# !/usr/bin/env python

“”“Web server for the Trendy Lights application.

The overall architecture looks like:

server.py script.js

| | | | | | | EE | <-> | App Engine | <-> | Browser | |\_\_\_\_\_\_| |\_\_\_\_\_\_\_\_\_\_\_\_| |\_\_\_\_\_\_\_\_\_|   / ‘- - - - - - - - - - - - - - -’

The code in this file runs on App Engine. It’s called when the user loads the web page and when details about a polygon are requested.

Our App Engine code does most of the communication with EE. It uses the EE Python library and the service account specified in config.py. The exception is that when the browser loads map tiles it talks directly with EE.

The basic flows are:

1. Initial page load

When the user first loads the application in their browser, their request is routed to the get() function in the MainHandler class by the framework we’re using, webapp2.

The get() function sends back the main web page (from index.html) along with information the browser needs to render an Earth Engine map and the IDs of the polygons to show on the map. This information is injected into the index.html template through a templating engine called Jinja2, which puts information from the Python context into the HTML for the user’s browser to receive.

Note: The polygon IDs are determined by looking at the static/polygons folder. To add support for another polygon, just add another GeoJSON file to that folder.

1. Getting details about a polygon

When the user clicks on a polygon, our JavaScript code (in static/script.js) running in their browser sends a request to our backend. webapp2 routes this request to the get() method in the DetailsHandler.

This method checks to see if the details for this polygon are cached. If yes, it returns them right away. If no, we generate a Wikipedia URL and use Earth Engine to compute the brightness trend for the region. We then store these results in a cache and return the result.

Note: The brightness trend is a list of points for the chart drawn by the Google Visualization API in a time series e.g. [[x1, y1], [x2, y2], …].

Note: memcache, the cache we are using, is a service provided by App Engine that temporarily stores small values in memory. Using it allows us to avoid needlessly requesting the same data from Earth Engine over and over again, which in turn helps us avoid exceeding our quota and respond to user requests more quickly.

“”"

import json import os

import config import ee import jinja2 import webapp2

from google.appengine.api import memcache

import urllib

# Web request handlers.

class MainHandler(webapp2.RequestHandler): “”“A servlet to handle requests to load the main Trendy Lights web page.”“”

def get(self, path=‘’): self.response.headers[’Content-Type’] = ’text/plain’ self.response.out.write(‘helloworld’)

class DetailsHandler(webapp2.RequestHandler): “”“A servlet to handle requests for details about a Polygon.”“”

def get(self): “”“Returns details about a polygon.”“” wdpaid = self.request.get(‘wdpaid’) content = Calculate\_water(wdpaid) self.response.headers[‘Content-Type’] = ‘application/json’ self.response.out.write(content)

class FeatureHandler(webapp2.RequestHandler): “”“A servlet to handle requests for details about a Polygon.”“”

def get(self): “”“Returns details about a polygon.”“” wdpaid = self.request.get(‘wdpaid’) content = Get\_feature(wdpaid) self.response.headers[‘Content-Type’] = ‘application/json’ self.response.out.write(content)

# Define webapp2 routing from URL paths to web request handlers. See:

# http://webapp-improved.appspot.com/tutorials/quickstart.html

app = webapp2.WSGIApplication([ (‘/getfeature’, FeatureHandler), (‘/details’, DetailsHandler), (‘/’, MainHandler),], debug=True)

# 24 hours after they are added. See:

# https://cloud.google.com/appengine/docs/python/memcache/

MEMCACHE\_EXPIRATION = 60 \* 60 \* 24

# Helpers

ppapi\_token = ‘4290b88825725a4d241c485d3b0b7cd7’

def get\_feature(wdpaid, token=ppapi\_token): url = “https://api.protectedplanet.net/v3/protected\_areas/{}?token={}&with\_geometry=1”.format(wdpaid, token) response = urllib.urlopen(url) data = json.loads(response.read()) data = data[‘protected\_area’] # geom geom = data[‘geojson’][‘geometry’] # attrs data.pop(‘geojson’) return ee.Feature(geom, data)

def Get\_feature(wdpaid): return json.dumps(get\_feature(wdpaid).getInfo())

def Calculate\_water(wdpaid):

my\_scale = 150 gsw = ee.Image(‘JRC/GSW1\_0/GlobalSurfaceWater’) transition = gsw.select(‘transition’) lookup\_names = zip(ee.List(gsw.get(‘transition\_class\_values’)).getInfo(), gsw.get(‘transition\_class\_names’).getInfo())

fc = ee.FeatureCollection(ee.List([get\_feature(wdpaid)])) surwater = ee.Image.pixelArea().addBands(transition) sums = surwater.reduceRegions( scale=my\_scale, collection=fc, reducer=ee.Reducer.sum().group(groupField=1, groupName=‘transition\_class\_value’) )

stats = ee.List(sums.first().get(‘groups’)).getInfo() return json.dumps(stats)

# Initialization.

# Use our App Engine service account’s credentials.

EE\_CREDENTIALS = ee.ServiceAccountCredentials( config.EE\_ACCOUNT, config.EE\_PRIVATE\_KEY\_FILE)

# Read the polygon IDs from the file system.

# Create the Jinja templating system we use to dynamically generate HTML. See:

# http://jinja.pocoo.org/docs/dev/

JINJA2\_ENVIRONMENT = jinja2.Environment( loader=jinja2.FileSystemLoader(os.path.dirname(**file**)), autoescape=True, extensions=[‘jinja2.ext.autoescape’])

# Initialize the EE API.

ee.Initialize(EE\_CREDENTIALS)