Tema 4. Menús

Tema 4. Menús

ÍNDICE.

Menú lateral deslizante (DrawerLayout / NavigationView)

Para crear un menú lateral deslizante tendremos que utilizar dos componentes en nuestro layout principal: un contenedor, llamado *DrawerLayout*, que albergará todos los elementos de dicho layout, y un navegador, llamado *NavigationView*, donde aparecerán las distintas opciones del menú.
Al pulsar sobre cada una de las diferentes opciones del menú se cargará el *fragment* correspondiente.

Para implementarlo, utilizaremos el layout *activity_main.xml* donde insertaremos estos dos componentes:

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.drawerlayout.widget.DrawerLayout
   xmlns:android="http://schemas.android.com/apk/res/android"
   xmlns:app="http://schemas.android.com/apk/res-auto"
   android:id="@+id/drawer_layout"
   android:layout_width="match_parent"
   android:layout_height="match_parent"
   android:fitsSystemWindows="true"
   tools:openDrawer="start">

    <!-- Layout principal de la actividad -->
    <include layout="@layout/content_main" />

    <!-- Layout del menú lateral (Navigation View) -->
    <com.google.android.material.navigation.NavigationView
       android:id="@+id/nav_view"
       android:layout_width="wrap_content"
       android:layout_height="match_parent"
       android:fitsSystemWindows="true"
       android:layout_gravity="start"
       app:menu="@menu/menu_navdrawer"/>
```

```
</androidx.drawerlayout.widget.DrawerLayout>
```

El layout principal de la actividad *content_main.xml,* contendrá, por ejemplo la *ToolBar* y un *FrameLayout* para el intercambio de fragments.:

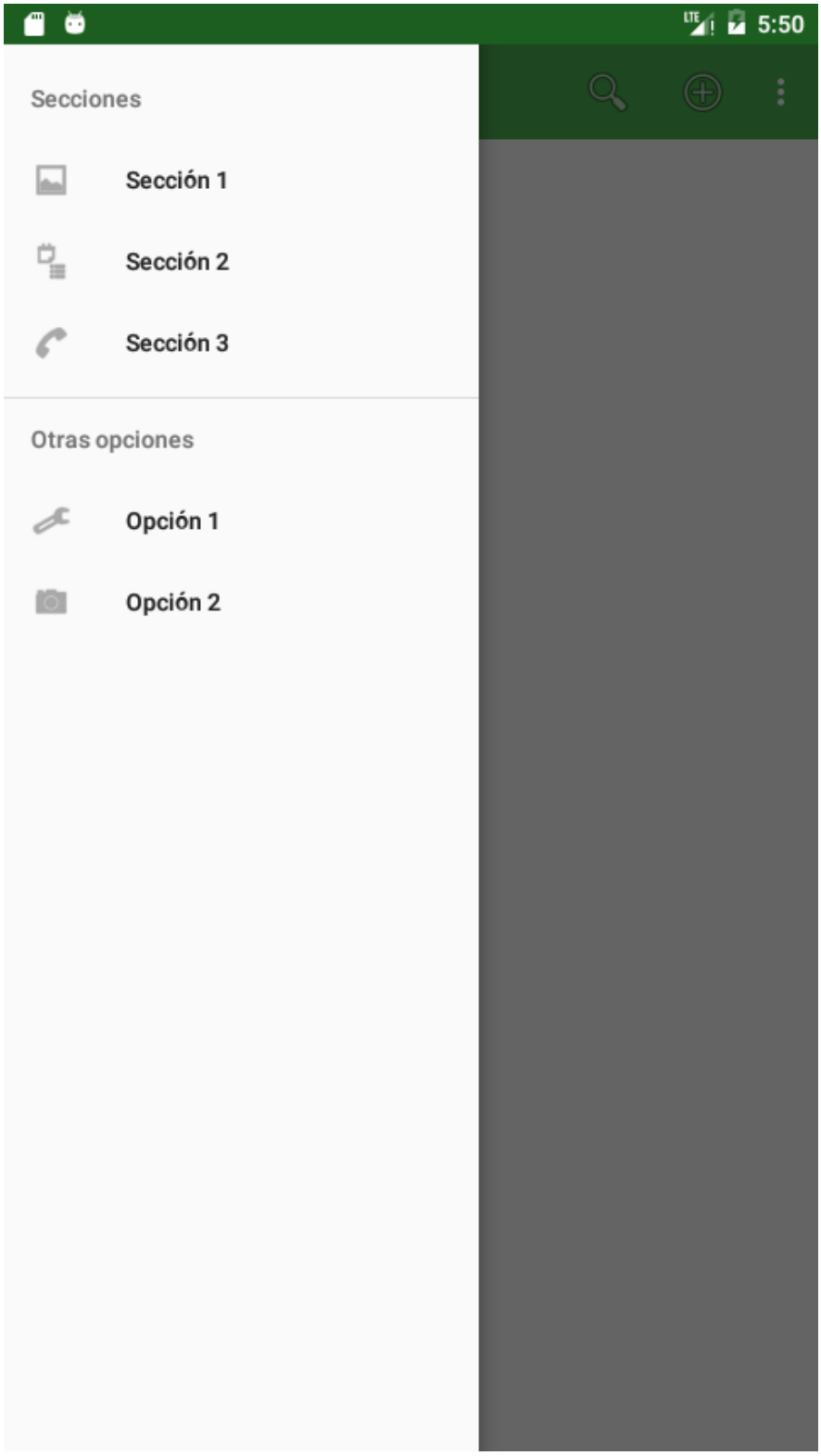
```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/a
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical">

    <androidx.appcompat.widget.Toolbar
        android:id="@+id/toolbar"
        android:layout_width="match_parent"
        android:layout_height="?attr/actionBarSize"
        android:background="?attr/colorPrimary"
        android:minHeight="?attr/actionBarSize"
        android:theme="?attr/actionBarTheme/>

    <FrameLayout
        android:id="@+id/content_frame"
        android:layout_width="match_parent"
        android:layout_height="match_parent"/>
</LinearLayout>
```

En el *NavigationView* enlazaremos el menú con opciones que queremos que aparezca al desplegarse (*menu/menu_navdrawer.xml*), por ejemplo el siguiente:

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android">
    <item
        android:id="@+id/navigation_header1"
        android:title="Secciones">
        <menu>
            <item
                android:id="@+id/menu_seccion_1"
                android:icon="@android:drawable/ic_menu_gallery"
```

```
                android:title="Sección 1"/>
            <item
                android:id="@+id/menu_seccion_2"
                android:icon="@android:drawable/ic_menu_agenda"
                android:title="Sección 2"/>
            <item
                android:id="@+id/menu_seccion_3"
                android:icon="@android:drawable/ic_menu_call"
                android:title="Sección 3"/>
        </menu>
    </item>
    <item
        android:id="@+id/navigation_header2"
        android:title="Otras opciones">
        <menu>
            <item
                android:id="@+id/menu_opcion_1"
                android:icon="@android:drawable/ic_menu_manage"
                android:title="Opción 1"/>
            <item
                android:id="@+id/menu_opcion_2"
                android:icon="@android:drawable/ic_menu_camera"
                android:title="Opción 2"/>
        </menu>
    </item>
</menu>
```

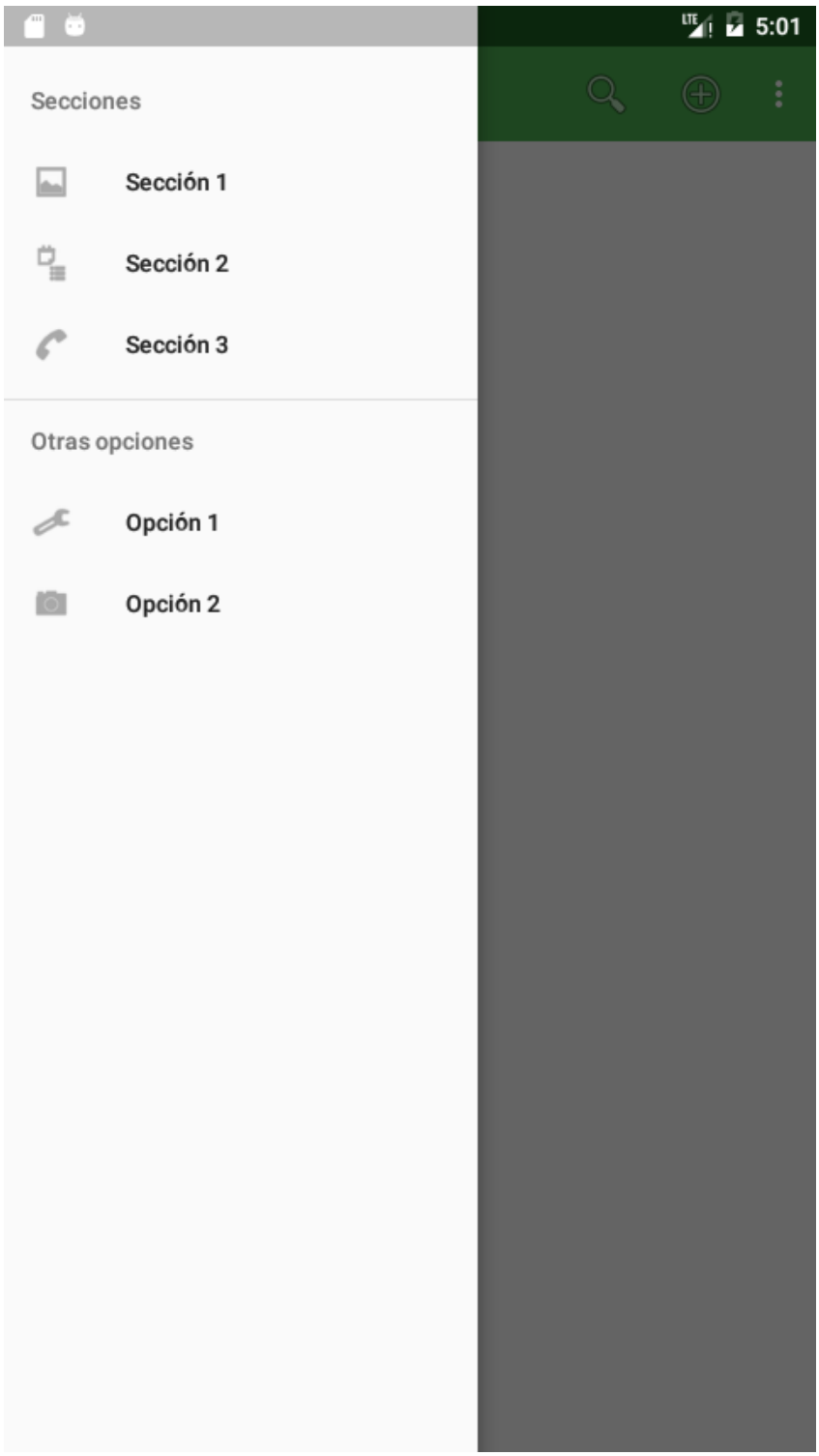
Al ejecutar visualizaremos lo siguiente:



Como puede observarse, el navegador lateral se sitúa por detrás de la StatusBar y por delante de la ToolBar. Sin embargo, el comportamiento deseable sería situarse por delante de ambas, para ello en el fichero *styles.xml* tendremos que definir a transparente el color de la barra de estado, de forma que conseguiremos el efecto deseado.

```
<item name="android:statusBarColor">@android:color/transparent</item>
```

El resultado obtenido será:



Ahora le daremos funcionalidad a las opciones del menú, de manera que al pulsar sobre cada una de ellas, saltaremos al fragment correspondiente.

Crearemos un nuevo layout que cargaremos como fragment 1 (*layout/fragment1.xml*).

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

    <TextView
        android:id="@+id/text"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_marginStart="8dp"
        android:layout_marginTop="8dp"
        android:layout_marginEnd="8dp"
        android:textAlignment="center"
        android:textSize="20sp"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent"/>
</androidx.constraintlayout.widget.ConstraintLayout>
```

Definiremos la clase *Fragment1.java* que inflará el layout anterior.

```
public class Fragment1 extends Fragment {
    public Fragment1() {
    }

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup c
        return inflater.inflate(R.layout.fragment1, container, fa
    }
}
```

En Kotlin (*Fragment1.kt*):

```
class Fragment1 : Fragment() {
```

```
    override fun onCreateView(
        inflater: LayoutInflater,
        container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View? {
        return inflater.inflate(R.layout.fragment1, container, fa
    }
}
```

Realizaremos el mismo procedimiento para los otros dos fragments.

Nota: realizaremos el ejemplo, sólo para las opciones del menú Secciones.

Finalmente, en la clase *MainActivity.java* añadiremos lo siguiente:

```
final DrawerLayout drawerLayout = (DrawerLayout) findViewById(R.i
NavigationView navView = (NavigationView)findViewById(R.id.nav_vi

navView.setNavigationItemSelectedListener(new NavigationView.OnNa

    @Override
    public boolean onNavigationItemSelected(MenuItem menuItem) {
        boolean fragmentTransaction = false;
        Fragment fragment = null;
        switch (menuItem.getItemId()) {
            case R.id.menu_seccion_1:
                fragment = new Fragment1();
                fragmentTransaction = true;
                break;
            case R.id.menu_seccion_2:
                fragment = new Fragment2();
                fragmentTransaction = true;
                break;
            case R.id.menu_seccion_3:
                fragment = new Fragment3();
                fragmentTransaction = true;
                break;
        }
        if(fragmentTransaction) {
            getSupportFragmentManager().beginTransaction().replac
```

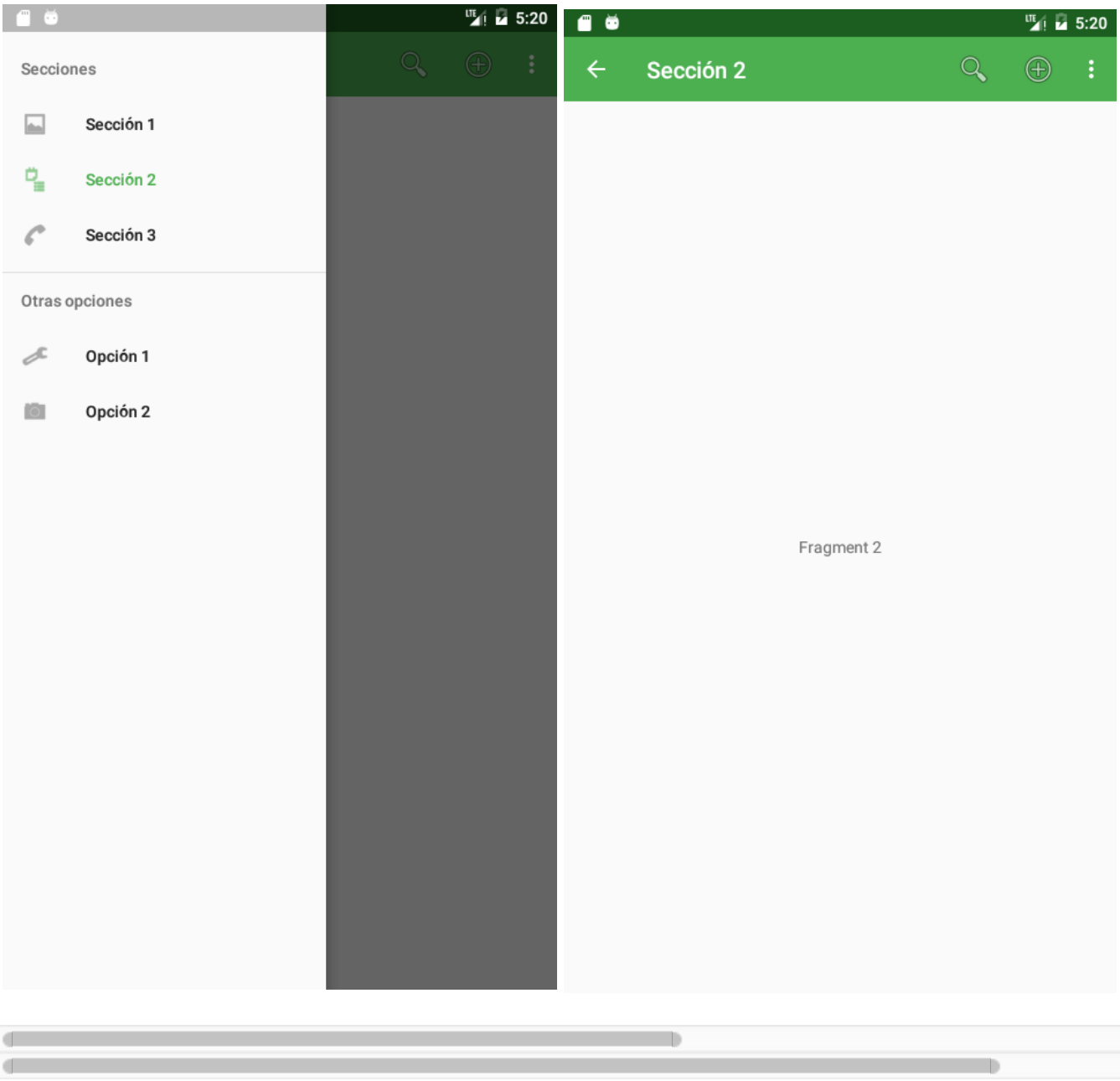
```
            menuItem.setChecked(true);
            getSupportActionBar().setTitle(menuItem.getTitle());
        }
        drawerLayout.closeDrawers();
        return true;
    }
});
```

En Kotlin (*MainActivity.kt*):

```
val drawerLayout = findViewById(R.id.drawer_layout) as DrawerLayo
val navView = findViewById(R.id.nav_view) as NavigationView

navView.setNavigationItemSelectedListener(object : NavigationView

    override fun onNavigationItemSelected(menuItem: MenuItem): Bo
        var fragmentTransaction = false
        var fragment: Fragment? = null

        when (menuItem.getItemId()) {
            R.id.menu_seccion_1 -> {
                fragment = Fragment1()
                fragmentTransaction = true
            }
           R.id.menu_seccion_2 -> {
                fragment = Fragment2()
                fragmentTransaction = true
            }
           R.id.menu_seccion_3 -> {
                fragment = Fragment3()
                fragmentTransaction = true
            }
        }
        if (fragmentTransaction) {
            supportFragmentManager.beginTransaction().replace(R.i
            menuItem.setChecked(true)
            supportActionBar!!.title = menuItem.title
        }
```

```
            drawerLayout.closeDrawers()
            return true
        }
    })
```

El resultado obtenido al pulsar sobre la Sección 2 será el siguiente:



Podríamos mejorar estéticamente el menú lateral deslizante, añadiendo una cabecera con una imagen y un título (nombre de la aplicación). Para ello, bastaría con crear un nuevo layout (*header.xml*) que contendría un *ImageView* y un *TextView*.

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

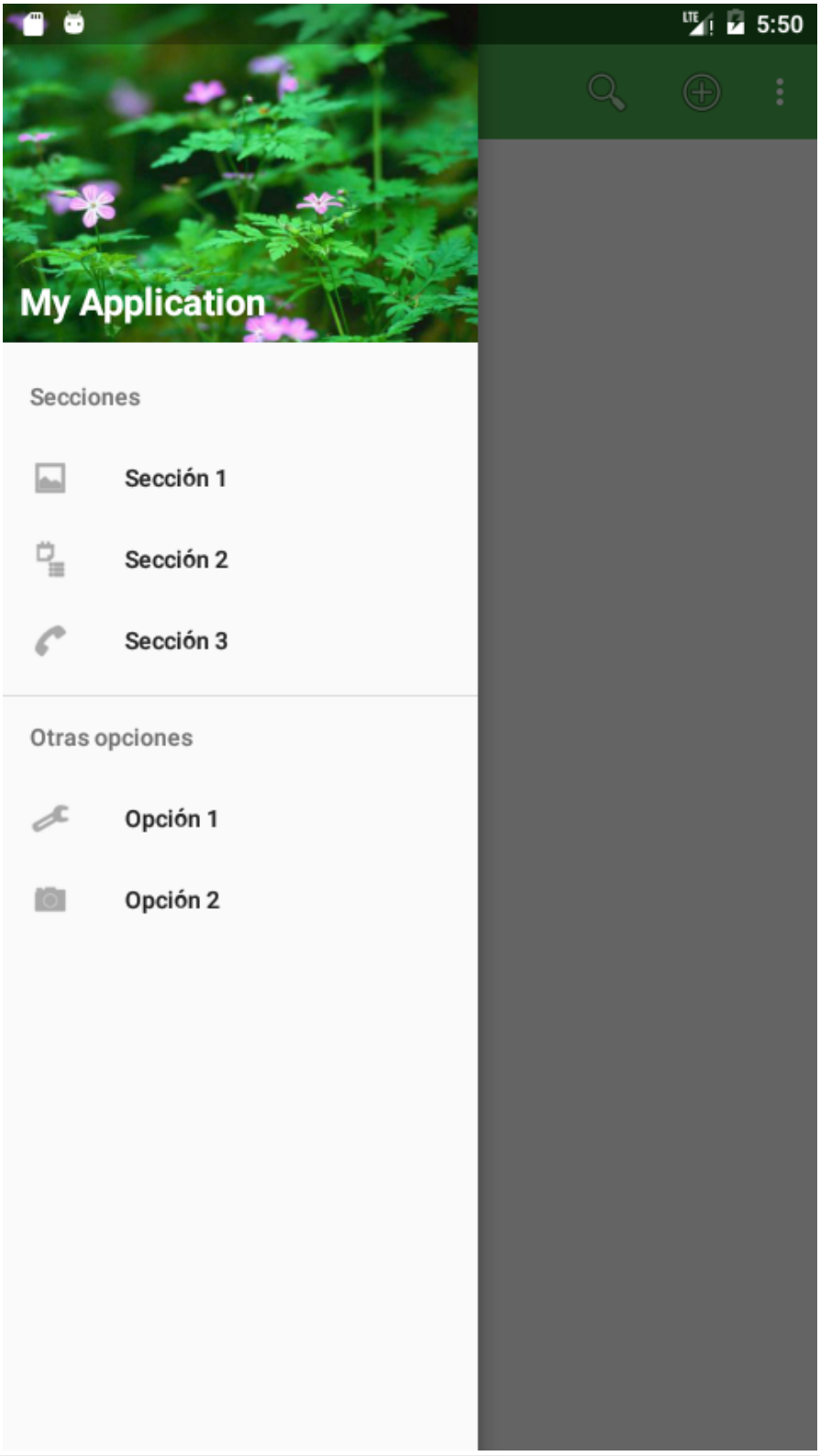
```
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <ImageView
        android:layout_width="match_parent"
        android:layout_height="200dp"
        android:scaleType="centerCrop"
        android:src="@mipmap/foto" />

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/app_name"
        android:textAppearance="@style/TextAppearance.AppCompat.L
        android:textStyle="bold"
        android:layout_gravity="bottom"
        android:layout_marginBottom="10dp"
        android:layout_marginLeft="10dp" />

</FrameLayout>
```

En el layout *main.xml*, dentro del componente *NavigationView,* añadiremos la cabecera creada anteriormente:

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.drawerlayout.widget.DrawerLayout
   xmlns:android="http://schemas.android.com/apk/res/android"
   xmlns:app="http://schemas.android.com/apk/res-auto"
   android:id="@+id/drawer_layout"
   android:layout_width="match_parent"
   android:layout_height="match_parent"
   android:fitsSystemWindows="true"
   tools:openDrawer="start">

   <!-- Layout principal de la actividad -->
   <include layout="@layout/content_main" />

   <!-- Layout del menú lateral (Navigation View) -->
```

```
    <com.google.android.material.navigation.NavigationView
        android:id="@+id/navview"
        android:layout_width="wrap_content"
        android:layout_height="match_parent"
        android:fitsSystemWindows="true"
        android:layout_gravity="start"
        app:menu="@menu/menu_navdrawer"
        app:headerLayout="@layout/header"/>

</androidx.drawerlayout.widget.DrawerLayout>
```

El resultado obtenido sería similar al indicado:



Para añadir en el lado izquierdo de la ToolBar el icono de la flecha (**up icon**), indicativo de la presencia de un navegador lateral deslizante, definiremos en la clase principal, la siguiente sentencia:

En Java:

```
getSupportActionBar().setDisplayHomeAsUpEnabled(true);
```

En Kotlin:

```
supportActionBar!!.setDisplayHomeAsUpEnabled(true)
```

Si por contra, deseamos añadir sobre la ToolBar el icono oficial de la presencia de un navegador lateral deslizante, el **hamburguer icon**, bastará con instanciar en la actividad principal, la clase ActionBarDrawerToggle como se indica:

En Java:

```
Toolbar toolbar = (Toolbar) findViewById(R.id.toolbar);
setSupportActionBar(toolbar);

DrawerLayout drawerLayout = (DrawerLayout) findViewById(R.id.draw

//icono hamburguesa
ActionBarDrawerToggle toggle = new ActionBarDrawerToggle(this, dr
//El listener realiza una pequeña animación sobre el icono al cer
drawerLayout.addDrawerListener(toggle);
toggle.syncState();
```

En Kotlin:

```
val toolbar = findViewById(R.id.toolbar) as Toolbar
setSupportActionBar(toolbar)

val drawerLayout = findViewById(R.id.drawer_layout) as DrawerLayo

val toggle = ActionBarDrawerToggle(this, drawerLayout, toolbar, 0
drawerLayout.addDrawerListener(toggle)
toggle.syncState()
```

Y para activar la apertura del navegador, tanto al pulsar el icono de la flecha como el de la hamburguesa (R.id.home en ambos casos), reescribiremos el siguiente método:

En Java:

```
  @Override
```

```
    public boolean onOptionsItemSelected(MenuItem item) {
        DrawerLayout drawerLayout = (DrawerLayout) findViewById(R
        switch(item.getItemId()) {
            case android.R.id.home:
                drawerLayout.openDrawer(GravityCompat.START);
                return true;
        }
        return super.onOptionsItemSelected(item);
    }
```

En Kotlin:

```
override fun onOptionsItemSelected(item: MenuItem): Boolean {
    val drawerLayout = findViewById(R.id.drawer_layout) as Drawer
    when (item.itemId) {
        android.R.id.home -> {
            drawerLayout.openDrawer(GravityCompat.START)
            return true
        }
    }
    return super.onOptionsItemSelected(item)
}
```

Menú con pestañas (TabLayout / TabItem)

El componente *TabLayout*, menú con pestañas (*TabItem*), aparece siempre debajo de la ToolBar. Para unificar ambos componentes como un todo, dispondremos de otro elemento, llamado *AppBarLayout* que contendrá a los dos anteriores.

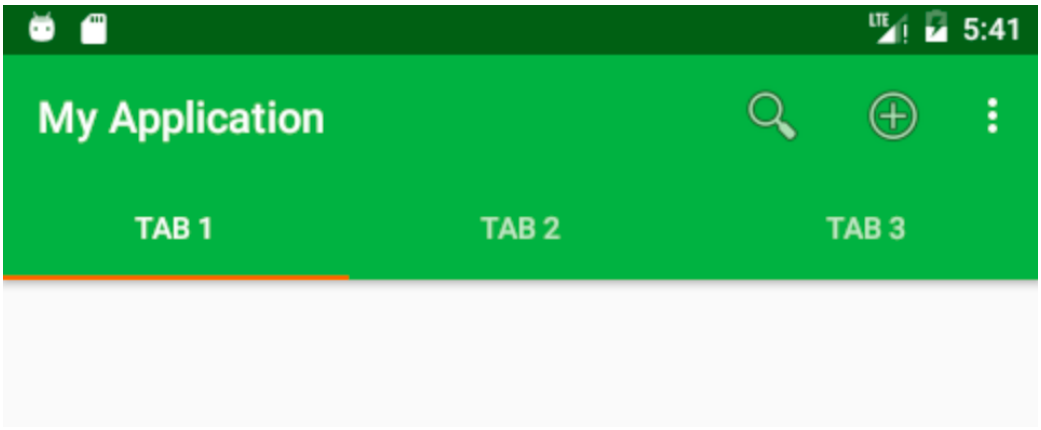
En el layout principal *activity_main.xml*, añadiríamos el siguiente código:

```
<com.google.android.material.appbar.AppBarLayout
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:theme="@style/AppTheme.AppBarOverlay">

    <androidx.appcompat.widget.Toolbar
        android:id="@+id/toolbar"
        android:layout_width="match_parent"
        android:layout_height="?attr/actionBarSize"
        android:background="?attr/colorPrimary"
        android:popupTheme="@style/AppTheme.PopupOverlay"/>

    <com.google.android.material.tabs.TabLayout
        android:id="@+id/appbartabs"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        app:tabMode="fixed">

        <com.google.android.material.tabs.TabItem
            android:id="@+id/tabItem"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="TAB 1" />

        <com.google.android.material.tabs.TabItem
            android:id="@+id/tabItem2"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
```

```
            android:text="TAB 2" />

        <com.google.android.material.tabs.TabItem
            android:id="@+id/tabItem3"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="TAB 3" />

    </com.google.android.material.tabs.TabLayout>

</com.google.android.material.appbar.AppBarLayout>
```

En el componente TabLayout podemos especificar el tipo de pestañas a utilizar: en modo fijo (fixed*)*, donde todas las pestañas ocupan el mismo espacio, o bien en modo scroll (scrollable) donde cada una ocupa el espacio que necesita.

El resultado en modo fixed será:



Para darle funcionalidad a las pestañas y poder pasar de una a otra, utilizaremos como en el caso del navegador lateral deslizante, los *fragments.*

Crearemos un nuevo layout que cargaremos como fragment 1 (*layout/fragment1.xml*).

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

    <TextView
```

```
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:text="Fragment 1"
        android:gravity="center" />
</FrameLayout>
```

Definiremos la clase *Fragment1.java* que inflará el layout anterior.

```
public class Fragment1 extends Fragment {
    public Fragment1() {
    }

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup c
        return inflater.inflate(R.layout.fragment1, container, fa
    }
}
```

En Kotlin (*Fragment1.kt*):

```
class Fragment1 : Fragment() {

    override fun onCreateView(
        inflater: LayoutInflater,
        container: ViewGroup?,
        savedInstanceState: Bundle?): View? {
        return inflater.inflate(R.layout.fragment1, container, fa
    }
}
```

Realizaremos el mismo procedimiento para los otros dos fragments.

Añadiremos en el layout principal *activity_main.xml*, un componente para realizar el intercambio de fragments. Lo situaremos por debajo del AppBarLayout.

```
<FrameLayout
    android:id="@+id/content_frame"
```

```
    android:layout_width="match_parent"
    android:layout_height="match_parent" >
</FrameLayout>
```

Finalmente, en la clase principal *MainActivity.java* saltaremos al fragment correspondiente al pulsar cada pestaña.

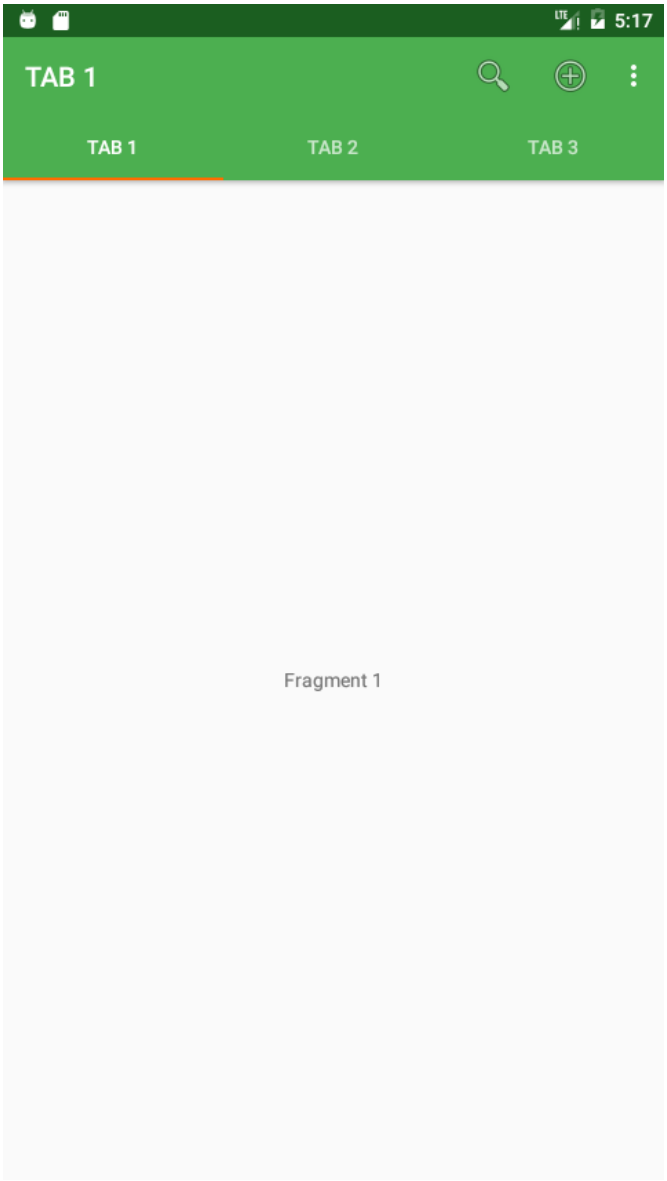
```
tabLayout.addOnTabSelectedListener(new TabLayout.OnTabSelectedLis
    boolean fragmentTransaction = false;
    Fragment fragment = null;
    @Override
    public void onTabSelected(TabLayout.Tab tab) {

        switch (tab.getPosition()) {
            case 0:
                fragment = new Fragment1();
                fragmentTransaction = true;
                break;
            case 1:
                fragment = new Fragment2();
                fragmentTransaction = true;
                break;
            case 2:
                fragment = new Fragment3();
                fragmentTransaction = true;
                break;
        }
        if(fragmentTransaction) {
            getSupportFragmentManager().beginTransaction().replac
            getSupportActionBar().setTitle(tab.getText());
        }
    }

    @Override
    public void onTabUnselected(TabLayout.Tab tab) {}

    @Override
    public void onTabReselected(TabLayout.Tab tab) {}
```

```
});
```

En Kotlin (*MainActivity.kt*):

```
tabLayout.addOnTabSelectedListener(object : TabLayout.OnTabSelect
      internal var fragmentTransaction = false
      internal var fragment: Fragment? = null

      override fun onTabSelected(tab: TabLayout.Tab) {
          when (tab.position) {
              0 -> {
                        fragment = Fragment1()
                        fragmentTransaction = true
              }
              1 -> {
                        fragment = Fragment2()
                        fragmentTransaction = true
              }
              2 -> {
                       fragment = Fragment3()
                       fragmentTransaction = true
              }
          }
          if (fragmentTransaction) {
                supportFragmentManager.beginTransaction().replace
                supportActionBar!!.setTitle(tab.text)
          }
      }

      override fun onTabUnselected(tab: TabLayout.Tab) {}

      override fun onTabReselected(tab: TabLayout.Tab) {}
})
```

El resultado obtenido para las pestañas 1 y 2, será:


5:17
TAB 1
TAB 1
TAB 2
TAB 3
Fragment 1


5:10
TAB 2
TAB 1
TAB 2
TAB 3
Fragment 2

Pantallas deslizantes (ViewPager)

Este componente nos permite deslizarnos por las distintas pantallas o páginas de una aplicación. Puede aparecer sólo o acompañado del componente *TabLayout*, en cuyo caso el deslizamiento por las pantallas tiene que ir sincronizado con el cambio de pestaña.

Utilizaremos los fragments para implementar las distintas pantallas o páginas.

ViewPager sin TabLayout.

En el layout principal *activity_main.xml*, aparecerá el elemento *ViewPager,* dentro del siguiente código:

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/a
    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="com.example.alicia.myapplication.MainActivity"

    <com.google.android.material.appbar.AppBarLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:theme="@style/ThemeOverlay.AppCompat.Dark.ActionB

        <androidx.appcompat.widget.Toolbar
            android:id="@+id/toolbar"
            android:layout_width="match_parent"
            android:layout_height="?attr/actionBarSize"
            android:background="?attr/colorPrimary"/>

    </com.google.android.material.appbar.AppBarLayout>

    <androidx.viewpager.widget.ViewPager
        android:id="@+id/pager"
```

```
        android:layout_width="match_parent"
        android:layout_height="match_parent" />

</LinearLayout>
```

Posteriormente, tendremos que crear un layout por cada página o pantalla deslizante que queramos implementar, así como la clase que permitirá inflar cada uno de esos layouts o fragments.

Nota: se obvia el código, ya que es equivalente al estudiado en el punto anterior, menú con pestañas.

A continuación definiremos el adaptador, clase *FragmentAdapterPager.java,* que extenderá de FragmentPagerAdapter. Este adaptador, como puede observarse permite definir el número de páginas deslizantes que tendrá nuestra aplicación, así como el fragment a cargar según la posición.

```
public class FragmentAdapterPager extends FragmentPagerAdapter {
    final int PAGE_COUNT = 3;

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

    @Override
    public int getCount() {
        return PAGE_COUNT;
    }

    @Override
    public Fragment getItem(int position) {
        Fragment fragment = null;

        switch (position) {
            case 0:
                fragment = new Fragment1();
                break;
            case 1:
                fragment = new Fragment2();
                break;
            case 2:
```

```
                fragment = new Fragment3();
                break;
        }

        return fragment;
    }
}
```

En Kotlin (*FragmentAdapterPager.kt*):

```
class FragmentAdapterPager(fm: FragmentManager) : FragmentPagerAd
    internal val PAGE_COUNT = 3

    override fun getCount(): Int {
        return PAGE_COUNT
    }

    override fun getItem(position: Int): Fragment {
        var fragment: Fragment? = null

        when (position) {
            0 -> fragment = Fragment1()
            1 -> fragment = Fragment2()
            2 -> fragment = Fragment2()
        }

        return fragment!!
    }
}
```

Finalmente, en la clase *MainActivity.java* situaremos el adaptador sobre el elemento *ViewPager*, como se indica:

```
ViewPager viewPager = (ViewPager) findViewById(R.id.pager);
viewPager.setAdapter(new FragmentAdapterPager(getSupportFragmentM
```

En Kotlin:

```
val viewPager = findViewById(R.id.pager) as ViewPager
viewPager.adapter = FragmentAdapterPager(supportFragmentManager)
```

ViewPager con TabLayout.

En el layout principal *activity_main.xml*, aparecerá el elemento *ViewPager,* dentro del siguiente código:

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

    <com.google.android.material.appbar.AppBarLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:theme="@style/ThemeOverlay.AppCompat.Dark.ActionB

        <androidx.appcompat.widget.Toolbar
            android:id="@+id/toolbar"
            android:layout_width="match_parent"
            android:layout_height="?attr/actionBarSize"
            android:background="?attr/colorPrimary"/>

        <com.google.android.material.tabs.TabLayout
            android:id="@+id/appbartabs"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"/>

    </com.google.android.material.appbar.AppBarLayout>

    <androidx.viewpager.widget.ViewPager
        android:id="@+id/pager"
        android:layout_width="match_parent"
        android:layout_height="match_parent"/>

</LinearLayout>
```

Posteriormente, tendremos que crear un layout por cada página o pantalla deslizante que queramos implementar, así como la clase que permitirá inflar cada uno de esos layouts o fragments.

Nota: se obvia el código, ya que es equivalente al estudiado en el punto anterior, menú con pestañas.

A continuación definiremos el adaptador, clase *FragmentAdapterPager.java,* que extenderá de FragmentPagerAdapter. El adaptador, permitirá definir el número de páginas deslizantes que tendrá nuestra aplicación y el número de pestañas a mostrar con su correspondiente título.

```
public class FragmentAdapterPager extends FragmentPagerAdapter {
    final int PAGE_COUNT = 3;
    private String tabTitles[] = new String[] { "Tab 1", "Tab 2",

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

    @Override
    public int getCount() {
        return PAGE_COUNT;
    }

    @Override
    public Fragment getItem(int position) {
        Fragment fragment = null;

        switch (position) {
            case 0:
                fragment = new Fragment1();
                break;
            case 1:
                fragment = new Fragment2();
                break;
            case 2:
                fragment = new Fragment3();
                break;
        }
```

```
        return fragment;
    }

    @Override
    public CharSequence getPageTitle(int position) {
        return tabTitles[position];
    }
}
```

En Kotlin (*FragmentAdapterPager.kt*):

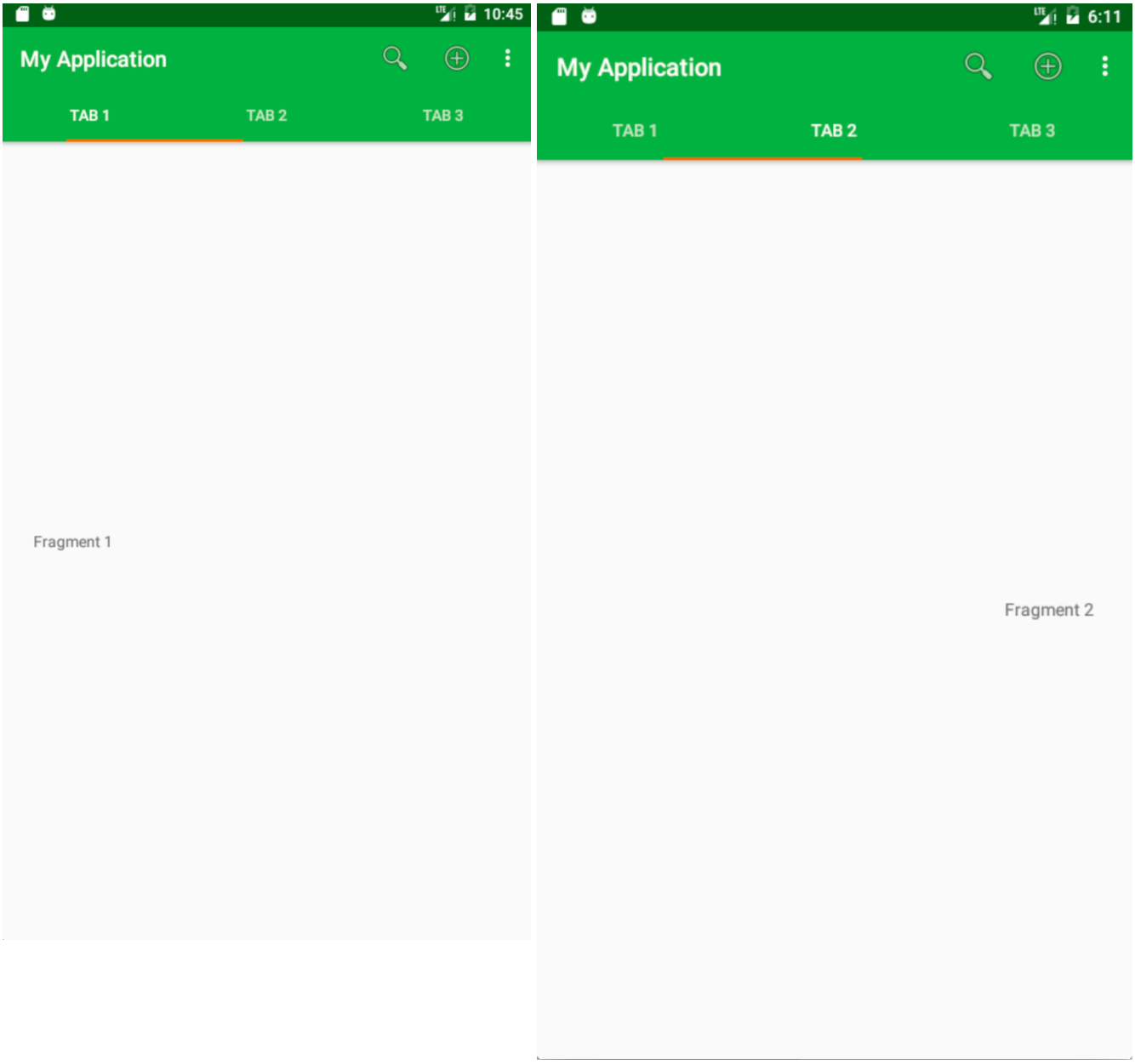
```
class FragmentAdapterPager(fm: FragmentManager) : FragmentPagerAda
    internal val PAGE_COUNT = 3
    private val tabTitles = arrayOf("Tab 1", "Tab 2", "Tab 3")

    override fun getCount(): Int {
        return PAGE_COUNT
    }

    override fun getItem(position: Int): Fragment {
        var fragment: Fragment? = null

        when (position) {
            0 -> fragment = Fragment1()
            1 -> fragment = Fragment2()
            2 -> fragment = Fragment2()
        }

        return fragment!!
    }

    override fun getPageTitle(position: Int): CharSequence? {
        return tabTitles[position]
    }
}
```

Finalmente, en la clase *MainActivity.java* situaremos el adaptador creado sobre el elemento *ViewPager*, y especificaremos que el componente *TabLayout* trabajará de forma sincronizada con él (método *setupWithViewPager*).

```
ViewPager viewPager = (ViewPager) findViewById(R.id.pager);
viewPager.setAdapter(new FragmentAdapterPager(getSupportFragmentM

TabLayout tabLayout = (TabLayout) findViewById(R.id.appbartabs);
tabLayout.setupWithViewPager(viewPager);
```

En Kotlin:

```
val viewPager = findViewById(R.id.pager) as ViewPager
viewPager.adapter = FragmentAdapterPager(supportFragmentManager)

val tabLayout = findViewById(R.id.appbartabs) as TabLayout
tabLayout.setupWithViewPager(viewPager)
```



Personalizar la animación deslizante de un ViewPager mediante PageTransformer.

Vamos a personalizar la animación entre páginas del ViewPager, para realizar una transición diferente a la que viene por defecto.

Para ello, implementaremos la interfaz *ViewPager.PageTransformer*. Esta interfaz presenta el método *transformPage()*, al cual se llama una vez por cada página visible y sus adyacentes. Por ejemplo, si la página 2 está visible y el usuario realiza el gesto de arrastrar hacia la página 3, el método transformPage() será llamado para las páginas 1, 2 y 3.

El método transformPage() transforma la página deseada, en base a la posición

(parámetro position) de dicha página en la pantalla. El parámetro position indica donde está localizada una determinada página con respecto al centro de la pantalla. Por ejemplo, la página visible en la pantalla presenta 0 en el parámetro position, si se desplaza al lado derecho de la pantalla, el valor position cambia a 1. Basándonos en la posición de las páginas en la pantalla, podemos crear animaciones deslizantes personalizadas entre páginas usando los métodos:setAlpha(), setTranslationX() o setScaleY().

Para cargar la animación personalizada, utilizaremos el código:

```
ViewPager viewPager = (ViewPager) findViewById(R.id.pager);
viewPager.setAdapter(new FragmentAdapterPager(getSupportFragmentM
viewPager.setPageTransformer(true, new ZoomPageTransformer());
```

En Kotlin:

```
val viewPager = findViewById(R.id.pager) as ViewPager
viewPager.adapter = FragmentAdapterPager(supportFragmentManager)
viewPager.setPageTransformer(true, ZoomPageTransformer())
```

La clase *ZoomPageTransformer* permite encoger la páginas del ViewPager, de manera que, a medida que una página se hace más cercana al centro de la pantalla vuelve a su tamaño original.

```
public class ZoomPageTransformer implements ViewPager.PageTransfo
    private static final float MIN_SCALE = 0.85f;
    private static final float MIN_ALPHA = 0.5f;

    public void transformPage(View view, float position) {
        int pageWidth = view.getWidth();
        int pageHeight = view.getHeight();

        if (position < -1) { // [-Infinito,-1)

            // Esta página desaparece de la pantalla hacia la izq
            view.setAlpha(0);

        } else if (position <= 1) { // [-1,1]
```

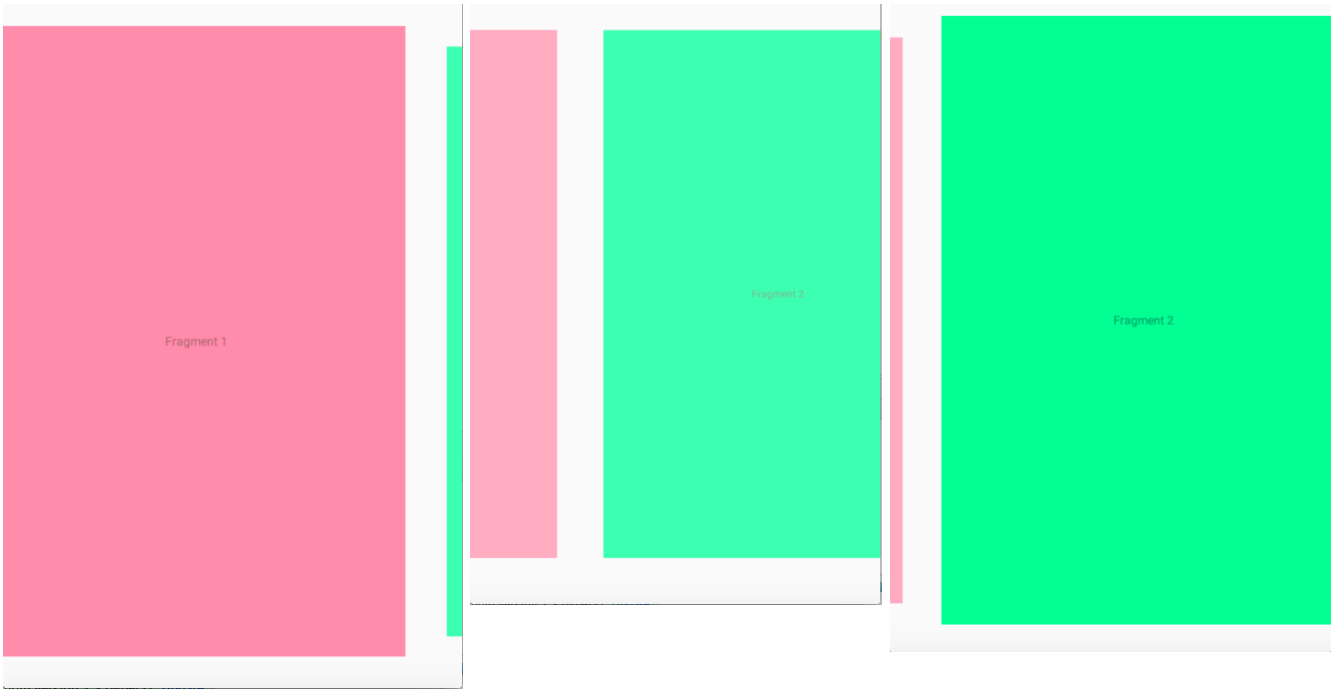
```
            // Modifica la transición deslizante encogiendo la pá
            float scaleFactor = Math.max(MIN_SCALE, 1 - Math.abs(
            float vertMargin = pageHeight * (1 - scaleFactor) / 2
            float horzMargin = pageWidth * (1 - scaleFactor) / 2;
            if (position < 0) {
                view.setTranslationX(horzMargin - vertMargin / 2)
            } else {
                view.setTranslationX(-horzMargin + vertMargin / 2
            }
            // Devuelve la página a su tamaño normal
            view.setScaleX(scaleFactor);
            view.setScaleY(scaleFactor);
            // Aparece la página con su tamaño normal
            view.setAlpha(MIN_ALPHA + (scaleFactor - MIN_SCALE) /

        } else { // (1,+Infinity]
            // Esta página desaparece de la pantalla hacia la der
            view.setAlpha(0);
        }
    }
}
```

En Kotlin:

```
class ZoomPageTransformer : ViewPager.PageTransformer {

    override fun transformPage(view: View, position: Float) {
        val pageWidth = view.getWidth()
        val pageHeight = view.getHeight()

        if (position < -1) { // [-Infinito,-1)

            // Esta página desaparece de la pantalla hacia la izq
            view.setAlpha(0F)

        } else if (position <= 1) { // [-1,1]

            // Modifica la transición deslizante encogiendo la pá
            val scaleFactor = Math.max(MIN_SCALE, 1 - Math.abs(po
```

```
            val vertMargin = pageHeight * (1 - scaleFactor) / 2
            val horzMargin = pageWidth * (1 - scaleFactor) / 2
            if (position < 0) {
                view.setTranslationX(horzMargin - vertMargin / 2)
            } else {
                view.setTranslationX(-horzMargin + vertMargin / 2
            }
            // Devuelve la página a su tamaño normal
            view.setScaleX(scaleFactor)
            view.setScaleY(scaleFactor)
            // Aparece la página con su tamaño normal
            view.setAlpha(MIN_ALPHA + (scaleFactor - MIN_SCALE) /

        } else { // (1,+Infinity]
            // Esta página desaparece de la pantalla hacia la der
            view.setAlpha(0F)
        }
    }

    companion object {
        private val MIN_SCALE = 0.85f
        private val MIN_ALPHA = 0.5f
    }
}
```

El efecto sería similar al siguiente:



Nota: otra posible animación Deep Page Transformer.

Menú contextual

El menú contextual es el que aparece al realizar una pulsación prolongada sobre cada uno de los items de un ListView o RecyclerView. Tenemos dos tipos de menú contextual: flotante o ActionMode.

Menú contextual flotante

El menú con opciones textuales aparece flotante al realizar una pulsación larga sobre un ítem del ListView o RecyclerView.

Ejemplo. Menú contextual flotante sobre el ítem de un ListView.

Nota: Se da por hecho que en layout principal de la aplicación hay un componente *ListView* con identificador listview.

Empezaremos implementando el menú que vamos a desplegar en la carpeta *menu*, por ejemplo (*menu_context.xml*):

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android">
    <item
        android:id="@+id/Editar"
        android:title="Editar" />
    <item
        android:id="@+id/Eliminar"
        android:title="Eliminar" />
    <item
        android:id="@+id/Compartir"
        android:title="Compartir" />
</menu>
```

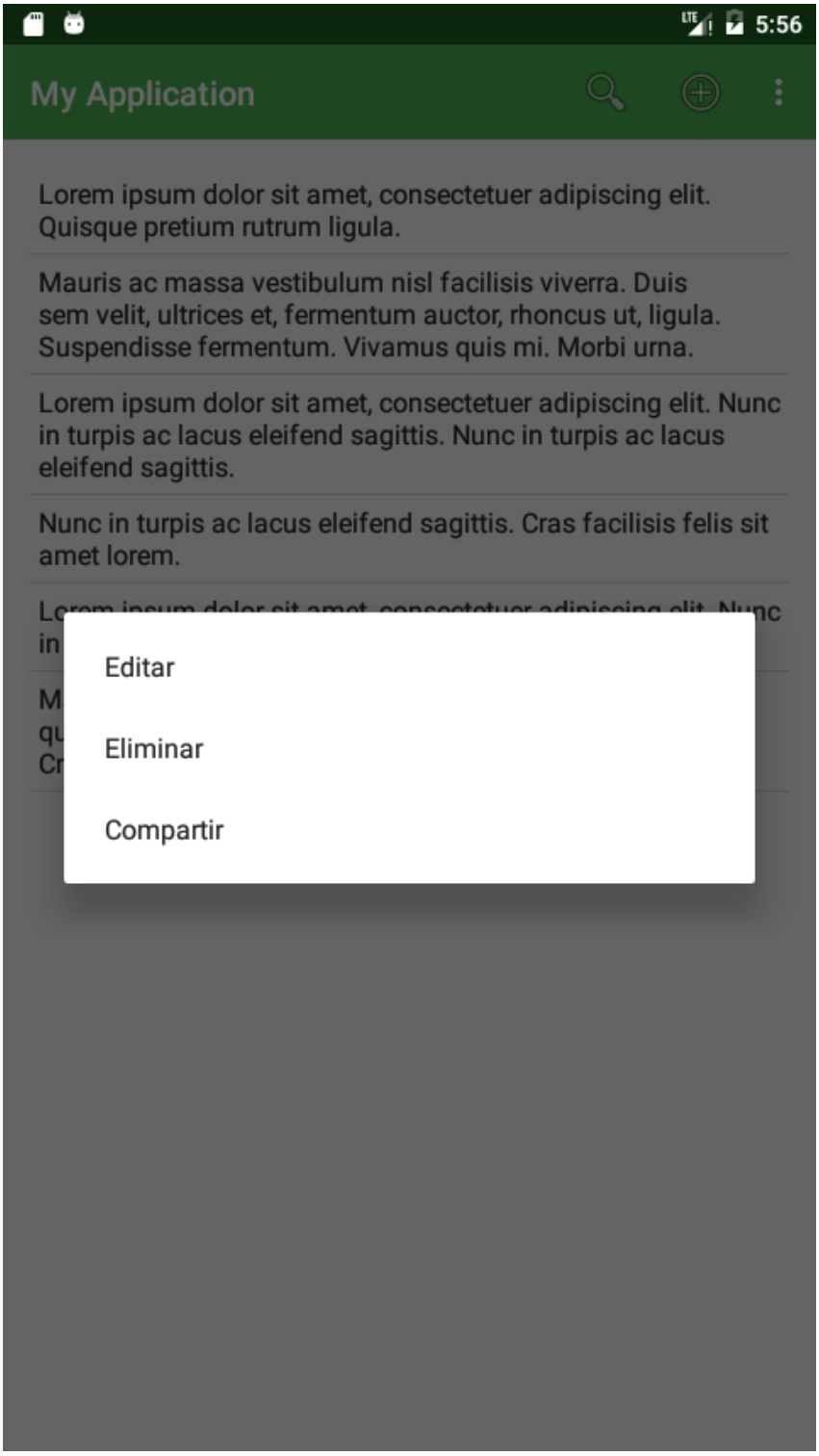
A continuación indicaremos en la clase principal, dentro del *onCreate()*, la sentencia que indicará la presencia de un menú contextual sobre el ListView:

```
ListView lista_tarjetas= (ListView) findViewById(R.id.listview);
registerForContextMenu(lista_tarjetas);
```

Finalmente, en el fichero *MainActivity.java*, fuera del *onCreate()*, implementaremos los métodos para inflar el menú creado anteriormente y, para darle funcionalidad a la opción elegida (en nuestro caso, sólo se mostrará un toast con el nombre de dicha opción):

```
@Override
public void onCreateContextMenu(ContextMenu menu, View v, Context
    super.onCreateContextMenu(menu, v, menuInfo);

    MenuInflater inflater = getMenuInflater();
    if (v.getId() == R.id.listview) {
        inflater.inflate(R.menu.menu_context, menu);
    }
}

@Override
public boolean onContextItemSelected(MenuItem item) {
    switch (item.getItemId()) {
        case R.id.Editar:
            Toast.makeText(this, "Editando", Toast.LENGTH_SHORT).
            return true;
        case R.id.Eliminar:
            Toast.makeText(this, "Eliminando", Toast.LENGTH_SHORT
            return true;
        case R.id.Compartir:
            Toast.makeText(this, "Compartiendo", Toast.LENGTH_SHO
            return true;
        default:
            return super.onOptionsItemSelected(item);
    }
}
```

El resultado obtenido:



Ejemplo. Menú contextual flotante sobre el ítem de un RecyclerView.

Similar al menú contextual sobre el ListView, aunque en este caso tendremos que realizar las operaciones dentro del adaptador, y no en la actividad o fragment donde se encuentre el componente. Concretamente, en la clase que extiende de RecyclerView.ViewHolder, tal y como se indica:

```
public class TarjViewHolder extends RecyclerView.ViewHolder implements
```

```
View.OnCreateContextMenuListener {
        public TarjViewHolder(View itemView) {
            super(itemView);
           itemView.setOnCreateContextMenuListener(this);
        }
       @Override
        public void onCreateContextMenu(ContextMenu contextMenu, View view,
ContextMenu.ContextMenuInfo contextMenuInfo) {
            contextMenu.add(0, 0, getAdapterPosition(), "Editar");     //groupId, itemId,
order, title
            contextMenu.add(0, 1, getAdapterPosition(), "Eliminar");
            contextMenu.add(0, 2, getAdapterPosition(), "Compartir");
        }
}
```

En Kotlin:

```
class TarjViewHolder(itemView: View) : RecyclerView.ViewHolder(it
  init {
     itemView.setOnCreateContextMenuListener(this)
  }
  override fun onCreateContextMenu(contextMenu: ContextMenu, view
     contextMenu.add(0, 0, adapterPosition, "Editar")     //group
     contextMenu.add(0, 1, adapterPosition, "Eliminar")
     contextMenu.add(0, 2, adapterPosition, "Compartir")
  }
}
```

Menú contextual ActionMode

El menú con opciones gráficas ActionMode, es la barra que aparece en lugar de la ToolBar o AppBar clásica, cuando el usuario realizar una pulsación larga sobre un ítem

del ListView o RecyclerView.

Ejemplo. Menú contextual ActionMode sobre el ítem de un ListView.

Para crear este tipo de menú contextual tendremos que definir primeramente una interfaz *ActionMode.CallBack*. Se llamará a esta interfaz cuando el usuario realice una pulsación larga sobre un item del *ListView*.

El código quedará:

```
private ActionMode.Callback modeCallBack = new ActionMode.Callbac

    public boolean onPrepareActionMode(ActionMode mode, Menu menu
        return false;
    }

    public void onDestroyActionMode(ActionMode mode) {
        mode = null;
    }

    public boolean onCreateActionMode(ActionMode mode, Menu menu)
        mode.setTitle("Options");
        mode.getMenuInflater().inflate(R.menu.menu_context, menu)
        return true;
    }

    public boolean onActionItemClicked(ActionMode mode, MenuItem
        int id = item.getItemId();

        switch (id) {
            case R.id.Compartir: {
                Toast.makeText(getApplicationContext(), "Comparti
                mode.finish();
                break;
            }
            case R.id.Eliminar: {
                Toast.makeText(getApplicationContext(), "Eliminan
                mode.finish();
                break;
            }
            case R.id.Editar: {
```

```
                Toast.makeText(getApplicationContext(), "Editando
                mode.finish();
                break;
            }
            default:
                return false;
        }
        return false;
    }
};
```

Nos interesan las lineas sombreadas en azul: la primera es donde crearemos la barra de acción contextual en la parte superior de la pantalla y la segunda será donde se indicará la acción a realizar en función del item del menú seleccionado en dicha barra (en este caso, por simplicidad, sólo se mostrará un toast con el nombre de dicha opción).

Crearemos el menú en *menu/menu_context*:

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android">
    <item
        android:id="@+id/Editar"
        android:title="Editar"
        android:icon="@android:drawable/ic_menu_edit" />
    <item
        android:id="@+id/Eliminar"
        android:icon="@android:drawable/ic_menu_delete"
        android:title="Eliminar" />
    <item
        android:id="@+id/Compartir"
        android:icon="@android:drawable/ic_menu_share"
        android:title="Compartir" />
</menu>
```

Mostraremos el menú contextual sobre el componente *ListView*:

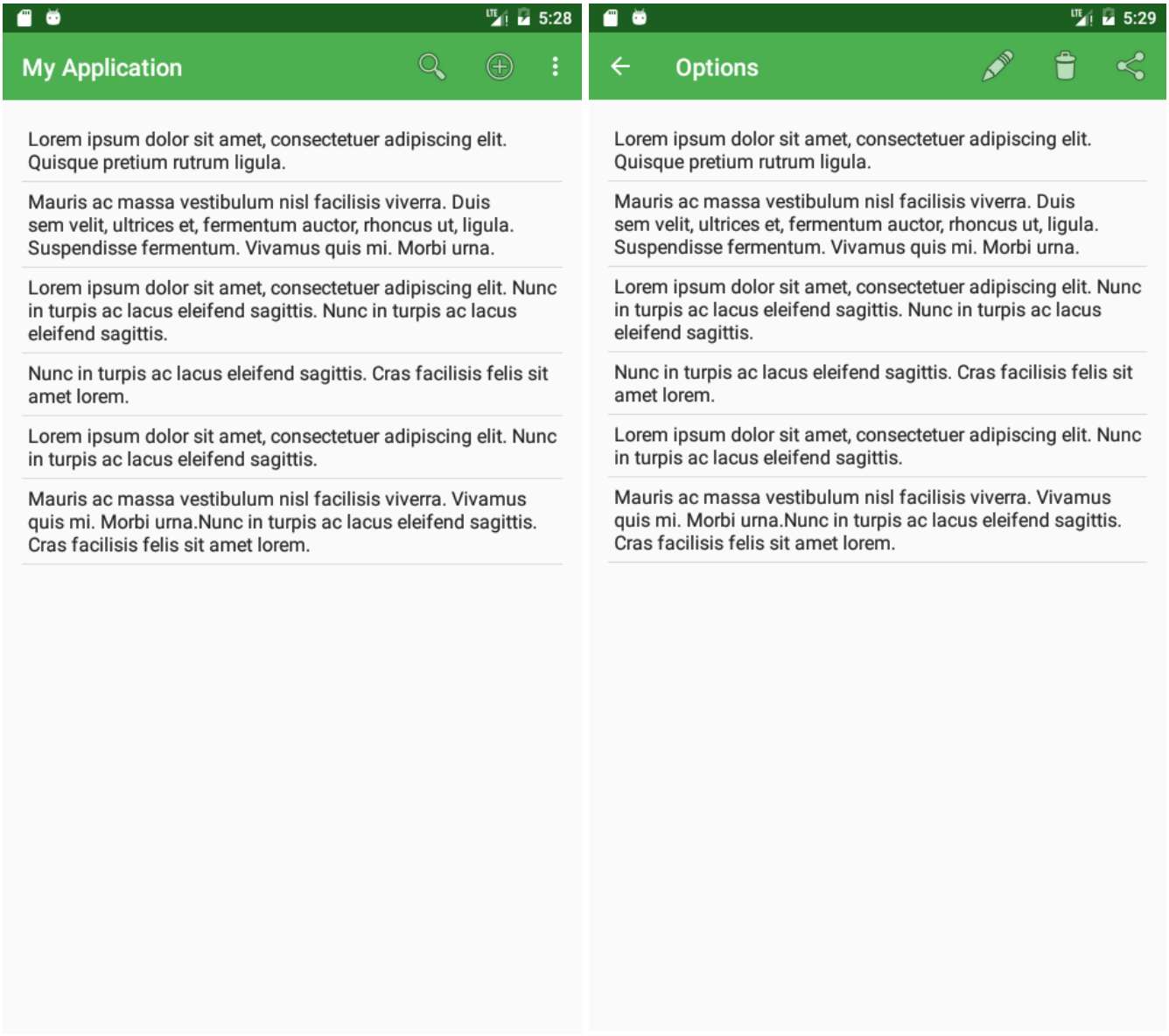
```
lista_tarjetas.setOnItemLongClickListener(new AdapterView.OnItemL
```

```
    public boolean onItemLongClick(AdapterView parent, View view,
        ActionMode mActionMode = startActionMode(modeCallBack);
        mActionMode.setTag(position);
        view.setSelected(true);
        return true;
    }
});
```

Por último, para conseguir que la barra de opciones *ActionMode* mantenga el color de la *ToolBar* y se sitúe en lugar de ésta, en vez de por encima, añadiremos en el fichero *values/styles.xml* las sentencias:

```
<item name="actionModeBackground">@color/colorPrimary</item>
<item name="windowActionModeOverlay">true</item>
```

El resultado obtenido será:



Ejemplo. Menú contextual ActionMode sobre el ítem de un RecyclerView.

Similar al ejemplo del ListView, pero en este caso, definiendo el código dentro del adaptador, concretamente en la clase que extiende de RecyclerView.ViewHolder.
En Kotlin:

```
class TarjViewHolder(itemView: View) : RecyclerView.ViewHolder(it

  init {
      itemView.setOnLongClickListener { view ->
        (view.context as AppCompatActivity).startSupportActionMod
        return true
      }
  }
```

```
    var modeCallBack: ActionMode.Callback = object : ActionMode.Cal
        override fun onActionItemClicked(mode: ActionMode?, item:
            val id = item?.itemId
            when (id) {
                R.id.action_renombrar -> {
                    Toast.makeText(itemView.context, "Renombrando
                    mode?.finish()
                }
                R.id.action_añadir -> {
                    Toast.makeText(itemView.context, "Añadiendo",
                    mode?.finish()
                }
                R.id.action_eliminar -> {
                    Toast.makeText(itemView.context, "Eliminando"
                    mode?.finish()
                }
                else -> return false
            }
            return false
        }

        override fun onPrepareActionMode(mode: ActionMode, menu:
            return false
        }

        override fun onDestroyActionMode(mode: ActionMode?) {
            var mode = mode
            mode = null
        }

        override fun onCreateActionMode(mode: ActionMode, menu: M
            mode.setTitle("Options")
            mode.getMenuInflater().inflate(R.menu.menu_tar, menu)
            return true
        }
    }
}
```