Django-geoprisma Documentation

Release 0.0.1

Groupe conseil Nutshimit-Nippour

April 21, 2015

Contents

Documentation:

CHAPTER 1

Overview

`Django-geoprisma` is an implementation of Geoprisma in Django.

What is Geoprisma? Geoprisma is a web mapping application featuring an access control to geospatial data via a proxy and a dynamic user interface via servder-side XSLT.

What is `Django-geoprisma`? `Django-geoprisma` use the concept of Geoprisma but adapt to Django's architecture.

1.1 Features

- Access control to geospatial data
- Dynamic user interface(UI) using Django's template system

1.2 Incoming

- Tests
- Better documentation

CHAPTER 2

Installation

Install `Django-geoprisma`

```
pip install django-geoprisma
```

2.1 Configuration

Add this to yours installed apps

```
INSTALLED_APPS = (
  ...
  'modeltranslation',
  'django.contrib.gis',
  'geoprisma',
  'geoprisma.acl',
)
```

Note: If you want to use the admin integration with django-modeltranslation, `modeltranslation` must be put before `django.contrib.admin` (only applies when using Django 1.7 or above).

Specify yours languages for django-modeltranslation

```
gettext = lambda s: s
LANGUAGES = (
  ('en',gettext('English')),
  ...
)
```

Your template context processors must look like this

```
TEMPLATE_CONTEXT_PROCESSORS = (
    "django.contrib.auth.context_processors.auth",
    "django.core.context_processors.debug",
    "django.core.context_processors.i18n",
    "django.core.context_processors.media",
    "django.core.context_processors.static",
    "django.core.context_processors.tz",
    "django.contrib.messages.context_processors.messages",
    "django.core.context_processors.request"
)
```

Sync your database

```
python manage.py syncdb
```

Load initial data

```
python manage.py loaddata default
```

Now you are ready to use geoprisma.

CHAPTER 3

User Guide

3.1 Concepts

In this section, you will learn more about the terms and words commonly used in `Django-geoprisma`.

3.1.1 Resource

A Resource is associated to a unique set of data, but it can have many different Web Services serving it in different ways. For example, a Resources named GMap_Roads could have its data served by many different kind of servers.

The Access Control on Resources is managed by the ACL.

3.1.2 DataStore

It's the equivalent of a layer of a server, i.e. a LAYER of a mapfile (MapServer), or a layer in a configuration file (FeatureServer, TileCache), etc. DataStores are linked to a single Service.

3.1.3 Service

A Service is a GeoSpacial Web Server, like WMS.

Here's a list of the current Service available in `Django-geoprisma`:

- WMS
- FeatureServer
- MapCache
- MapServer
- GYMO (Google, Yahoo, Microsoft, OpenStreetMap)
- File
- WFS
- HTTPRequest

3.1.4 Widget

A widget is a feature of the UI application. Each widget can be associated to one or more Resources. The Resource has to have the according service defined by the Widget ServiceType to be able to use the widget properly.

For example : A QueryOnClick widget that has Action:read and ServiceType:WMS could be added to a Resource if that Resource has a WMS Service configured and would be added to the UI if the User has the read Permission to the related Resource.

3.1.5 ACL

ACL stands for Access Control List. It's the application responsible of managing the Permissions, Actions, Roles and Resources in `Django-geoprisma`.

3.1.6 Proxy

The Proxy of `Django-geoprisma` is the only point of entry for any queries made. For example, if we would want to make a WMS 'GetMap' query to a WMS Service that is defined in the Config, it must be made through the Proxy.

3.1.7 Template

A Template is a HTML file used as a UI skeleton. It has a list supported widgets and where to draw them. It uses Django's template language and the ExtJS API.

3.1.8 MapContext

A MapContext is basically a list of Resource inside a specific geospatial context, resulting in a map.

3.1.9 Application

An Application consist of a list of Widget and a specific Template, resulting in a complete webmapping application, but without any data layer.

3.1.10 Session

Combining a MapContext with a Application results in a complete webmapping application (with the Application defined widgets) using and having data layers in it (from the MapContext defined resources). This combinaison is called a Session.

CHAPTER 4

Development

4.1 Overview

Here we describe the development process overview.

4.1.1 You want to report a bug or suggest a improvement

Just go to https://github.com/groupe-conseil-nutshimit-nippour/django-geoprisma/issues and create new one.

4.1.2 How do I get involved ?

It's simple! If you want to fix a bug, extend documentation or whatever you think is appropriate for the project and involves changes, just fork the project at github (https://github.com/groupe-conseil-nutshimit-nippour/django-geoprisma), create a separate branch, hack on it, publish changes at your fork and create a pull request.

Here is a quick how to:

1. Fork a project: https://github.com/groupe-conseil-nutshimit-nippour/django-geoprisma/fork
2. Checkout project to your local machine:

   ```
   $ git clone git@github.com:YOUR_NAME/django-geoprisma.git
   ```

3. Create a new branch with name describing change you are going to work on:

   ```
   $ git checkout -b bugfix/support-for-custom-model
   ```

4. Perform changes at newly created branch. Remember to include tests (if this is code related change) and run test suite. See *running tests documentation*.
5. (Optional) Squash commits. If you have multiple commits and it doesn't make much sense to have them separated (and it usually makes little sense), please consider merging all changes into single commit. Please see https://help.github.com/articles/interactive-rebase if you need help with that.
6. Publish changes:

   ```
   $ git push origin YOUR_BRANCH_NAME
   ```

7. Create a Pull Request (https://help.github.com/articles/creating-a-pull-request). Usually it's as simple as opening up https://github.com/YOUR_NAME/django-geoprisma and clicking on review button for newly created branch. There you can make final review of your changes and if everything seems fine, create a Pull Request.

4.2 Example project

This example project demonstrates how to use Django-geoprisma. You can use this example project for development or just test features.

4.2.1 Usage

1. Clone the Django-geoprisma repo or download the archive.
2. Go to the example_project directory into the Django-geoprisma folder.
3. Install requirements

   ```
   $ pip install -r requirements.txt
   ```

4. You need to create a database. Don't forget to put your database configurations in settings.py

   ```
   $ python manage.py syncdb
   ```

5. Load default geoprisma data

   ```
   $ python manage.py loaddata default
   ```

6. Load example data

   ```
   $ python manage.py loaddata example
   ```

7. Create a superuser

   ```
   $ python manage.py createsuperuser
   ```

8. Run the django server

   ```
   $ python manage.py runserver
   ```

9. Sign in into admin with the superuser. `http://yourserverurl/admin`
10. Enter the url `http://yourserverurl/map/ws_example/1` for open the geoprisma example application.

4.3 Testing

4.3.1 Running tests

using runtests.py:

```
$ python runtests.py
```

4.3.2 Coverage support

Coverage is a tool for measuring code coverage of Python programs. It is great for tests and we use it as a backup - we try to cover 100% of the code used by `django-geoprisma`. This of course does *NOT* mean that if all of the codebase is covered by tests we can be sure there is no bug (as specification of almost all applications requeries some unique scenarios to be tested). On the other hand it definitely helps to track missing parts.

To run test with coverage support:

```
$ coverage run runtests.py
```

To show the report:

```
$ coverage report
```

Note:

You need to install coverage using: $ pip install coverage

CHAPTER 5

License

CHAPTER 6

Indices and tables

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