CSC 346 Fall 2014

September 18, 2014

CSC 345 : Assignment 3

Dropbox Web Site

Due: Wednesday October 1 at 11:59pm

Overview

For this assignment you are to create a web site implemented using Python and Flask on Google App Engine and whose content is stored in Dropbox.

Dropbox

- Create a Dropbox App at https://www.dropbox.com/developers/apps. Be sure to select "Dropbox API App", "Files and datastores", and "Yes - my app only needs access to files it creates". You may name your app anything you like, but be aware that a directory/folder is created with that same name so you may want to avoid special characters and whitespace. The "Yes - my app…" answer is particularly important -- it causes a folder named Apps/<app name> to be created and limits your app's access to that folder. This prevents a bug or exploit in your app from accessing and modifying your entire Dropbox folder.
- Create an access token for your app. Go to the "App Console" for the app you just created and midway down the page it says "Generated access token" followed by a "Generate" button. Select the button to generate an access token. Save this token in a safe place; do NOT embed it in your source code.
- Download and install the Dropbox SDK.

Google App Engine

- If you haven't done so already go to Google App Engine at https://developers.google.com/appengine/ and do the Python/Flask tutorial. The project you create will serve as the basis for your web site.
- In order for your Google app to make a connection to your Dropbox app you must enable billing for your Google app. You shouldn't go over the free quota, but it must be enabled anyway. Go to https://appengine.google.com, select your app, and select Billing on the left to set it up.

- Edit the app.yaml file for your project so that the “handlers” and “libraries” sections contain the following:

  ```
  handlers:
  - url: .*
    script: main.app
    secure: always

  libraries:
  - name: jinja2
    version: "2.6"
  - name: markupsafe
    version: "0.15"
  - name: ssl
    version: latest
  ```

  The “secure: always” says to always use HTTPS, and the “name: ssl” information says that your app needs the latest version of the ssl module (it is used internally by the dropbox package).

- Copy the “dropbox” package directory from the Dropbox SDK to your Google app directory. When your app does “import dropbox” it will get the necessary files from here (Google, for obvious reasons, does not provide the dropbox package by default.)
- Download the urllib3 package from https://pypi.python.org/pypi/urllib3. This too is used by the dropbox module and not provided by Google. Unzip/untar it and copy the urlib3 directory it contains to your Google app directory.

Web App

- Implement your web server app by modifying the main.py you downloaded as part of the Google App Engine tutorial.
- Store your Dropbox token in a config file and access it via the ConfigParser module as demonstrated in lecture. The name of the config file must be “settings.cfg” and it must have the same contents as in lecture, i.e. a “Dropbox” section with a “token” option.

- When your app receives a request for a URL use the dropbox module to fetch the corresponding file. There is a bit of a trick to this in that you can't enumerate all possible files in your Flask application. Flask does allow for variable names in the app.route annotation, e.g.:

  ```
  @app.route('/<path:name>')
  def process(name):
          …
  ```

  The <path:name> says that the remainder of the URL (the rest of the path) should be stored in the variable *name* and passed to the function as a parameter. You can then look for a file by this name in Dropbox. You'll need to put a '/' at the beginning of the path you pass to Dropbox, e.g. '/' + name.

- You must also provide a special 'changes.txt' URL that returns the last 20 changes to the Dropbox folder contents. This is a not a real file in Dropbox, but instead the contents are automatically generated by your URL handler via the "delta" dropbox method. It should maintain a FIFO data structure that contains 20 changes, so that when the file is requested only the deltas since the last request are fetched and added to the FIFO. The contents of the FIFO are then used to create the file contents. Each delta is a single line in the file of the form:

  date file action

  where *date* is the time at which the delta was fetched (in the format returned by the time.ctime() python method, *file* is the name of the file (or directory), and *action* is what happened to the file, e.g. "modified" or "deleted". These fields are separated by tab ('\t') characters. Note that webhooks are not used to implement this functionality.

- If the URL specifies a non-existent file your web server app should return 404 (the HTTP 'not found' error code). This is done by calling abort(404).

- When the path specified in the URL is a directory your web server app should automatically look for an index.html file in that directory and return its contents. If index.html does not exist your app should return 404.

Turnin

- Turn in only the files you wrote or modified for this assignment to the appropriate D2L dropbox. Do not turn in all files, and especially do not turn in your settings.cfg file.