Django Powerbank Documentation

Release 0.4.1

Janusz Skonieczny

Nov 26, 2018

Contents

Contents:

CHAPTER 1

Django Powerbank

Documentation: https://django-powerbank.readthedocs.io.

1.1 Work in progress

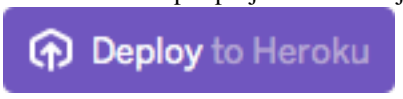
This package was created with wooyek/cookiecutter-django-app project template. It may fool you by having a lot of cookie filling just ready from the start. Be warned! It may not be ready for eating!

1.2 Features

- Pending :D

1.3 Demo

To run an example project for this django reusable app, click the button below and start a demo serwer on Heroku



1.4 Quickstart

Install Django Powerbank:

```
pip install django-powerbank
```

Add it to your *INSTALLED_APPS*:

```
INSTALLED_APPS = (
    ...
    'django_powerbank.apps.DjangoPowerbankConfig',
    ...
)
```

Add Django Powerbank's URL patterns:

```
from django_powerbank import urls as django_powerbank_urls


urlpatterns = [
    ...
    url(r'^', include(django_powerbank_urls)),
    ...
]
```

1.5 Running Tests

Does the code actually work?

```
$ pipenv install --dev
$ pipenv shell
$ tox
```

We recommend using pipenv but a legacy approach to creating virtualenv and installing requirements should also work. Please install *requirements/development.txt* to setup virtual env for testing and development.

1.6 Credits

This package was created with Cookiecutter and the wooyek/cookiecutter-django-app project template.

CHAPTER 2

Installation

2.1 Stable release

To install Django Powerbank, run this command in your terminal:

```
$ pip install django-powerbank
```

This is the preferred method to install Django Powerbank, as it will always install the most recent stable release.

If you don't have pip installed, this Python installation guide can guide you through the process.

2.2 From sources

You can either clone the public repository:

```
$ git clone git://github.com/wooyek/django-powerbank
```

Or download the download source from project website. Once you have a copy of the source, you can install it with:

```
$ python setup.py install
```

CHAPTER 3

Usage

To use Django Powerbank in a project, add it to your *INSTALLED_APPS*:

```
INSTALLED_APPS = (
    ...
    'django_powerbank.apps.DjangoPowerbankConfig',
    ...
)
```

Add Django Powerbank's URL patterns:

```
from django_powerbank import urls as django_powerbank_urls


urlpatterns = [
    ...
    url(r'^', include(django_powerbank_urls)),
    ...
]
```

CHAPTER 4

Api docs

4.1 django_powerbank package

4.1.1 Subpackages

django_powerbank.core package

Submodules

django_powerbank.core.validators module

class django_powerbank.core.validators.**MaxCurrentYearValidator**
Bases: object

deconstruct()
Return a 3-tuple of class import path, positional arguments, and keyword arguments.

message = 'Ensure this value is less than or equal to %(limit_value)s.'

class django_powerbank.core.validators.**MaxTodayDateValidator**
Bases: object

deconstruct()
Return a 3-tuple of class import path, positional arguments, and keyword arguments.

message = 'Ensure this value is less than or equal to %(limit_value)s.'

class django_powerbank.core.validators.**MsisdnValidator**(*code=None*, *inverse_match=None*, *flags=None*)
Bases: django.core.validators.RegexValidator

class django_powerbank.core.validators.**SmartUrlValidator**
Bases: object

deconstruct()
Return a 3-tuple of class import path, positional arguments, and keyword arguments.

Module contents

django_powerbank.db package

Subpackages

django_powerbank.db.models package

Subpackages

django_powerbank.db.models.fields package

Submodules

django_powerbank.db.models.fields.pl module

class django_powerbank.db.models.fields.pl.**PlNipField**(**args*, ***kwargs*)
Bases: django.db.models.fields.CharField

formfield(***kwargs*)
Return a django.forms.Field instance for this field.

to_python(*value*)
Convert the input value into the expected Python data type, raising django.core.exceptions.ValidationError if the data can't be converted. Return the converted value. Subclasses should override this.

class django_powerbank.db.models.fields.pl.**PlRegonField**(**args*, ***kwargs*)
Bases: django.db.models.fields.CharField

formfield(***kwargs*)
Return a django.forms.Field instance for this field.

to_python(*value*)
Convert the input value into the expected Python data type, raising django.core.exceptions.ValidationError if the data can't be converted. Return the converted value. Subclasses should override this.

Module contents

class django_powerbank.db.models.fields.**AutoSlugField**(*source_field=None*, *keep_existing=False*, *source_fallback=False*, **args*, ***kwargs*)
Bases: *django_powerbank.db.models.fields.SourceFieldMixin*, django.db.models.fields.SlugField

get_slug_value(*model_instance*)

get_source_value(*model_instance*)

pre_save(*model_instance*, *add*)
Return field's value just before saving.

class django_powerbank.db.models.fields.**BinaryMaskEnum**
Bases: *django_powerbank.db.models.fields.ChoicesIntEnum*

An enumeration.

class django_powerbank.db.models.fields.**ChoicesIntEnum**
Bases: enum.IntEnum

Extends IntEum with django choices generation capability

class django_powerbank.db.models.fields.**JSONField**(**args*, ***kwargs*)
Bases: django.db.models.fields.TextField

Simple JSON field that stores python structures as JSON strings on database.

from_db_value(*value*, *expression*, *connection*, *context*)

get_prep_value(*value*)
Convert value to JSON string before save

to_python(*value*)
Convert the input JSON value into python structures, raises django.core.exceptions.ValidationError if the data can't be converted.

validate(*value*, *model_instance*)
Check value is a valid JSON string, raise ValidationError on error.

value_from_object(*obj*)
Return the value of this field in the given model instance.

value_to_string(*obj*)
Return value from object converted to string properly

class django_powerbank.db.models.fields.**MarkDownField**(*source_field=None*, **args*, ***kwargs*)
Bases: *django_powerbank.db.models.fields.SourceFieldMixin*, django.db.models.fields.TextField

contribute_to_class(*cls*, *name*, *private_only=False*)
Register the field with the model class it belongs to.

If private_only is True, create a separate instance of this field for every subclass of cls, even if cls is not an abstract model.

from_db_value(*value*, *expression*, *connection*)

pre_save(*model_instance*, *add*)
Return field's value just before saving.

to_python(*value*)
Convert the input value into the expected Python data type, raising django.core.exceptions.ValidationError if the data can't be converted. Return the converted value. Subclasses should override this.

class django_powerbank.db.models.fields.**PhoneField**(**args*, ***kwargs*)
Bases: django.db.models.fields.CharField

default_validators = [<django_powerbank.core.validators.MsisdnValidator object>]

formfield(***kwargs*)
Return a django.forms.Field instance for this field.

to_python(*value*)
Convert the input value into the expected Python data type, raising django.core.exceptions.ValidationError if the data can't be converted. Return the converted value. Subclasses should override this.

class django_powerbank.db.models.fields.**SecretField**(*source_field=None*, **args*, ***kwargs*)

Bases: *django_powerbank.db.models.fields.SourceFieldMixin*, django.db.models.fields.CharField

pre_save(*model_instance*, *add*)
Return field's value just before saving.

class django_powerbank.db.models.fields.**SourceFieldMixin**(*source_field=None*, **args*, ***kwargs*)

Bases: object

check(***kwargs*)

class django_powerbank.db.models.fields.**UniqueSlugField**(*source_field=None*, *keep_existing=False*, **args*, ***kwargs*)

Bases: *django_powerbank.db.models.fields.AutoSlugField*, django.db.models.fields.SlugField

get_slug_value(*model_instance*)

Submodules

django_powerbank.db.models.base module

class django_powerbank.db.models.base.**BaseModel**(**args*, ***kwargs*)

Bases: django.db.models.base.Model

class Meta

Bases: object

abstract = False

get_field_name(*field_name*)

get_field_names()

populate(***kwargs*)
Populate an instance from keyword arguments.

Each keyword argument will be used to set a corresponding field. Keywords must refer to valid property name. This is similar to passing keyword arguments to the Model constructor.

to_dict(*include=None*, *exclude=None*)
Return a dict containing the entity's property values.

Parameters

- **include** – Optional set of property names to include, default all.
- **exclude** – Optional set of property names to skip, default none. A name contained in both include and exclude is excluded.

django_powerbank.db.models.query module

class django_powerbank.db.models.query.**ApproxQuerySet**(*model=None*, *query=None*, *using=None*, *hints=None*)

Bases: django.db.models.query.QuerySet

count()
Perform a SELECT COUNT() and return the number of records as an integer.

If the QuerySet is already fully cached, return the length of the cached results set to avoid multiple SELECT COUNT(*) calls.

classmethod wrap_query(*qry*)

class django_powerbank.db.models.query.**TableStatusQuerySet**(*model=None*, *query=None*, *using=None*, *hints=None*)
Bases: *django_powerbank.db.models.query.ApproxQuerySet*

Module contents

Module contents

django_powerbank.forms package

Submodules

django_powerbank.forms.fields module

class django_powerbank.forms.fields.**DateRangeField**(*input_formats=None*, **args*, ***kwargs*)
Bases: django.forms.fields.DateField

to_python(*value*)
Validate that the input can be converted to a date. Return a Python datetime.date object.

widget
alias of django.forms.widgets.Input

class django_powerbank.forms.fields.**PhoneField**(***, *max_length=None*, *min_length=None*, *strip=True*, *empty_value=''*, ***kwargs*)
Bases: django.forms.fields.CharField

widget
alias of *django_powerbank.forms.widgets.PhoneInput*

class django_powerbank.forms.fields.**TypeaheadField**(***, *max_length=None*, *min_length=None*, *strip=True*, *empty_value=''*, ***kwargs*)
Bases: django.forms.fields.CharField

widget
alias of *django_powerbank.forms.widgets.Typeahead*

django_powerbank.forms.widgets module

class django_powerbank.forms.widgets.**PhoneInput**(*attrs=None*)
Bases: django.forms.widgets.TextInput

input_type = 'tel'

media

class django_powerbank.forms.widgets.**SelectizeBase**(*attrs=None*, *url=None*, *allow_create=False*, *value_field='text'*, *label_field='text'*, *search_field='text'*, *plugins=[]*, *close_after_select=True*)

Bases: django.forms.widgets.Input

build_attrs(*base_attrs*, *extra_attrs=None*, ***kwargs*)
Build an attribute dictionary.

get_context(*name*, *value*, *attrs*)

media

class django_powerbank.forms.widgets.**SelectizeSelect**(*attrs=None*, *url=None*, *allow_create=False*, *value_field='text'*, *label_field='text'*, *search_field='text'*, *plugins=[]*, *close_after_select=True*)

Bases: *django_powerbank.forms.widgets.SelectizeBase*, django.forms.widgets.Select

A selectize.js field

It requires selectize.js and headjs to be avaialable in the browser. See a template below to se why. You can provide your own template to use selectize.js in a different way.

media

optgroups(*name*, *value*, *attrs=None*)
Return a list of optgroups for this widget.

template_name = 'django_powerbank/forms/widgets/selectize/select.html'

class django_powerbank.forms.widgets.**SelectizeTags**(*attrs=None*, *url=None*, *allow_create=False*, *value_field='text'*, *label_field='text'*, *search_field='text'*, *plugins=[]*, *close_after_select=True*)

Bases: *django_powerbank.forms.widgets.SelectizeBase*

media

template_name = 'django_powerbank/forms/widgets/selectize/tags.html'

class django_powerbank.forms.widgets.**Typeahead**(*attrs=None*, *url=None*)

Bases: django.forms.widgets.Input

build_attrs(*base_attrs*, *extra_attrs=None*, ***kwargs*)
Build an attribute dictionary.

media

Module contents

django_powerbank.testing package

Submodules

django_powerbank.testing.base module

class django_powerbank.testing.base.**AdminUserTestCase**
Bases: *django_powerbank.testing.base.AssertionsMx*

setUp()

class django_powerbank.testing.base.**AssertionsMx**
Bases: object

assertNoFormErrors(*response*, *form_context_key='form'*)

class django_powerbank.testing.base.**MigrationsCheck**(*methodName='runTest'*)
Bases: *django_powerbank.testing.base.MigrationsCheckMx*, unittest.case.TestCase

class django_powerbank.testing.base.**MigrationsCheckMx**
Bases: object

setUp()

tearDown()

test_missing_migrations()

class django_powerbank.testing.base.**StaffUserTestCase**
Bases: *django_powerbank.testing.base.AssertionsMx*

setUp()

class django_powerbank.testing.base.**UserTestCase**
Bases: *django_powerbank.testing.base.AssertionsMx*

setUp()

django_powerbank.testing.factories module

class django_powerbank.testing.factories.**UserFactory**
Bases: factory.django.DjangoModelFactory

email = <factory.faker.Faker object>

first_name = <factory.faker.Faker object>

is_active = True

is_staff = False

last_name = <factory.faker.Faker object>

username = <factory.declarations.Sequence object>

django_powerbank.testing.utils module

django_powerbank.testing.utils.**model_to_request_data_dict**(*model*)
Removes fields with None value. Test client will serialize them into 'None' strings that will cause validation errors.

Module contents

django_powerbank.views package

Submodules

django_powerbank.views.auth module

class django_powerbank.views.auth.**AbstractAccessView**(***kwargs*)
Bases: *django_powerbank.views.ExceptionResponseView*

Allows you to handle authorization before dispatch is called

check_authorization(**args*, ***kwargs*)

dispatch(*request*, **args*, ***kwargs*)

class django_powerbank.views.auth.**AbstractAuthorizedView**(***kwargs*)
Bases: *django_powerbank.views.auth.AuthenticatedView*

check_authorization(**args*, ***kwargs*)

forbidden_message = 'You are not authorized to view this page'

get_forbidden_message()

handle_forbidden()

is_authorized(**args*, ***kwargs*)

django_powerbank.views.auth.**AccessMixin**
alias of *django_powerbank.views.auth.AuthenticatedView*

class django_powerbank.views.auth.**AuthenticatedView**(***kwargs*)
Bases: *django_powerbank.views.auth.AbstractAccessView*

redirects unauthenticated users to login

check_authorization(**args*, ***kwargs*)

handle_anonymous(**args*, ***kwargs*)

is_authenticated(**args*, ***kwargs*)

class django_powerbank.views.auth.**StaffRequiredMixin**(***kwargs*)
Bases: *django_powerbank.views.auth.AbstractAuthorizedView*

is_authorized(**args*, ***kwargs*)

django_powerbank.views.generic module

class django_powerbank.views.generic.**FilterMixin**
Bases: django.views.generic.list.MultipleObjectMixin

get_queryset()
Return the list of items for this view.

The return value must be an iterable and may be an instance of *QuerySet* in which case *QuerySet* specific behavior will be enabled.

django_powerbank.views.mixins module

class django_powerbank.views.mixins.**ReturnUrlMx**(***kwargs*)
Bases: django.views.generic.base.ContextMixin, django.views.generic.base.View

dispatch(*request*, **args*, ***kwargs*)
Does request processing for return_url query parameter and redirects with it's missing

We can't do that in the get method, as it does not exist in the View base class and child mixins implementing get do not call super().get

get_context_data(***kwargs*)

get_success_url()

Module contents

exception django_powerbank.views.**ExceptionResponse**(*response*)
Bases: Exception

Generic exception for signalling that instead of default error handling we should use attached response

class django_powerbank.views.**ExceptionResponseView**(***kwargs*)
Bases: django.views.generic.base.View

dispatch(*request*, **args*, ***kwargs*)

exception django_powerbank.views.**Http302**(*to*, **args*, ***kwargs*)
Bases: *django_powerbank.views.ExceptionResponse*

Wraps a redirect shortcut call into the ResponseException

exception django_powerbank.views.**Http400**(*response*)
Bases: *django_powerbank.views.ExceptionResponse*

exception django_powerbank.views.**Http401**(*response*)
Bases: *django_powerbank.views.ExceptionResponse*

exception django_powerbank.views.**Http403**(*response*)
Bases: django.core.exceptions.PermissionDenied, *django_powerbank.views.ExceptionResponse*

A convenience wrapper around :py:class:~django.core.exceptions.PermissionDenied‘

4.1.2 Submodules

4.1.3 django_powerbank.app_settings module

4.1.4 django_powerbank.apps module

class django_powerbank.apps.**DjangoPowerbankConfig**(*app_name*, *app_module*)
Bases: django.apps.config.AppConfig

name = 'django_powerbank'

ready()
Override this method in subclasses to run code when Django starts.

verbose_name = 'Django Powerbank'

django_powerbank.apps.**setup_app_settings**()

4.1.5 Module contents

CHAPTER 5

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:

5.1 Types of Contributions

5.1.1 Report Bugs

Report bugs at https://github.com/wooyek/django-powerbank/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.

5.1.2 Fix Bugs

Look through the GitHub issues for bugs. Anything tagged with "bug" and "help wanted" is open to whoever wants to implement it.

5.1.3 Implement Features

Look through the GitHub issues for features. Anything tagged with "enhancement" and "help wanted" is open to whoever wants to implement it.

5.1.4 Write Documentation

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

5.1.5 Submit Feedback

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

5.2 Get Started!

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

1. Fork the *django-powerbank* repo on github.com
2. Clone your fork locally:

```
$ git clone git@github.com:your_name_here/django-powerbank.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-powerbank
$ cd django-powerbank/
$ 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-powerbank tests
$ tox -e check
$ python setup.py test or 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.

5.3 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 versions mentioned in tox.ini file. Check https://travis-ci.org/wooyek/django-powerbank/pull_requests and make sure that the tests pass for all supported Python versions.

5.4 Tips

To run a subset of tests:

```
$ pytest tests.test_models
```

CHAPTER 6

Credits

6.1 Development Lead

- @wooyek: Janusz Skonieczny <js+pypi@bravelabs.pl>

6.2 Contributors

None yet. Why not be the first?

CHAPTER 7

History

7.1 0.2.1 (2017-07-14)

- First release on PyPI.

CHAPTER 8

Indices and tables

- genindex
- modindex
- search

Python Module Index

Index

E

F

G

H

I

J

L

M

U

V

W