Django Test Tools Documentation

Release 1.10.6

Luis Carlos Berrocal

Aug 23, 2020

Contents

Contents:

CHAPTER 1

Django Test Tools

Simple tests tools to make testing faster and easier. Most of the tools are to do a quick scaffolding for tests.

The tools presume a naming convention:

- **Tests:** Are named with the convention **TestCaseModelName**. For a model named *Poll* the test would be generated as the testing class would be *TestCasePoll*
- **Factories:** Are named with the convention **ModelName**. For a model named *Poll* the test would be generated as the testing class would be *PollFactory*
- **Serializers:** Are named with the convention **TestCaseSerializer**. For a model named *Poll* the test would be generated as the testing class would be *PollSerializer*

Compatibility matrix:

Python version	Django 1.11.x	Django 2.2.x	Django 3.0.x
3.7	x	x	x
3.6	x	x	x

1.1 Documentation

The full documentation is at https://django-test-tools.readthedocs.io.

1.2 Quickstart

Install Django Test Tools:

```
pip install django-test-tools
```

In your settings.py file add it to your *INSTALLED_APPS*

```
INSTALLED_APPS = (
    ...
    'django_test_tools.apps.DjangoTestToolsConfig',
    ...
)
```

Create an output folder in the root folder of you project, name it what ever you want, and add the settings variable **TEST_OUTPUT_PATH** pointing to it.

```
import environ

ROOT_DIR = (
    environ.Path(__file__) - 3
)  # (my_project/config/settings/base.py - 3 = alpha_clinic/)
APPS_DIR = ROOT_DIR.path("my_project")
TEST_OUTPUT_PATH = ROOT_DIR.path("output").root
```

1.3 Features

1.3.1 Factory Generator

To create Factory Boy style factories.

For a django project named polling_app with an app name poll the following command will generate the scaffolding for the tests for all the models in th app polls.

```
$  python manage.py generate_factories polling_app.polls
```

For the following models

```
class OperatingSystem(models.Model):
    name = models.CharField(max_length=20)
    version = models.CharField(max_length=5)
    licenses_available = models.IntegerField()
    cost = models.DecimalField(decimal_places=2, max_digits=7)

    class Meta:
        unique_together = ('name', 'version')


class Server(models.Model):
    PRODUCTION = 'PROD'
    DEVELOPMENT = 'DEV'
    USE_CHOICES = ((PRODUCTION, 'Prod'),
                   (DEVELOPMENT, 'Dev'))
    name = models.CharField(max_length=20, unique=True)
```

(continues on next page)

(continued from previous page)

```
    notes = models.TextField()
    virtual = models.BooleanField()
    ip_address = models.GenericIPAddressField()
    created = models.DateTimeField()
    online_date = models.DateField()
    operating_system = models.ForeignKey(OperatingSystem, related_name='servers', on_
↪delete=models.CASCADE)
    server_id = models.CharField(max_length=6)
    use = models.CharField(max_length=4, choices=USE_CHOICES, default=DEVELOPMENT)
    comments = models.TextField(null=True, blank=True)
```

running *python manage.py generate_factories example.servers > ./output/factories.py* will create the following factories

```
import string

from random import randint
from pytz import timezone

from django.conf import settings

from factory import Iterator
from factory import LazyAttribute
from factory import SubFactory
from factory import lazy_attribute
from factory.django import DjangoModelFactory, FileField
from factory.fuzzy import FuzzyText, FuzzyInteger
from faker import Factory as FakerFactory

from example.servers.models import OperatingSystem, Server

faker = FakerFactory.create()


class OperatingSystemFactory(DjangoModelFactory):
    class Meta:
        model = OperatingSystem

    name = LazyAttribute(lambda x: faker.text(max_nb_chars=20))
    version = LazyAttribute(lambda x: faker.text(max_nb_chars=5))
    licenses_available = LazyAttribute(lambda o: randint(1, 100))
    cost = LazyAttribute(lambda x: faker.pydecimal(left_digits=5, right_digits=2,␣
↪positive=True))

class ServerFactory(DjangoModelFactory):
    class Meta:
        model = Server

    name = LazyAttribute(lambda x: faker.text(max_nb_chars=20))
    notes = LazyAttribute(lambda x: faker.paragraph(nb_sentences=3, variable_nb_
↪sentences=True))
    virtual = Iterator([True, False])
    ip_address = LazyAttribute(lambda o: faker.ipv4(network=False))
    created = LazyAttribute(lambda x: faker.date_time_between(start_date="-1y", end_
↪date="now",
                                                              tzinfo=timezone(settings.
↪TIME_ZONE)))
```

(continues on next page)

(continued from previous page)

```
    online_date = LazyAttribute(lambda x: faker.date_time_between(start_date="-1y",␣
↪end_date="now",
                                                                  tzinfo=timezone(settings.
↪TIME_ZONE)))
    operating_system = SubFactory(OperatingSystemFactory)
    server_id = LazyAttribute(lambda x: FuzzyText(length=6, chars=string.digits).
↪fuzz())
    use = Iterator(Server.CHOICES, getter=lambda x: x[0])
    comments = LazyAttribute(lambda x: faker.paragraph(nb_sentences=3, variable_nb_
↪sentences=True))
```

Important the use attribute is created incorrectly. **When you use choices you need to manually change it** to USE_CHOICES.

```
use = Iterator(Server.USE_CHOICES, getter=lambda x: x[0])
```

1.3.2 Model Test Case Generator

```
$  python manage.py generate_model_test_cases project.app
```

1.3.3 Serializer Generator

```
$ python manage.py generate_serializers project.app -s ModelSerializer
```

1.4 Running Tests

Does the code actually work?

```
source <YOURVIRTUALENV>/bin/activate
(myenv) $ pip install tox
(myenv) $ tox
```

1.5 Pushing code to Pypi

1. Setup environment

   ```
   source./venv/bin/activate
   ```

2. Updated version. Instead of patch you could also use **major** o **minor** depending on the level of the release.

   ```
   $ make patch
   ```

3. Check the .travis.yml to make sure the versions of Djago are the latests. Check in https://www.djangoproject.com/download/ for the latest versions.
4. Check setup.py for Django and Python versions.
5. Close the git-flow release manually.

6. Push to repo, Travis CI should deploy to pypi

```
make travis-push
```

1.6 Credits

Tools used in rendering this package:

- Cookiecutter
- cookiecutter-djangopackage

CHAPTER 2

Installation

At the command line:

```
$ easy_install django-test-tools
```

Or, if you have virtualenvwrapper installed:

```
$ mkvirtualenv django-test-tools
$ pip install django-test-tools
```

CHAPTER 3

Usage

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

```
pip install django-test-tools
```

In your settings.py file add it to your *INSTALLED_APPS*

```
INSTALLED_APPS = (
    ...
    'django_test_tools.apps.DjangoTestToolsConfig',
    ...
)
```

Create an output folder in the root folder of you project, name it what ever you want, and add the settings variable **TEST_OUTPUT_PATH** point to it.

```
import environ

ROOT_DIR = (
    environ.Path(__file__) - 3
)  # (my_project/config/settings/base.py - 3 = alpha_clinic/)
APPS_DIR = ROOT_DIR.path("my_project")
TEST_OUTPUT_PATH = ROOT_DIR.path("output").root
```

CHAPTER 4

django_test_tools package

4.1 Subpackages

4.1.1 django_test_tools.doc_utils package

Submodules

django_test_tools.doc_utils.folder_structure module

django_test_tools.doc_utils.folder_structure.**create_folder_structure**(*doc_base_folder*, *project_name*)

django_test_tools.doc_utils.folder_structure.**get_module_files**(*folder*)

django_test_tools.doc_utils.folder_structure.**write_template**(*data*, *folder*, *output_file*, *template*)

Module contents

4.1.2 django_test_tools.flake8 package

Submodules

django_test_tools.flake8.parsers module

class django_test_tools.flake8.parsers.**Flake8Parser**

Bases: object

3 E124 closing bracket does not match visual indentation 6 E127 continuation line over-indented for visual indent 11 E128 continuation line under-indented for visual indent 2 E221 multiple spaces before operator 1 E222 multiple spaces after operator 10 E225 missing whitespace around operator 6 E231 missing whitespace after ‘,’ 2 E251 unexpected spaces around keyword / parameter equals 4 E261 at least two spaces before inline

comment 4 E262 inline comment should start with '# ' 8 E265 block comment should start with '# ' 4 E266 too many leading '#' for block comment 2 E271 multiple spaces after keyword 5 E302 expected 2 blank lines, found 1 7 E303 too many blank lines (3) 2 E402 module level import not at top of file 8 E501 line too long (123 > 120 characters) 17 F401 'django.contrib.admin' imported but unused 25 F405 'env' may be undefined, or defined from star imports: .base 1 F811 redefinition of unused 'RemarksManager' from line 3 7 F841 local variable 'response' is assigned to but never used 2 W293 blank line contains whitespace 6 W391 blank line at end of file

parse_summary(*filename*)

write_summary(*source_filename*, *target_filename*)

class django_test_tools.flake8.parsers.**RadonParser**
Bases: object

config/settings/test.py LOC: 61 LLOC: 12 SLOC: 23 Comments: 23 Single comments: 22 Multi: 4 Blank: 12 - Comment Stats

(C % L): 38% (C % S): 100% (C + M % L): 44%

** **Total** ** LOC: 2149 LLOC: 894 SLOC: 1311 Comments: 335 Single comments: 310 Multi: 128 Blank: 400 - Comment Stats

(C % L): 16% (C % S): 26% (C + M % L): 22%

parse_totals(*filename*)

write_totals(*source_filename*, *target_filename*)

Module contents

4.1.3 django_test_tools.generators package

Submodules

django_test_tools.generators.model_test_gen module

class django_test_tools.generators.model_test_gen.**AppModelsTestCaseGenerator**(*app*)
Bases: object

class django_test_tools.generators.model_test_gen.**ModelTestCaseGenerator**(*model*)
Bases: object

django_test_tools.generators.serializer_gen module

class django_test_tools.generators.serializer_gen.**AppSerializerGenerator**(*app*, *serializer_class='ModelSeriali*
Bases: object

class django_test_tools.generators.serializer_gen.**SerializerGenerator**(*model*, *serializer_class='ModelSerializer'*)
Bases: object

Module contents

4.1.4 django_test_tools.git package

Submodules

django_test_tools.git.helpers module

class django_test_tools.git.helpers.**GenericCVS**
Bases: object

classmethod commit(*message*)

classmethod is_usable()

class django_test_tools.git.helpers.**Git**
Bases: *django_test_tools.git.helpers.GenericCVS*

Option Description of Output %H Commit hash %h Abbreviated commit hash %T Tree hash %t Abbreviated tree hash %P Parent hashes %p Abbreviated parent hashes %an Author name %ae Author email %ad Author date (format respects the –date=option) %ar Author date, relative %cn Committer name %ce Committer email %cd Committer date %cr Committer date, relative %s Subject

classmethod add_path(*path*)

classmethod assert_nondirty()

classmethod latest_tag_info()

report()

classmethod tag(*name*, *message*)

Module contents

4.1.5 django_test_tools.management package

Subpackages

django_test_tools.management.commands package

Submodules

django_test_tools.management.commands.generate_factories module

class django_test_tools.management.commands.generate_factories.**Command**(*stdout=None*, *stderr=None*, *no_color=False*, *force_color=False*)
Bases: django.core.management.base.BaseCommand

$ python manage.py generate_factories project.app

add_arguments(*parser*)
Entry point for subclassed commands to add custom arguments.

handle(**args*, ***options*)
The actual logic of the command. Subclasses must implement this method.

class django_test_tools.management.commands.generate_factories.**ModelFactoryGenerator**(*model*)
Bases: object

django_test_tools.management.commands.generate_model_test_cases module

class django_test_tools.management.commands.generate_model_test_cases.**Command**(*stdout=None*, *stderr=None*, *no_color=False*, *force_color=False*)
Bases: django.core.management.base.BaseCommand

$ python manage.py

add_arguments(*parser*)
Entry point for subclassed commands to add custom arguments.

handle(**args*, ***options*)
The actual logic of the command. Subclasses must implement this method.

django_test_tools.management.commands.generate_serializers module

class django_test_tools.management.commands.generate_serializers.**Command**(*stdout=None*, *stderr=None*, *no_color=False*, *force_color=False*)
Bases: django.core.management.base.BaseCommand

$ python manage.py generate_serializers project.app -s ModelSerializer

add_arguments(*parser*)
Entry point for subclassed commands to add custom arguments.

handle(**args*, ***options*)
The actual logic of the command. Subclasses must implement this method.

django_test_tools.management.commands.parse_qc_files module

class django_test_tools.management.commands.parse_qc_files.**Command**(*stdout=None*, *stderr=None*, *no_color=False*, *force_color=False*)
Bases: django.core.management.base.BaseCommand

$ python manage.py

add_arguments(*parser*)
Entry point for subclassed commands to add custom arguments.

handle(**args*, ***options*)
The actual logic of the command. Subclasses must implement this method.

Module contents

Module contents

4.2 Submodules

4.3 django_test_tools.app_manager module

class django_test_tools.app_manager.**DjangoAppManager**
Bases: object

get_app(*app_name*)

get_app_data(*app_name*)
Read application data converts into a dictionary

Parameters **app_name** – Application name

Returns Dictionary with application data

get_installed_apps()

get_model(*app_name*, *model_name*)

get_project_apps(*project_name*)

4.4 django_test_tools.apps module

class django_test_tools.apps.**DjangoTestToolsConfig**(*app_name*, *app_module*)
Bases: django.apps.config.AppConfig

name = 'django_test_tools'

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

4.5 django_test_tools.assert_utils module

class django_test_tools.assert_utils.**AssertionWriter**(***kwargs*)
Bases: object

This class generates assertions using Django practice of putting actual value first and then expected value.

add_regular_expression(*name*, *pattern*, ***kwargs*)

write_assert_list(*dictionary_list*, *variable_name*, ***kwargs*)
Function to generate assertions for a dictionary or list content. :param kwargs: :param dictionary_list: :param variable_name: :return:

django_test_tools.assert_utils.**write_assert_list**(*filename*, *dictionary_list*, *variable_name*)
Function to generate assertions for a dictionary or list content. :param filename: :param dictionary_list: :param variable_name: :return:

Note: Deprecated: Use assert_utils.write_assertions instead

---

django_test_tools.assert_utils.**write_assertions**(*dictionary_list*, *variable_name*, ***kwargs*)

Writes assertions using Django practice of putting actual value first and then expected value to a file. If no filename is supplied it will generate a file in the settings.TEST_OUTPUT_PATH folder with the **variable_name** and the current date. By default key named created and modified will be excluded.

Parameters

- **dictionary_list** – <list> or <dict> dictionary or list of values
- **variable_name** – <str> name of the variable
- **kwargs** – filename <str>String. Full path to the output file.
- **kwargs** – excluded_keys <list>list of strings. List with keys to exclude
- **kwargs** – type_only <boolean> Check only for types instead of values. Default false

Returns filename string.

4.6 django_test_tools.excel module

class django_test_tools.excel.**ExcelAdapter**

Bases: object

classmethod convert_to_dict(*filename*, *sheet_name=None*)

Reads an Excel file and converts every row into a dictionary. All values are converted to strings. Assumes first row contains the name of the attributes.

Parameters

- **filename** – <str> Excel filename
- **sheet_name** – <str> Name of the sheet

Returns <list> A list of dictionaries.

classmethod convert_to_list(*filename*, *sheet_name=None*, *has_header=True*)

Reads an Excel file and converts every row into a dictionary. All values are converted to strings. Assumes first row contains the name of the attributes.

Parameters

- **filename** – <str> Excel filename
- **sheet_name** – <str> Name of the sheet

Returns <list> A list of dictionaries.

4.7 django_test_tools.file_utils module

class django_test_tools.file_utils.**TemporaryFolder**(*base_name*, *delete_on_exit=True*)

Bases: object

write(*filename*, *content*)

django_test_tools.file_utils.**add_date**(*filename*, ***kwargs*)

Adds to a filename the current date and time in '%Y%m%d_%H%M' format. For a filename /my/path/myexcel.xlsx the function would return /my/path/myexcel_20170101_1305.xlsx. If the file already exists the function will add seconds to the date to attempt to get a unique name.

The function will detect if another file exists with the same name if it exist it will append seconds to the filename. For example if file /my/path/myexcel_20170101_1305.xlsx alread exist thte function will return /my/path/myexcel_20170101_130521.xlsx.

Parameters

- **filename** – string with fullpath to file or just the filename
- **kwargs** – dictionary. date_position: suffix or preffix, extension: string to replace extension

Returns string with full path string including the date and time

django_test_tools.file_utils.**compare_file_content**(**args*, ***kwargs*)

django_test_tools.file_utils.**create_dated**(*filename*)

Based on the filename will create a full path filename including the date and time in '%Y%m%d_%H%M' format. The path to the filename will be set in the TEST_OUTPUT_PATH settings variable.

If the TEST_OUTPUT_PATH folder doesn't exist the function will create it.

Parameters **filename** – base filename. my_excel_data.xlsx for example

Returns string, full path to file with date and time in the TEST_OUTPUT_PATH folder

django_test_tools.file_utils.**hash_file**(*filename*, *algorithm='sha1'*, *block_size=65536*)

Creates a unique hash for a file.

Parameters

- **filename** – String with the full path to the file
- **algorithm** – String Algorithm to create the hash
- **block_size** – int for the size of the block while reading the file

Returns string the hash for the file

django_test_tools.file_utils.**json_serial**(*obj*)

JSON serializer for objects not serializable by default json code taken from: https://stackoverflow.com/questions/11875770/how-to-overcome-datetime-datetime-not-json-serializable

django_test_tools.file_utils.**parametrized**(*dec*)

Need to study this code. Got it from http://stackoverflow.com/questions/5929107/python-decorators-with-parameters

Parameters **dec** –

Returns

django_test_tools.file_utils.**serialize_data**(*data*, *output_file=None*, *format='json'*, ***kwargs*)

Quick function to serialize a data to file. The data keys will be saved in an alphabetical order for consistency purposes. If no output_file is supplied the function will created a dated file in the settings.TEST_OUTPUT_PATH folder. if the output_file is a folder the dated file will be created on the supplied folder with the serialized date. if the output_file is a file the data will be serialized to thar file

Parameters

- **data** – Dictionary or list to serialize
- **format** – Format to serialize to. Currently json is the only one supported
- **output_file** – File to output the data to
- **kwargs** –

django_test_tools.file_utils.**shorten_path**(*path*, *level=2*, *current_level=1*)
This method shortens the path by eliminating the folders on top.

```
filename = '/user/documents/personal/file.txt'
shortened = shorten_path(filename)
self.assertEqual(shortened, 'personal/file.txt')
```

Parameters

- **path** – string full path for the filename
- **level** – int, number of levels to show.
- **current_level** – int, recursing level.

Returns string shortened path

django_test_tools.file_utils.**temporary_file**(**args*, ***kwargs*)

django_test_tools.file_utils.**temporary_files**(**args*, ***kwargs*)

4.8 django_test_tools.mixins module

class django_test_tools.mixins.**JWTTestMixin**
Bases: object

delete_with_token(*url*, *access_token*)

get_access_token(*user*)

get_with_token(*url*, *access_token*)

put_with_token(*url*, *access_token*, *data*)

class django_test_tools.mixins.**TestCommandMixin**
Bases: object

This mixin helps capture the output of a command written with the stdout.write() method and the stderr.write

```
class TestYourCommand(TestCommandMixin, TestCase):

    def test_your_command_action(self):
        call_command('your_command', 'your_argument', stdout=self.content,␣
↪stderr=self.error_content)
        results = self.get_results()
        self.assertEqual(23, len(results))
```

get_errors()

get_results(*content=None*)

setUp()

class django_test_tools.mixins.**TestFixtureMixin**(*app_name=None*, ***kwargs*)
Bases: object

This a mixin to add to test cases to easily access fixtures. It assumes the the you have a package for your tests named **tests** and your **fixtures** are in a folder named fixtures within your tests package and that you have settings variable named APPS_DIR pointing tou your applications folder (this is created by Cookiecutter by default). Your tests package should look like this.

├── clinics

│ ├── __init__.py │ ├── admin.py │ ├── apps.py │ ├── exceptions.py │ ├── forms.py │ ├── migrations
│ ├── models.py │ ├── tests │ │ ├── __init__.py │ │ ├── factories.py │ │ ├── fixtures │ │ │ ├──
data.json │ │ │ ├── model_data.txt │ │ ├── test_forms.py │ │ ├── test_models.py │ ├── urls.py │ └──
views.py

For the above exmple

```
class TestClinicAdapter(TestFixtureMixin, TestCase):

    def setUp(self) -> None:
        self.app_name = 'clinics'

    def test_parse(self):
        clinics_dictionary = self.get_fixture_json('data.json')
        ...

    def test_read(self):
        filename = self.get_fixture_fullpath('model.txt')
        ...
```

app_name = None

get_fixture_fullpath(*fixture_filename*)
Get full patch for the fixture file

Parameters **fixture_filename** – <str> name of the fixture file

Returns <str> full path to fixture file

get_fixture_json(*fixture_filename*)
Reads a file and returns the json content.

Parameters **fixture_filename** – <str> filename

Returns <dict> With the file content

class django_test_tools.mixins.**TestOutputMixin**
Bases: object

clean_output = True

clean_output_folder(*dated_filename*)

get_csv_content(*filename*, *delimiter=', '*, *encoding='utf-8'*)

get_excel_content(*filename*, *sheet_name=None*)
Reads the content of an excel file and returns the content a as list of row lists. :param filename: string full path to the filename :param sheet_name: string. Name of the sheet to read if None will read the active sheet :return: a list containing a list of values for every row.

get_txt_content(*filename*, *encoding='utf-8'*)

4.9 django_test_tools.models module

4.10 django_test_tools.urls module

4.11 django_test_tools.utils module

class django_test_tools.utils.**SpanishDate**
Bases: object

parse(*str_date*)

to_string(*m_date*)

class django_test_tools.utils.**Timer**
Bases: object

Class to measure time elapsed

Example:

```
def test_performance(self):
    with Timer() as stopwatch:
        web_service = WebServiceUtil()
        web_service.consume_date(12)
    elapsed_milliseconds = stopwatch.elapsed*1000
    logger.debug('Elapsed: {} ms'.format(elapsed_milliseconds))
    self.assertTrue(elapsed_milliseconds <= 500)
```

get_elapsed_time()

get_elapsed_time_str()

reset()

running

start()

stop()

django_test_tools.utils.**add_date_to_filename**(*filename*, ***kwargs*)

Adds to a filename the current date and time in '%Y%m%d_%H%M' format. For a filename /my/path/myexcel.xlsx the function would return /my/path/myexcel_20170101_1305.xlsx. If the file already exists the function will add seconds to the date to attempt to get a unique name.

param filename string with fullpath to file or just the filename

param kwargs dictionary. date_position: suffix or preffix, extension: string to replace extension

return string with full path string incluiding the date and time

Note: Deprecated: Should use django_test_tools.file_utils.add_date() function

class django_test_tools.utils.**cd**(*newPath*)
Bases: object

Context manager for changing the current working directory

django_test_tools.utils.**convert_to_snake_case**(*camel_case*)
Converts a CamelCase name to snake case. ..code-block:: python

camel_case = 'OperatingSystemLongName' snake_case = convert_to_snake_case(camel_case) self.assertEqual(snake_case, 'operating_system_long_name')

Parameters **camel_case** – string. Camel case name

Returns string. Snake case name

django_test_tools.utils.**create_output_filename_with_date**(*filename*)

Based on the filename will create a full path filename incluidn the date and time in '%Y%m%d_%H%M' format. The path to the filename will be set in the TEST_OUTPUT_PATH settings variable.

param filename base filename. my_excel_data.xlsx for example

return string, full path to file with date and time in the TEST_OUTPUT_PATH folder

---

Note: Deprecated: Should use django_test_tools.file_utils.create_dated() function

---

django_test_tools.utils.**daterange**(*start_date*, *end_date*)
DEPRECATED use utils.weekdays() function instead :param start_date: :param end_date: :return:

django_test_tools.utils.**datetime_to_local_time**(*date_time*)
Converts a naive date to a time zoned date based in hte setting.TIME_ZONE variable. If the date has already a time zone it will localize the date. :param date_time: <date> or <datetime> to be localized :return: localized non naive datetime

django_test_tools.utils.**deprecated**(*func*)

This is a decorator which can be used to mark functions as deprecated. It will result in a warning being emitted when the function is used.

from: https://wiki.python.org/moin/PythonDecoratorLibrary#CA-92953dfd597a5cffc650d5a379452bb3b022cdd0_7

django_test_tools.utils.**dict_compare**(*d1*, *d2*)

django_test_tools.utils.**force_date_to_datetime**(*unconverted_date*, *tzinfo=<UTC>*)

django_test_tools.utils.**load_json_file**(*filename*)

django_test_tools.utils.**parse_spanish_date**(*str_date*)

django_test_tools.utils.**versiontuple**(*v*)

django_test_tools.utils.**weekdays**(*start_date*, *end_date*)
Returns a generator with the dates of the week days between the start and end date

```
start_date = datetime.date(2016, 10, 3)  # Monday
end_date = datetime.date(2016, 10, 7)  # Friday
days = list(weekdays(start_date, end_date))
self.assertEqual(5, len(days))
```

Parameters

- **start_date** – date. Start date
- **end_date** – date. End date

4.12 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/luiscberrocal/django-test-tools/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" is open to whoever wants to implement it.

5.1.3 Implement Features

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

5.1.4 Write Documentation

Django Test Tools could always use more documentation, whether as part of the official Django Test Tools 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/luiscberrocal/django-test-tools/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-test-tools* for local development.

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

```
$ git clone git@github.com:your_name_here/django-test-tools.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-test-tools
$ cd django-test-tools/
$ 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_test_tools 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.

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

5.4 Tips

To run a subset of tests:

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

CHAPTER 6

Credits

6.1 Development Lead

- Luis Carlos Berrocal <luis.berrocal.1942@gmail.com>

6.2 Contributors

Issis Itzel Montilla <issis.montilla@gmail.com>

CHAPTER 7

History

7.1 0.1.0 (2017-04-26)

- First release on PyPI.

Python Module Index

d

Index

E

F

G

H