django-genia Documentation

Release 0.1.dev1

Taylor Mitchell

July 15, 2012

CONTENTS

django-genia is a relatively simple set of models and helpers for handling generational data.

CHAPTER

ONE

CONTRIBUTING

This project is open source and licensed under the BSD license. It is hosted at GitHub so head over to the project page if you want to report an issue, request a feature or contribute to its development.

CHAPTER

TWO

CONTENTS

2.1 Introduction

2.1.1 What is generational data?

Generational data refers to data within a Django app that is relevant together at a given point in time. We may create different records at different times, but we only want to display the records that go together. Each of these snapshots we call a *generation*.

2.1.2 Why would I want to use django-genia?

Sometimes you have data that takes a long time to create/import but that only makes sense when loaded completely. While the data is loading you may not want your users to see it.

django-genia makes it easy to support this workflow.

django-genia gives you simple hooks to put each data import into its own generation, and then make the generation active at the end. By default, queries against your generational models will only return records in the active generation.

2.1.3 Why not just import everything in one big transaction?

Good question! For our purposes, we need to commit transactions as we go for a number of reasons. The biggest is that we like to import data in parallel when possible and some of the tables we reference are shared. If you don't commit as you go, you're more likely to encounter `IntegrityError` and its ilk.

2.2 Usage

2.2.1 Getting Started

Prerequisites

- Django version 1.2 or greater (may work with earlier versions but untested)

Installation

Note: We assume you're using pip+virtualenv+virtualenvwrapper. If not, start now.

Install django-genia:

```
$ pip install django-genia
```

Setup

The easiest way to get started is to edit your Django app's `models.py` to use `genia.models.GenerationalModelMixin`:

```
from django.db import models
from genia.models import GenerationalModelMixin

class Person(GenerationalModelMixin, models.Model):
    name = models.CharField(max_length=100)

    def __unicode__(self):
        return self.name
```

This mixin will add a `generation` field to your model which is a `ForeignKey` to a `genia.models.Generation`. It also overrides your model's default manager to use `genia.models.GenerationalModelManager`.

If you need to customize this (e.g. your model already has its own custom manager), you can use the following template:

```
from django.db import models
from genia.models import Generation, GenerationalModelManager

class Person(models.Model):
    name = models.CharField(max_length=100)
    generation = models.ForeignKey(Generation)
    objects = YourCustomManager()
    active_objects = GenerationalModelManager()

    def __unicode__(self):
        return self.name
```

If your model defines fields with `unique=True`, they may not work as expected. In this case, you should update your model definition to use `unique_together` along with the `generation` field.

Usage

The basic use case is if you have to import a lot of data at once into an app, but you don't want it to "go live" until it's all done loading. In this case, you would follow the following workflow:

1. Create a new generation
2. Start importing your data against this new generation
3. Once the import completes, set the new generation to be active

Here's an interactive session showing this workflow:

```
>>> from genia.models import Generation
>>> from my_app.loader import load_people
>>> new_gen = Generation('my_app')
>>> load_people(generation=new_gen)
...
>>> new_gen.make_active()
```

2.3 API Reference

2.3.1 genia.models

Django models, managers and mixins

For supporting generational data with django-genia

class `genia.models.`**`Generation`**`(`**args*, ***kwargs*`)`
Bases: `django.db.models.base.Model`

Stores history of data generations for a given app

Parameters

- **id** (*AutoField*) – Id
- **app_name** (*CharField*) – App name
- **index** (*IntegerField*) – Generation number for this app
- **created** (*DateTimeField*) – Created
- **last_updated** (*DateTimeField*) – Timestamp of the last time the data was changed. Manually set.
- **active** (*BooleanField*) – Is this the active generation for the given app? Note that only one generation can be active at a time

`make_active``()`
Set the model to be the active generation for its app

If there is already an active generation, it will be unset.

`save``(`**args*, ***kwargs*`)`
Override base model save method

Automatically sets the index of a new generation to the right (incremented) value.

class `genia.models.`**`GenerationManager`**
Bases: `django.db.models.manager.Manager`

Manager for generation objects

`active``(`*app_name*`)`
Get the one active generation for the given app

Parameters **app_name** (*String*) – Name of the app

Returns Active generation

Return type Generation object

class `genia.models.`**`GenerationalModelManager`**

Bases: `django.db.models.manager.Manager`

Manager for models that are implementing generational data

`get_query_set``()`

Override the base get_query_set

Returns Current QuerySet filtered to only objects in the active generation

Return type QuerySet object

class `genia.models.`**`GenerationalModelMixin`**`(`**args*, ***kwargs*`)`

Bases: `django.db.models.base.Model`

Mixin class for models that represent generational data

Parameters **generation_id** (ForeignKey to `Generation`) – Generation

classmethod **`active_generation`**`()`

Get the active generation for this model's application data

Returns Active generation for the model's app

Return type Generation object

2.3.2 genia.utils

Helper functions and other misc code that is shared within django-genia

`genia.utils.`**`get_app_name_for_model`**`(`*model*`)`

Extract the app name from a model class

Parameters **model** – Generational model

Returns App name containing model

Return type String

CHAPTER

THREE

INDICES AND TABLES

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

PYTHON MODULE INDEX