Autentificación en servicios remotos

Índice

Los servicios REST estan fuertemente vinculados al protocolo HTTP, por lo que los mecanismos de seguridad utilizados también deberían ser los que define dicho protocolo. Pueden utilizar los diferentes tipos de autentificación definidos en HTTP: *Basic*, *Digest*, y *X.509*. Sin embargo, cuando se trata de servicios que se dejan disponibles para que cualquier desarrollador pueda acceder a ellos, utilizar directamente estos mecanismos básicos de seguridad puede resultar peligroso. En estos casos la autentificación suele realizarse mediante el protocolo OAuth.

1. Seguridad HTTP básica

Vamos a ver en primer lugar cómo acceder a servicios protegidos con seguridad HTTP estándar. En estos casos, deberemos proporcionar en la llamada al servicio las cabeceras de autentificación al servidor, con los credenciales que nos den acceso a las operaciones solicitadas.

Por ejemplo, desde un cliente Android en el que utilicemos la API de red estándar de Java SE deberemos definir un `Authenticator` que proporcione estos datos:

```
Authenticator.setDefault(new Authenticator() {
    protected PasswordAuthentication getPasswordAuthentication() {
        return new PasswordAuthentication (
              "usuario", "password".toCharArray());
    }
});
```

En caso de que utilicemos HttpClient de Apache, se especificará de la siguiente forma:

```
DefaultHttpClient client = new DefaultHttpClient();
client.getCredentialsProvider().setCredentials(
    new AuthScope("jtech.ua.es", 80),
    new UsernamePasswordCredentials("usuario", "password")
);
```

Aquí además de las credenciales, hay que indicar el ámbito al que se aplican (host y puerto).

Para quienes no estén muy familiarizados con la seguridad en HTTP, conviene mencionar el funcionamiento del protocolo a grandes rasgos. Cuando realizamos una petición HTTP a un recurso protegido con seguridad básica, HTTP nos devuelve una respuesta indicándonos que necesitamos autentificarnos para acceder. Es entonces cuando el cliente solicita al usuario las credenciales (usuario y password), y entonces se realiza una nueva petición con dichas credenciales incluidas en una cabecera *Authorization*. Si las credenciales son válidas, el servidor nos dará acceso al contenido solicitado.

Este es el funcionamiento habitual de la autentificación. En el caso del acceso mediante HttpClient que hemos visto anteriormente, el funcionamiento es el mismo, cuando el servidor nos pida autentificarnos la librería lanzará una nueva petición con las credenciales especificadas en el proveedor de credenciales.

Sin embargo, si sabemos de antemano que un recurso va a necesitar autentificación, podemos también autentificarnos de forma preventiva. La autentificación preventiva consiste en mandar las credenciales en la primera petición, antes de que el servidor nos las solicite. Con esto ahorramos una petición, pero podríamos estar mandando las credenciales en casos en los que no resulta necesario.

Con HttpClient podemos activar o desactivar la autentificación preventiva con el siguiente método:

```
client.getParams().setAuthenticationPreemptive(true);
```

En iOS, la autentificación la deberá hacer el delegado de la conexión. Para ello deberemos implementar el método `connection:didReceiveAuthenticationChallenge:`. Este método será invocado cuando el servidor nos pida autentificarnos. En ese caso deberemos crear un objeto `NSURLCredential` a partir de nuestras credenciales (usuario y password).

A continuación vemos un ejemplo típico de implementación:

```
- (void)connection:(NSURLConnection *)connection
    didReceiveAuthenticationChallenge:
        (NSURLAuthenticationChallenge *)challenge
{
    if(challenge.previousFailureCount == 0) {
        NSURLCredential *cred =
            [NSURLCredential credentialWithUser:@"mi_login"
                                       password:@"mi_password"
                                    persistence:NSURLCredentialPersistenceNone];
        [[challenge sender] useCredential: cred
               forAuthenticationChallenge:challenge];
    } else {
        [[challenge sender] cancelAuthenticationChallenge: challenge];
    }
}
```

Podemos observar que comprobamos los fallos previos de autentificación que hemos tenido. Es decir, si con los credenciales que tenemos en el código ha fallado la autentificación, será mejor que cancelemos el acceso, ya que si volvemos a intentar acceder con los mismos credenciales vamos a tener el mismo error. En caso de que sea el primer intento, creamos los credenciales (podemos ver que se puede indicar que se guarden de forma persistente para los próximos accesos), y los utilizamos para responder al reto de autentificación (`NSURLAuthenticationChallenge`).

Este método nos permite autentificarnos siempre que realicemos una conexión asíncrona. Sin embargo, si queremos conectar de forma síncrona, o queremos autentificarnos de forma preventiva, esto resultará bastante más complejo, ya que deberemos añadir las cabeceras de autentificación manualmente a la petición. La cabecera que necesitamos es `Authorization`, a la que le proporcionaremos el tipo de autentificación y los credenciales (login y password). Por ejemplo, para utilizar seguridad de tipo BASIC como en los casos anteriores, esta cabecerá tendrá la siguiente forma:

```
Authorization: Basic login_base64:password_base64
```

La mayor dificultad radica en que el login y el password no se incluyen directamente, sino que deben estar codificados en base64. Necesitaremos por lo tanto una función que se encargue de realizar esta codificación, y no hay ninguna implementada en el SDK de iOS. Esto no es demasiado complejo de implementar, pero si no estamos familiarizados con este formato podemos encontrar diferentes implementaciones. Una de ellas es la implementación de Matt Gallagher:

```
http://cocoawithlove.com/2009/06/base64-encoding-options-on-mac-and.html
```

Esta implementación es bastante elegante, ya que se introduce como una categoría de `NSData`, lo cual añade a esta clase la capacidad de realizar la codificación y descodificación. Con ella podemos implementar la codificación de la siguiente forma:

```
NSString *cred = [NSString stringWithFormat:@"%@:%@", login, password];
NSString *credBase64 = [[cred dataUsingEncoding:NSUTF8StringEncoding]
                                                  base64EncodedString];
NSString *authHeader = [NSString stringWithFormat:@"Basic %@",
                                                 credBase64];

NSMutableURLRequest *theRequest = [NSMutableURLRequest
                                        requestWithURL: url];
[theRequest addValue:authHeader
  forHTTPHeaderField:@"Authorization"];
```

Donde `login` y `password` son variables de tipo `NSString` donde tenemos almacenados nuestros credenciales, y `url` es de tipo `NSURL` y contiene la URL a la que conectar. En el código podemos ver que primero se genera la cadena de credenciales, concatenando `login:password`. A continuación esta cadena se codifica en base64, y con esto ya podemos añadir una cabecera `Authorization` cuyo valor sea la cadena `Basic` *<credenciales_base64>*.

Si en lugar de acceder a estos servicios desde una aplicación nativa, lo hacemos desde Javascript, deberemos asegurarnos de que el usuario se ha autentificado previamente en el sitio web que está haciendo la llamada AJAX al servicio REST seguro.

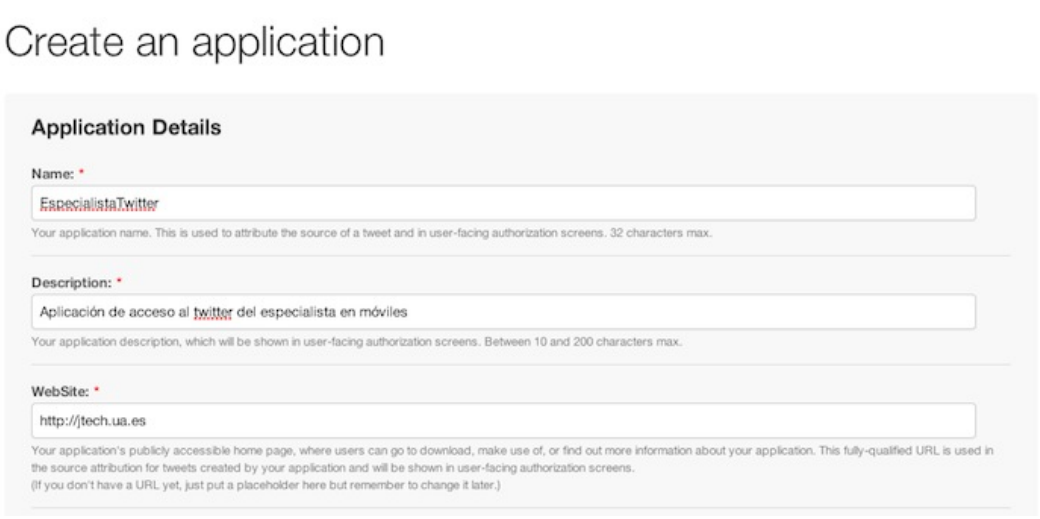
2. Protocolo OAuth

En la anterior sección hemos visto cómo acceder a servicios web que requieren autentificación, proporcionando las credenciales necesarias en la petición HTTP. Esto es apropiado cuando accedemos a nuestros propios servicios, sin embargo, si queremos permitir que terceros puedan acceder a ellos esto puede resultar problemático. Esto se debe a que la aplicación que acceda a nuestro servicio estaría recogiendo el login y el password del usuario, y enviándolas a nuestro servicio. Cuando el servicio responda la aplicación sabrá si las credenciales eran válidas o no. Es decir, dicha aplicación contará con las credenciales validadas de un usuario, y podría tratar dicha información de forma inadecuada o fraudulenta.

Por lo tanto, deberíamos evitar que las credenciales del usuario pasen por una aplicación que no controlemos nosotros. Para ello contamos con el protocolo *OAuth* (*Open Authorization*), que se encarga de autentificar usuarios en aplicaciones de terceros de forma segura. Este protocolo lo encontraremos en gran cantidad de servicios, como Twitter o Facebook. Vamos a ver a continuación el funcionamiento de este protocolo y algún ejemplo de utilización.

Supongamos que queremos que nuestra aplicación acceda a Twitter en nombre del usuario que esté utilizándola. Este es el típico escenario en el que se utiliza *OAuth*. En este caso nuestra aplicación se comportará como cliente de Twitter, y deberá tener unos credenciales que la identifiquen como tal. Para ello deberemos dar de alta nuestra aplicación en Twitter mediante el siguiente formulario:

```
https://dev.twitter.com/apps/new
```

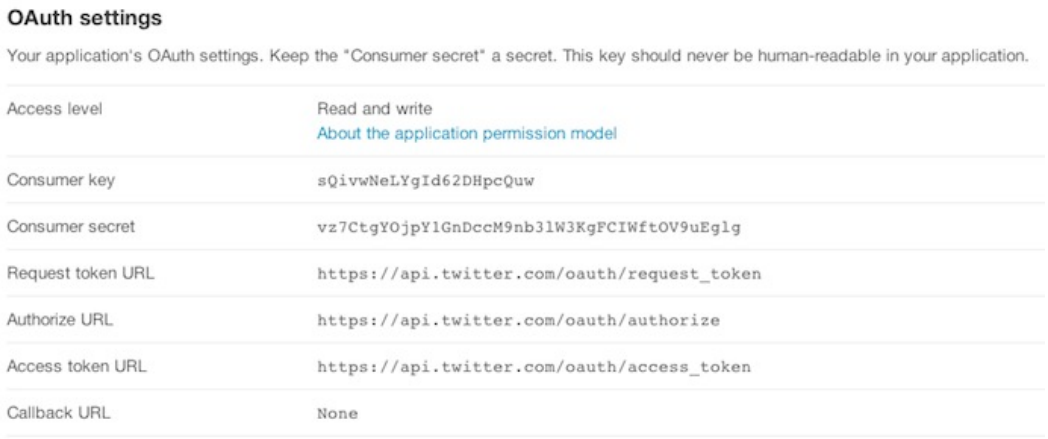


Alta como consumidor Twitter

Importante

Para poder acceder al servicio mediante OAuth de la forma que vamos a ver a continuación, será imprescindible haber rellenado en este formulario el campo *Callback URL*. Será suficiente con poner cualquier URL válida (puede coincidir con *WebSite*).

Tras registrarnos, nos da una serie de datos para nuestra aplicación:



OAuth settings

Your application's OAuth settings. Keep the "Consumer secret" a secret. This key should never be human-readable in your application.

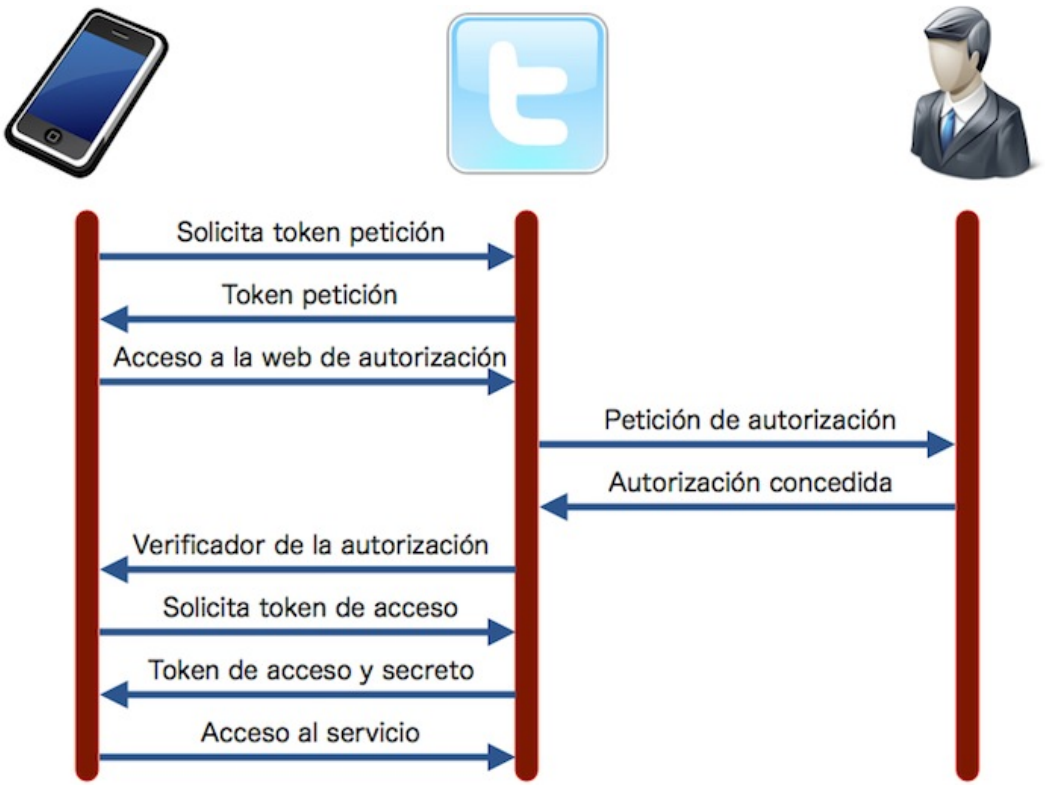
Access level	Read and write About the application permission model
Consumer key	`sQivwNeLYgId62DHpcQuw`
Consumer secret	`vz7CtgYOjpYlGnDccM9nb3lW3KgFCIWftOV9uEglg`
Request token URL	`https://api.twitter.com/oauth/request_token`
Authorize URL	`https://api.twitter.com/oauth/authorize`
Access token URL	`https://api.twitter.com/oauth/access_token`
Callback URL	`None`

Datos para el consumidor OAuth

Cuando una aplicación accede en nombre de un usuario a un servicio mediante *OAuth*, no sólo el usuario final debe autentificarse en el servicio, sino que la aplicación también debe hacerlo. Para esto se nos proporcionan unas credenciales al registrar nuestra aplicación en el servicio al que vamos a acceder (en el ejemplo anterior Twitter), conocidas como `oauth_consumer_key` y `oauth_consumer_secret`. Estas credenciales identifican a nuestra aplicación en el servicio, por lo que éste último tendrá un control sobre las aplicaciones que están accediendo en nombre de usuarios, y ninguna de ellas podrá acceder sin haberse registrado antes. Cuando accedamos al servicio desde nuestra aplicación, deberemos incluir estas credenciales en la petición para que el servicio conozca la identidad de la aplicación que está accediendo. Podemos pensar en `oauth_consumer_key` como el nombre de usuario único con el que nuestra aplicación accede al servicio, y en `oauth_consumer_secret` como el password asociado a dicho usuario.

Una vez contamos con estos datos, vamos a ver cómo se realizaría el proceso de autentificación. Vemos que el servicio nos proporciona diferentes URLs para realizar este proceso:

URL	Descripción
Request token URL	Solicita un token de petición acceso para nuestra aplicación, dadas nuestras credenciales.
Authorize URL	Solicita al usuario que autorice el acceso al servicio a través de nuestra aplicación. Se proporciona el token generado por la URL anterior, y si el usuario autoriza la petición, nos devuelve un código de autorización.
Access token URL	Genera el token que da acceso al servicio a partir del código de autorización obtenido mediante la URL anterior.



Ejemplo de escenario del protocolo OAuth

Vamos a ver paso a paso cómo se realiza este proceso:

1. Lo primero que debemos hacer es solicitar un token de petición de acceso para nuestra aplicación accediendo a *Request token URL*. A dicha URL le pasaremos nuestras credenciales, incluyendo `oauth_consumer_key` como parámetro de la petición, y firmando la petición entera mediante `oauth_consumer_secret` (este código se utilizará siempre para firmar las peticiones, no se enviará directamente ya que debe ser secreto). Como respuesta nos dará un token de petición (`oauth_token`) y una clave secreta asociada (`oauth_token_secret`):

   ```
   oauth_token=ab3cd9j4ks73hf7g
   oauth_token_secret=ZXhhbXBsZS5jb20
   ```

   Estos datos no se pueden utilizar directamente para acceder al servicio, antes deberemos pedirle autorización al usuario para que nos permita acceder en su nombre. Deberemos convertir el token de petición actual (no autorizado), en un token de petición autorizado por el usuario. Esto lo haremos en el siguiente paso.
2. Una vez tengamos el token de petición de acceso, deberemos solicitar que el usuario autorice que nuestra aplicación pueda acceder al servicio, pero esto no podemos hacerlo directamente desde nuestra aplicación, sino que el usuario deberá darnos la autorización a través del servicio final al que accederemos. Para ello tenemos la *Authorize URL*. Deberemos acceder a dicha URL en un navegador, proporcionando el token obtenido en el paso anterior. Al hacer esto, nos redirigirá a la página de login

del servicio (por ejemplo la página de login de Twitter), y le pedirá al usuario introducir sus credenciales en el servicio y le preguntará si desea autorizarnos a acceder en su nombre. Todo esto se estará haciendo fuera de nuestra aplicación. En caso de una aplicación web nos habrá redirigido a la URL de un sitio web distinto (por ejemplo `www.twitter.com`), pero en una aplicación móvil nativa tendremos que abrir un navegador. La cuestión es, ¿cómo volvemos a nuestra aplicación tras recibir la autorización del usuario? Para que esto sea posible, en la petición podemos proporcionar también una URL de *callback* como parámetro. Esta es la URL a la que el servicio al que accedemos nos llevará tras autorizarnos el usuario, pasándonos como parámetro de la query un token de autorización (`oauth_token`) y un verificador (`oauth_token_verifier`). Pero en el caso de los móviles es más complejo, ya que tenemos una aplicación nativa, no un navegador, y queremos que el token sea devuelto a nuestra aplicación, y no que el navegador nos redirija a otra página. Podemos conseguir esto poniendo como esquema de la URL de *callback* un esquema propio creado por nosotros, y haciendo que nuestra actividad "escuche" la peticiones a dicho tipo de URLs, como por ejemplo podría ser `oauth-jtech-twitter://callback`:

```
<activity android:name=".AutentificacionOAuthActivity"
          android:launchMode="singleTask">
  <intent-filter>
    <action android:name="android.intent.action.VIEW" />
    <category android:name="android.intent.category.DEFAULT" />
    <category android:name="android.intent.category.BROWSABLE" />
    <data android:scheme="oauth-jtech-twitter" android:host="callback" />
  </intent-filter>
</activity>
```

Cuando se produzca un nuevo *intent* en `AutentificacionOAuthActivity` (`onNewIntent`), comprobaremos si la URL de la que viene es la del *callback*, y de ser así obtendremos de los parámetros de la query los datos `oauth_token` y `oauth_token_verifier`.

3. El paso final consiste en, dado el token autorizado de petición de acceso obtenido en el paso anterior, solicitar un token de acceso válido para que nuestra aplicación pueda acceder al servicio en nombre del usuario. Para ello se utiliza la URL *Access token URL*. Recibirá como parámetros de entrada nuestra clave de autentificación (`oauth_consumer_key`), y el token de petición autorizado en el paso anterior junto a su verificador (`oauth_token` y `oauth_token_verifier`). Como resultado obtendremos el token de acceso (`oauth_token`) y su clave secreta asociada (`oauth_token_secret`). Con estos datos ya podremos acceder a las operaciones del servicio.
4. Para acceder al servicio deberemos proporcionar en los mensajes de petición una cabecera `Authorization` con los tokens obtenidos anteriormente: las credenciales de nuestra aplicación obtenidas en el registro (`oauth_consumer_key`), y el token de acceso obtenido en el paso anterior (`oauth_token`). Además, dicha petición deberá estar firmada utilizando como clave de firmado una combinación de las claves secretas asociadas a los tokens anteriores: `oauth_consumer_secret` y `oauth_token_secret`.

Importante

La clave secreta no debe compartirse con nadie. Si pudiésemos pensar que ha sido comprometida, deberemos invalidarla y generar una nueva.

Encontramos diferentes librerías para trabajar con *OAuth* tanto en Android como en iOS. En Android podemos encontrar por ejemplo *Signpost* (http://code.google.com/p/oauth-signpost/) o *Scribe* (http://github.com/fernandezpablo85/scribe-java). Para iOS encontramos librerías como *RSOAuth* (https://github.com/rsieiro/RSOAuthEngine) o *MPOAuth* (https://github.com/thekarladam/MPOAuth).

A continuación veremos en un ejemplo cómo utilizar *Signpost* para acceder a Twitter desde una aplicación Android.

3. Acceso a servicios de terceros

Un ejemplo práctico de acceso a servicios proporcionados por terceros es Twitter. Anteriormente hemos visto cómo acceder a sus servicios para obtener el *public timeline*. Sin embargo, nos ofrece muchas más operaciones, y muchas de ellas requieren que estemos autentificados como usuarios, como por ejemplo publicar nuevos mensajes.

La autentificación se realiza utilizando *OAuth*, por lo que veremos un ejemplo práctico de uso de este protocolo.

Podemos recurrir a la documentación de la API REST publicada por diferentes servicios e implementar las comunicaciones de forma manual como hemos visto anteriormente.

Servicio	URL desarrolladores
Twitter	https://dev.twitter.com//
LinkedIn	https://developer.linkedin.com/rest
Facebook	https://developers.facebook.com/
Dropbox	https://www.dropbox.com/developers

En algunas de estas páginas podemos encontrar también SDKs para las distintas plataformas móviles, como ocurre por ejemplo en el caso de Facebook o Dropbox. Si no contamos con un SDK oficial, podemos implementar nosotros el acceso a los servicios REST o bien buscar una librería proporcionada por terceros.

Tanto en Android como en iOS hay diferentes librerías que nos facilitan el acceso a los servicios de Twitter y de otras redes. Por ejemplo, en iOS contamos con **ShareKit** (http://getsharekit.com/) que nos permite acceder a diferentes servicios, como Facebook, Twitter o Google Reader. De forma similar, en Android encontramos **AndroidLibs** (http://androidlibs.com/sociallib.html), que soporta Facebook,

Twitter, LinkedIn y Google Buzz. También encontramos librerías específicas para servicios concretos, como **JTwitter** (http://www.winterwell.com/software/jtwitter.php) o **Twitter4J** (http://twitter4j.org/en/index.jsp).

A modo de ejemplo, vamos a ver a continuación cómo podemos implementar el acceso a Twitter desde una aplicación Android utilizando **Signpost** y **Twitter4J**. Tras esto, veremos la forma de acceder a Twitter desde iOS. En este caso será más sencillo, ya que desde iOS 5.0 el acceso a Twitter se encuentra integrado en el SDK, por lo que no será necesario incluir ninguna librería adicional.

3.1. Acceso a Twitter desde Android

El primer paso para hacer que nuestra aplicación pueda acceder a Twitter será darla de alta en este servicio, tal como hemos visto anteriormente, para así poder realizar la autentificación mediante *OAuth*. Una vez hecho esto tendremos nuestra clave de consumidor y las URLs de acceso. Podemos introducir esta información como constantes en nuestra actividad (vamos a suponer que estamos creando una actividad cuyo nombre es `TwitterActivity`):

```
private final static String OAUTH_CONSUMER_KEY = "sQivwNeLYgId62DHpcQuw";
private final static String OAUTH_CONSUMER_SECRET =
                        "vz7CtgYOjpY1GnDccM9nb3lW3KgFCIWftOV9uEglg";

private final static String REQUEST_TOKEN_URL =
                        "https://api.twitter.com/oauth/request_token";
private final static String AUTHORIZE_URL =
                        "https://api.twitter.com/oauth/authorize";
private final static String ACCESS_TOKEN_URL =
                        "https://api.twitter.com/oauth/access_token";
```

Primer paso. Para obtener el token de acceso *OAuth*, lo primero que deberemos hacer es obtener el token de petición. Crearemos un objeto de tipo `OAuthConsumer` y otro de tipo `OAuthProvider`. El primero de ellos representa al consumidor (nuestra aplicación), y se crea a partir de nuestro credenciales (clave y secreto del consumidor). El segundo representa el proveedor (Twitter en este caso), y se crea a partir de las URLs para la autentificación mediante *OAuth*.

```
OAuthConsumer consumer;
OAuthProvider provider;

...

consumer = new CommonsHttpOAuthConsumer(OAUTH_CONSUMER_KEY,
                                        OAUTH_CONSUMER_SECRET);
provider = new CommonsHttpOAuthProvider(REQUEST_TOKEN_URL,
                                        ACCESS_TOKEN_URL,
                                        AUTHORIZE_URL);

String url = provider.retrieveRequestToken(consumer, CALLBACK_URL);
```

Una vez creados el consumidor y proveedor, solicitamos el token de petición *OAuth*. Para ello llamamos sobre el proveedor a `retrieveRequestToken`, proporcionando la

información sobre el consumidor y la URL de *callback* que utilizaremos. Nos devuelve una URL que llevará ya incluido como parámetro el token de petición. Es decir, al llamar a este método se estará conectando a la *Request token URL*, y una vez obtenga el token de petición, creará la URL de autorización adjuntando dicho token como parámetro. Esta URL tendrá la siguiente forma:

```
https://api.twitter.com/oauth/authorize?
    oauth_token=vF7W9KKYwXgOYjFhpoXyRN66d89AxSXxLL9W0hBl8
```

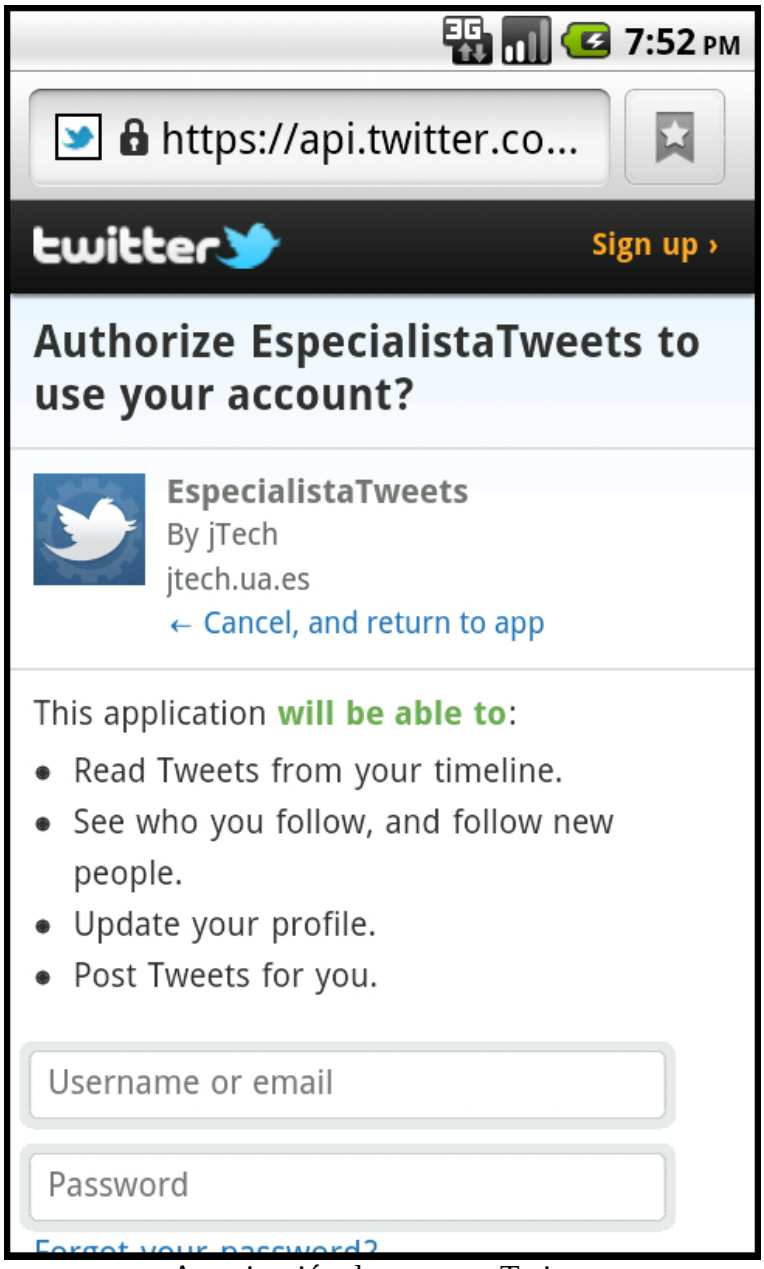
Segundo paso. A continuación debemos solicitar autorización para nuestro token de acceso. La autorización nos la debe dar el usuario conectando directamente a través de Twitter. Es decir, tendremos que abrir un navegador y acceder en él a la URL anterior:

```
Intent intent = new Intent(Intent.ACTION_VIEW, Uri.parse(url));
```

Recomendación

Es recomendable añadir algunos *flags* al `intent` anterior, por ejemplo para evitar que se guarde en la historia, ya que no querremos que al volver atrás vuelva a aparecer la pantalla de login.

Veremos una página como la siguiente:



Autorización de acceso a Twitter

Cuando introduzcamos los datos y pulsemos el botón *Authorize app*, si los credenciales son correctos nos llevará a la URL de *callback*, a la que se le pasará como parámetro la clave autorizada. Aquí es donde viene la parte más compleja, ya que queremos que la URL de *callback* no nos lleve a una página en el navegador, sino que nos envíe de vuelta a la aplicación Android. Para ello podemos hacer que nuestra actividad tenga un filtro que haga que se invoque al visualizar un determinado tipo de URL, tal como hemos visto anteriormente. Podemos declararla de la siguiente forma en el `AndroidManifest.xml`:

```
<activity android:name=".TwitterActivity"
          android:label="Acceso a Twitter"
          android:launchMode="singleTask">
  <intent-filter>
    <action android:name="android.intent.action.VIEW" />
    <category android:name="android.intent.category.DEFAULT" />
    <category android:name="android.intent.category.BROWSABLE" />
    <data android:scheme="oauth-jtech-twitter" android:host="callback" />
  </intent-filter>
</activity>
```

Podemos ver que la actividad la hemos declarado como `singleTask`, ya que queremos que cuando se acceda a la URL de *callback*, se vuelva a la actividad que ya teníamos abierta, y no se cree una nueva en la pila. En la actividad podemos especificar también la dirección de *callback* como constante:

```
private final static String CALLBACK_SCHEME = "oauth-jtech-twitter";
private final static String CALLBACK_URL = CALLBACK_SCHEME +
                                           "://callback";
```

Hemos de recordar que esta dirección de *callback* se ha utilizado en el paso anterior al llamar a `retrieveRequestToken` (es decir, no se utiliza la dirección de *callback* por defecto que se indicó al dar de alta nuestra aplicación en Twitter). Ahora cuando confirmemos nuestro datos en la pantalla del navegador, y autoricemos a la aplicación, al cargar la URL de *callback* nos llevará se nuevo a la actividad `TwitterActivity` y se ejecutará el método `onNewIntent` de la misma. En este método deberemos comprobar si la `Uri` del `intent` que nos ha llegado es la de *callback*, y en tal caso continuaremos con la autentificación:

```
protected void onNewIntent(Intent intent) {
  super.onNewIntent(intent);

  Uri uri = intent.getData();
  if (uri != null && uri.getScheme().equals(CALLBACK_SCHEME)) {
    // Solicitar token de acceso
    ...
  }
}
```

La `Uri` de *callback* obtenida tendrá la siguiente forma:

```
oauth-jtech-twitter://callback?
  oauth_token=vF7W9KKYwXgOYjFhpoXyRN66d89AxSXxLL9W0hBl8&
  oauth_verifier=2pXSOjjwJ48Ues51iQm2K5PDTI6WS8xkWJzcgABoQ
```

Tercer paso. Para finalizar la autentificación deberemos solicitar el token de acceso. Para ello extraeremos de la URL anterior el verificador *OAuth*, y solicitaremos al proveedor el token de acceso proporcionándole como entrada dicho verificador de la autorización del usuario:

```
String oauthVerifier = uri.getQueryParameter(OAuth.OAUTH_VERIFIER);
provider.retrieveAccessToken(consumer, oauthVerifier);

String oauthToken = consumer.getToken();
String oauthTokenSecret = consumer.getTokenSecret();
```

El token de acceso se solicita llamando al método `retrieveAccessToken` del proveedor, pasándole como parámetro la información del consumidor y el verificador obtenido en el paso anterior. Eso llamará a la *Access token URL*, y nos devolverá el token de acceso autorizado junto a su secreto asociado. Ahora ya podremos acceder a Twitter utilizando estos datos. La información de autentificación que deberemos proporcionar a Twitter es:

- Clave del consumidor y secreto asociado
- Token de acceso y secreto asociado

Esta información se añadirá al cliente de Twitter4J de la siguiente forma:

```
Twitter twitter = TwitterFactory.getSingleton();

twitter.setOAuthConsumer(OAUTH_CONSUMER_KEY, OAUTH_CONSUMER_SECRET);

AccessToken token = new AccessToken(oauthToken, oauthTokenSecret);
twitter.setOAuthAccessToken(token);
```

Esto sólo es necesario hacerlo una vez, a partir de este momento podremos utilizar el *singleton* `Twitter` todas las veces que queramos sin necesidad de volver a especificar la información de acceso. Es importante destacar que el token de acceso no tiene caducidad, por lo que será recomendable guardarlo en las *shared preferences* para así tenerlo disponible para futuras ejecuciones de la aplicación, sin tener que volver a solicitarlo. Sólo dejará de funcionar si el usuario revoca el acceso a Twitter para nuestra aplicación.

Ahora por ejemplo podremos publicar un nuevo *tweet* de la siguiente forma:

```
Twitter twitter = TwitterFactory.getSingleton();
twitter.updateStatus(mensaje);
```

Importante

Debemos tener en cuenta que los métodos `retrieveRequestToken` y `retrieveAccessToken` establecen una conexión de red para obtener los tokens de petición y acceso respectivamente. Por lo tanto, deberemos llamar a estos métodos siempre desde hilos en segundo plano (podemos utilizar una `AsyncTask` como hemos visto anteriormente). Lo mismo se aplica a las llamadas a operaciones de Twitter, como es el caso de `updateStatus`.

3.2. Acceso a Twitter desde iOS

Como hemos visto, en iOS también encontramos diferentes librerías para acceder a Twitter. Sin embargo, a partir de iOS 5.0 el acceso a Twitter se incorpora al SDK, por lo que no será necesario introducir ninguna librería adicional, y simplificará enormemente el acceso a este servicio, ya que el propio sistema operativo se ocupará de la gestión de las cuentas de usuario para acceder a Twitter, lo cual evitará que tengamos que preocuparnos nosotros de implementar el acceso mediante *OAuth*.

Con la API de iOS 5.0 SDK tenemos dos formas de utilizar Twitter: utilizar un formulario predefinido para componer *tweets* (`TWTweetComposeViewController`), o

enviar los *tweets* directamente desde el código (`TWRequest`). Las dos clases anteriores pertenecen el *Twitter framework*, que deberemos incluir en nuestra aplicación para acceder a Twitter utilizando esta API.

La forma más sencilla de integrar Twitter en nuestra aplicación es utilizar el formulario predefinido. Hacer esto es tan sencillo como añadir las siguientes líneas:

```
TWTweetComposeViewController *controller =
    [[TWTweetComposeViewController alloc] init];
[controller setInitialText: @"Texto por defecto"];
[self presentModalViewController:controller animated: YES];
```

Como podemos ver, podemos añadir un texto inicial por defecto, pero el usuario podrá modificarlo y no se nos permitirá alterar lo que haya introducido finalmente el usuario. Es decir, el usuario tendrá siempre la última palabra sobre lo que se va a publicar en Twitter.

Si necesitamos conocer cuándo se ha enviado el *tweet* o cuando se ha cancelado, podemos añadir un *completion handler* de la siguiente forma:

```
TWTweetComposeViewController *controller =
    [[TWTweetComposeViewController alloc] init];
[controller setInitialText: @"Texto por defecto"];
[controller setCompletionHandler:
  ^(TWTweetComposeViewControllerResult result) {
    switch(result) {
        case TWTweetComposeViewControllerResultDone:
            NSLog(@"Tweet enviado");
            break;
        case TWTweetComposeViewControllerResultCancelled:
            NSLog(@"Tweet cancelado");
            break;
    }
    [self dismissModalViewControllerAnimated:YES];
  }];
[self presentModalViewController:controller animated: YES];
```

Si queremos enviar un *tweet* desde el código de la aplicación, sin necesitar la interacción del usuario, deberemos utilizar la clase `TWRequest`. Esta clase nos permitirá acceder a toda la API REST de forma sencilla, no estando limitada únicamente al envío de *tweets* como en el caso simplificado anterior.

Antes de acceder, necesitaremos seleccionar la cuenta de usuario que utilizaremos. Para ello necesitaremos añadir a nuestro proyecto el *framework Accounts*. Las cuentas de usuario de Twitter se gestionan a nivel de sistema operativo, así que lo único que tendremos que hacer es seleccionar una de las disponibles. Para ello podemos acceder al almacen de cuentas (`ACAccountStore`), y solicitar la lista de cuentas de tipo Twitter (`ACAccountTypeIdentifierTwitter`). Puede que haya más de una cuenta, en el siguiente ejemplo por simplicidad cogemos siempre la primera de la lista, pero podríamos por ejemplo presentar una interfaz al usuario que le permita seleccionar la cuenta que desea utilizar.

```
ACAccountStore *accountStore = [[ACAccountStore alloc] init];
ACAccountType *accountType = [accountStore
    accountTypeWithAccountTypeIdentifier:ACAccountTypeIdentifierTwitter];
[accountStore requestAccessToAccountsWithType:accountType
```

```
                withCompletionHandler:^(BOOL granted, NSError *error) {
    if(granted) {
        NSArray *accountsArray =
            [accountStore accountsWithAccountType:accountType];

        if ([accountsArray count] > 0) {
            ACAccount *twitterAccount = [accountsArray objectAtIndex:0];

            ...
        }
    }
}];
```

Una vez obtenida la cuenta a utilizar (`ACAccount`), ya podemos acceder a Twitter mediante `TWRequest`. Este objeto se inicializa a partir de la URL de los servicios de Twitter a la que queramos acceder, los parámetros que queramos enviar, y el método HTTP a utilizar (consultar la documentación de los servicios REST de Twitter, https://dev.twitter.com//, para ver la lista completa de operaciones, junto con su URL, lista de parámetros admitidos, y método HTTP a utilizar). Tras inicializarlo, deberemos establecer la cuenta de Twitter a utilizar. A continuación vemos la forma de enviar un nuevo *tweet*:

```
TWRequest *postRequest = [[TWRequest alloc]
    initWithURL:
        [NSURL URLWithString:
            @"http://api.twitter.com/1/statuses/update.json"]
    parameters:
        [NSDictionary dictionaryWithObject:@"Tweet predefinido"
                                    forKey:@"status"]
    requestMethod:TWRequestMethodPOST];
[postRequest setAccount:twitterAccount];
```

Una vez tenemos este objeto, podemos enviar la petición HTTP al servicio firmada correctamente con nuestro token de acceso *OAuth* llamando al siguiente método:

```
[postRequest performRequestWithHandler: ^(NSData *responseData,
                    NSHTTPURLResponse *urlResponse, NSError *error) {
    NSLog(@"Tweet enviado");
}];
```

Tenemos que proporcionar también un *handler* al que se le notificará el resultado del envío.

Hemos visto que en este caso no nos ha hecho falta utilizar nuestra clave de consumidor, ya que la autentificación se está realizando a nivel de sistema operativo. Es decir, el consumidor en este caso es el propio sistema iOS, que incorporará su propia clave de consumidor con la que se accede a Twitter. El usuario configurará su cuenta en los ajustes del teléfono, y desde el código lo único que deberemos hacer es seleccionar una de las cuentas disponibles.