Retrieved September 29, 2018, from https://www.learn2crack.com/2017/07/android-material-design-tabs.html

Me, D. B. (2018, January 16). Android Material Design Tabs (Tab Layout) with Swipe. Retrieved September 29, 2018, from https://medium.com/@droidbyme/android-material-design-tabs-tab-layout-with-swipe-884085ae80ff

Tab Layout. (n.d.). Retrieved September 28, 2018, from https://material.io/develop/android/components/tab-layout/

TabLayout Tutorial with Example in Android Studio. (n.d.). Retrieved October 1, 2018, from <https://abhiandroid.com/materialdesign/tablayout-example-android-studio.html>

Tabs. (n.d.). Retrieved September 28, 2018, from https://material.io/design/components/tabs.html#