IST 516: Final-Report

Team 3

John Tyndall (jbt8@psu.edu)

Dwayne Jackson (doj5082@psu.edu)

Justin Piro (jsp21@psu.edu)

Kirk MacKenzie (kum225@psu.edu)

Sean Barbour (srb5032@psu.edu)

Fall 2013

Dr. Lee

Project Website: http://www.personal.psu.edu/kum225

Project Presentation: http://psu.voicethread.com/share/5254669

Table of Contents

Table of Contents 2
Section 3-1: Detailed Design of the Project Idea 3
Section 3-2: Overall Architecture Diagram 4
Section 3-3: List of XML/Web Technologies Demonstrated 6
Section 3-4: CalendarWx - Working Application 6
Section 3-5: Project Outline 7
Section 3-6: Demo Information 7
Section 3-7: Meeting Log 8
References 13
Appendix 1: CalendarWx Python code 14
Appendix 2: Main Python code 18
Appendix 3: Overview of Team Members' Responsibilities 22

Section 3-1: Detailed Design of the Project Idea

As traditional utilities and services move to the Internet, users must organize and keep track of the data and information served from several different sources. Often, information from one service complements the information from another; however, these services remain separate. One example is a calendar web application, which lists and displays details about upcoming appointments; another is weather forecast information, which informs users about impending conditions.

Currently, there are no web applications that combine weather forecasts and calendar appointments in such a way that a calendar user can view forecasts of appointment locations within the same interface. Google Calendar provides some of this functionality; however, it only displays weather information for the location associated with a Google Account. For example, if a user lives in Fairbanks, AL but has an appointment in El Paso, TX, the forecast may suggest entirely different conditions (i.e., it would show the weather in Fairbanks).

CalendarWx is our solution to this problem. It integrates Google Web Services and ATOM/RSS feeds from the National Weather Service to present a calendar user with an information mash-up. In doing so, it provides a unique and valuable tool to assist in the scheduling of appointments, making of travel plans, and planning of outdoor activities. Our application saves time and simplifies the process by allowing users to view information from several different sources in location.

At the heart of our design are the Google Web services and platform APIs, namely Calendar and Maps. In order to leverage these services in CalendarWx, we needed to choose a coding language to interact with them. Python was chosen given its familiarity from our use in previous coursework. Once this was determined, we had to familiarize ourselves with Google's Cloud Console and OAuth2. These services all work together to grant access to our web app for use of Google's Calendar API and provide a means of authenticating the end user. This was important to us because it not only created a secure environment in which to log in, but took the responsibility of owning sensitive account information out of our hands.

After authenticating, the user's calendar events are extracted from his or her Google Calendar. These events are redisplayed with one major addition: weather forecast information for that appointment date at that appointment's location. If an event does not have a location, then we do not display any weather information (the event could be at the user's home location or a different location). This is achieved by using the Google Maps Web Service Geocoding API. This allows us to generate the latitude and longitude values necessary to present to the National Weather Service, which will provide the weather for those coordinates. After that is completed, we use the NWS ATOM/RSS feed to obtain a seven-day forecast for the user's appointment location. Finally after all the data is collected, it is displayed using Jinja2 to generate an HTML environment from a given template.

Section 3-2: Overall Architecture Diagram

By making use of Google Cloud and Web Services, we were able to leverage a number of different services and keep the number of service providers to a minimum. Originally, we believed that we would need to maintain our own accounts to achieve our goal; however, using OAuth 2.0 made this unnecessary. Thus, all that is required for our application is a web host and the services provided by Google and the National Weather Service.

A diagram displaying this architecture is shown below.

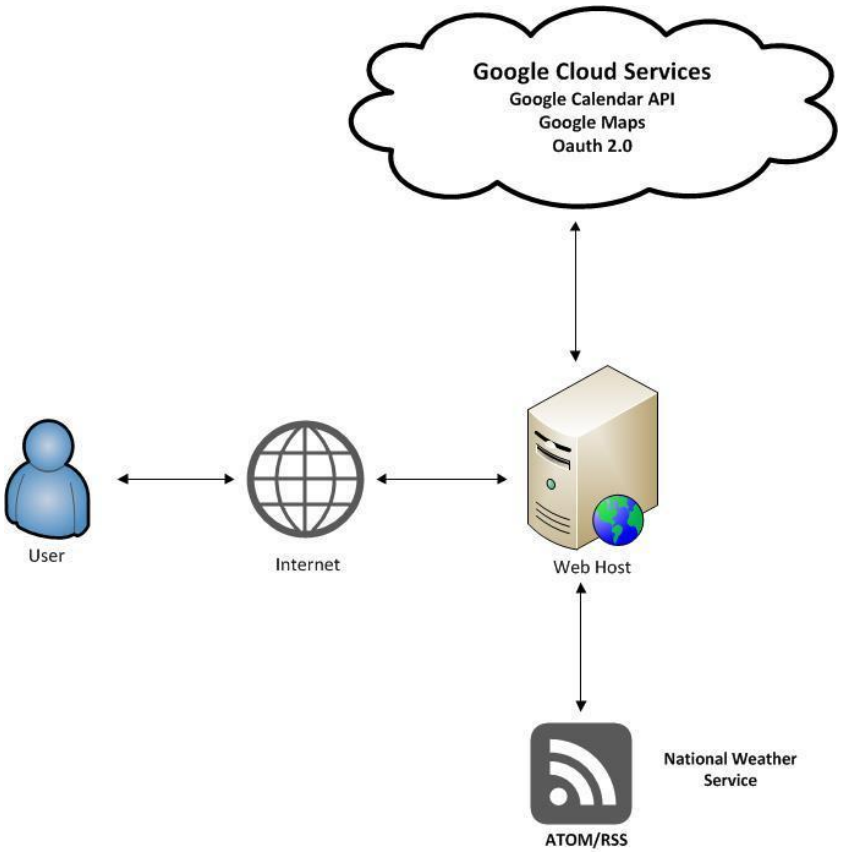

Figure 1: Architecture Diagram

When a user enters the CalendarWx application, he or she will be immediately sent to a Google authentication screen to log into a Google Account. During this time, two different authentications are occurring: first, CalendarWx authenticates with the Google Cloud Console and second, the user authenticate to Google and allows our application access to his or her data. CalendarWx then takes that information and combines it with parsed XML data from the National Weather Service, then presents it to the user.

The figure below shows this workflow.

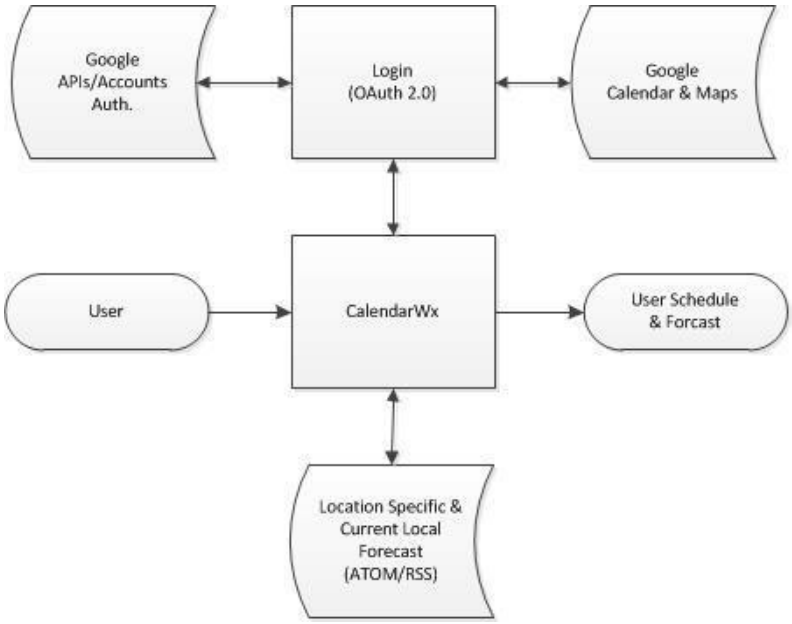

Figure 2: Workflow Diagram

Section 3-3: List of XML/Web Technologies Demonstrated

Several technologies and concepts from class are used to implement our project. These are listed in the following table.

Technology	**Description**
OAuth 2.0	Means of authentication for Google APIs and user accounts.
XML parsing	Weather forecast data was presented to us in XML format which we needed to parse and format to display.
ATOM/RSS Feed Reader	Method of distributing weather forecast data from the NWS.
Google Maps Geocoder	Google Web Service used to retrieve the latitude and longitude of each even location from Google Calendar.
Google Calendar API	Google Apps Platform API used to retrieve event information from the user's google calendar.
JINJA2	Template engine used to generate CalendarWx Agenda View.

Table 1: List of Technologies

Section 3-4: CalendarWx - Working Application

CalendarWx has the primary features listed below. These goals will be met if the user has a Google Account, utilizes the Google Calendar application, and chooses to input a location for all of their calendar events. Appointments without a location specified will not have weather information displayed.

1. Users log into CalendarWx with individual Google Accounts.
2. Appointments will display weather information at the appointment's location, as provided by the NWS RSS feed.
3. Current seven-day forecast will display for the user's current location.

Section 3-5: Project Outline

Please refer to the appendix, Overview of Team Members' Responsibilities, for a detailed outline and timeline.

Section 3-6: Demo Information

Once major entities for CalendarWx are in a functional state, we can begin our testing and debugging phases. To assist our group in the testing and debugging process, we created a simple testing worksheet to ensure that the requirements were properly addressed. The worksheet is incorporated within the appendix of this document. During our design process, Kirk had some difficulties with enabling OAuth in order to connect to the Google Calendar. Justin had the same code and was able to get the OAuth API to work as well as extract calendar entries. Kirk determined that there was either a firewall or other software that was preventing him from successfully receiving the OAuth credentials. All the other pieces of code were available to operate and create the calendar and extract the weather. The demo will show a mockup of the initial design, and we have included two main Python scripts in Appendix 1 and Appendix 3. Appendix 1 contains the code that uses the Google Geocoder API to take an address as input and return the latitude and longitude of the location. Those parameters are then used to extract the NWS forecast for the selected address. After receiving the seven-day forecast, it is

loaded into a Python dictionary and used to create a table view of the weather. Appendix 2 contains the code that implements the OAuth Google Calendar credentials and retrieves the calendar events.

Section 3-7: Meeting Log

- **Meeting 10/20/13**
  - Team had a sync to discuss status report.
- **Attendance**
  - Justin, John, Kirk, Dwayne, Sean
- **Summary**
  - Everyone will populate assigned sections for the status report.

- **Meeting 11/4-11/8**
  - Team had quick sync via email on when to meet next for next project deadline Task 2.
- **Attendance**
  - Justin, John, Kirk, Dwayne, Sean
- **Summary**
  - We decided to meet Sat Nov. 9 at 4pm EST
  - Had talks on if we should look to leverage Drupal content management system and look to modify a module to suit our needs for the project.
  - A new doc, IST516_Team3_Planning-Report_and_Task2_R2 was added to Google Drive on 11/5.

- **Meeting 11/9/13**
  - The team had meeting via Google Hangouts.
- **Attendance**
  - Justin, John, Kirk, Dwayne, Sean
- **Summary**
  - We decided to keep focus on or original idea as the needed skill set for Drupal custom module modification look to take a significant amount of time.
- **Action Plan**
  - We divided up the sections for task 2 evenly so we would not have any duplication of work. Everyone will try to have areas completed by mid-week.

- **Meeting 11/12-11/14**
  - Team had syncs via email.
- **Attendance**
  - Justin, John, Kirk, Dwayne, Sean
- **Summary**

- We decided to meet Sat 16th between 1pm-2pm EST

- **Meeting 11/16/13**
  - Team meeting via Google Hangouts @ 4pm EST
- **Attendance**
  - John, Sean, Justin, Dwayne, Kirk
  - All team members made the meeting except for one team member who updated us prior that he would be traveling.
- **Summary**
  - As we wanted to ensure the team was still on track with development efforts, we updated the Team Dev lead. We compiled the below summary of main items we need to discuss and address during next team meeting set for 11/17/13.
    - From our findings with the Google Calendar API, we might have limitations that will only allow us out-of-the-box to display the current conditions for that day. If our understanding is correct then we would like to make sure that this is expressed in the mid-term report, ideally in the Overall Architecture Diagram section or the Features App section.
    - We also need to make sure our diagram reflects at a high level what our finished product will look like.
- **Action Plan**
  - Send mail out to all team members with summary of today's meeting.
  - Have another meeting on 11/17.

- **Meeting 11/17/13**
  - Team meeting via Google Hangouts @ 2pm EST
- **Attendance:**
  - Justin, John, Kirk, Dwayne, Sean
- **Summary:**
  - Team Dev lead provided a mockup screenshot of what the GUI might look like and also sample code that retrieves calendar entries from a Google Calendar. We discussed in detail items written in ATOM, which allows the client to retrieve data from an RSS feed.
- **Action Plan**
  - Item 1: Will look to write the Python code to retrieve the 7 day forecast.
  - Item 2: Everyone will add to needed sections in the current doc shared out via Google Drive.
  - Item 3: Current diagrams we have in document will need to be updated as well to reflect areas we spoke of changes being made to.

- **Meeting 11/18-11/22, 11/24**
  - Syncs via email regarding next steps, in addition to status updates on each person's assigned area.
  - After feedback on Task 2 grade, we decided to meet on 11/24 @ 2pm EST via Google Hangouts.
- **Attendance**
  - Justin, John, Kirk, Dwayne,Sean

- **Summary**
  - Discussed feedback provided from Task 2. Had specifics to query Dr. Lee for better understanding of what is needed for Final Report.

- **Meeting 12/2-12/6**
  - Syncs via email.
- **Attendance**
  - Justin, John, Kirk, Dwayne,Sean
- **Summary**
  - As we hit some authentication issues via Google, we all decided it was best to request an extension. Email was composed on 12/3/13. We round-robined the email until everyone was approved, and content was sent to Dr. Lee.
- **Action Plan**
  - All hands on deck until we have authentication issues resolved.
  - We worked to ensure everyone's dev environment had consistency across Eclipse Juno, Kepler PyDev, and Java versions. Also await response from Dr.Lee on extension. We look to meet on 12/7 and 12/8.

- **Meeting 12/7**
  - Team meeting via Google Hangouts.
- **Summary**
  - We discussed our individual findings for our authentication to Google issues. We came to agreement on what looked to be the strongest findings. As class was granted an extension, we will have the other three team members take primary role on other major items that are due.
  - We then looked to have the Dev lead and secondary Dev continue with troubleshooting the issues local in their dev labs to find root cause and resolve issues.
    - Presentation slide deck design and outline.
    - Final Report structure.
    - Research and selection on software/service we will leverage to perform recording of presentation. Front-runner we discussed today is VoiceThread. Primary lead may need to email Dr. Lee for confirmation.
- **Action Plan**
  - Dev lead and secondary Dev will focus on coding road block faced. We will reconnect for meeting on 12/8 for updates and make adjustments to duties as needed.
  - Presentation lead will have deck design and sample structure populated for 12/8 meeting.
  - Final report will be created and structure laid out populated with needed information that we have currently ready.
  - Software/Service we selected and high-level recap on use will be ready for 12/8 meeting.

- **Meeting 12/8**
  - Team meeting via Google Hangouts.
- **Attendance:**
  - Justin, John, Kirk, Dwayne, Sean

- **Summary**
  - Secondary Dev lead identified root cause in local dev lab for our authentication to Google issue. Demo was provided via Google Hangouts of code working pulling needed information.
  - Presentation slide deck design and outline.
  - Final Report structure.
  - We have approval to use VoiceThread. High level how-to was provided during meeting.
- **Action Plan**
  - Primary Dev Lead will work on deploying the needed fix to the public demo we are developing in workspace.
  - Presentation slide deck design and outline created. We talked this over deciding the breakdown of slides and most important items of focus:
    - Demo.
    - Technologies used.
    - Primary features which make our Web App unique.
  - Final report draft was shared out to Google Drive with updated Meeting Log.
  - We must start recording as soon as we have the following complete:
    - Development
    - Presentation
  - Next meeting on 12/9.

- **Meeting 12/9**
  - Team meeting via Google Hangouts.
- **Attendance**
  - Justin, John, Dwayne, Sean
- **Summary**
  - We discussed the pending items for the project.
  - Primary and secondary Dev are still working with the deployment of code in the public work space. As we had a teammate missing due to uncontrollable events, we will meet again on 12/10.
- **Action Plan**
  - Presentation lead will continue to work on main aspects of presentation.
  - Primary Dev lead and secondary Dev lead will continue to work on deployment.
  - Final Report leads will continue work on populating the needed items.
  - We will look to meet on 12/11 to start VoiceThread recording if all goes well with recording. Devs will update.

- **Meeting 12/10**
  - Team meeting via Google Hangouts.
- **Attendance**
  - Justin, John, Dwayne, Kirk, Sean
- **Summary**
  - We discussed primary issues we are still facing while deploying. We came to agreement on potential work-around we look to use.

- During the presentation we all reviewed each slide and made final adjustments to the presentation via Google Docs. We also selected and assigned a person to the set of slides they would speak to during presentation.

- **Meeting 12/11**
  - Team meeting via Google Hangouts.
- **Attendance**
  - Justin, John, Dwayne, Kirk, Sean
- **Summary**
  - As a team, we reviewed and edited the slides of our presentation. Then uploaded to VoiceThread.
  - We divvied up the PowerPoint slides for our audio recordings.

- **Meeting 12/12**
  - Team meeting via Google hangouts
- **Attendance**
  - Justin, John, Dwayne, Kirk, Sean
- **Summary**
  - Team reviewed online presentation together. We discussed and changed certain parts of voice recordings.
  - We ran through sections of the final report, verifying the correctness and cohesiveness.
  - We finalized the project website.
  - We came to the conclusion that the project was ready for submission.

References

Briggs, Jason R. (2012-11-26). Python for Kids: A Playful Introduction to Programming. No Starch Press. Kindle Edition.

Google. "Google Accounts Authentication and Authorization." *Using OAuth 2.0 to Access Google APIs*. Developers.google.com, n.d. Web. 7 Nov. 2013., from http://https://developers.google.com/accounts/docs/OAuth2

Google. "Google Calendar API," Retrieved November 7, 2013, from https://developers.google.com/google-apps/calendar/

Google. "Google Code Playground," Retrieved November 7, 2013, from https://code.google.com/apis/ajax/playground/

Jinja. "Welcome to Jinja2," Retrieved November 20, 2013, from http://jinja.pocoo.org/docs/

National Weather Service. "NWS Public Alerts in XML/CAP and ATOM Formats," Retrieved November 7, 2013, from http://alerts.weather.gov

Appendix 1: CalendarWx Python code

```
#!/usr/bin/env python
#
#
"""CalendarWx is a calendar concept that uses calendar location entries
   and retrieves the weather for the event.

The CalendarWx application uses entries posted in a users calendar
along with the location of the appointment or event and retrieves
the weather forecast for the associated event time in order to inform
the user of what the weather will be during the event.
"""

import httplib2
import logging
import os
import time

import urllib2
import xml.etree.ElementTree as ET

from apiclient import feedparser

def addressToLatLong( address ):

    """Extract the latitude and longitude from a selected address using
       Google's Geocode API.

    Args:
    address: string, the location of the scheduled calendar event.

    Returns: The latitude and longitude of the given location.
    """

    geocodeURL="http://maps.googleapis.com/maps/api/geocode/xml?address=%s&sensor=false" % address

    response = urllib2.urlopen(geocodeURL)
    xmlgeocode = response.read()

    root = ET.fromstring(xmlgeocode)

    for result in root.iter('result'):
        for geo in result.iter('geometry'):
            for locat in geo.iter('location'):
                lat = locat.find('lat').text
                lng = locat.find('lng').text

    return (lat,lng)

def extractWeatherFcst(lat, lng):

    """Extract the weather forecast from the NWS based on the given
       latitude and longitude..

    Args:
    lat: string, the latitude of the selected location.
    lng: string, the longitude of the selected location.
```

```
    Returns: The latitude and longitude of the given location.
    """
    fcst_search = retrieveWxForecast(lat, lng)

    root = ET.fromstring(fcst_search)

    # Create a list of forecast times to use them as keys for a dictionary
    # lookup to be used when integrating the calendar events.
    fcst_times = [];

    # Create the forecast dictionary that will associate a forecast time with
    # the forecast for that time period.
    fcst_dict = {};

    # Loop through the output of the XML file and parse the information.
    for result in root.iter('data'):

        # Loop through the forecast time layouts.
        for times in result.iter('time-layout'):

            # Find the forecast output format.
            time_layout = times.find('layout-key').text

            # Only retrieve the 12 hour forecast information.
            if time_layout.find('p12h') > 0:

                # Retrieve all the forecast times, format them into Python
                # time structures and load then into a list.
                for period in times.iterfind('start-valid-time'):

                    dashPos = period.text.rfind('-')
                    ftime = period.text[0:dashPos]

                    fcstTime = time.strptime(ftime, "%Y-%m-%dT%H:%M:%S")
                    fcst_times.append(fcstTime)

        # Iterate through the forecast information and extract the
        # forecast text.
        for geo in result.iter('wordedForecast'):

            pIdx = 0
            # Get all of the forecasts for every time period and create the
            # dictionary entries.
            for tmp in geo.iterfind('text'):
                fcst_dict[fcst_times[pIdx]]= tmp.text
                pIdx += 1

#    for dict in iter(fcst_dict):
#        print dict, fcst_dict[dict]

    return fcst_dict, fcst_times

def retrieveWxForecast(lat, lng):

    """Retrieve the weather forecast for the given location.

    Args:
    lat: string, the latitude of the selected location.
    lng: string, the longitude of the selected location.
```

```
    Returns: The XML forecast information.
    """


       ndfdTestUrl3='http://forecast.weather.gov/MapClick.php?lat='+lat+'&lon='+lng+'&unit=0&lg=english&F
       cstType=dwml'
    fcst_xml = urllib2.urlopen(ndfdTestUrl3).read()

    return fcst_xml

def formatCalendarEntry(location, wx_dict, time_list):

    """Format the HTML calendar entry output to be in a table format.

    Args:
    location: string, The address location
    event_dict, TODO: Add in  the calendar entries to be inserted into calendar table.
    wx_dict: dict, The dictionary containing the forecast time and wording.
    time_list: list, The list of available time periods from the forecast.

    Returns: The HTML table output to be display on the main page.
    """

    event_dict = {}
#{'Outdoor Ice Skating':time.strftime('%Y%m%d-%H:%M:%S','20131211-04:00:00'),
#                  'Outdoor Ice Skating':time.strftime('%Y%m%d-%H:%M:%S','20131213-03:00:00'),
#                  'Outdoor Ice Skating':time.strftime('%Y%m%d-%H:%M:%S','20131214-12:00:00')
#                  }



    cal_entries = '<table width="1000">\n'
    cal_entries += '<tr>\n<th>Date</th><th>Scheduled</th><th>Event</th><th>Location</th><th>Daytime
       Weather</th><th>Nighttime Weather</th>\n</tr>\n'

    for in_time in time_list:
#        print in_time, wx_dict[in_time]
        dict_time = time.strftime('%Y%m%d-%H:%M:%S',in_time)
        cal_entries +=
       '<tr><td>'+dict_time+'</td><td></td><td></td><td>'+location+'</td><td></td><td>'+wx_dict[in_time]+
       '</td><td></td></tr>\n'

    cal_entries += '</table>\n'

    return cal_entries



def retrieveCurrentWeather(address):

    """Retrieve the current weather from an ATOM/RSS feed to be displayed
       for the current day's weather conditions on the calendar.

       Args: string, The address of the user's current location.

       Returns: The current conditions for the given location.
    """
    rssURL = 'http://w1.weather.gov/xml/current_obs/'

    # Use the
```

```
    lat, lng = addressToLatLong(address)

    wx_search = retrieveWxForecast(lat, lng)

    root = ET.fromstring(wx_search)

    # Loop through the output of the XML file and parse the information.
#    for result in root.iter('data'):
#
#        print result
#        if result.find('current observations') > 0:
#
#            # Loop through the weather information.
#            for wx in result.iter('moreWeatherInformation'):
#                print wx

    d = feedparser.parse('http://w1.weather.gov/xml/current_obs/KBOS.rss')
    d.feed

#    print d.feed
    return d.feed.title
```

Appendix 2: Main Python code

```
##!/usr/bin/env python
#
# Copyright 2013 Google Inc.
#
# Licensed under the Apache License, Version 2.0 (the "License");
# you may not use this file except in compliance with the License.
# You may obtain a copy of the License at
#
#     http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
# See the License for the specific language governing permissions and
# limitations under the License.
#
"""Starting template for Google App Engine applications.

Use this project as a starting point if you are just beginning to build a Google
App Engine project. Remember to download the OAuth 2.0 client secrets which can
be obtained from the Developer Console <https://code.google.com/apis/console/>
and save them as 'client_secrets.json' in the project directory.
"""


import httplib2
import logging
import os
import argparse


from apiclient import discovery
from oauth2client import appengine
from oauth2client import client


from oauth2client.file import Storage
from oauth2client.tools import run_flow
from oauth2client.client import flow_from_clientsecrets


from google.appengine.api import memcache


import webapp2
import jinja2
import calendarwx



JINJA_ENVIRONMENT = jinja2.Environment(
    loader=jinja2.FileSystemLoader(os.path.dirname(__file__)),
    autoescape=True,
    extensions=['jinja2.ext.autoescape'])


# CLIENT_SECRETS, name of a file containing the OAuth 2.0 information for this
# application, including client_id and client_secret, which are found
# on the API Access tab on the Google APIs
# Console <http://code.google.com/apis/console>


CLIENT_SECRETS = os.path.join(os.path.dirname(__file__), 'client_secrets.json')
# Helpful message to display in the browser if the CLIENT_SECRETS file
# is missing.
MISSING_CLIENT_SECRETS_MESSAGE = """
<h1>Warning: Please configure OAuth 2.0</h1>
```

```
<p>
To make this sample run you will need to populate the client_secrets.json file
found at:
</p>
<p>
<code>%s</code>.
</p>
<p>with information found on the <a
href="https://code.google.com/apis/console">APIs Console</a>.
</p>
""" % CLIENT_SECRETS

service = discovery.build('calendar', 'v3')#, http=http)

decorator = appengine.oauth2decorator_from_clientsecrets(
    CLIENT_SECRETS,
    scope=[
      'https://www.googleapis.com/auth/calendar',
      'https://www.googleapis.com/auth/calendar.readonly',
    ],
    message=MISSING_CLIENT_SECRETS_MESSAGE)



FLOW = flow_from_clientsecrets(CLIENT_SECRETS,
                               scope='https://www.googleapis.com/auth/calendar',
                               redirect_uri='http://example.com/auth_return')

"""
The ArgumentParser will be used to inject parameter arguments into
the run_flow method that initializes the client credentials.
"""
argparser = argparse.ArgumentParser(add_help=False)
argparser.add_argument('--auth_host_name', default='localhost',
                       help='Hostname when running a local web server.')
argparser.add_argument('--noauth_local_webserver',
                       default=False, help='Do not run a local web server.')
argparser.add_argument('--auth_host_port', default=[8080, 10080], type=int,
                       nargs='*', help='Port web server should listen on.')
argparser.add_argument('--logging_level', default='ERROR',
                       choices=['DEBUG', 'INFO', 'WARNING', 'ERROR',
                                'CRITICAL'],
                       help='Set the logging level of detail.')



# If the Credentials don't exist or are invalid, run through the native client
# flow. The Storage object will ensure that if successful the good
# Credentials will get written back to a file.
storage = Storage('calendar.dat')

credentials = storage.get()
#print credentials
#if credentials is None or credentials.invalid == True:
#    credentials = run_flow(FLOW, storage, argparser.parse_args([]))
#print credentials

# Create an httplib2.Http object to handle our HTTP requests and authorize it
# with our good Credentials.
http = httplib2.Http()
#http = credentials.authorize(http)

# Build a service object for interacting with the API. Visit
# the Google Cloud Console
# to get a developerKey for your own application.
```

```
# SAVE BELOW!!!!
page_token = None

#events = service.events().list(
#        calendarId='kirkmk71@gmail.com',
##        calendarId='justin.piro@gmail.com',
#        singleEvents=True,
#        maxResults=3,
#        orderBy='startTime',
#        timeMin='2013-11-01T00:00:00-08:00',
#        timeMax='2013-12-31T00:00:00-08:00',
#        pageToken=page_token,
#        ).execute()

#return events
# print 'PRINT EVENTS'
#events={u'nextPageToken': u'EiUKGjhudmg3MGpiajRxM2Jsc2djNTc2azVsaDdnGICA3bWrzLoC', u'kind':
      u'calendar#events', u'defaultReminders': [{u'minutes': 10, u'method': u'popup'}, {u'minutes': 10,
      u'method': u'sms'}], u'items': [{u'status': u'confirmed', u'kind': u'calendar#event', u'end':
      {u'date': u'2013-11-04'}, u'created': u'2013-11-03T04:10:12.000Z', u'iCalUID':
      u'qikg029jfsp91bkoblbr71oir8@google.com', u'reminders': {u'overrides': [{u'minutes': 30,
      u'method': u'popup'}], u'useDefault': False}, u'htmlLink':
      u'https://www.google.com/calendar/event?eid=cWlrZzAyOWpmc3A5MWJrb2JsYnI3MW9pcjhfMjAxMzExMDMganVzdG
      luLnBpcm9AbQ', u'sequence': 1, u'updated': u'2013-11-08T20:50:17.702Z', u'visibility': u'private',
      u'summary': u'Feed Oscar', u'start': {u'date': u'2013-11-03'}, u'etag':
      u'\"zOs4HjqIVxd3GdBeTTwy4VkH_no/Mjc2NzM5NTYyOTg4NjAwMA\"', u'originalStartTime': {u'date': u'2013-
      11-03'}, u'transparency': u'transparent', u'recurringEventId': u'qikg029jfsp91bkoblbr71oir8',
      u'organizer': {u'self': True, u'email': u'justin.piro@gmail.com'}, u'creator': {u'self': True,
      u'email': u'justin.piro@gmail.com'}, u'id': u'qikg029jfsp91bkoblbr71oir8_20131103'}, {u'status':
      u'confirmed', u'kind': u'calendar#event', u'end': {u'timeZone': u'America/New_York', u'dateTime':
      u'2013-11-04T22:00:00-05:00'}, u'created': u'2013-09-23T13:43:23.000Z', u'iCalUID':
      u'uafhdfknhh1uooc618achik2jo@google.com', u'reminders': {u'overrides': [{u'minutes': 10,
      u'method': u'popup'}, {u'minutes': 10, u'method': u'sms'}], u'useDefault': False}, u'htmlLink':
      u'https://www.google.com/calendar/event?eid=dWFmaGRma25oaDF1b29jNjE4YWNoaWsyam9fMjAxMzExMDVUMDIwMD
      AwWiBqdXN0aW4ucGlyb0Bt', u'sequence': 1, u'updated': u'2013-11-05T01:51:34.400Z', u'summary':
      u'Team Canceron Adobe Connect Telecon', u'guestsCanInviteOthers': False, u'start': {u'timeZone':
      u'America/New_York', u'dateTime': u'2013-11-04T21:00:00-05:00'}, u'etag':
      u'\"zOs4HjqIVxd3GdBeTTwy4VkH_no/Mjc2NzIzMjU4NzUzMTAwMA\"', u'originalStartTime': {u'timeZone':
      u'America/New_York', u'dateTime': u'2013-11-04T21:00:00-05:00'}, u'location': u'Adobe Connect',
      u'recurringEventId': u'uafhdfknhh1uooc618achik2jo', u'attendees': [{u'email': u'asw195@psu.edu',
      u'responseStatus': u'needsAction'}, {u'displayName': u'John Colt', u'email':
      u'johnccolt@gmail.com', u'responseStatus': u'needsAction'}, {u'self': True, u'displayName':
      u'Justin Piro', u'email': u'justin.piro@gmail.com', u'responseStatus': u'declined'}, {u'email':
      u'szelistowski@raytheon.com', u'responseStatus': u'needsAction'}, {u'displayName': u'James
      Farrell', u'email': u'jsmfarrell@gmail.com', u'responseStatus': u'needsAction'}, {u'displayName':
      u'Joel Swenson', u'email': u'joel.w.swenson@gmail.com', u'responseStatus': u'needsAction'},
      {u'organizer': True, u'displayName': u'Bob Szelistowski', u'email': u'bski.largo@gmail.com',
      u'responseStatus': u'accepted'}, {u'email': u'wayne@signallaw.com', u'responseStatus':
      u'needsAction'}], u'organizer': {u'displayName': u'Bob Szelistowski', u'email':
      u'bski.largo@gmail.com'}, u'creator': {u'email': u'bski.largo@gmail.com'}, u'id':
      u'uafhdfknhh1uooc618achik2jo_20131105T020000Z'}, {u'status': u'confirmed', u'kind':
      u'calendar#event', u'end': {u'date': u'2013-11-06'}, u'created': u'2013-11-03T04:11:10.000Z',
      u'iCalUID': u'8nvh70jbj4q3blsgc576k5lh7g@google.com', u'reminders': {u'overrides': [{u'minutes':
      1440, u'method': u'popup'}], u'useDefault': False}, u'htmlLink':
      u'https://www.google.com/calendar/event?eid=OG52aDcwamJqNHEzYmxzZ2M1NzZrNWxoN2cganVzdGluLnBpcm9AbQ
      ', u'sequence': 0, u'updated': u'2013-11-03T04:11:11.095Z', u'summary': u'Vacation Day', u'start':
      {u'date': u'2013-11-05'}, u'etag': u'\"zOs4HjqIVxd3GdBeTTwy4VkH_no/Mjc2NjkwMzc0MjA5NzAwMA\"',
      u'organizer': {u'self': True, u'email': u'justin.piro@gmail.com'}, u'creator': {u'self': True,
      u'email': u'justin.piro@gmail.com'}, u'id': u'8nvh70jbj4q3blsgc576k5lh7g'}], u'updated': u'2013-
      12-05T22:07:00.428Z', u'summary': u'Justin Piro', u'etag':
      u'\"zOs4HjqIVxd3GdBeTTwy4VkH_no/Hj1en0tzQL3JaoAOuGFxkL4ecHg\"', u'timeZone': u'America/New_York',
      u'accessRole': u'owner'}
# print events
#format(events[string])
#target.write(events)
output = "never made it"
# Loop until all pages have been processed.
#while events != None:
```

```
output = "I'm in"
    # Get the next page.
#    response = events.execute()
    # Accessing the response like a dict object with an 'items' key
    # returns a list of item objects (events).
#for event in events.get('items', []):
        # The event object is a dict object with a 'summary' key.
#        output.write(repr(
#    print event
#    for sum in event.get('summary', 'NO SUMMARY'):
    #) + '\n')
        # Get the next events object by passing the previous events object to
        # the list_next method.
#        events = service.events().list_next(events, response)
#        print sum

class MainHandler(webapp2.RequestHandler):

  @decorator.oauth_aware
  def get(self):
    variables = {
        'url': decorator.authorize_url(),
        'has_credentials': decorator.has_credentials(),
        'calendar_entry': cal_entry,
        'current_wx': current_wx
        }
    template = JINJA_ENVIRONMENT.get_template('main.html')
    self.response.write(template.render(variables))



app = webapp2.WSGIApplication(
    [
     ('/', MainHandler),
     (decorator.callback_path, decorator.callback_handler()),
    ],
    debug=True)



#print events
#def main():

location = "Boston,MA"

lat, lng = calendarwx.addressToLatLong(location)

wx_dict, time_list = calendarwx.extractWeatherFcst(lat,lng)

cal_entry = calendarwx.formatCalendarEntry(location, wx_dict, time_list)

current_wx = calendarwx.retrieveCurrentWeather(location)
```

Appendix 3: Overview of Team Members' Responsibilities

Task	Description	Author(s)	Time Line	Current Status
Task1	**Planning Report**	All team members	Week of 10/14-10/20	Status: Completed
Part:1 (1-1)	Motivation	All team members	Week of 10/14-10/20	Status: Completed
Part:1 (1-2)	Primary focus of the Web App?	All team members	Week of 10/14-10/20	Status: Completed
Part:1 (1-3)	Major components of the Web App	All team members	Week of 10/14-10/20	Status: Completed
Part:1 (1-4)	How did we arrive at the current design choice?	All team members	Week of 10/14-10/20	Status: Completed
Part:1 (1-5)	What were some alternative design approaches that we considered, but did not use and why?	All team members	Week of 10/14-10/20	Status: Completed
Part:1 (1-6)	High level summary of design/build approach	All team members	Week of 10/14-10/20	Status: Completed
Part:1 (1-7)	Project Outline	All team members	Week of 10/14-10/20	Status: Completed
Part:1 (1-8)	Team Web Page **Link:** www.personal.psu.edu/kum225/	Kirk/John	Week of 10/7-10/13	Status: Completed
Task2	**Midterm Report**	All team members	11/17	Status: Completed
Part:2 (2-1)	Detailed design of the project idea	All team members	Week of 11/4-11/10	Status: Completed
Part:2 (2-2)	Overall Architecture Diagram	All team members	Week of 11/4-11/10	Status: Completed
Part:2 (2-3)	List of XML/Web Technologies Demonstrated	All team members	Week of 11/4-11/10	Status: Completed
Part:2 (2-4)	Features of the App	All team members	Week of 11/4-11/10	Status: Completed
Part:2 (2-5)	Evaluation	All team members	Week of 11/11-11/17	Status: Completed
Part:2 (2-6)	Updated Changes from Planning Report	All team members	Week of 11/11-11/17	Status: Completed
Part:2 (2-7)	Task 2 Meeting Log Notes	All team members	Week of 11/4-11/10 and 11/11-11/17	Status: Completed
Task3	**Final Report**	All team members	Week of 12/9-12/12	Status: Completed
Part:3 (3-1)	Detailed Design of the Project Idea	All team members	Week of 12/9-12/12	Status: Completed
Part:3 (3-2)	Overall Architecture Diagram	All team members	Week of 12/9-12/12	Status: Completed
Part:3 (3-3)	List of XML/Web Technologies Demonstrated	All team members	Week of 12/9-12/12	Status: Completed
Part:3 (3-4)	Implementation Details	All team members	Week of 12/9-12/12	Status: Completed
Part:3 (3-5)	Project Outline	All team members	Week of 12/9-12/12	Status: Completed
Part:3 (3-6)	Demo Information	All team members	Week of 12/9-12/12	Status: Completed

Part:3 (3-7)	Meeting Logs	Dwayne	Week of 12/9-12/12	Status: Completed
Task4	**Presentation Video**	All team members	Week of 12/9-12/12	Status: Completed
PowerPoint	PowerPoint Creation/Editing	Lead: Sean All team members	Week of 12/9-12/12	Status: Completed
Video	Video Presentation Creation/Editing	All team members	Week of 12/9-12/12	Status: Completed
Appendix 1:	Web Retrieval Technologies Implemented	All team members	Week of 12/9-12/12	Status: Completed
Appendix 2:	CalendarWx Python Code	Leads: Kirk/Justin Team members review	Week of 12/9-12/12	Status: Completed
Appendix 3:	Main Python Code	Leads: Kirk/Justin Team members review	Week of 12/9-12/12	Status: Completed
Appendix 4:	Table of each individual team member's major and minor responsibilities.	Lead: Dwayne	Week of 12/9-12/12	Status: Completed
References	Books and/or refereed articles.	All team members	Week of 12/9-12/12	Status: Completed
Editing and Revision	Finalize document for submission.	Lead: John All team members	12/12	Status: Completed