QML OAuth

This article describes a simple OAuth authentication module for QML. OAuth (Open Authorization) is an open standard for authorization that allows users to share their private resources stored on one site with another site, without having to give the first site the username and password of the site where their data is stored. It is widely used by popular social media sites including Facebook, Twitter and LinkedIn to share resources like photos, music and contact lists.

03 Apr 2011

This example has been tested on the http://developer.sna.pr/oauth API.

How does OAuth work?

There are some good explanations of the protocol in the Beginner's Guide to OAuth – Part II : Protocol Workflow and on oauth.net.

As a simple example, consider a photo editing site that wants to allow its users to edit photos stored on their favourite photo sharing service. OAuth provides a mechanism where the user is redirected to the *photo sharing* site in order to enter their credentials and approve limited access to the photo editing site. The user is then redirected back to the photo editing site along with a token which provides it the access it needs.

The WebView

Therefore, we can simply use the `WebView` component to display the authentication page (url property), and intercept the token sent with the `onUrlChanged` signal. Like this:

```
WebView {
        id: loginView
        url: "http://sna.pr/ext/oauth/authorize/?redirect_uri=http://sna.pr&client_id=YOUR_CLIENT_ID&response_type=code"
        onUrlChanged: OAuth.urlChanged(url)
}
```

and set the redirect url to wherever, since we intercept it and are only interested in the parameters.

Intercept the URL redirection

This means that the web view will be displayed, will ask the user to authenticate, and then to allow our application to be connected. If the user allows it, the view is redirected to a URL with a "code" parameter (and the `urlChanged` method called). It looks like this:

```
function urlChanged(url) {
    var authorized = false;
    var mUrl = url.toString();
    var code = "";
    if (mUrl.indexOf("http://sna.pr") > -1) {
        var query = mUrl.substring(mUrl.indexOf('?') + 1);
        var vars = query.split("&");
        for (var i=0;i<vars.length;i++) {
            var pair = vars[i].split("=");
            if (pair[0] == "code") {
                authorized = true;
                code = pair[1];
            }
        }
    }
    if (authorized) {
        requestPermanentToken(code);
    }
}
```

This simply parses the url to get the return code.

Request a permanent Token

This code is then used to request a permanent token, the result being a json string containing the `access_token`:

```
function requestPermanentToken(code) {
        var req = new XMLHttpRequest();
        req.onreadystatechange = function() {
            if (req.readyState == 4) {
                var result = eval('(' + req.responseText + ')');
                if (result.error == null) {
                    token = result.access_token
                }
            }
        }
        req.open("POST", "https://sna.pr/ext/oauth/access_token/", true);
        req.send("grant_type=authorization_code"+
                 "&client_id=YOUR_CLIENT_ID"+
                 "&client_secret=YOUR_CLIENT_SECRET"+
                 "&code=" + code + "&redirect_uri=http://sna.pr");
}
```

Nothing complex here, simply calling the right service to get the permanent token.

(C) Copyright Nokia 2013. All rights reserved.

Save it

Then, of course, it is easy to save it after parsing the json result object :

```
var db = openDatabaseSync("Token", "1.0", "the token", 1);
var dataStr = "INSERT INTO Token VALUES(?)";
var data = [result.access_token];
db.transaction(function(tx) {
    tx.executeSql('CREATE TABLE IF NOT EXISTS Token(token TEXT)');
    tx.executeSql(dataStr, data);
});
```

Add a little function to retrieve the saved token:

```
function checkToken() {
    var db = openDatabaseSync("Token", "1.0", "the token", 1);
    var dataStr = "SELECT * FROM Token";
    db.transaction(function(tx) {
        tx.executeSql('CREATE TABLE IF NOT EXISTS Token(token TEXT)');
        var rs = tx.executeSql(dataStr);
        if (rs.rows.item(0).token) {
            snaprOAuth.token = rs.rows.item(0).token
            snaprOAuth.state = "AuthDone"
        } else {
            snaprOAuth.state = "Login"
        }
    });
}
```

Put it all together

And finally an item to contain states and the token property, plus putting all the javascript in a **oauth.js** file:

```
import QtQuick 1.0
import QtWebKit 1.0
import Qt 4.7
import "oauth.js" as OAuth

Item {
    id: snaprOAuth
    property string nextState: "AuthDone"
    anchors.fill: parent
    Component.onCompleted: OAuth.checkToken()
    property string token: ""
    WebView {
        id: loginView
        visible: false
        anchors.fill: parent
        onUrlChanged: OAuth.urlChanged(url)
    }

    states: [
        State {
            name: "Login"
            PropertyChanges {
                target: loginView
                visible: true
                url: "http://sna.pr/ext/oauth/authorize/"+
                     "?redirect_uri=http://sna.pr&client_id=YOUR_CLIENT_ID&response_type=code"
            }
        },
        State {
            name: "AuthDone"
            PropertyChanges {
                target: loginView
                visible: false
                opacity: 0
            }
            PropertyChanges {
                target: parent
                state: nextState
            }
        }
    ]
}
```

And there you have it.

The `nextState` property is used to notify the parent that includes this. This script first checks if the token exists, if not if fires the web view with the authentication page, and later notifies the parent when it's all done. I find this easier than some other solutions involving a local HTTP server.

(C) Copyright Nokia 2013. All rights reserved.

(C) Copyright Nokia 2013. All rights reserved.