Animaciones básicas: CoordinatorLayout

Material Design pone mucho énfasis no sólo en el aspecto estático de los elementos de la interfaz de usuario, sino también en el movimiento de éstos por la pantalla. Gran parte de las aplicaciones actuales muestran animaciones aquí y allá, no siempre con sentido, pero siempre intentando reforzar la experiencia del usuario.

Algunas de estas animaciones, a fuerza de utilizarlas, se han vuelto ya un clásico en nuestros dispositivos, y es éste uno de los motivos que han llevado a Google a incluir en la nueva librería de diseño (*Design Support Library*) nuevos componentes que facilitan en gran medida su implementación, permitiéndonos así centrarnos más en la funcionalidad principal de nuestras aplicaciones, y no tanto en su comportamiento puramente visual.

En este nuevo artículo del curso veremos algunas de las tareas que nos facilitan estos nuevos componentes, y para ello iremos construyendo poco a poco una aplicación de ejemplo en la que introduciremos progresivamente estos elementos. Para tampoco empezar de cero, me basaré en código ya desarrollado en artículos anteriores.

Vamos a empezar por crear una nueva aplicación en blanco en Android Studio a la que aplicaremos el tema `Theme.AppCompat.Light.NoActionBar`, definiremos los colores principales del tema, y añadiremos de forma explícita un componente `Toolbar` que funcione como *action bar* (o *app bar*, según la nueva terminología). Esto lo explicamos en detalle en el toolbar (II) sobre la

action bar. Obviaré en esta ocasión los “trucos” para conseguir la sombra en versiones de Android anteriores a Lollipop.

A continuación añadiremos a nuestro layout una lista de elementos utilizando el control `RecyclerView`. En el artículo dedicado a este componente vimos los pasos necesarios para ello, creando un adaptador personalizado (`AdaptadorTitulares`), utilizando uno de los `LayoutManager` estandar, y un `ItemDecorator` extraído de los ejemplos del SDK. Para este ejemplo copiaré directamente el código que ya utilizamos en dicho artículo.

Nuestro layout XML en este momento quedaría como sigue:

```
1
2
3       <LinearLayout
4           xmlns:android="http://schemas.android.com/apk/res/android"
5           xmlns:tools="http://schemas.android.com/tools"
6           xmlns:app="http://schemas.android.com/apk/res-auto"
7           android:layout_width="match_parent"
8           android:layout_height="match_parent"
9           android:orientation="vertical"
10          tools:context=".MainActivity" >
11
12          <android.support.v7.widget.Toolbar
13              android:id="@+id/appbar"
14              android:layout_height="?attr/actionBarSize"
15              android:layout_width="match_parent"
16              android:minHeight="?attr/actionBarSize"
17              android:background="?attr/colorPrimary"
18              android:elevation="4dp"
19              android:theme="@style/ThemeOverlay.AppCompat.Dark.ActionBar"
20              app:popupTheme="@style/ThemeOverlay.AppCompat.Light" >
21
22          </android.support.v7.widget.Toolbar>
23
24          <android.support.v7.widget.RecyclerView
25              android:id="@+id/lstLista"
26              android:layout_width="match_parent"
27              android:layout_height="match_parent" />
28
29      </LinearLayout>
```

Y el código java de la actividad principal quedaría de la siguiente forma:

```
1
2
3     public class MainActivity extends AppCompatActivity {
4
5         private RecyclerView lstLista;
6
7         @Override
8         protected void onCreate(Bundle savedInstanceState) {
9             super.onCreate(savedInstanceState);
10            setContentView(R.layout.activity_main);
11
12            //App bar
13            Toolbar toolbar = (Toolbar) findViewById(R.id.appbar);
14            setSupportActionBar(toolbar);
15            getSupportActionBar().setTitle("Mi Aplicación");
16
17            //RecyclerView
18            RecyclerView lstLista = (RecyclerView)findViewById(R.id.lstLista);
19
20            ArrayList<Titular> datos = new ArrayList<>();
21            for(int i=0; i<50; i++)
22                datos.add(new Titular("Título " + i, "Subtítulo item " + i));
23
24            AdaptadorTitulares adaptador = new AdaptadorTitulares(datos);
25            lstLista.setAdapter(adaptador);
26
27            lstLista.setLayoutManager(
28                new LinearLayoutManager(this, LinearLayoutManager.VERTICAL, false));
29            lstLista.addItemDecoration(
30                new DividerItemDecoration(this, DividerItemDecoration.VERTICAL_LIST));
31            lstLista.setItemAnimator(new DefaultItemAnimator());
32        }
33
        //...
    }
```

Hasta aquí nada nuevo. A continuación vamos a introducir ya uno de los nuevos componentes que nos proporciona la nueva librería de diseño para controlar la animación de nuestra interfaz, y más concretamente de la *action bar*. Este nuevo componente, `AppBarLayout`, ya lo comentamos de pasada en el “toolbar(III)”, donde vimos que entre otras cosas nos ayudaba con la integración de pestañas bajo la action bar.

Por el momento vamos a introducir un `AppBarLayout` en nuestro layout principal que envuelva al control `Toolbar`. Dentro de poco veremos qué papel va a jugar este componente en todo esto.

Seguidamente vamos a introducir el elemento más importante en todo este tema, y que da nombre a este artículo, el nuevo `CoordinatorLayout`. Este nuevo tipo de layout es el que va a controlar, de una forma algo *mágica*, la animación de algunos de los elementos de la interfaz, y está pensado para usarlo como contenedor principal en nuestros layouts XML.

En la práctica, basta con añadirlo como elemento padre de nuestro layout. En nuestro caso, sustituiremos el `LinearLayout` principal por un contenedor de tipo `CoordinatorLayout`. Veamos cómo queda el layout con los dos últimos cambios realizados:

```
1
2       <android.support.design.widget.CoordinatorLayout
3           xmlns:android="http://schemas.android.com/apk/res/android"
4           xmlns:tools="http://schemas.android.com/tools"
5           xmlns:app="http://schemas.android.com/apk/res-auto"
6           android:layout_width="match_parent"
7           android:layout_height="match_parent"
8           tools:context=".MainActivity" >
9
10          <android.support.design.widget.AppBarLayout
11              android:id="@+id/appbarLayout"
12              android:layout_width="match_parent"
13              android:layout_height="wrap_content"
14              android:theme="@style/ThemeOverlay.AppCompat.Dark.ActionBar" >
15
16              <android.support.v7.widget.Toolbar
17                  android:id="@+id/appbar"
18                  android:layout_height="?attr/actionBarSize"
19                  android:layout_width="match_parent"
20                  android:minHeight="?attr/actionBarSize"
21                  android:background="?attr/colorPrimary"
22                  android:elevation="4dp"
23                  android:theme="@style/ThemeOverlay.AppCompat.Dark.ActionBar"
24                  app:popupTheme="@style/ThemeOverlay.AppCompat.Light" >
25
26              </android.support.v7.widget.Toolbar>
27
28          </android.support.design.widget.AppBarLayout>
29
30          <android.support.v7.widget.RecyclerView
31              android:id="@+id/lstLista"
32              android:layout_width="match_parent"
33              android:layout_height="match_parent" />
34
35      </android.support.design.widget.CoordinatorLayout>
36
```

32
33
34

Si ejecutamos la aplicación en este momento veremos que por ahora nada ha cambiado: una action bar estática y la lista sobre la que podemos desplazarnos. La diferencia es que con los nuevos elementos introducidos estamos en disposición de cambiar muy fácilmente el comportamiento de la action bar para que responda a los eventos de otros elementos, como por ejemplo el scroll en la lista. Un comportamiento clásico en este tipo de interfaces es ocultar la action bar una vez comenzamos a desplazarnos hacia abajo en la lista y volver a mostrarla en cuanto hacemos scroll en sentido contrario. Pues bien, hacer esto es ahora tan sencillo como añadir al `RecyclerView` una nueva propiedad `layout_behavior` con valor "`@string/appbar_scrolling_view_behavior`", y al `Toolbar` la propiedad `layout_scrollFlags` con valor "`scroll|enterAlways`". Simplificando el tema:

- La primera avisará al `CoordinatorLayout` de que debe estar atento y reaccionar al scroll del control `RecyclerView`.
- La segunda le indicará cómo debe actuar la action bar como respuesta a dicho evento de scroll, en este caso usamos la combinación de dos valores:
    - `scroll`: Indica que la action bar debe ocultarse cuando se haga scroll hacia abajo en la lista.
    - `enterAlways`: Indica que la action bar debe mostrarse en cuanto se comience a hacer scroll hacia arriba en la lista, lo que se conoce como *quick return*. Si no se utiliza este indicativo, la action bar sólo se mostraría al alcanzarse el primer elemento de la lista.

Veamos cómo quedaría el layout XML:

```
<android.support.design.widget.CoordinatorLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity" >

    <android.support.design.widget.AppBarLayout
        android:id="@+id/appbarLayout"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:theme="@style/ThemeOverlay.AppCompat.Dark.ActionBar" >

        <android.support.v7.widget.Toolbar
            android:id="@+id/appbar"
            android:layout_height="?attr/actionBarSize"
            android:layout_width="match_parent"
            android:minHeight="?attr/actionBarSize"
            android:background="?attr/colorPrimary"
            android:elevation="4dp"
            app:layout_scrollFlags="scroll|enterAlways"
            android:theme="@style/ThemeOverlay.AppCompat.Dark.ActionBar"
            app:popupTheme="@style/ThemeOverlay.AppCompat.Light" >

        </android.support.v7.widget.Toolbar>

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

    <android.support.v7.widget.RecyclerView
        android:id="@+id/lstLista"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        app:layout_behavior="@string/appbar_scrolling_view_behavior" />

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

En el siguiente video se observa el comportamiento de la action bar estableciendo el atributo `layout_scrollFlags` al valor "`scroll|enterAlways`" y también sólo "`scroll`", para observar la diferencia:

https://youtu.be/d-ek7YpvAGU

¿Pero cómo funciona todo esto si añadimos algún elemento a nuestra interfaz que esté ligado a la action bar? Incluyamos por ejemplo una

barra de pestañas (tabs) como explicamos en el tercer artículo sobre la action bar. Añadimos primero el control `TabLayout` bajo el `Toolbar` (y dentro del `AppBarLayout`) en nuestro layout XML, acordándonos de eliminar el atributo `elevation` del toolbar:

```
...
<android.support.design.widget.AppBarLayout
    android:id="@+id/appbarLayout"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:theme="@style/ThemeOverlay.AppCompat.Dark.ActionBar" >

    <android.support.v7.widget.Toolbar
        android:id="@+id/appbar"
        android:layout_height="?attr/actionBarSize"
        android:layout_width="match_parent"
        android:minHeight="?attr/actionBarSize"
        android:background="?attr/colorPrimary"
        app:layout_scrollFlags="scroll|enterAlways"
        android:theme="@style/ThemeOverlay.AppCompat.Dark.ActionBar"
        app:popupTheme="@style/ThemeOverlay.AppCompat.Light" >

    </android.support.v7.widget.Toolbar>

    <android.support.design.widget.TabLayout
        android:id="@+id/appbartabs"
        android:layout_width="match_parent"
        android:layout_height="wrap_content" />

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

Y lo configuramos desde el `onCreate()` de la actividad con tres pestañas fijas de muestra, aunque esta vez sin ninguna funcionalidad para no complicar más el ejemplo:

```
@Override
protected void onCreate(Bundle savedInstanceState) {

    //...

    //TabLayout
    TabLayout tabLayout = (TabLayout) findViewById(R.id.appbartabs);
    tabLayout.setTabMode(TabLayout.MODE_FIXED);
    tabLayout.addTab(tabLayout.newTab().setText("Tab 1"));
    tabLayout.addTab(tabLayout.newTab().setText("Tab 2"));
    tabLayout.addTab(tabLayout.newTab().setText("Tab 3"));
}
```

```
11
12
```

Si volvemos a ejecutar ahora la aplicación veremos cómo al hacer scroll en la lista, la action bar desaparece de la pantalla igual que en el ejemplo anterior, pero que la barra de pestañas queda fija en la parte superior. Esto es un patrón muy común en muchas aplicaciones, por ejemplo la propia tienda de Google Play, pero podríamos querer que las pestañas también desaparecieran de pantalla para dejar más espacio a la lista principal mientras se hace scroll. Esto sería tan sencillo como añadir el atributo `layout_scrollFlags` también al control `TabLayout`, con los mismos valores que en el toolbar:

```
1
2
3
4
5
```
```
    <android.support.design.widget.TabLayout
        android:id="@+id/appbartabs"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        app:layout_scrollFlags="scroll|enterAlways" />
```

En el siguiente video se pueden ver las diferencias en el comportamiento del bloque de pestañas con y sin *scroll flags*:

https://youtu.be/3Dxd1jr7jiM

Sigamos añadiendo elementos a la interfaz. En primer lugar vamos a eliminar las pestañas que acabamos de introducir para que no nos molesten (quitamos el control `TabLayout` del layout XML y las lineas de configuración del `onCreate`), y vamos a añadir un botón flotante (`FloatingActionButton`) tal cómo vimos al final del artículo sobre botones.

Colocaremos este botón bajo el `RecyclerView` en nuestro layout XML, con la siguiente definición:

```
1
2
3
4
```
```
    <android.support.design.widget.FloatingActionButton
        android:id="@+id/btnFab"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_margin="16dp"
```

```
5         android:layout_gravity="end|bottom"
6         android:src="@drawable/ic_fab_icon"
7         app:borderWidth="0dp" />
8
```

Queremos que el botón aparezca en la esquina inferior derecha de la pantalla, por lo que establecemos un `layout_gravity` con valor "`end|bottom`" con un margen de `16dp`. Como respuesta a la pulsación de este botón vamos a mostrar una notificación de tipo *snackbar*, como ya explicaremos en detalle.

```
//Floating Action Button
btnFab = (FloatingActionButton)findViewById(R.id.btnFab);
btnFab.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {
        Snackbar.make(view, "Esto es una prueba",
Snackbar.LENGTH_LONG).show();
    }
});
```

Como ya veremos, un snackbar se muestra en la parte inferior de la pantalla, por lo que en condiciones normales aparecería por encima del botón flotante que acabamos de introducir en la interfaz. Sin embargo, si ejecutamos la aplicación y pulsamos el botón nos encontraremos con una grata sorpresa:

https://youtu.be/krgTe8Ve3pI

En el video anterior podemos ver cómo al mostrarse el snackbar el botón flotante se desplaza automáticamente hacia arriba para dejar espacio a la notificación, y cuando ésta desparece el botón vuelve a desplazarse a su posición original. Además, como ya vimos, podemos descartar la notificación con el gesto de deslizamiento hacia la derecha. Pues bien, todo esto se lo debemos una vez más al nuevo control `CoordinatorLayout`, que no sólo hace posible las animaciones sino que controla cuándo éstas pueden o deben afectar a

otros elementos de la interfaz, en este caso a un botón flotante. Y todo ello sin escribir ni una sólo línea de código específica para estas tareas.

Sigamos “complicando” el tema. Un patrón también muy típico en muchas aplicaciones actuales es utilizar una action bar extendida, que inicialmente ocupa buena parte de la pantalla, y hacer que ésta se contraiga hasta su tamaño mínimo al hacer scroll sobre el contenido principal de la actividad. Podemos ver este comportamiento por ejemplo en la aplicación Play Music.

Pues bien, Google también contempla esta posibilidad en la nueva librería de diseño, y lo hace introduciendo otro tipo adicional de contenedor, llamado `CollapsingToolbarLayout`, que siempre junto al ya mencionado `CoordinatorLayout` nos facilitará enormemente la tarea. Este nuevo componente debe colocarse dentro del `AppBarLayout` y conteniendo al `Toolbar`.

```
...
<android.support.design.widget.AppBarLayout
    android:id="@+id/appbarLayout"
    android:layout_width="match_parent"
    android:layout_height="192dp"
    android:theme="@style/ThemeOverlay.AppCompat.Dark.ActionBar" >

    <android.support.design.widget.CollapsingToolbarLayout
        android:id="@+id/ctlLayout"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        app:layout_scrollFlags="scroll|exitUntilCollapsed">

        <android.support.v7.widget.Toolbar
            xmlns:android="http://schemas.android.com/apk/res/android"
            xmlns:app="http://schemas.android.com/apk/res-auto"
            android:id="@+id/appbar"
            android:layout_height="?attr/actionBarSize"
            android:layout_width="match_parent"
            android:minHeight="?attr/actionBarSize"
            android:background="?attr/colorPrimary"
            android:theme="@style/ThemeOverlay.AppCompat.Dark.ActionBar"
            app:popupTheme="@style/ThemeOverlay.AppCompat.Light"
            app:layout_collapseMode="pin" >

        </android.support.v7.widget.Toolbar>
```

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

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

Comentemos un poco el código anterior. Lo primero que observamos es que es el atributo `layout_height` del `AppBarLayout` el que recibe la altura de la action bar extendida, en este caso de ejemplo indicaremos `192dp`.

También observamos que hemos añadido al `Toolbar` un nuevo atributo `layout_collapseMode` con valor "`pin`". Con esto conseguiremos que los elementos mostrados en la action bar (botones de acción, menú de overflow, …) no se vean afectados por la expansión/contracción de la misma, es decir, que se queden siempre fijados a la parte superior de la pantalla salvo cuando la propia action bar desaparece.

Vemos también que ya no nos hace falta utilizar el atributo `layout_scrollFlags` en el `Toolbar`, ya que este atributo lo "desplazaremos" al nuevo componente `CollapsingToolbarLayout`, donde podremos utilizar algunos valores ya conocidos, como "`scroll`" o "`enterAlways`", y algunos nuevos como "`exitUntilCollapsed`" o "`enterAlwaysCollapsed`". Dependiendo de la combinación de valores utilizada para este atributo el comportamiento de la action bar al contraerse/expandirse y salir de la pantalla será distinto. Algunas combinaciones típicas serían las siguientes:

1. `scroll|enterAlways`: al hacer scroll hacia arriba la action bar se contrae hasta su tamaño mínimo y posteriormente desaparece

de la pantalla, tras esto comienza el scroll real de la lista. Al hacer scroll en sentido contrario la action bar comienza a aparecer inmediatamente, se expande, y cuando alcanza su tamaño máximo comienza el scroll de la lista.

2. `scroll|enterAlwaysCollapsed`: Similar a la opción 1, pero la action bar sólo comienza a expandirse cuando la lista ha alcanzado su primer elemento.
3. `scroll|enterAlways|enterAlwaysCollapsed`: Similar a la opción 2, pero en esta ocasión la action bar aparece en su tamaño mínimo inmediatamente al hacer scroll hacia abajo, y solo se expande al llegar al primer elemento de la lista.
4. `scroll|exitUntilCollapsed`: Similar a la opción 3 con la diferencia de que la action bar no llega a desaparecer, tan solo se contrae y expande hasta su tamaño mínimo, priorizándose además el scroll de la lista.

En el siguiente video veremos mejor las diferencias entre las opciones anteriores:

https://youtu.be/pp6s2zWQJ8c

Vemos además en los videos anteriores que el título de la action bar cambia de tamaño y se desplaza de forma coordinada con la expansión de la action bar. Esto lo conseguimos estableciendo el título de la actión bar sobre el componente `CollapsingToolbarLayout`, en vez de sobre el `Toolbar`:

```
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        //App bar
        Toolbar toolbar = (Toolbar) findViewById(R.id.appbar);
        setSupportActionBar(toolbar);
        //getSupportActionBar().setTitle("Mi Aplicación");

```

```
        //...

        //CollapsingToolbarLayout
        ctlLayout = (CollapsingToolbarLayout)findViewById(R.id.ctlLayout);
        ctlLayout.setTitle("Mi Aplicación");
    }
```

Otra posibilidad que nos proporciona el nuevo componente `CoordinatorLayout` en combinación con `AppBarLayout` y `CollapsingToolbarLayout` es “fijar” algunos componentes a otros componentes móviles de la interfaz, de forma que ambos se muevan de forma coordinada. Un caso típico sería fijar un botón flotante a la parte inferior de una action bar “expandible”.

Para hacer esto vamos a modificar un poco la definición del FloatingActionButton que ya teníamos incluido en nuestro layout XML. Primero vamos a eliminar su atributo`layout_gravity`, y lo sustituiremos por otros dos nuevos: `layout_anchor` y`layout_anchorGravity`. El primero de ellos nos permitirá indicar el control al que fijaremos el botón, y el segundo su alineación respecto a dicho control. En nuestro caso queremos fijarlo a la parte inferior derecha de la action bar, por lo que quedaría de la siguiente forma:

```
    <android.support.design.widget.FloatingActionButton
        android:id="@+id/btnFab"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_margin="16dp"
        android:src="@drawable/ic_fab_icon"
        app:borderWidth="0dp"
        app:layout_anchor="@id/appbarLayout"
        app:layout_anchorGravity="bottom|right|end" />
```

Un detalle a tener en cuenta es que en la propiedad `layout_anchor` indicaremos el ID del `AppBarLayout` y no del `Toolbar` en sí.

Tan sólo con esto, si volvemos a ejecutar una vez más la aplicación veremos cómo el botón flotante queda fijado a la action bar y se mueve y/o desaparece de forma totalmente coordinada con ésta:

https://youtu.be/FWlgdVbHDQk

Por último, vamos a ver cómo podemos añadir una imagen de fondo a la action bar extendida que además realice el típico efecto de *parallax* al contraerse y expandirse.

Para conseguir esto tan sólo tenemos que añadir un control `ImageView` dentro del `CollapsingToolbarLayout`, justo antes del `Toolbar`, al que añadiremos el atributo`layout_collapseMode` con valor "`parallax`". También tendremos que eliminar el atributo `background` del `Toolbar` y añadir al `CollapsingToolbarLayout` el atributo `contentScrim` con el color que queramos que tenga el toolbar una vez contraído (normalmente el *primaryColor* del tema, el mismo que antes teníamos asignado al background del toolbar):

```
1     <android.support.design.widget.CollapsingToolbarLayout
2         android:id="@+id/ctlLayout"
3         android:layout_width="match_parent"
4         android:layout_height="match_parent"
5         app:layout_scrollFlags="scroll|exitUntilCollapsed"
6         app:contentScrim="?attr/colorPrimary" >
7
8         <ImageView
9             android:id="@+id/imgToolbar"
10            android:layout_width="match_parent"
11            android:layout_height="match_parent"
12            android:scaleType="centerCrop"
13            android:src="@drawable/img_toolbar"
14            app:layout_collapseMode="parallax" />
15
16        <android.support.v7.widget.Toolbar
17            android:id="@+id/appbar"
18            android:layout_height="?attr/actionBarSize"
19            android:layout_width="match_parent"
20            android:minHeight="?attr/actionBarSize"
21            app:layout_scrollFlags="scroll|enterAlways"
22            android:theme="@style/ThemeOverlay.AppCompat.Dark.ActionBar"
23            app:popupTheme="@style/ThemeOverlay.AppCompat.Light"
24            app:layout_collapseMode="pin" >
```

```
21
22          </android.support.v7.widget.Toolbar>
23
24      </android.support.design.widget.CollapsingToolbarLayout>
25
26
27
28
```

Si ejecutamos por última vez la aplicación, veremos cómo la imagen aparece como fondo de la action bar y cómo ésta desaparece una vez contraida quedando el color que hemos utilizado como `contentScrim`:

Y con esto terminaríamos el artículo dedicado al nuevo componente `CoordinatorLayout` y sus elementos asociados `AppBarLayout` y `CollapsingToolabrLayout`. Espero que os sirva para dar más vida a vuestras aplicaciones.

Puedes consultar y/o descargar el código completo de los ejemplos desarrollados en este artículo accediendo a la página del curso en GitHub.