CS5300 S16 **TR 11:40-12:55** **Gates G01**

Project 1a: A Simple Website Maintaining Sessions

1. Introduction

This is the beginning of first group programming project in CS5300. **This first part of the Project should be done individually.** That is, we want every student to implement it independently and turn it in to CMS. For Part b of the project (which will be posted shortly), you will transform this code into a distributed session state database running in Amazon AWS. For that part of the Project you should form **groups of 3 persons**. **You may form a group of 4 persons** if you wish; but you should talk to Professor Demers first. **You may also form a 1- or 2-person group,** but a small group will be expected to do all the work of a 3-person group.

The next project (Project 1b) will use AWS EC2, AWS SimpleDB or DynamoDB, and other Amazon Web Services to build a scalable and fault-tolerant website. This site will be very simple -- it will do nothing but maintain and display session state. It will run in in multiple Apache Tomcat instances, which will run Java Servlet (and/or JSP) code implementing the behavior of your site, together with additional Java code your group will write to implement a scalable, distributed, fault-tolerant, in-memory session state database using algorithms closely related to the SSM system discussed in lecture.

In the current assignment (Project 1a) you will write some of the Servlet or JSP code you will need, and test that code on your own machine (with no load balancer or distributed session state database).

Warning: This is an easy program if you have experience with Eclipse/Java/Tomcat. But without that experience, the entry barrier can be pretty high. So start this project right away!

2. Preliminaries

Make sure you have the JDK (not just a JRE) Version 1.8e from Oracle.

Download and install Apache Tomcat, Version 8 preferred.

In order to take advantage of Amazon's tools and APIs for this and later assignments, you should use the Eclipse IDE. Download and installation instructions for Eclipse can be found here. A description of the AWS Toolkit for Eclipse, together with links to the Eclipse download site and instructions for obtaining and installing the AWS Toolkit, are available here.

> **Note:** you want the Java EE version of Eclipse, not the plain Java version.

There are many places on the Web to find instructions for writing and deploying Java Servlets and JSPs. A very extensive set of tutorials can be found at the Marty Hall Course Materials Site. Some of this material describes older versions of Java, but this shouldn't cause problems, as the APIs you'll need for this project have not changed.

3. Site Behavior

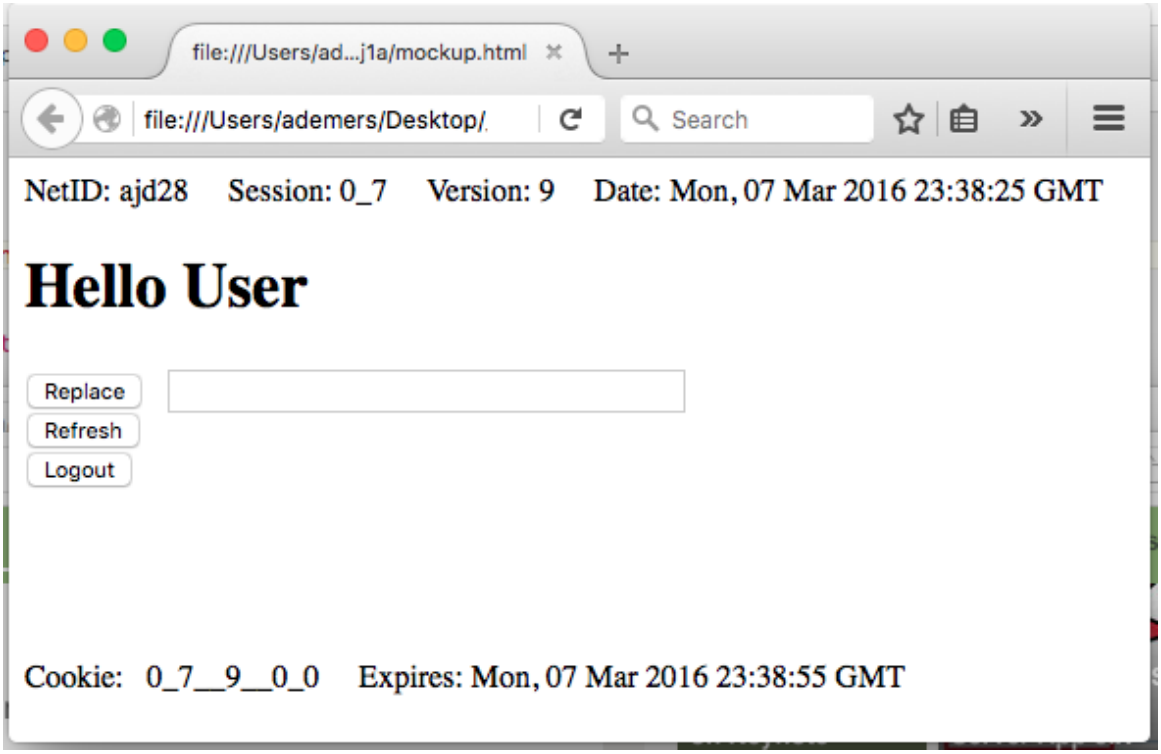
All your site needs to do is maintain and display a small session state, with session timeout implemented.

The session state is just a character string, which we call the "message". When a user first accesses the site, a new session state is created with the message set to a default value like "Hello, User!" The message is displayed on the returned page, along with the session ID, the current time and the expiration time of the

session, and an HTML form allowing the user to choose among 3 requests:

- **Refresh**
  Redisplay the session message, with an updated session expiration time;
- **Replace**
  Replace the message with a new one (that the user typed into an HTML form field), and display the (new) message and expiration time;
- **LogOut**
  Terminate the session.

Your site need not be beautiful; in fact it can be as ugly as this mockup:



For debugging and grading purposes the mockup displays the value of its session cookie in the lower left corner.

4. Implementing Session Management

Session management should use the standard technique of passing a cookie back and forth between client and server, as discussed in Lecture. The cookie must contain enough information to identify the session, but it should not contain the actual session state (i.e. the message), which is held at the server. **You must manage session cookies manually -- you may not use the Java `HttpSession` interface,** even though it is provided automatically by Apache Tomcat, because its Session objects do not have a few features (like version numbers) that will be necessary for Project 1b. So you should generate and manipulate session cookies in your own code, as Java `Cookie` objects, using methods from the `HttpServletRequest` and `HttpServletResponse` classes.

The name of your cookie should not conflict with any other software; something like

```
CS5300PROJ1SESSION
```

should be fine.

The value of your session cookie (which, like all cookie values, is just a character string) will have several parts:

- session ID
  A unique session ID must be generated by the application server for the first client request of each session (it is easy to recognize the *first* client request, because it has no session cookie). Eventually (for Project 1b) your site will consist of multiple application servers, making it a bit tricky to guarantee that a newly generated session ID is globally unique -- simply incrementing a counter independently at each application server is clearly not sufficient, nor does it work to use the system clock.
- version number
  As in the SSM paper, this identifies the version of the session data. For the SSM-like protocol we will be using in Project 1b, the version must be incremented on *every* request, including requests that don't change the message.
  Note: the session expiration time -- the time at which the `CS5300PROJ1SESSION` cookie expires -- also changes with *every* request. Both these properties imply that every request must return a new cookie to the client.
- location metadata
  Recall, in the SSM paper, this is a set of `WQ` identifiers of brick servers containing the specified version of the session data. For this assignment there is only a single application server, which holds all the session data in memory, so there is no real need for location metadata. But for Project 1b there will be multiple places where session data might be stored, so your cookie format will eventually need to support location metadata.

You can avoid the need to "canonicalize" the cookie values by restricting them to "safe" characters such as letters, digits and ".-_" and then parsing the values into their separate components as needed.

Of course, even a single-server site needs to be able to support multiple concurrent sessions and multiple concurrent client requests. Thus you need a "session data table", keyed by session ID (and possibly, depending on your design, by version as well) containing

> < sessionID, version, message, expiration-timestamp >

tuples. This will presumably be a Java `HashTable` or `HashMap`. In order to process a request, your Servlet code will first retrieve session state from this table, then do the necessary processing, then construct the new session cookie for return to the client and store the updated session state back into the session state table.

Some important considerations:

- The Tomcat container runs Servlet methods concurrently, in separate threads; be careful about synchronization of accesses to the session state table. Running with no synchronization could of course trash the contents of your state table; but simply locking the table for the duration of each request would serialize all the application threads. This might not be noticeable in this assignment, as the actual work done by each request probably finishes in much less than a millisecond, but in a real system it would degrade throughput unacceptably, so we expect you to get it right.
- You need to do something to remove timed-out sessions from the table; otherwise the table will grow without bound. (That's one reason for the timestamp field in the table entries) You can do this in a separate "daemon" thread, or you can piggyback this work on the request-processing threads.
- You should also limit the size of the session state value. We suggest an upper bound of 512 bytes; this will ensure that a session state value will fit in a single UDP network packet, which will be convenient for Project 1b.

5. Submit Your Work

Create a file, in `zip` archive format, named `solution.zip` This archive should contain

- a `README` file.
  This may be in `.pdf` or `.txt` format. This file should include anything we need to know to grade

your assignment. In particular, it should briefly describe the overall structure of your solution, including your cookie format, the design of your session table, how you garbage collect timed-out sessions, what functionality is implemented in each source file.

- your Java and/or JSP source code.
  Include understandable high-level comments.
- a deployable `.war` file. **Note:** your app needs to be deployable by just dropping the `.war` file into Tomcat&rrsquo;s app directory.
  Your `README` file should have any special instructions needed for us to run this.
- at least four screenshot files:
  - screenshot1: browser window right after connecting to your site;
  - screenshot2: browser window after clicking **Refresh**;
  - screenshot3: browser window after adding some text and clicking **Replace**;
  - screenshot4: browser window after clicking **Logout**;

Submit your `solution.zip` file using CMS by the specified deadline.

6. A Head Start for Project 1b

This part is optional -- you don't need to hand anything in.

To deploy and run your Servlet code on any standards-compliant Servlet container, you (or Eclipse) can create a `.war` archive. You already know how to install and run a Tomcat server on an EC2 instance, so you should be able to run your Project 1a solution there.

> **Don't forget to shut down your site after you have finished playing with it, so it will stop consuming money from your AWS account.**

At the end of each work session it is a good idea to use the AWS Management Console to make sure you have no running EC2 instances.