Django-VueFormGenerator Documentation

Release 0.2.3

Tanner Hobson

Mar 29, 2019

Contents

Contents:

CHAPTER 1

Django-VueFormGenerator

A package to help bridge the gap between Django's Forms and VueFormGenerator's Schemas using DjangoRest-Framework.

1.1 Documentation

The full documentation is at https://django-vueformgenerator.readthedocs.org.

1.2 Quickstart

Install Django-VueFormGenerator:

```
pip install django-vueformgenerator
```

Then use it in a project:

```
from django_vueformgenerator.schema import Schema
from django import forms
import json

class TestForm(forms.Form):
    title = forms.CharField(max_length=128)
    content = forms.TextField(max_length=1280)

form = TestForm()  # or TestForm(data={'title':'My Title'})
schema = Schema().render(form)
print(json.dumps(schema))
```

1.3 Features

- TODO

1.4 Running Tests

Does the code actually work?

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

1.5 Credits

Tools used in rendering this package:

- Cookiecutter
- cookiecutter-djangopackage

CHAPTER 2

Installation

At the command line:

```
$ easy_install django-vueformgenerator
```

Or, if you have virtualenvwrapper installed:

```
$ mkvirtualenv django-vueformgenerator
$ pip install django-vueformgenerator
```

CHAPTER 3

Usage

To use Django-VueFormGenerator in a project:

```
from django_vueformgenerator.schema import Schema
from django import forms
import json

class TestForm(forms.Form):
    title = forms.CharField(max_length=128)
    content = forms.TextField(max_length=1280)

form = TestForm()  # or TestForm(data={'title': 'My Title'})
schema = Schema().render(form)
print(json.dumps(schema))
```

Then on the frontend, you can use the schema directly:

```
<template>
    <vue-form-generator :schema="schema" :model="model"></vue-form-generator>
</template>

<script>
export default {
    // ...
    data() {
        return { /* schema object from Django-VueFormGenerator */ };
    }
    // ...
}
</script>
```

Currently, the fields that are implemented are:

- *django.forms.CharField* which maps to *{ "type": "text" }*
- *django.forms.TextField* which maps to *{ "type": "textArea" }*

- *django.forms.BooleanField* which maps to *{ "type": "checkbox" }*
- *django.forms.IntegerField* which maps to *{ "type": "number" }*
- Any field which uses *django.forms.Field(choices=(...))* which maps to *{ "type": "select" }*

For more information, check:

- *VueFormGenerator's documentation <https://icebob.gitbooks.io/vueformgenerator/content/fields/>*
- *Django Form's documentation <https://docs.djangoproject.com/en/1.10/ref/forms/fields/>*

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:

4.1 Types of Contributions

4.1.1 Report Bugs

Report bugs at https://github.com/player1537/django-vueformgenerator/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.

4.1.2 Fix Bugs

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

4.1.3 Implement Features

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

4.1.4 Write Documentation

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

4.1.5 Submit Feedback

The best way to send feedback is to file an issue at https://github.com/player1537/django-vueformgenerator/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 :)

4.2 Get Started!

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

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

```
$ git clone git@github.com:your_name_here/django-vueformgenerator.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-vueformgenerator
$ cd django-vueformgenerator/
$ 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 django_vueformgenerator 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.

4.3 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 2.6, 2.7, and 3.3, and for PyPy. Check https://travis-ci.org/player1537/django-vueformgenerator/pull_requests and make sure that the tests pass for all supported Python versions.

4.4 Tips

To run a subset of tests:

```
$ python -m unittest tests.test_django_vueformgenerator
```

CHAPTER 5

Credits

5.1 Development Lead

- Tanner Hobson <thobson125@gmail.com>

5.2 Contributors

None yet. Why not be the first?

CHAPTER 6

History

6.1 0.2.3 (2016-11-03)

- Add support for dotted models. Use this feature by defining your Form with a field that has a name with double-underscores (e.g. "foo__bar__baz", which will become "foo.bar.baz" in the schema's model field).

6.2 0.2.2 (2016-11-01)

- Fix implementation of using initial data in forms. Previously, if you used *CharField(initial='foo')* then this information would not be preserved when creating the schema.

6.3 0.2.1 (2016-10-27)

- Fix bug in tests so that the tests run successfully in Python 2.7.

6.4 0.2.0 (2016-10-25)

- Add ability to use existing data in form
- DEPRECATED: Any code which previously used *Schema().render(MyForm)* should now use *Schema().render(MyForm())* (in other words, *render()* accepts an instance of a form, rather than a form itself). To check if you are calling the function against contract, you can run your code with *python -Wd* (e.g. *python -Wd manage.py runserver*).

6.5 0.1.1 (2016-10-18)

- Add additional tests for schema generation
- Add components for numbers and for selecting between choices
- Add Python 2 support
- Add better documentation
- Fix exception raised on bad widget

6.6 0.1.0 (2016-10-11)

- First release on PyPI.