REST Web Service Example in PHP

This is an example of a PHP web service that implements a REST interface. The web service is implemented using a base class that you can reuse to implement other web services. The rest of this document describes the files that make up the example and how they are used.

Overview

The example includes both a PHP web service and a client written using HTML, CSS and JavaScript. The client is provided both as a way of demonstrating the use of the web service and as an example of using Ajax and JQuery in a web application. There is no need for the web service to be called from JavaScript. It could just as easily be called from C# (as you will see later in the module) and other languages.

The files that implement the web service are as follows:

RestService.php	The base class that implements the core web service functionality.
BooksWebService.php	Inherits from RestService.php. Implements the functionality specific to this web service.
api.php	The main web service file
Book.php	Class that implements the Book object
dbinfo.php	The database configuration details. The variables in here should be changed to reflect the connection information for the MySQL database you are using.
.htaccess	Configuration file for use when deployed to Apache web servers
000-default.conf	Updated site configuration file for Apache to enable use of .htaccess files.
wsbooks_mysql.sql	The SQL script to create the tables needed for this application.

The files that implement the client are as follows:

Index.html	The main page for the client. This is a single page web site.
Content\	
bootstrap.min.css	The minified Bootstrap CSS file
Site.css	Site specific CSS
Scripts\	
bootstrap.min.js	The minified Bootstrap JavaScript file
jquery-1.10.2.min.js	The minified jQuery file
modernizr-2.6.2.js	Modernizr JavaScript file that emulates newer CSS features on older browsers
webservicedemo.js	The JavaScript specific to this web site.

Setting Up the Application

This assumes you are running this demo on the Linux virtual machine setup on Azure for this module. If you are running this on another Linux system, there might be some changes needed to the process, particularly when configuring Apache to enable .htaccess files.

First copy all the files for this demo to your home folder on the virtual machine using WinSCP.

Now, connect to your virtual machine using PuTTY and copy all of the files over to the Apache root folder using the following two commands:

```
sudo cp -r * /var/www/html
sudo cp .htaccess /var/www/html
```

The next step is to enable the use of rewrite rules on Apache. Use the following commands to enable the rewrite rules and restart Apache:

```
sudo a2enmod rewrite
sudo systemctl restart apache2
```

We now need to enable the use of .htaccess files so that we can specify the rewrite rules that we want. If you have configured a virtual machine just as specified in this module, you can simply execute the following two commands to do this.

```
sudo cp 000-default.conf /etc/apache2/sites-available
sudo systemctl restart apache2
```

If you are on another system, follow the instructions for enabling the use of .htaccess files on your system.

The last thing you need to do is setup the database. The web service used in this example supports basic CRUD operations (Create, Read, Update and Delete) on a simple database of books. It uses a simple table in a MySQL relational database. Before running this demonstration, you need to create a database in MySQL called wsbooks. You should then run the script provided in wsbooks_mysql.sql against the database to create the books table needed for this application. Finally, you need to update the variables in dbinfo.php to reflect the connection parameters needed for the MySQL database you are using.

The API

The web service uses the standard HTTP methods GET, PUT, POST and DELETE. The URLs use a directory structure notation. The operations currently supported are:

GET /books	Retrieve a list of all books in the database in JSON format
GET /books/id	Retrieve a book with a specific id
GET /books/year/month/day	Retrieve a list of all books published after the specified date
POST /books	Add a new book to the database
PUT /books	Update the book with the specified id (id comes from the book in the message body)
DELETE /books/id	Delete the book with the specified id

Data is passed to the web service in the HTTP Request message body and returned in the HTTP Response message body. All data is in JSON format.

The Role of the .htaccess Configuration Files

The actual main entry point for the web service is in api.php and it expects to receive the URL as a parameter. For example, in order for the web service to work, the HTTP request:

    GET /books/1

needs to be converted into the following:

    GET api.php?q=/books/1

This action is performed by the web server itself following rules in the .htaccess configuration file to 'rewrite' the URL, i.e. reformat the URL that comes from the client into the format expected by the web service. The actual configuration file you use depends on the web server you are deploying to. If you are deploying to an Apache web server, the .htaccess file is used. It contains the following lines:

```
Options +FollowSymLinks
RewriteEngine on
RewriteCond %{REQUEST_FILENAME} !-f
RewriteCond %{REQUEST_FILENAME} !-d
RewriteCond %{REQUEST_URI} !=/favicon.ico
RewriteRule ^(.*)$ api.php?q=$1 [L,QSA]
```

These lines perform the required rewriting of the URL. You can find out more about the contents of .htaccess files at http://httpd.apache.org/docs/current/howto/htaccess.html

Implementation of the Web Service

The class RestService.php provides the core implementation of a REST web service. To use it, you just need to inherit from this class and override the functions performGet, performPost, performPut and performDelete with your own implementations.

For each call to the web service, api.php simply creates an instance of your class that inherits from RestService.php (in this case BooksRestService) and then calls the method handleRawRequest.

The code is quite extensively commented and you are encouraged to work your way through it.

The Client

The client makes extensive use of Bootstrap. This is a framework originally developed by Twitter for web developers that assists in the development of responsive web sites that adjust to different sizes of screens. More information can be found at: http://getbootstrap.com/

The client site is implemented as one page (index.html) and one file containing JavaScript that is specific to the site (webservicedemo.js). This consists of a series of AJAX calls to the web service in response to actions by the user. DIVs in the main page are updated with the results of the calls to the web service.