Django Database Views Documentation

Release 0.1.2

Ahmed Nassar

Mar 21, 2017

Contents

Contents:

CHAPTER 1

django-database-views

Serve your single page Javascript applications from Django.

Documentation

The full documentation is at https://django-database-views.readthedocs.io.

Requirements

- Django > 1.8
- A database engine such as MySQL

Quickstart

Install django-database-views using pip:

```
pip install django-database-views
```

Add it to your installed apps:

```
INSTALLED_APPS = (
    ...
    'database_views.apps.DatabaseViewsConfig',
    ...
)
```

Create a model to store versions for your index template in your app's models.py:

```
from database_views.models import AbstractTemplate


class IndexTemplate(AbstractTemplate):

    class Meta:
        db_table = 'your_table_name' # For example 'index_template'.
```

Create a class-based view for your single page app in your app's views.py and assign your model to its *model* property:

```
from database_views.views import DatabaseTemplateView
from database_views.views import CachedTemplateResponse
from myapp.models import IndexTemplate


class IndexView(DatabaseTemplateView):
    app_name = 'main'
    model = IndexTemplate
    response_class = CachedTemplateResponse
```

Add a route for your index page view in your project's urls.py file:

```
from myapp.views import IndexView

urlpatterns = [
    ...
    url(r'^$', IndexView.as_view())
    ...
]
```

That's it!! Go to your new route and you should see your single page app's index template served. Please ensure that you configure the serving of your app's static assets properly.

Features

- Easily serve your single page javascript applications from Django.
- Optionally cache your templates for a configurable amount of time.
- Works with ember-cli-deploy and more specifically with ember-cli-deploy-mysql.

Running Tests

To run tests use the following commands from the root of this project:

```
source <YOURVIRTUALENV>/bin/activate
(myenv) $ pip install -r requirements_test.txt
(myenv) $ py.test
```

Credits

Tools used in rendering this package:

- Cookiecutter
- cookiecutter-djangopackage

CHAPTER 2

Installation

At the command line:

```
$ easy_install django-database-views
```

Or, if you have virtualenvwrapper installed:

```
$ mkvirtualenv django-database-views
$ pip install django-database-views
```

CHAPTER 3

Usage

Deploying Your Single Page Application

You can publish a new template to your database in any way you choose. We assume that you will use this to serve an Ember application, and the index template has been deployed to the database. The easiest way to deploy it is to use ember-cli-deploy-mysql.

If you are not using Ember you can still use this project to serve your application. You just have to properly deploy your template to your MySQL database and set it as the current one. For more details about the template model schema. see the Usage section.

Models

AbstractTemplate

The *AbstractTemplate* model represents a single template stored in the database. This model creates a table in the database to hold all template-related information. This model has the following fields defined:

- key: A unique identifier for the template.
- value: The template content.
- created_at: Creation date and time.
- gitsha: Reserved for future use.
- deployer: Reserved for future use.

This schema is based on the implementation of the django-cli-deploy-mysql package. According to the deploy strategy implemented by that package, a template version is activated by creating a record with the key *current* and the key of the current version as its value. So for the current template to work properly your table has to have two records in it. One for the template itself and another one for the current vesion pointer.

AbstractApplicationTemplate

By default, this package is implemented to be compatible with Ember CLI deploy tools. However, you can still use this package to implement a custom solution or change the way it works by default.

This model allows you to implement a model to store template for multiple applications. This model extends the *AbstractTemplate* model and adds the following fields to it:

- app_name: The name of the application that owns this template.
- current: A boolean flag for the current version.

Cache Time to Live (TTL) Configuration

By default, if the *CachedTemplateResponse* class is used it will cache the contents of the template for a week. To configure a different cache TTL for the template response, set the *TEMPLATE_CACHE_TTL* setting in your settings module. For example:

```
TEMPLATE_CACHE_TTL = 24 * 60 * 60  # Cache template response for 24 hours.
```

Disabling Cache

It is recommended to cache template responses so the application doesn't query the database every time the template is served. If for any reason you need to disable the caching feature of the template use the *DatabaseTemplateResponse* class instead as follows:

```
from database_views.views import DatabaseTemplateView
from database_views.response import DatabaseTemplateResponse
from myapp.models import IndexTemplate


class IndexView(DatabaseTemplateView):
    model = IndexTemplate
    response_class = DatabaseTemplateResponse
```

CHAPTER 4

Contributing

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

You can contribute in many ways:

Types of Contributions

Report Bugs

Report bugs at https://github.com/a7madnassar/django-database-views/issues.

If you are reporting a bug, please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

Fix Bugs

Look through the GitHub issues for bugs. Anything tagged with “bug” is open to whoever wants to implement it.

Implement Features

Look through the GitHub issues for features. Anything tagged with “feature” is open to whoever wants to implement it.

Write Documentation

Django Database Views could always use more documentation, whether as part of the official Django Database Views docs, in docstrings, or even on the web in blog posts, articles, and such.

Submit Feedback

The best way to send feedback is to file an issue at https://github.com/a7madnassar/django-database-views/issues.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that contributions are welcome :)

Get Started!

Ready to contribute? Here's how to set up *django-database-views* for local development.

1. Fork the *django-database-views* repo on GitHub.
2. Clone your fork locally:

```
$ git clone https://github.com/a7madnassar/django-database-views.git
```

3. Install your local copy into a virtualenv. Assuming you have virtualenvwrapper installed, this is how you set up your fork for local development:

```
$ mkvirtualenv django-database-views
$ cd django-database-views/
$ python setup.py develop
```

4. Create a branch for local development:

```
$ git checkout -b name-of-your-bugfix-or-feature
```

   Now you can make your changes locally.

5. When you're done making changes, check that your changes pass flake8 and the tests, including testing other Python versions with tox:

```
$ flake8 database_views tests
$ python setup.py test
$ tox
```

   To get flake8 and tox, just pip install them into your virtualenv.

6. Commit your changes and push your branch to GitHub:

```
$ git add .
$ git commit -m "Your detailed description of your changes."
$ git push origin name-of-your-bugfix-or-feature
```

7. Submit a pull request through the GitHub website.

Pull Request Guidelines

Before you submit a pull request, check that it meets these guidelines:

1. The pull request should include tests.
2. If the pull request adds functionality, the docs should be updated. Put your new functionality into a function with a docstring, and add the feature to the list in README.rst.
3. The pull request should work for Python 2.7, and 3.4, and for PyPy. Check https://travis-ci.org/a7madnassar/django-database-views/pull_requests and make sure that the tests pass for all supported Python versions.

CHAPTER 5

Credits

Development Lead

- Ahmed Nassar <ektebli@yahoo.com>

Contributors

None yet. Why not be the first?

CHAPTER 6

History

0.1.0 (2017-03-10)

- First release on PyPI.