# dj-geocoding Documentation

***Release 0.2.1***

**Ben Lopatin**

December 02, 2014

Contents

dj-geocoding is a Django app that has two goals:

1. Provide basic geodata functionality without requiring PostGIS
2. Make geocoding accessible and easy to integrate in the Django admin

PostGIS is fantastic but isn't always available.

Contents:

CHAPTER 1

# Installing dj-geocoding

Install the package and add it to your project's requirements file:

```
pip install dj-geocoding
```

The app does not install any models so there's no need to add to your own project's *INSTALLED_APPS* list.

## 1.1 Configuration

For bulk geocoding the app currently uses Geocodio. Future versions are expected to support additional bulk geocoding services.

Add your API key to your project settings:

```
GEOCODIO_API_KEY = 'jskd823jqjdkjdj191'
```

**Note:** Ensure that this key is added to either a non-source controlled settings file or better yet is loaded via an environment variable. Secrets like this should never go in source control.

## 1.2 App configuration

See *configuring admin-based geocoding* for project integration guidance.

CHAPTER 2

# Geocoding in the Django admin

With a few additions to your own app's code you can enable filtering and geocoding directly in the Django admin interface.

You don't *need* to make every change identified here, but if you do not then you should ensure that your own code answers the implicit assumptions covered the configuration outlined.

## 2.1 Models and mixins

The *GeocoderMixin* provides a single method, *geocode*, which is useful in your own code but not used by the admin interface.

If your model does not already have latitude and longitude Decimal fields then build them with the *GeoFieldsModel* as the base model (or using the *GeoBase* class which combines the *GeoFieldsModel* and *GeocoderMixin*. The *GeoFieldsModel* adds the two Decimal fields required, the *point* property for getting and setting the location by a single tuple, and the *has_geolocation* method.

You'll also want to add a *get_display_address* method to your model. This method should take no extra arguments and should return a single string that contains the formatted address. E.g.:

```
class MyModel(GeoBase):
    street_address = models.CharField(max_length=100)
    city = models.CharField(max_length=30)
    state = models.CharField(max_length=2)
    zip = models.CharField(max_length=9)

    def get_display_address(self):
        return ", ".join([self.street_address, self.city, self.state, self.zip])
```

## 2.2 ModelAdmin configuration

The admin geocoding action lets you select one or more locations from your model in the Django admin and geolocate them using the admin's action dropdown.

To enable the admin action, use the *GeolocateMixin* mixin class when defining your *ModelAdmin* class.:

```
class MyModelAdmin(admin.ModelAdmin, GeolocateMixin):
    pass
```

As of version 0.2.1 the app provides only a count of those locations it tried to geocode; it does not discern between successful and failed attempts in its success message.

### 2.2.1 Filtering and listing

It's helpful to be able to see which locations are geocoded at a glance, and better yet, filter your list accordingly.

The *GeoFieldsModel* provides the annotated *has_geolocation* method which can be used as a *list_display* item:

```
class MyAdmin(admin.ModelAdmin):
    list_display = ('name', 'has_geolocation')
```

The annotation means that Django will display the proper visual indicators for whether this is true or not for each location.

The *GeocodedFilter* class assumes only that your model has latitude and longitude fields.:

```
class MyAdmin(admin.ModelAdmin):
    list_display = ('name', 'has_geolocation')
    list_filter = (GeocodedFilter,)
```

CHAPTER 3

# Indices and tables

- *genindex*
- *modindex*
- *search*