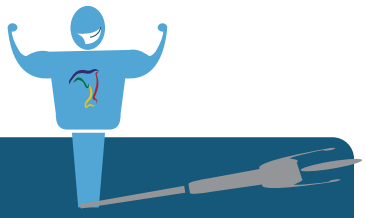
autentia

Avenida de Castilla,1 - Edificio Best Point - Oficina 21B
28830 San Fernando de Henares (Madrid)
tel./fax: +34 91 675 33 06
info@autentia.com - **www.autentia.com**

¿**Qué ofrece** Autentia Real Business Solutions S.L?

Somos su empresa de **Soporte a Desarrollo Informático**.
Ese apoyo que siempre quiso tener...

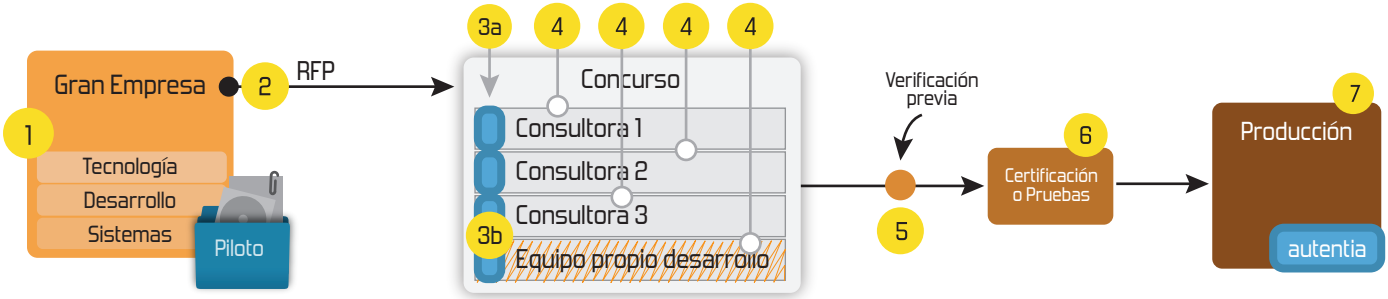
1. Desarrollo de componentes y proyectos a medida



2. Auditoría de código y recomendaciones de mejora

3. Arranque de proyectos basados en nuevas tecnologías

1. Definición de frameworks corporativos.
2. Transferencia de conocimiento de nuevas arquitecturas.
3. Soporte al arranque de proyectos.
4. Auditoría preventiva periódica de calidad.
5. Revisión previa a la certificación de proyectos.
6. Extensión de capacidad de equipos de calidad.
7. Identificación de problemas en producción.



4. Cursos de formación (impartidos por desarrolladores en activo)

Spring MVC, JSF-PrimeFaces /RichFaces, HTML5, CSS3, JavaScript-jQuery

Gestor portales (Liferay)
Gestor de contenidos (Alfresco)
Aplicaciones híbridas

Control de autenticación y acceso (Spring Security)
UDDI
Web Services
Rest Services
Social SSO
SSO (Cas)

JPA-Hibernate, MyBatis
Motor de búsqueda empresarial (Solr)
ETL (Talend)

Dirección de Proyectos Informáticos.
Metodologías ágiles
Patrones de diseño
TDD

Tareas programadas (Quartz)
Gestor documental (Alfresco)
Inversión de control (Spring)

BPM (jBPM o Bonita)
Generación de informes (JasperReport)
ESB (Open ESB)

Compartimos nuestro conocimiento en:
www.adictosaltrabajo.com

Para más información visítenos en:
www.autentia.com

Francisco J. Arroyo

Consultor tecnológico de desarrollo de proyectos informáticos.

Puedes encontrarme en Autentia: Ofrecemos servicios de soporte a desarrollo, factoría y formación

Somos expertos en Java/JEE

Ver todos los tutoriales del autor

Fecha de publicación del tutorial: 2012-11-06 Tutorial visitado 1 veces Descargar en PDF

Integrar Barcode Scanner en nuestra aplicación Android

0. Índice de contenidos.

1. Introducción

En este tutorial, vamos a ver como podemos leer códigos qr desde nuestra aplicación de **Android**.

Para quien no conozca qué son los códigos QR, comentaré de forma reducida que, son códigos de barra en los que cabe bastante más información que en los códigos de barra tradicionales. Seguramente os sonará la siguiente imagen.

Para realizar la lectura de los códigos, vamos a utilizar una aplicación que problablemente ya conoceréis: Barcode Scanner, que para mi gusto y con permiso de Goggles (aunque ésta es capaz de reconocer objetos), es la mejor aplicación para leer códigos de este tipo.

2. Entorno

- Hardware
  - Mackbook Pro
    - Intel Core i7 2Ghz
    - 8GB RAM
    - 500GB HD
    - Sistema Operativo: Mac OS X (10.8.2)
  - Galaxy Nexus
    - Android 4.1.1 (Jelly Bean).
- Software
  - IntelliJ 11 Ultimate

3. Desarrollo de la aplicación.

Manos a la obra:

El primer detalle es que no tenemos que indicar ningún permiso en el fichero **AndroidManifest.xml** de la aplicación, ya que nuestra aplicación no va a acceder a ningún recurso, sólo lanzaremos un intent que hará el trabajo por nosotros. Nota: Si vuestra aplicación necesita acceder a la cámara o a internet, tendréis que añadir igualmente los permisos en el AndroidManifest.xml.

El segundo paso es agregar un par de clases a nuestro proyecto. Estas dos clases son:

- IntentIntegrator
- IntentResult

Y ahora os voy a poner ya el código, para que veáis de qué forma tan sencilla podemos leer un código qr :).

Empezamos por el código de la vista:

view plain print ?

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android">
             //oculto para facilitar la lectura

    <ImageButton android:background="@drawable/buton_selector"
        //oculto para facilitar la lectura
    </ImageButton>

    <TextView android:id="@+id/tvFormat">
        //oculto para facilitar la lectura

    </TextView>

    <TextView android:id="@+id/tvResult">
        //oculto para facilitar la lectura
    </TextView>

</LinearLayout>
```

Si os fijáis en el layout, tenemos un botón (**ImageButton**) y dos **TextView**. El botón lo usaremos para iniciar el intent que lanzará la aplicación encargada de leer el qr-code. Los dos TextViews los usaremos para mostrar la lectura.

Y terminos con el código que ejecuta la acción. Aquí esta toda la clase del Activity (excepto los imports).

view plain print ?

```
public class QrReaderActity extends Activity {

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        configureButtonReader();
    }

    private void configureButtonReader() {
        final ImageButton buttonReader = (ImageButton)findViewById(R.id.btReader);
        buttonReader.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                new IntentIntegrator(QrReaderActity.this).initiateScan();
            }
        });
    }

    @Override
    public void onActivityResult(int requestCode, int resultCode, Intent intent) {
        final IntentResult scanResult = IntentIntegrator.parseActivityResult(requestCode, resultCode,
        handleResult(scanResult);
    }

    private void handleResult(IntentResult scanResult) {
        if (scanResult != null) {
            updateUITextViews(scanResult.getContents(), scanResult.getFormatName());
        } else {
            Toast.makeText(this, "No se ha leído nada :(", Toast.LENGTH_SHORT).show();
        }
    }

    private void updateUITextViews(String scan_result, String scan_result_format) {
        ((TextView)findViewById(R.id.tvFormat)).setText(scan_result_format);
        final TextView tvResult = (TextView)findViewById(R.id.tvResult);
        tvResult.setText(scan_result);
        Linkify.addLinks(tvResult, Linkify.ALL);
    }
}
```

Lo que hay que destacar de este código están en las líneas:

- **15**: Lo que hace es inicializar el **Intent** que las propiedades que necesita para que se ejecute Barcode Scanner. Al no pasar parámetros, intentará leer todos los tipos diferentes de códigos de barra que es capaz. Además es importante que especifiquemos nuestro activity, para que luego podamos gestionar el resultado de la lectura.
- **20 a 24**: Cuando llamamos a **initiateScan()**, lo que hace es empezar un Activity con **startActivityForResult**. Entonces si redefinimos el método **onActivityResult**, podemos gestionar el resultado. En este caso instanciamos un **IntentResult** parseando el intent que hemos recibido. Esta clase nos ayuda a extraer el resultado del intent.
- **28**: Extraemos el resultado del **IntentResult** con **scanResult.getContents()** y **scanResult.getFormatName()** y lo pintamos en la pantalla.

Como diría un profesor que tuve de la carrera: Facilote, facilote!! :P

Para que veáis que no os engaño, os muestro unas capturas del proceso :)

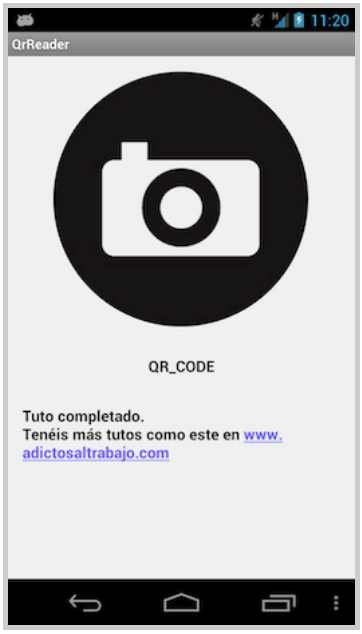
» Jugando con JSON en Java y la librería GSON. Parte 2

» Android Beam

» Como hacer nuestros test más legibles con Hamcrest

» Ejemplo de Viewpager para android

» Uso de las anotaciones @Embeddable, @Embedded, @AttributeOverrides, @AssociationOverrides



4. Detalles a tener en cuenta.

La implementación que hemos hecho, nos obliga a tener instalado **Barcode Scanner** en el dispositivo. Si intentamos ejecutar nuestra aplicación, nos saldrá un **alert** indicando que no lo tenemos instalado y preguntándonos si lo queremos instalar. ¿Que ocurre si tenemos otras aplicaciones que son capaces de leer códigos qr y no queremos instalar otra más?

El comportamiento de la aplicación que acabamos de hacer se debe a la clase que hemos utilizado. Si lo que queremos es que el teléfono nos muestre una lista de las aplicaciones disponibles, tendremos que hacerlo de la siguiente manera.

view plain print ?

```
01. private void configureButtonReader() {
02.     final ImageButton buttonReader = (ImageButton)findViewById(R.id.btReader);
03.     buttonReader.setOnClickListener(new View.OnClickListener() {
04.         @Override
05.         public void onClick(View view) {
06.             try {
07.                 //new IntentIntegrator(QrReaderActity.this).initiateScan();
08.                 Intent intent = new Intent("com.google.zxing.client.android.SCAN");
09.                 intent.putExtra("SCAN_MODE", "QR_CODE_MODE");
10.                 startActivityForResult(intent, 0);
11.             } catch (ActivityNotFoundException exception) {
12.                 Toast.makeText(viewGroup.getContext(), "Error", Toast.LENGTH_SHORT).show();
13.             }
14.         }
15.     });
16. }
```

y

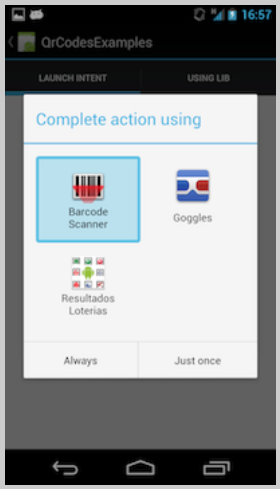
view plain print ?

```
01. @Override
02. public void onActivityResult(int requestCode, int resultCode, Intent data) {
03.     super.onActivityResult(requestCode, resultCode, data);
04.     if((requestCode == 0) && (resultCode == -1)) {
05.         updateUITextViews(data.getStringExtra("SCAN_RESULT"), data.getStringExtra("SCAN_RESULT_FORMA
06.     } else {
07.         Toast.makeText(viewGroup.getContext(), "Error", Toast.LENGTH_SHORT).show();
08.     }
```

```
09. }
```

Lo que hemos hecho es comentar la clase que usábamos para lanzar el intent y hemos creado el nuestro. El problema de esto es que tenemos que gestionar nosotros que ocurre en caso de qué no haya ninguna aplicación disponible y parsear nosotros el intent devuelto por la aplicación.

Os muestro una captura para que veáis lo que mostraría en mi caso



5. Conclusiones.

Si la aplicación que estamos desarrollando, necesita recibir datos del exterior, quizás sea una opción el que los recoja de códigos qr, ya que hemos visto que con muy poco esfuerzo, nuestra aplicación será totalmente capaz de leer estos códigos.

Para cualquier comentario, duda o sugerencia, tenéis el formulario que aparece a continuación.

Un saludo.