SER421 Online Lab 2 Fall 2017 due Monday 10/30 11:59:00PM

THIS LAB IS TO BE COMPLETED INDIVIDUALLY

Learning Outcomes:

1. Understand the nature of HTTP traffic, including the use of HTTP Verbs, request/response headers, and cookies on the modern web
2. Understand the awakward nature of CGI-based programming

Activity 1: HTTP (30%)

Using the developer tools in Chrome or Firefox, visit these URLs and answer the questions below:

1. http://www.msnbc.com
2. http://www.bing.com (type in any search phrase and hit return)
3. https://www.cnn.com

For each URL (only the initial URL), use the dev tool to find out the values in the HTTP response header. Specifically:

- Does the site set cookie(s)? List them.
- Does the site use persistent connections? Capture the network traffic with the site that suggests what it does and explain.
- Does the site compress its content? Again capture the network traffic that suggests this and explain.
- How many URLs are loaded based on the initial page load?
- Capture the response header paste it into your solution for each URL.
- You may put your answers in a Word/OpenOffice document named <asurite>_lab2.[docx|odt]

If these are sites that you have visited in the past, please make sure you clear your browser cache and cookies before recording your network traffic. Same goes if you "redo" your access to the site for this lab, we need a clean browser state.

Activity 2: CGI Programming (70%)

You are to build a small fictional new site for *NEW News Inc.* using the Commong Gateway Interface (CGI). New News provides news stories for casual and subscribed readers alike. It also provides a community reporter interface, where any registered reporter in the field can provide news stories. The main functionality centers on a content management system, with flavors of CRUD (Create, Read, Update, Delete) functionality for various roles.

Requirements:

Roles:

1. There are 3 roles in the system: “Guest”, “Subscriber”, and “Reporter”. Here is what each can do:
    a. Guests can read any public news story
    b. Subscribers can read any news story
    c. Reporters can read public news stories and news stories they have posted, but not news stories posted by other reporters.
    d. Reporters can create public and private (for Subscribers only), edit, and delete stories.

Login/Logout (use HTTP POST on any forms):

2. For this lab there is no real authentication. On any Guest page, provide a “Login” link that goes to a page that presents a username field in an HTML form. If the user is in the persistence store, then log that user in (track the login) under the Role the user was previously assigned. Obviously this is not how real authentication should work!
3. If the user does not exist in the persistence store, then the application should ask if the new user should be created and under what role (Subscriber or Reporter, Guests do not have accounts).
4. All screens for a logged in user (by definition, a Subscriber or Reporter) should indicate the user’s name, the user’s role, and provide a “Logout” link that logs the user out and returns the user to the landing page as a Guest. If the user is a Guest then the screen should indicate a Guest at the top, and a link that returns the user to the landing page.

CRUD operations (use HTTP GET or POST as you feel is best):

5. A View News (the landing) page will display the titles of all news stories for all users. This page should do the following:
    a. If the user in her/his present role can View the story, then the title should be a hyperlink to that story.
    b. If the user can Edit the story, then there should be an Edit button *next to* the story on the right that takes the user to a page to edit the story and complete the updates (or cancel).
    c. If the user can Delete the story, then there should be a Delete button *next to* the story on the right that takes the user to a page asking for confirmation.
6. Reporters should have a hyperlink in the page navigation to Create a new story. The link should take them to an Add story page, where the Reporter should be able to add a title, add a story, and indicate whether it is public or for subscribers only.

An example of this application is available http://lead2.poly.asu.edu:8080/NewNews/news (it is functional except it does NOT implement the "edit" feature of an existing news story). You will not replicate the URLs or necessarily the style of this application, but it gives you an idea of the functionality.

Constraints

The intent of this activity is for you develop CGIs, and to manage login state. With this in mind, here are your nonfunctional requirements:

1. The CGI server we will use the npm module called cgi available at https://www.npmjs.com/package/cgi. Make sure you get this package and none other.
2. You will write exactly ONE CGI program to handle all web functionality. You will have to think about how to map incoming information to the proper part of your program. This means specifically that you will need to pass some information in every request to your server to indicate what functionality you want.
3. The news file must be named news.xml and be in the same directory as your CGI program. Similarly, users and roles should be stored in a JSON file in the same directory and named newsusers.json. We will post examples of the files on Blackboard.
4. You may use any technique you like (hidden form fields, URL rewriting, Cookies) to manage the logged in state of the user (logged in, username, user role). You may explain in your README your design decision.
5. You are not allowed to use ANY client-side (in the browser) Javascript or CSS ANYWHERE in this activity. You do not have to make the application aesthetically pleasing. The CGI program must render complete pages.
6. You may use any one of the following for the CGI program: bash, sh, C, Python, Perl, or Java8 (note if you use Java you wil need to invoke Java from a script). All code must be portable. Ask if you have questions.
7. Your server should return proper client error messages for improper access to the server. Specifically, the HTTP response codes 400, 403, 404, and 405 should be returned in those situations where a web browser or other client is trying to improperly access the server (wrong HTTP verb, improperly specified request, unauthorized access to a page requiring proper authentication, and so on.
8. General programming best practices apply. Code should be readable, well-documented, and efficient. You need to create a portable solution (no absolute file paths and no hardcoded values). The solution should exhibit proper behavior even if hit by multiple users from multiple machines at the same time.

Hints:

- First write a "happy day case" soluton that has nothing to do with the web. That is, write it so it can run from the command-line for the main behaviors you expect.
- Since you are writing a single program to generate all the features of this application, your program will have to *route* incoming requests to different parts of your program. Think about the steps this will require, and what your main decision logic will be, and how to make modular these "parts". Of course since you can choose from a number of languages for this task, the actual way you do this will be language-dependent.
- 3rd-party libraries are allowed (for example, to parse XML and/or JSON). But your solution must be a CGI in part 2. Your 3rd party libraries must be described completely and URLs given for how to install. 3rd party libraries must also be portable (no Mac-only, Windows-only, or Linux-only code or libraries, we have to be able to test your code!)
- The CGI Wikipedia page is actually pretty detailed as to what CGI is and how it has evolved. There are a lot of CGI articles and tutorials out there, though I find most are language specific.
- My personal hint: don't just run to Google and type in "CGI". Understand what CGI is, and most importantly how it passes information to and from the parent server. Specifically:
    - What is the QUERY_STRING environment variable and how does it get set?
    - What other environment variables are used by CGI?
    - How is standard input (stdin) used as the mechanism for HTTP POST payloads?
    - How is standard output (stdout) used to write HTTP response headers and content back through the interface

  Take the time to understand these things before opening up your code editor. It is part of the objective of the lab to see how awkward this type of I/O is, but also how it can help you re-use legacy (old) programs!

Submission:

- Submit a zipfile (.zip only, not .rar, not .7z, not .tar.gz, not .tgz; note that Java's "jar" tool actually creates zipfiles) named <asurite>_lab2.zip. Note that your ASURITE is the one for ASU's computer systems, not your 10-digit id. I will deduct 5 points for not following the naming convention as it creates more work for us when grading.
- You may include a file named README.txt that explains anything you think we should know before grading.
- Please submit stable solutions. Resist the temptation to change code last minute, and put the proper files in your submission. Absolutely no late or resubmissions. If your solution does not work we will not try to fix it! Code that does not compile or throws runtime errors will result in severe point deductions and only limited partial credit!
- I strongly suggest taking your own zipfile submission and installing, compiling, configuring and running your solution in a clean directory on your system. "It works on my laptop" is not an accepted reason for a grade appeal.
- Keep in mind you can submit as many times as you want but we always grade the last submission. So don't get shut out at the submission deadline!
- This is an individual lab, no sharing of any part of your answers in any form.