EE461L – Software Engineering and Design Laboratory

Homework 1: Build your Own WebApp Blog

Due: February 19, 2016, Friday, 11:59pm

Objective. In this homework assignment, you will build your own web app, completely from scratch. The goal is to make sure you are comfortable with the tools and methods you will use throughout the semester.

Collaboration. This homework will be performed in teams of two.

The Assignment. In this homework, you will build "blog" app on Google AppEngine. You will submit a link to your working webapp, and we will interact with that app to test your submission. Here are the detailed requirements (to achieve a perfect grade, you must meet all of these requirements):

Landing page: The link you submit should be of the form http://<blogappid>.appspot.com. This page should be nicely formatted. It should include a header with at least one image that describes the blog, separated visually from a set of the most recent blog entries. Each blog entry should be displayed by displaying the title of the post (in some different font than the body), the content of the post, the user who posted it, and the date when it was posted. Look at some example blogs to see how they're generally formatted, if you're not familiar. You don't need to be fancy with side panels and such. You need to do something interesting with the look and feel of your blog, though. You could use an external style sheet to define the style (change some colors, font styles, etc.). Or you could go a step further and use an external tool, e.g., bootstrap (http://getbootstrap.com) or bootswatch (http://bootswatch.com/).

Login: The landing page should have a place for the visitor to sign in (using the Google Users Service). If a visitor is NOT signed in, he or she should be allowed to view the blog posts but not create additional posts.

Posting: Once a visitor is logged in, he or she should be able to create a new blog post. This means your page should have a link somewhere that allows the visitor to create a new post, which will likely take him or her to a new page. This page will have a place to enter a title of the blog post and the content of a blog post (two differently sized, labeled text boxes will suffice). You'll also need a button that the user can click to "submit" his or her new entry. Once the user clicks submit, he or she should be taken back to the landing page, and the newest blog entry should be displayed at the top of the list of entries. It probably wouldn't hurt to throw a Cancel button on that page as well, that allows the user to cancel a started post. Reuse the style sheet that you use for the landing page so that you get some uniformity among the pages. Posts should be persisted in the datastore. You can use any method you like, including Objectify or some other third party library.

Access to Posts: While the landing page should list the most recent handful of posts (say 3-5), it should also be possible to access all posts somehow. The easiest way to do this would be to allow the user to click a link that says "List All Posts" or something similar and be taken to a page that shows all posts on the blog, the most recent ones on top. You are required to do

at least this; you can choose to implement more sophisticated browsing if you want (e.g., the "Previous Posts" and "Later Posts" links you sometimes see on blogs).

A Cron Job: A What? A Cron job! You're going to teach yourself how to make your webapp do something predefined on a regular schedule, akin to `cron` in Unix. Your landing page should include a link that allows a visitor to "subscribe." Every day at 5pm Central Time, your app should send (via email) a digest of any new posts in the last 24 hours to all subscribed users. If there are no new posts, no email should be sent. **Your landing page should also include an unsubscribe link. We're going to subscribe to your blogs, but we really want to be able to unsubscribe, so test this link HARD to make sure it works.**
Here are some places to start:
The Google App Engine developers page for the Cron Service for Java:
https://developers.google.com/appengine/docs/java/config/cron
A slightly more detailed tutorial that includes servlet code:
http://goo.gl/kCv6FG

What to Submit. Before submitting your assignment, you and your partner will have to join the same HW1 group within Canvas. Once you do this, either one of you can submit the assignment, and it will be associated with both of your profiles. You should submit a **report** that explains how you implemented this assignment. This report should include the following:

- The URL of your app. In addition to embedding the URL in your report, you should also submit it as a comment to your submission (that just makes it easier for us to grab ad test). We'll test it by looking at it, examining its available source, and interacting with it (we'll sign in, post, subscribe, etc.). At this point you do not need to submit your source, but we reserve the right to request to inspect it at a later point. Keep a snapshot.
- A full description of the functionality of your blog with salient implementation details and screenshots. Don't go overboard, but make sure you hit the highlights.
- If you implemented more features than the required ones, please list them and describe them concisely in your own words.
- A short (definitely less than a page) *collaboration statement* that succinctly describes how you and your partner divided the work. For each team member, assess strengths and weaknesses. This assessment is required (if you don't submit it at all, your grade will be affected), but its contents will not influence your grade. Instead, it is your opportunity to reflect on where you need to focus more. For example, the following statement is the kind of thing we might expect: "Since Alice did all of the work developing our css, Bob is less prepared to create a css from scratch." Or "Alice should practice more with the Objectify API; Bob ended up doing most of the work for storing data." Or "Wow, Alice is fantastic at this JSP stuff!"

Grading Rubric. The following is the rough point allocation for the pieces of the assignment. The entire assignment is worth 100 points.

- The basic landing page is structured correctly, satisfies the stated requirements, displays posts, displays all of the required content of posts, etc. – 25 points.
- The basic landing page is not ugly and the style is applied to all pages – 5 points.
- The app gives the ability to access all posts from the beginning of time – 10 points

- Login works with the Google Users API – 10 points.
- Posting new blog entries works, from clicking on the New Post link, to typing the new post, to submitting it – 20 points.
- The subscribe link works and the email comes – 15 points.
- The unsubscribe link works and the emails stop coming – 10 points.
- Implementation of additional features – 5 points.

Some Recommended Materials.

- Official tutorials: https://cloud.google.com/appengine/docs/java
- Sanderson, D. (2012). *Programming Google App Engine*. 2nd Edition. Sebastapool, CA: O'Reilly.
- JavaTPoint-Servlet: http://www.javatpoint.com/servlet-tutorial

Here are some tips on this homework:

1. When you are creating a new project ₩ using the Web Application wizard in Eclipse, uncheck the "Use Google Web Toolkit" checkbox, and make sure the "Use Google App Engine" checkbox is checked (If you leave the GWT checkbox checked, the new project will be created with GWT starter files and requires the GWT plugin in your browser which is no longer supported by most web browsers).

2. You will have to compile your code with JDK1.7. Google App Engine provides Java 7 VMs in a sandboxed environment and attempting to visit a site that has been compiled using the Java 8 compiler would result in a browser window displaying the GAE 500 Server error.

3. Specification of static files, also called resource files, is recommended but not required. For your purpose, pushing all files in the WAR to the application servers is acceptable.

4. You can use the deployment descriptor to require secure connections (https) for certain URL paths. It is recommended but it is optional.

5. Certain kind of text field checking is necessary (e.g., a blog post's subject cannot be empty). You can choose your own approach to implementing this functionality.

6. Please make sure there is consistency between the content of your website and the corresponding data in the datastore. You can show it by giving examples of GQL (a SQL-like language for retrieving entities or keys from the App Engine scalable datastore) queries on the Administration Console.

7. Most likely Google's cloud executing environment will be different from the Maven on your computer. Examining logs on your admin console is an important way for debugging and troubleshooting.