LOCATION
LOCATION
LOCATION

JULIA GRACE

@JEWELIA

WHOAMI

- @jewelia
- 1st Engineering hire at Tindie
  - tindie.com
  - “Etsy for Hardware Hackers”
- BS & MS in Computer Science
- Veteran of a few startups & IBM Research
- This talk goes over work I did in Fall 2012 at WeddingLovely.

WHAT YOU'LL GET OUT OF THIS TALK

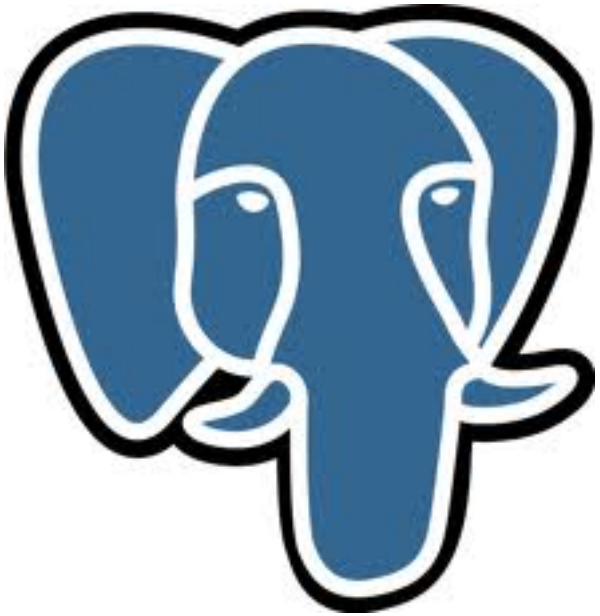
- We'll walk through how to encode, store and search geospatial data.
  - Think: input box that allow users to input any geographic data (e.g. city, state, zip, country).
- Queries for data nearby or within a radius.
- You see this all over the web.
- Example: Yelp

OUR STACK

Python 2.7

Django 1.4.3

PostgreSQL 9.2



BEFORE MOVING TO SPATIAL DB...

- Hard coded list of strings in a list shown in a dropdown.
- Cheap, easy and works most of the time.

```
LOCATIONS = [
    ('United States',
        [
            ('albany', 'Albany, NY'),
            ('albuquerque', 'Albuquerque, NM'),
            ('anchorage', 'Anchorage, AK'),
            ('annapolis', 'Annapolis, MD'),
            ('atlanta', 'Atlanta, GA'),
            ('austin', 'Austin, TX'),
            ('baltimore', 'Baltimore, MD'),
            ('boston', 'Boston, MA'),
        ]
    )
]

location = models.CharField(
        max_length=100,
        choices=settings.LOCATIONS)
```

AFTER MOVING TO SPATIAL DB...

- "Omnibox" that can handle (almost) any input.

```
location = PointField(
        help_text="Represented as (longitude, latitude)",
        geography=True,
        dim=3)
```

BE SMART (LISTS AIN'T ALL BAD)

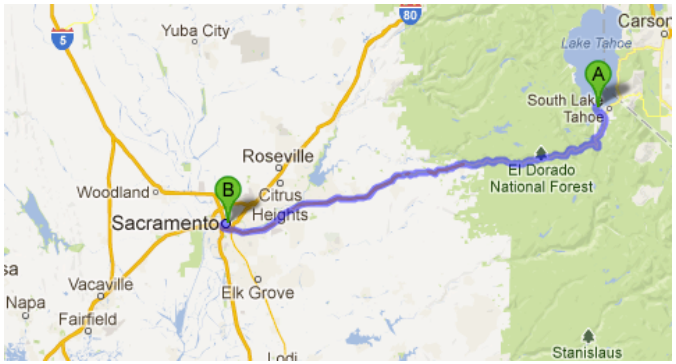
- **Not every application needs a spatial database and the ability to do distance lookups.**
- **Did I mention that lists are:**
  - Fast to implement.
  - Need less infrastructure.
  - Might get you 75% (or even 90%) of the way there.

...BUT LISTS WERE NOT IDEAL FOR OUR USER-FACING SEARCH

- **Good short term solution. Bad long term solution**
- **Alphabetically distant cities might be geographically close.**

```
LOCATIONS = [
    ('United States',
        [
            ('lake_tahoe', 'Lake Tahoe, CA'),
            ('little_rock', 'Little Rock, AR'),
            ('los_angeles', 'Los Angeles, CA'),
            ('memphis', 'Memphis, TN'),
            ('miami', 'Miami, FL'),
            ('montgomery', 'Montgomery, AL'),
            ('nashville', 'Nashville, TN'),
            ('new_orleans', 'New Orleans, LA'),
            ('new_york', 'New York, NY'),
            ('norfolk', 'Norfolk, VA'),
            ('phoenix', 'Phoenix, AZ'),
            ('pittsburgh', 'Pittsburgh, PA'),
            ('portland', 'Portland, OR'),
            ('providence', 'Providence, RI'),
            ('raleigh', 'Raleigh, NC'),
            ('san_antonio', 'San Antonio, TX'),
            ('scranton', 'Scranton, PA'),
            ('sacramento', 'Sacramento, CA'),
        ]
    )
]
```

Lake Tahoe and Sacramento are “close” enough that search results should contain objects in both locations.



… JUST TO NOTE

- **I had no idea how to do this when I started.**
  - No extensive experience with spatial databases or geocoding.

- **Hopefully the lessons I learned will save you time/energy/cash money.**

...TO THE SPATIAL DB AND BEYOND!

Process of converting an existing Django application (or building a new application) to use GeoDjango and handle location input:

(1) Database

(2) Application layer

(3) Front-end

(4) Bonus Round!

(1) DATABASE: OPTIONS

1. **No spatial database and do the math ourselves.**
   - Store lat/long at decimals and mathematically compute distance between them using Haversine formula:

```
haversineDistance[p1_, p2_] :=
 Module[{d, a, r, lat1, lon1, lat2, lon2, dlon, dlat, c},
  r = 6371000;
  lat1 = p1[[1]] Degree; lon1 = p1[[2]] Degree;
  lat2 = p2[[1]] Degree; lon2 = p2[[2]] Degree;
  dlat = lat2 - lat1; dlon = lon2 - lon1;
  a = Sin[dlat/2]^2 + Cos[lat1]*Cos[lat2]*Sin[dlon/2]^2;
  c = 2*ArcTan[Sqrt[1 - a], Sqrt[a]];
  d = r*c;
  N[d]
  ]
```

Haversine formula in Mathematica

2. **Use a spatial database (e.g. PostGIS) and compute distances at the DB level (but then we might as well just be writing straight SQL):**

```
SELECT ST_Distance(a.geom, b.geom) FROM points_table a, points_table b WHERE a.id='x' AND b.id='y';
```

3. **Spatial database + GeoDjango.**
   - GeoDjango = Django's API for accessing geographic data and doing distance lookups (among other calculations) on that data.

(1) DATABASE: INSTALL POSTGRESQL 9.2

- We were running PostgreSQL 9.1.3 on it's on Ubuntu 12.04 LTS (precise).
- Decided to upgrade to 9.2 after struggling with 9.1.3
- At the time (10/2012), 9.2 was not available via aptitude or apt-get.

```
sudo add-apt-repository ppa:pitti/postgresql
sudo apt-get update
sudo apt-get install postgres-9.2 postgresql-common postgresql-server-dev-9.2
```

- Reference: http://askubuntu.com/questions/186610/how-do-i-upgrade-to-postgres-9-2

(1) DATABASE: INSTALL SPATIAL LIBRARIES

- List of needed libraries: https://docs.djangoproject.com/en/dev/ref/contrib/gis/install/geolibs/
- Versions matter! When using PostgreSQL 9.2 you must use (in this order):
  - GEOS 3.3.3+
  - GDAL 1.9+
  - PostGIS 2.0+
- Don't install PostGIS before GEOS or face certain doom.
- Complete list of versions that play well: http://trac.osgeo.org/postgis/wiki/UsersWikiPostgreSQLPostGIS

Installing Geospatial libraries

GeoDjango uses and/or provides interfaces for the following open source geospatial libraries:

Program	Description	Required	Supported Versions
GEOS	Geometry Engine Open Source	Yes	3.3, 3.2, 3.1, 3.0
PROJ.4	Cartographic Projections library	Yes (PostgreSQL and SQLite only)	4.8, 4.7, 4.6, 4.5, 4.4
GDAL	Geospatial Data Abstraction Library	No (but, required for SQLite)	1.9, 1.8, 1.7, 1.6, 1.5
GeoIP	IP-based geolocation library	No	1.4
PostGIS	Spatial extensions for PostgreSQL	Yes (PostgreSQL only)	2.0, 1.5, 1.4, 1.3
SpatiaLite	Spatial extensions for SQLite	Yes (SQLite only)	3.0, 2.4, 2.3

(1) DATABASE: INSTALL PROPER DEPENDENCIES FOR YOUR OS

Install PostGIS dependencies:

```
apt-get install libxml2-dev proj libjson0-dev xsltproc docbook-xsl docbook-mathml
```

Build GEOS 3.3.x:

```
wget http://download.osgeo.org/geos/geos-3.3.8.tar.bz2
tar xvfj geos-3.3.8.tar.bz2
cd geos-3.3.8
./configure
make
sudo make install
```

Build PostGIS:

```
wget http://download.osgeo.org/postgis/source/postgis-2.0.3.tar.gz
tar xfvz postgis-2.0.3.tar.gz
cd postgis-2.0.3
./configure
make
sudo make install
sudo ldconfig
sudo make comments-install
```

Reference (with detailed explanations I can't fit here):
http://trac.osgeo.org/postgis/wiki/UsersWikiPostGIS20Ubuntu1204src

(1) DATABASE: LOAD UP THOSE SPATIAL LIBS

1. Start PostgreSQL and create a DB (if you have an existing DB you'd like to use, you can skip this step).

```
createdb YOUR_DB_NAME
```

2. Create the spatial database template:

```
createlang -d YOUR_DB_NAME plpgsql
```

3. Load PostGIS SQL routines:

```
psql -d YOUR_DB_NAME -f /usr/share/postgresql/9.2/contrib/postgis-2.0/postgis.sql
psql -d YOUR_DB_NAME -f /usr/share/postgresql/9.2/contrib/postgis-2.0/spatial_ref_sys.sql
```

4. Enable users to alter spatial tables:

```
psql YOUR_DB_NAME
YOUR_DB_NAME=# GRANT ALL ON geometry_columns TO PUBLIC;
YOUR_DB_NAME=# GRANT ALL ON geography_columns TO PUBLIC;
YOUR_DB_NAME=# GRANT ALL ON spatial_ref_sys TO PUBLIC;
```

(1) DATABASE: STACK SCRIPT TO INSTALL POSTGRES 9.2 + SPATIAL DB

If you're on Linode, here's a stack script I wrote:
http://www.linode.com/stackscripts/view/?StackScriptID=5425

(sorry for the eye test)

```
source <ssinclude StackScriptID="123"> ## lib-system-ubuntu {
    # Set hostname
    if [ "$HOSTNAME" ]; then
        system_update_hostname "$HOSTNAME"
    fi

    # Add users
    if [ "$USERNAME" ]; then
        system_add_user "$USERNAME" "$PASSWORD" "sudo"
        system_user_add_ssh_key "$USERNAME" "$SSH_KEY"
    fi

    if [ "$DO_DEPLOY" == "Yes" ]; then
        system_add_user "$DEPLOY_USERNAME" "$PASSWORD" "sudo"
        system_user_add_ssh_key "$DEPLOY_USERNAME" "$SSH_KEY"
    fi

    # SSH
    system_sshd_passwordauthentication "no"
    system_sshd_permitrootlogin "no"
    system_sshd_pubkeyauthentication "yes"
## }

# Essentials
aptitude -y install python-software-properties
aptitude -y install build-essential python-dev git-core mailutils

# Install PostgreSQL 9.2?
if [ "$DO_POSTGRESQL" == "Yes" ]; then
    add-apt-repository ppa:pitti/postgresql -y
    apt-get update
    apt-get install postgresql-9.2 postgresql-common postgresql-server-dev-9.2

    #prereqs for postgis
    apt-get install libxml2-dev proj libjson0-dev xsltproc docbook-xsl docbook-mathml


    if [ "$POSTGRESQL_USER" -a "$POSTGRESQL_DATABASE" ]; then
        # Create postgres user
        echo "CREATE ROLE $POSTGRESQL_USER WITH LOGIN ENCRYPTED PASSWORD '$POSTGRESQL_PASSWORD';" | sudo -u postgres psql

        # Create postgres database
        sudo -u postgres createdb --owner "$POSTGRESQL_USER" "$POSTGRESQL_DATABASE"
    fi
fi

# Notify email
if [ "$NOTIFY_EMAIL" ]; then
    IP_ADDRESS="$(system_primary_ip)"
    mail -s "Linode Stackscript Deployed" $NOTIFY_EMAIL << EOF
Your server is deployed at $IP_ADDRESS.

Happy querying!
EOF
fi;
```

(1) DATABASE: DOES YOUR SPATIAL DB WORK?

- How to verify PostGIS *actually works*:

```
psql -d YOUR_DB_NAME -c "select PostGIS_full_version()"

                                                postgis_full_version
-----------------------------------------------------------------------------------------------------
 POSTGIS="2.0.1 r9979" GEOS="3.3.5-CAPI-1.7.5" PROJ="Rel. 4.8.0, 6 March 2012" LIBXML="2.7.3"
(1 row)
```

- I did not install the raster libraries, so you can ignore the warnings that may appear.

HOOK UP POSTGRES + POSTGIS TO DJANGO

- **PostGIS 2.0 doesn't play well with Django: https://code.djangoproject.com/ticket/16455**
- **Modify the PostGIS DB adapter**
  - Copy postgis/ directory from https://github.com/django/django/tree/master/django/contrib/gis/db/backends/postgis to your local development directory.
  - I copied it into a lib/postgis/ in my Django project.
  - Update settings.py (next slide has example) to point to the new DB adapter.
  - Make these changes: https://code.djangoproject.com/attachment/ticket/16455/16455-r17171-v4.patch

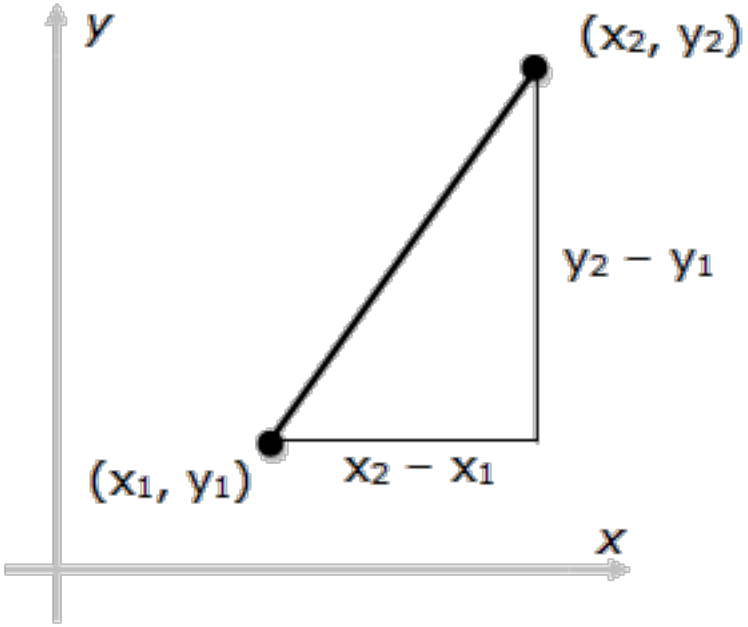
HOOK UP POSTGRES + POSTGIS TO DJANGO

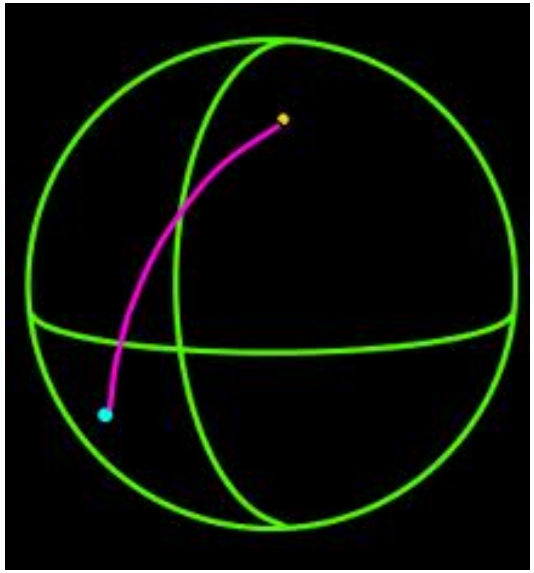
- **Example of settings.py pointing to local copy of the postgis DB adapter:**

```
# from settings.py

DATABASES = {
 'default': {
     'ENGINE': 'lib.postgis',
     'NAME': 'your_db_name',
     'USER': 'your_username',
     'PASSWORD': '',
     'HOST': '127.0.0.1',
     'PORT': '5432',
 }
}

# When using a custom backend, South needs to know what it is
SOUTH_DATABASE_ADAPTERS = {
    'default':'south.db.postgresql_psycopg2'
}
```

GEOGRAPHY INTERLUDE

- Distance between 2 points on a plane is not computationally intensive to calculate.
- ..but the Earth isn't flat and doing geometric calculations require more complex mathematics.





(2) APPLICATION LAYER: FUN BEGINS

- **Modifications to your existing models:**

```
from django.contrib.gis.db import models as geomodels

class Location(models.Model):

    # … existing attributes go here
    name = models.CharField(max_length=255)

    # GeoDjango PointField used to store a point with 3 dimensions
    point = geomodels.PointField(geography=True, dim=3, blank=True, null=True)

    # You MUST use GeoManager to make spatial queries
    objects = geomodels.GeoManager()
```

geography=true uses spherical representation of the Earth instead of plane (flat) representation. For short distances plane will work (and is faster to compute) but for longer distances you should account for the curvature of the Earth or else you're distances will inaccurate.

More info: https://docs.djangoproject.com/en/dev/ref/contrib/gis/model-api/#geography

(2) APPLICATION LAYER: POINT FIELD EXAMPLE

- Point = longitude/latitude representation of a point on Earth.
- Creating and saving a Point in Django ORM:

```
from decimal import Decimal
from django.contrib.gis.geos import Point, fromstr

from yourapp.models import Location

latitude = Decimal(37.3542)
longitude = Decimal(121.9542)

# 2 ways of storing points
# Both of these are equivalent
point = fromstr("POINT(%s %s)" % (longitude, latitude))
point = Point(longitude, latitude)

# Location is an object you define in your models.py
location = Location.objects.create(point=point)
```

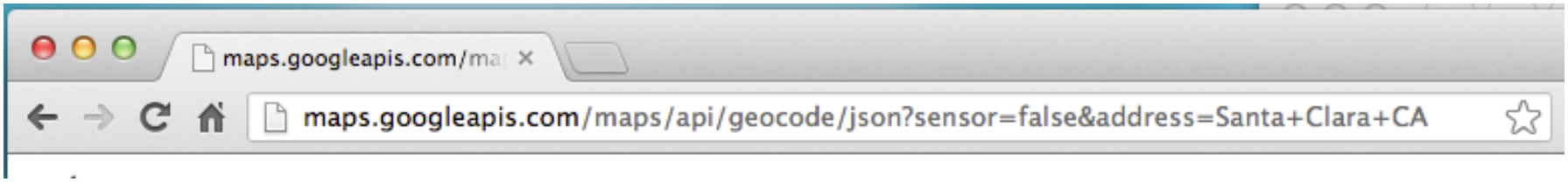
INTERLUDE: WHAT IS GEOCODING

- **Process of translating data (e.g. strings such as "94040" or "Santa Clara") and finding the associated geographic coordinates such as latitude/longitude.**

- **Many public and free APIs to do this for you.**
- **One of most popular is Google's Geocoding API: https://developers.google.com/maps/documentation/geocoding/**

INTERLUDE: WHAT IS GEOCODING

Type this into your browser:



BOOM!

- Response: a lot of JSON data!
- You might not need all of it; I used:
  - formatted_address
  - location lat/long

- If you are going to be spending a lot of time reading JSON in a web browser, here are some plugins to make your life easier:
- https://twitter.com/jewelia/status/257997860451258369

julia h grace
@jewelia

Here some sweet plugins that make in-browser JSON much more human readable: FF (addons.mozilla.org/en-US/firefox/...) & Chrome (chrome.google.com/webstore/detai...)


```
{
    "results": [
        {
            "address_components": [
                {
                    "long_name": "Santa Clara",
                    "short_name": "Santa Clara",
                    "types": [
                        "locality",
                        "political"
                    ]
                },
                {
                    "long_name": "Santa Clara",
                    "short_name": "Santa Clara",
                    "types": [
                        "administrative_area_level_2",
                        "political"
                    ]
                },
                {
                    "long_name": "California",
                    "short_name": "CA",
                    "types": [
                        "administrative_area_level_1",
                        "political"
                    ]
                },
                {
                    "long_name": "United States",
                    "short_name": "US",
                    "types": [
                        "country",
                        "political"
                    ]
                }
            ],
            "formatted_address": "Santa Clara, CA, USA",
            "geometry": {
                "bounds": {
                    "northeast": {
                        "lat": 37.418939,
                        "lng": -121.9297351
                    },
                    "southwest": {
                        "lat": 37.3228419,
                        "lng": -122.00537
                    }
                },
                "location": {
                    "lat": 37.3541079,
                    "lng": -121.9552356
                },
                "location_type": "APPROXIMATE",
                "viewport": {
                    "northeast": {
                        "lat": 37.418939,
                        "lng": -121.9297351
                    },
                    "southwest": {
                        "lat": 37.3228419,
                        "lng": -122.00537
                    }
                }
            },
            "types": [
                "locality",
                "political"
            ]
        }
    ],
    "status": "OK"
}
```

(2) APPLICATION LAYER: NEED TO GET LAT/LONG FOR LEGACY STRING DATA?

- **What if you need to geocode legacy data (e.g. you stored "San Francisco, CA, USA)?**

- **Simple example using Google's Geocoding API: https://developers.google.com/maps/documentation/geocoding/**

(2) APPLICATION LAYER: NEED TO GET LAT/LONG FOR LEGACY STRING DATA?

```python
import requests

from decimal import Decimal
from django.contrib.gis.geos import Point

from yourapp.models import Location

def Geocode():
    url = 'https://maps.googleapis.com/maps/api/geocode/json'

    # Location data must be '+' delimited
    location_string = 'Sacramento+CA'

    # Must tell Google you're not a sensor overlord
    payload = { 'sensor':'false','address':location_string }

    r = requests.get(url, params=payload)
    data = r.json
    if data['status'] == 'OK':
        latitude = data['results'][0]['geometry']['location']['lat']
        longitude = data['results'][0]['geometry']['location']['lng']

    # Location is an object you define in your models.py
    location = Location.objects.create(point=Point(Decimal(lng), Decimal(lat)))
```

TIP: POINTS ARE STORED AS GEOMETRIES

```
your_db=# select point from YOUR_DB;

                    point
-----------------------------------------------------
0101000020E610000058CBF852D3C351C087F0790FE12D4540
```

WTF? Where are the lat/long values?

```
your_db=# select ST_AsText(point) from YOUR_DB;

           st_astext
-----------------------------
 POINT(-71.0597732 42.3584308)
```

More fancy PostGIS functions for your pleasure:
http://postgis.refractions.net/documentation/manual-1.5/ch08.html#PostGI

(2) APPLICATION LAYER: MAKING SPATIAL QUERIES

- Query for all objects within a specified radius.
- Great for situations where having no results is okay (if you have no data within the radius specified).

```
import django.contrib.gis.db import models as geomodels
import django.contrib.gis.measure.D

max_distance = 25
ref_point = geomodels.Point(-71.0597732 42.3584308)

# Example querying for locations <=25 miles of a lat/long
locations = Location.objects.filter(point__distance_lte=(ref_point, D(mi=max_distance)))

# Example querying for locations <=25 km of a lat/long
locations = Location.objects.filter(point__distance_lte=(ref_point, D(km=max_distance)))
```

distance_lte = distance less than equal
distance_gte = distance greater than equal
Full list of lookups:
https://docs.djangoproject.com/en/dev/ref/contrib/gis/db-api/#spatial-lookup-compatibility

(2) APPLICATION LAYER: MAKING SPATIAL QUERIES

- Query for all objects sorted by distance from a lat/long.
- Useful for times when you don't know if querying for objects within a radius (e.g. 25 miles) will return any results.
- This guarantees you will have results (if you have data :)

```
from decimal import Decimal
from django.contrib.gis.geos import Point, fromstr
import django.contrib.gis.measure.D

from yourapp.models import Location

latitude = Decimal(37.3542)
longitude = Decimal(121.9542)
point = Point(longitude, latitude)

# All locations sorted by distance from a lat/long
locations = Location.objects.all().distance(point).order_by('distance')
```

(3) FRONT END: GEOCODING USER INPUT

- **Two options: Geocode client side or server side**
- **Client side**
  - You have to write JS.
  - Many free geocoding APIs (like Google) rate limit you by IP address, so geocoding client side will likely mean you won't get rate limited if you have a lot of different users.
  - The terms of service of the Google Geocoding API require you to display a Google Map.
- **Server side**
  - You get to write Python (this *is* PyCon…)
  - You probably will be rate limited.

YET ANOTHER INTERLUDE

- My next startup idea
- Pre-order yours today!

(3) FRONT END: CLIENT SIDE GEOCODING USER INPUT

- **Forms.py**

```
class LocationForm(forms.Form):
    user_entered_address = forms.CharField(required=False, max_length=255)
    full_address = forms.CharField(max_length=255, required=False, widget=forms.HiddenInput())
    latitude = forms.CharField(required=False, widget=forms.HiddenInput())
    longitude = forms.CharField(required=False, widget=forms.HiddenInput())
```

HiddenInput because we are going to query for this data client side via Google Geocoding API.

- **HTML template:**

```
<form id="search_form" action="/search/" method="GET" class="form autosubmit">
    {{ form.user_entered_address }}
</form>
```

(3) FRONT END: GEOCODING USER INPUT

- Example based on https://developers.google.com/maps/documentation/javascript/geocoding
- Geocode client side, append the lat/long data to the form before submission:

```
<script src="http://maps.google.com/maps/api/js?sensor=false"></script>
<script type="text/javascript">
$(document).ready(function() {
    var geocoder = new google.maps.Geocoder();
    $('#search_form').submit(function(e){
        e.preventDefault();
        var onSuccess = function(results, status) {
            if (status == google.maps.GeocoderStatus.OK) {
                result = results[0].geometry.location;
                $(this).prepend('<input type="hidden" name="latitude" value="' + result.lat() + '">');
                $(this).prepend('<input type="hidden" name="longitude" value="' + result.lng() + '">');
                $(this).prepend('<input type="hidden" name="formatted_address" value="' +
                                results[0].formatted_address + '">');
            }
           $(this).trigger('submit');
        }
        geocoder.geocode({'address': addr}, onSuccess);
    });
});
</script>
```

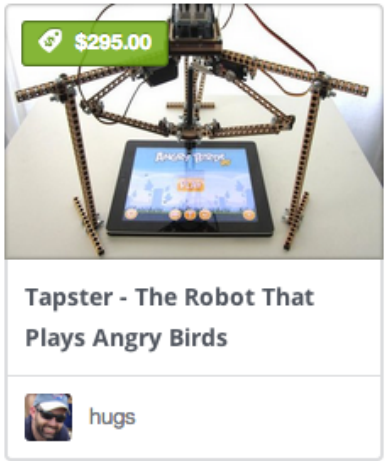
(3) FRONT END: GEOCODING USER INPUT

- **Views.py**

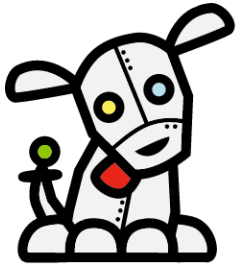
```
def search(request, directory):
    if request.method == 'GET':
        form = SearchForm(request.REQUEST)
        if form.is_valid():

            # Let's limit our distance to objects within 25 miles
            max_distance = 25

            # Pull geocoded input off the form
            formatted_address = form.cleaned_data['formatted_address']
            lat = form.cleaned_data['latitude']
            lng = form.cleaned_data['longitude']

            # Create a Point
            ref_pnt = fromstr("POINT(%s %s)" % (lng, lat))

            # Query for all objects within 25 miles from the point
            location = Location.objects.filter(point__distance_lte=(ref_pnt, D(mi=max_distance)))

            # Add to context and render a template
            return render_to_response('/search/results.html', {
                'locations': locations,
            }, context_instance=RequestContext(request))
```

FINAL INTERLUDE

- **Fancy things with middleware we tried at Tindie.**
- **Tindie has over 500 products from 200 sellers *worldwide*.**
- **Shipping rates *can* vary significantly based on country.**
- **Auto-detect country to show shipping rates?**



Robots shipped all around the world!



COUNTRIES

Australia
Canada
Germany
India
Ireland
Italy
Japan
Lithuania
Macedonia, The Former Yugoslav Republic of
Netherlands
Slovenia
South Africa
Switzerland
United Kingdom
United States

BONUS: AUTO DETECT LOCATION THROUGH MIDDLEWARE

- **There are services that map IP address to country.**
  - We used http://ipinfodb.com/

- IP address : **98.207.195.205**
- Country : UNITED STATES
- State/Province : CALIFORNIA
- City : LOS ALTOS
- Zip or postal code : 94022

Pretty darn accurate….

BONUS: AUTO DETECT LOCATION THROUGH MIDDLEWARE

- Add a Django Middleware to set the location in the session:
- If no location can be determined, default to US:

```
# Supply your own api_key
IPINFODB_API_KEY = "YOUR_KEY_HERE"
IPINFODB_URL = "http://api.ipinfodb.com/v3/ip-country/?key=%(api_key)s&ip=%(ip_addr)s&format=json"

class CountrySessionMiddleware(object):
    def process_request(self, request):
        if not "country" in request.session.keys():
            url = IPINFODB_URL % {
                "api_key": IPINFODB_API_KEY,
                "ip_addr": request.environ["REMOTE_ADDR"]
            }
            try:
                json_response = requests.get(url)
                data = json.loads(json_response.content)
                country = data.get("countryCode")
                if not country or country == "-":
                    country = "US"
                request.session["country"] = country
            except (requests.ConnectionError, requests.Timeout):
                pass

        return None
```

THANK YOU

This talk would not have been possible without feedback and input from these awesome people:

- **Tracy Osborn**
- **Andrey Petrov**
- **Kenneth Love**
- **Lynn Root**

...and PyLadies!

pyladies
ladies who love
coding in Python

QUESTIONS?

- **You can always find me at**
  - @jewelia
  - jewelia@gmail.com

- **I also have stickers, lots of stickers**