CARTOMAPPING – BACKEND

La aplicación nativa tiene dos funcionalidades importantes:

- La capacidad de realizar un SELECT a la base de datos
- La capacidad de realizar un INSERT en la base de datos.

Para ello, se ha desarrollado una arquitectura basada en Flask, mod_wsgi y diversos conocimientos sobre Apache, Postgres y Python, además de una librería denominada “psycopg2”.

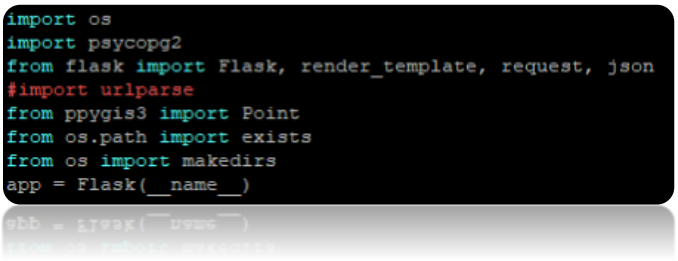
DESARROLLO DE LA APLICACIÓN FLASK

Flask es un microframework para Python basado en Werkzeug, Jinja 2 y buenas intenciones.

La aplicación Flask se encuentra en el directorio “html” del servidor. En él, se encuentran dos directorios importantes: “templates” y “static”. El primero contiene las plantillas html y el segundo los scripts, estilos, datos y demás. Sin embargo, lo único que necesitamos para la aplicación son varios archivos en el directorio de Flask:

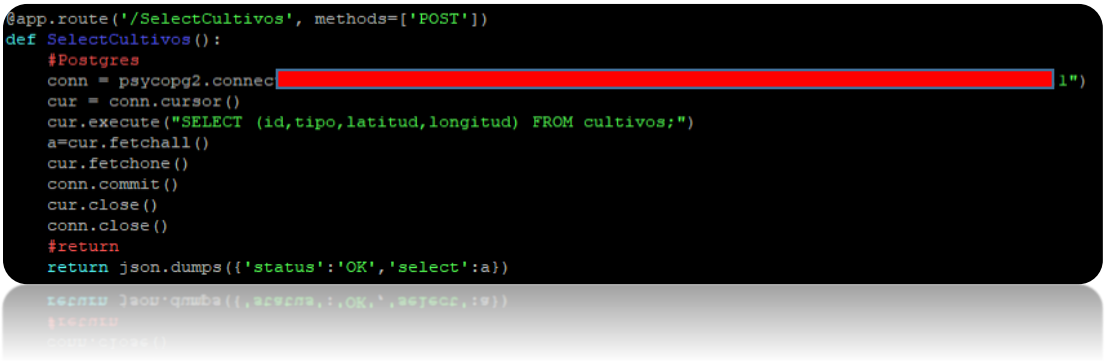
- Flaskapp.py

Se importan las librerias.



```
import os
import psycopg2
from flask import Flask, render_template, request, json
#import urlparse
from ppygis3 import Point
from os.path import exists
from os import makedirs
app = Flask(__name__)
```

Se define el routing para el SELECT de la capa CULTIVOS.

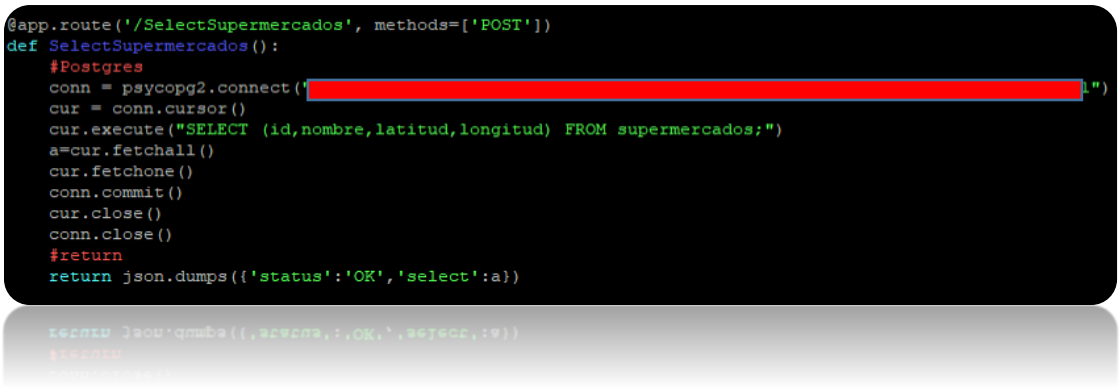


```
@app.route('/SelectCultivos', methods=['POST'])
def SelectCultivos():
    #Postgres
    conn = psycopg2.connec                                                    l")
    cur = conn.cursor()
    cur.execute("SELECT (id,tipo,latitud,longitud) FROM cultivos;")
    a=cur.fetchall()
    cur.fetchone()
    conn.commit()
    cur.close()
    conn.close()
    #return
    return json.dumps({'status':'OK','select':a})
```

Se define el routing para el SELECT de la capa PUNTOS DE INTERES.

```
@app.route('/SelectPuntos', methods=['POST'])
def SelectPuntos():
    #Postgres
    conn = psycopg2.connect("[redacted]1")
    cur = conn.cursor()
    cur.execute("SELECT (id,nombre,descripcion,latitud,longitud) FROM puntos_interes;")
    a=cur.fetchall()
    cur.fetchone()
    conn.commit()
    cur.close()
    conn.close()
    #return
    return json.dumps({'status':'OK','select':a})
```

Se define el routing para el SELECT de la capa SUPERMERCADOS.



```
@app.route('/SelectSupermercados', methods=['POST'])
def SelectSupermercados():
    #Postgres
    conn = psycopg2.connect("[redacted]1")
    cur = conn.cursor()
    cur.execute("SELECT (id,nombre,latitud,longitud) FROM supermercados;")
    a=cur.fetchall()
    cur.fetchone()
    conn.commit()
    cur.close()
    conn.close()
    #return
    return json.dumps({'status':'OK','select':a})
```

Las tres funciones devuelven un json con los puntos de la base de datos.

Se define el routing para el INSERT de la capa CULTIVOS.

```
@app.route('/InsertCultivos', methods=['POST'])
def InsertCultivos():
    SQL1 = """INSERT INTO cultivos (tipo,latitud,longitud) values (%s,%s,%s);"""
    tipo1 = request.form['tipoCultivo']
    lat1 = float(request.form['latitud1'])
    lon1 = float(request.form['longitud1'])
    data1 = (tipo1,lat1,lon1,)
    conn1 = psycopg2.connect("[redacted]")
    cur1 = conn1.cursor()
    cur1.execute(SQL1,data1)
    conn1.commit()
    cur1.close()
    conn1.close()
    return json.dumps({'insert':'OK'})
```

Se define el routing para el INSERT de la capa PUNTOS DE INTERES.

```
@app.route('/InsertPuntos', methods=['POST'])
def InsertPuntos():
    SQL2 = """INSERT INTO puntos_interes (nombre,descripcion,latitud,longitud) values (%s,%s,%s,%s);"""
    nombre2 = request.form['nombrePunto']
    descripcion = request.form['descripcion']
    lat2 = float(request.form['latitud2'])
    lon2 = float(request.form['longitud2'])
    data2 = (nombre2,descripcion,lat2,lon2,)
    conn2 = psycopg2.connect("[redacted]")
    cur2 = conn2.cursor()
    cur2.execute(SQL2,data2)
    conn2.commit()
    cur2.close()
    conn2.close()
    return json.dumps({'insert':'OK'})
```

Se define el routing para el INSERT de la capa SUPERMERCADOS.

```
@app.route('/InsertSupermercados', methods=['POST'])
def InsertSupermercados():
    SQL3 = """INSERT INTO supermercados (nombre,latitud,longitud) values (%s,%s,%s);"""
    nombre3 = request.form['nombreSupermercado']
    lat3 = float(request.form['latitud3'])
    lon3 = float(request.form['longitud3'])
    data3 = (nombre3,lat3,lon3,)
    conn3 = psycopg2.connect("[redacted]")
    cur3 = conn3.cursor()
    cur3.execute(SQL3,data3)
    conn3.commit()
    cur3.close()
    conn3.close()
    return json.dumps({'insert':'OK'})
```

El programa termina en:

```
if __name__ == '__main__':
  app.run()
```

- Flaskapp.wgsi

Creamos este archive para cargar la aplicación.

```
import sys
sys.path.insert(0, '/var/www/html/Flask')

from flaskapp import app as application
```

Habilitar mod_wsgi

El servidor apache muestra páginas html de forma predeterminada, pero para servir contenido dinámico desde una aplicación Flask tendremos que hacer algunos cambios. En el archivo de configuración de Apache (/etc/apache2/sites-enabled:000-default.conf), añadimos lo siguiente:

```
VirtualHost *:80>
      # The ServerName directive sets the request scheme, hostname and port that
      # the server uses to identify itself. This is used when creating
      # redirection URLs. In the context of virtual hosts, the ServerName
      # specifies what hostname must appear in the request's Host: header to
      # match this virtual host. For the default virtual host (this file) this
      # value is not decisive as it is used as a last resort host regardless.
      # However, you must set it for any further virtual host explicitly.
      #ServerName www.example.com

      ServerAdmin webmaster@localhost
      DocumentRoot /var/www/html

      WSGIDaemonProcess flaskapp threads=5
      WSGIScriptAlias / /var/www/html/Flask/flaskapp.wsgi

      <Directory Flask>
       Header set Access-Control-Allow-Origin "*"
       WSGIProcessGroup flaskapp
       WSGIApplicationGroup %{GLOBAL}
       Order deny,allow
       Allow from all
      </Directory>

      # Available loglevels: trace8, ..., trace1, debug, info, notice, warn,
      # error, crit, alert, emerg.
      # It is also possible to configure the loglevel for particular
      # modules, e.g.
      #LogLevel info ssl:warn

      ErrorLog ${APACHE_LOG_DIR}/error.log
      CustomLog ${APACHE_LOG_DIR}/access.log combined

      # For most configuration files from conf-available/, which are
      # enabled or disabled at a global level, it is possible to
      # include a line for only one particular virtual host. For example the
      # following line enables the CGI configuration for this host only
      # after it has been globally disabled with "a2disconf".
      #Include conf-available/serve-cgi-bin.conf
/VirtualHost>
```

Para finalizer, reestablecemos el servidor:

```
$ sudo apachectl restart
```

Configuración de la aplicación de Flask para Elastic Beanstalk de AWS

Primero, se crea un entorno virtual con virtualenv y se utilizará para instalar Flask y sus dependencias. Al utilizar un entorno virtual, se puede determinar exactamente qué paquetes son necesarios para la aplicación a fin de que los paquetes necesarios se instalen en las instancias EC2 que ejecutan la aplicación.

Posteriormente, implementamos el sitio con la CLI de EB, con el fin de crear un entorno e implementar la aplicación de Flask. Para más infomación, consultar este enlace d AWS:

https://docs.aws.amazon.com/es_es/elasticbeanstalk/latest/dg/create-deploy-python-flask.html