Lab Exercise 4 – RESTful Web Services and Deployment in the Google Cloud

CIS656 – Fall 2012

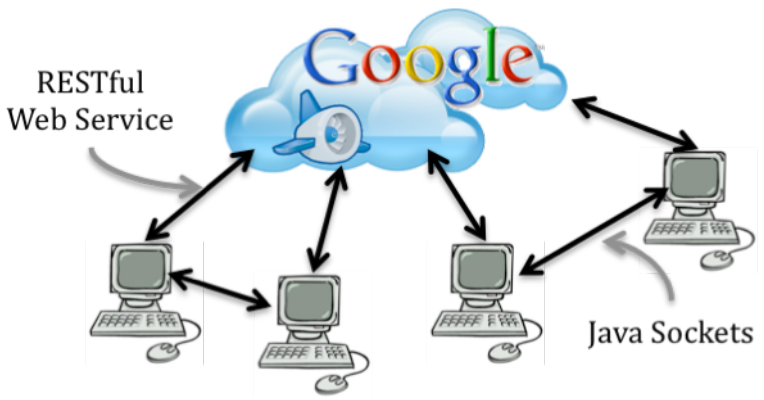
Due Date: November 14, 2012 (see below for description of deliverables.)
Grade Weight: 25 points (out of a total of 100 Lab points)

Objective

The objective of this project is to give hands-on experience building a RESTful web services interface and deploying it in the Google cloud. Successful completion of this lab will increase the students' understanding and appreciation of the topics covered in the lectures in the area of distributed web-based systems and cloud computing. In particular, students will gain not only a conceptual understanding of the differences of REST vs. "Big Web Services" (e.g. SOAP, WS-* stack) but will have hands-on experiencing designing and building a RESTful web service using the RESTlet Java framework.

Lab Description

In previous projects the student has implemented an instant messaging service using two different software architectures: a centralized client/server architecture implement using Java RMI, and a decentralized peer-to-peer architecture implemented using Chord. This lab will focus on another centralized implementation of the same application where the server is deployed in the Google cloud and accessed via RESTful web services.



.

Figure 1 Instant Messaging Architecture

The architecture of the instant messaging system the student will build is shown in Figure 1 above. Like Lab 2 completed earlier this semester, there are essentially two major types of components in this system:

1. Presence Service – The Presence Service is essentially a web server the exposes a resource-oriented (e.g. RESTful) web services interface to clients. The abstract interface (PresenceService.java) and related data structures (RegistrationInfo.java) have been specified *a priori*, and can be downloaded from the class website. Note that these have evolved slightly since Lab 2, so the student will want to use the versions distributed with this assignment. The Presence Service will ultimately be deployed in the Google cloud and accessed by remote chat clients via RESTful web services.
2. ChatClient - The Chat Client is the application that end users will run when they wish to see who is on-line, and/or have on-line discussions with other users. The ChatClient will use HTTP requests to register its own presence with the server, and also to determine the location and status of other ChatClients. As in Lab 2, the actual exchange of messages among clients will be direct via Java Sockets.

The interactions between these components are quite simple. When a user executes a ChatClient process, they will be expected to conform to the following conventions:

```
java [necessary java flags go here] ChatClient {user} [host[:port]]
```

where user refers to the name of the user invoking the client, and host:port refers to the host and port number associated with the Presence Service.

Upon startup, the ChatClient will register its presence with the Presence Service. The ChatClient will implement a very simple command-line-interface that interprets and executes six commands.

- friends – When this command is entered, the client determines (via the Presence Service) which users are registered with the presence server and prints out the users' names and whether they are available or not available (via the status field in the registration information).
- talk {username} {message} - When this command is entered, the client 1) first checks to see if the user is present *and available* (via the PresenceService). 2) If the user is registered and available, a connection to the target client is established, based on their registration info, and they are sent the given message. Note that when a client receives a message, it will simply print it out to the console, and re-prompt the user to enter his/her next command. (i.e. you will need multiple threads of execution here within your client process.)
- broadcast {message} – The client broadcasts the message to every user that is currently registered and available. Clients should NOT broadcast the message to themselves.
- busy – The client updates its registration with the presence server, indicating it is not currently available. If the client is already in not available when this command is entered, nothing needs to be done, though it would be good to prompt the user and indicate they already are not available. A client that is busy should not receive any messages whether they be sent with the talk or the broadcast command.

- available – The client updates its registration information with the presence server, indicating it is now available. If the client is already available when this command is entered, nothing needs to be done, though it would be good to prompt the user and indicate they are already registered as available.
- exit – When this command is entered, the ChatClient will unregister itself with the PresenceService and terminate.

Presence Server RESTful Interface

Students will implement the PresenceService abstract interface against the following URI scheme:

URI	GET	PUT	DELETE	POST
/v1/users	Returns a list of resources representing all users currently logged in on the system.	NA	NA	Registers a new user on the system. Entity is standard web form encoding.
/v1/users/{name}	Returns a representation of the user resource {name}.	Updates the status of resource {name}. HTTP request entity is standard web form encoding.	Deletes the user resource {name}	NA

Table 1. Application resource interfaces.

The RESTlet Framework

You will need the "Google AppEngine" edition of the RESTlet framework version 2.0.15. A link to the download is available on this page:

http://www.restlet.org/downloads

Google AppEngine Deployment

Your final implementation must be deployed via Google AppEngine:

http://code.google.com/appengine/

You will need (one time access) to an SMS capable mobile phone in order to signup for a Google App engine account. If you don't have a mobile phone, try setting up a free Google Voice account and using its SMS feature. **Do NOT use your GVSU gmail account when setting up your account. You MUST use a normal gmail account.** If you attempt to setup your GAE account with your GVSU account and enter your mobile, the account creation will fail and you will no longer be able to use that mobile # to create a free GAE account!

The instructor will provide demos Google App Engine and there are additional tutorials available on the App Engine site and as well as restlet.org, and provide you with sample code.

Implementation Suggestions

You need to come up to speed on both Google AppEngine and the RESTlet framework in order to complete this project. You might want to follow the following suggestions as a sort of roadmap so you don't spend too much time digging around for what you need:

1. To come up to speed on Google AppEngine (GAE) visit the developer site (http://code.google.com/appengine/) and follow the "Get Started" instructions over on the right.
2. If you are using Eclipse (recommended) you can simply install the Google Plug-in for Eclipse. (http://code.google.com/appengine/downloads.html#Download_the_Google_Plugin_for_Eclipse). There is no need to install the SDK separately.
3. Once you have your development environment setup and have created your GAE account, go ahead and complete the "Getting Started: Java" tutorial. (http://code.google.com/appengine/docs/java/gettingstarted/)
4. Once you are comfortable with GAE, download the (GAE Edition) of the RESTlet 2.0.15 framework (http://www.restlet.org/downloads.)
5. Create a new GAE project in Eclipse, and copy the necessary RESTlet jars to your project (you can mimic the settings in the sample code covered in lecture.)
6. Proceed to implement / test your chat server & client using the sample code covered in lecture as your reference model.

Deliverables

There are two deliverables required in order to receive full credit for this lab exercise.

Provide a Demo to Instructor

Provide a demo of the working system to the instructor during office hours before or on the designated due date.

Electronically Submit the Source Code

Archive all the source code needed to build and execute your implementation in a zip file and submit it to BlackBoard. The timestamp on your submission must be no later than the due date to receive credit.

References

http://en.wikipedia.org/wiki/Representational_State_Transfer - REST overview

http://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm - Chapter 5 of Roy Fielding's Ph.D. dissertation where REST was first presented.

http://www.restlet.org/ - home of the RESTlet framework that you will download and use for this lab.

http://www.themobilemontage.com/teaching/screencast-tutorials - find links to a three part screencast ("The Anatomy of a RESTful Web Service") on developing with the RESTlet framework put together by the instructor and links to the example source code.

L. Richardson and S. Ruby. RESTful Web Services. O'Reilly 2007 – The best book available on REST. Optional with regard to this project, but worth every penny if you are interested in developing a deeper understanding of REST.