Web Services and Cloud-Based Systems

Assignment 1 - Web Services

Tom Peerdeman & Laurens Verspeek
10266186 & 10184465

April 9, 2015

1 Calculator Service Implementation

The calculator service is written in PHP. PHP has builtin classes for SOAP servers and SOAP clients. However, the function to automatically generate WSDL files is not yet implemented in this SOAP extension of PHP. To overcome this problem we decided to use a library called PhpWsdl to automatically create the WSDL file. This library was also able to create a SOAP server from an existing PHP class and deploy it. It uses the builtin PHP classes for SOAP servers to do this.

The class used to create the SOAP server is called *calculator.class.php* and containts the implementation of 4 methods: *float add(float a, float b), float sub(float a, float b), float div(float a, float b), float mul(float a, float b)*. All the methods make use of float variables to support calculations on real numbers. The file *server.php* creates and runs the SOAP server with the use of the PhpWsdl library. It also creates documentation for the deployed service and automatically generates the WSDL file which can be found in *server.php?WSDL*. The library is also able to generate a client class automatically (example of such a client can be found in *generatedClient.class.php*). We also have a client which uses the builtin SoapClient class from PHP. This client is created from the WSDL file. The client can be found in *client.php*.

2 Answer question 3: Stateful Calculator

A stateful service means that the service has a memory of the past. Previous transactions are remembered and may affect the current transaction. In the context of our calculator service, stateful means that the service should keep track of what the client has already calculated previously. This enables the client to perform further calculations with the results of their previous calculations. This could be implemented in the service by giving each client an id and a state variable. This variable contains the current (last executed) calculation result. The service has to store this state variable among with the user id (e.g. in memory or in a database). When a client performs a request, he has to send his user id among with the request. The state variable for his id will then get updated. Whenever the client wants to use the previous result in his current calculation, he can just simply request a calculation with just one variable. Then the other variable will get filled in by the stored state variable.

3 URL-shortener service implementation

The URL shortener service is again written in PHP. PHP is a good choice for REST services as PHP has built in support for parsing the parameters of the different HTTP request methods. However PHP on its own cannot parse urls in the form of /:id, as required by the service specification. To solve this the apache redirect module is used. Any url following the /:id pattern is redirected on the server side to a php script with an id parameter. So for example
http://host.com/100 is internally called as http://host.com/index.php?id=100.

The next problem is the statefullness of the service. The service requires that the server remembers the URLs added to the shortener service. The service is thus statefull. By default PHP is stateless, however there are a couple of extensions to make PHP statefull. The extension used is SQLite. SQLite is a very small database stored on the server's file system. This database can be altered, and queried using standard SQL queries.
Using SQLite also solved the issue of unique ids for the URLs. SQL can mark columns in the table as primary. This means that all values in this column have to be unique. If the datatype of this column is integer, the column also gets the auto increment property. This means that for every insert this field is automaticly filled with an unique id, which is the highest existing id plus one. By returning this id to the user we always get an unique id.

Not mentioned in the specification is the behaviour when a duplicate URL is inserted. To save ids it was decided to only give out one id per unique URL. This means that when a duplicate URL is inserted into the system, the id of the original insertion of that URL is returned. While this policy works very well for simple add and get requests it can be, with the current implementation, a bit troublesome with update and delete requests. If an user decides to delete or update an url, any other user using this url is also impacted by this change. This can be solved by creating a system for multiple users, as explained in the next section. In this way we can return the same id when a duplicate URL is inserted, but give out a different id if the original inserting user differs from the new inserting user.

4 Answer question 3: Consider implementing URL shortener for multiple users

The URL shortener service makes no distinction between different clients. This does not mean that it can not support multiple users, only that these users all share the same available ids for URLs. As the ids are stored as a integer, this means that the maximum number of unique URLS which can be shortened is $2^{32} - 1$ (Maximum size of unsigned 32-bit integer). To be able to differentiate multiple users, the service can store a user id for each user among with the id of the URL and the URL itself. Whenever a user requests a URL with the id of a URL, he also has to provide his user id. Now that the service can make distinction between different users, it can start counting for the URL id at zero for each new user and only has to check for duplicates within the entries with the same user id. This way, each user can shorten $2^{32} - 1$ unique URLs, instead of $2^{32} - 1$ unique URLs for all users combined.