Script.

**Introduction**

For assignment 2 in this module, I decided to develop an application demonstrating an advanced technique using a web development framework and evaluate how well I feel that framework fitted to the needs of the assignment.

The web framework that I have decided to use is Phalcon. Phalcon is an open source, full stack framework for PHP 5 written as a C-extension, with its functionality exposed as PHP classes.

The application that I decided to develop demonstrates an Authentication and Authorization standards when accessing a web service. This standard is known as the OAuth2 protocol, which replaced the original OAuth protocol in 2006.

OAuth2 is a protocol that lets external applications request authorization to private details of a user without getting their password by requesting access in the form of an access token. This is preferred over Basic Authentication because tokens have what is known as ‘scope’. Scope means that tokens are limited to specific pieces of data, which can be revoked by users at any time.

**Going through the Application**

In this application, I have developed 3 small websites along with a simple web service to demonstrate how OAuth2 is used. The websites include a simple client application, the authentication server and finally a user account backend to control access to authorized applications. I should also say at this time that all of the visual websites have used twitter bootstrap for the styling.

So to demonstrate the application flow that is used in order to request access to private data we start on the client applications website which as you can see just contains a simple button. When we press this button the user is then redirected to the authentication server at ‘/oauth/authorize’ with a query string with the initial details regarding this request. The details that have been send are the client id, which is the unique identifier for the client requesting access, scope, which is the data that the client is requesting, a state variable, which is an un-guessable random string which is used to protect against cross-site request forgery attacks, and finally a redirect URI, which is the website that we should redirect to after authentication is completed i.e the user has accepted or declined access.

I should say at this point that in a live system a client will have to sign up as a developer in order to gain a client id and secret in order to make requests and during this process they will have to provide a default redirect URI which will be used if one is not supplied or the one supplied is not valid. The supplied redirect URI in the initial request has to have the same hostname as the default one or the default one will be used. This is to prevent callbacks being made to another website.

If all of the provided information is valid this log in form will be displayed, otherwise the server will send a 403 Forbidden status.

At this point the user will log in with his normal credentials, but this is done not done on the client server meaning that the client never has access to the users log in credentials.

After the user successfully logs in, the authentication will display all of the permissions that the client is asking for and we can accept a certain set of permissions or decline the request outright.

If we decline the request, an error message is passed back to the client and this has to be deal with accordingly, in this case it just redirects back to the initial page.

If the user accepts the request however, a temporary code is passed back to the client with the original state string. The code can be then exchanged for an access token with the accepted scopes attached to it. The access token is then used in any web service requests to get information on the user that authorized us access, in this case it calls the ‘/user’ api which returns the users profile information.

I have also developed a simple backend for a user which allows users to alter permissions given to client applications. This initial page uses the same login information to log in to the backend.

This then takes us to the user’s backend and at the bottom of the page we can see the applications that this user has given access to. We can then alter the permissions this client has or revoke permissions entirely, meaning they will have to re-request for permission if they try to access this users profile using the access token gained in a previous authentication request.

**Talk about the code behind the application.**

The code behind the application is again separated into 4 different projects, I chose to take this route as in a live application generally these 4 websites would be hosted on four different servers.

As you can see, the example website is a full Phalcon website in which each route is in the fashion controllername/actionname. There are two actions in the index controller, the root page or index/index which as you can see checks to see if the client has already gained a token from a previous request. If so we go straight to the callback action. As we saw on the view for this page, it is just a simple button which redirects and this section checks if that form has been posted as builds up the initial request to send to the authorize url to the server. The callback action in this case performs a post request to exchange the retrieved temp code for a persistent access token and this is then used to call the user api using the token.

In the auth server, we took a slightly different approach and this is known as a ‘Mirco’ application which Phalcon designed to be a lightweight website which handles a small number of routes which is perfect for the authentication server as it only has three routes.

The first route is the authorize route where the client redirects to, to allow the user to accept or decline the permissions. Firstly we have to check that all of the input is valid and build a request object from this. Then we determine if the client already has a token defined for this user after they have logged in and then push the view to the client.

The background route handles an ajax request in order to log the user in and also accept or decline the permissions, this is based on the type parameter in the route. The reason I chose to handle this by ajax as ajax is a same-server request, meaning the login request couldn’t be redirected to another website. In the login type, it just checks the users credentials are correct or not and then displays the permissions requested page. And the authorize type checks to see if the user accepted or declined the request and returns the request to the redirect uri accordingly.

The access\_token route essentially handles exchanging a temp code for an access token, it will first check to see if this client had a previous token and just add the new permissions to it or generate a new token and then just return it to the client.

The api website is also a micro-site as it only has two routes which are the two dummy functions that this api provides which are user and messages.

Users juat returns the users profile information and messages just returns any messages that this user has left.

Both of these routes need to have an access token provided and check which user/client pair that token is for and if that token has the right permissions to gain the data it is requesting. If it has the data is returned or if not a 403 forbidden status is sent.

**Evalution of the web development framework.**

As stated above, Phalcon takes a MVC based approach to web development, with a mass of built in classes that can be injected into the application at any time using dependency management injection. This is shown here when we set the default database connection to use for our models. This means that all of the models can gain access to the default db service automatically and can perform operations on the database. A model maps to a database table at a higher level of abstraction and without performing raw SQL statements on the database, this means that we are protected against SQL injection attacks. Another feature that Phalcon models possess are events that are triggered during database operations which can halt the operation if necessary, some of which are used in the Token model to perform final validation checks on the data before the database operation is completed. This also makes operations to the database extremely reliable.

The routing and request objects also made it easy to not only send and receive requests, but also to carry out some advanced operations such as add custom headers and set the status code of the request or apply filters to post or get parameters in order to strip out html or unwanted characters s shown here. This meant that I could very easily access what data was valid from requests and handle it appropriately.

The biggest advantage of using Phalcon over other web development frameworks was its dependency injection mechanism as I explained before. The dependency manager not only allows us to add new services which can be accesses locally, but we can also define custom core functionality such as the dispatcher as shown here by adding custom event triggers during the execution process. This means that we have total control over every aspect of the application from when the route is provided to when it served to the end client after matching a controller/action.

Obviously, Phalcon has some drawbacks as well. The biggest of which is that it is a not a well-known framework when comparing it to other PHP frameworks such as the Zend framework or Symfony and does lack in some of the features that are supplied by both of these frameworks, for example, internationalization is an area where Phalcon lack. Another massive drawback of using this framework, or any other PHP framework over ASP.NET MVC is the lack of testing support that is available. Also the features that the Entity Framework provide to auto-generate database tables from a Model is also not included, but personally I feel that Phalcon has the best PHP Model implementation as it is the only one that has event triggers to database events.