django-whatever Documentation

Release 0.3.1

coagulant

Oct 12, 2021

Contents

Note: django-whatever is a friendly fork of `django-any` package written by Mikhail Podgurskiy (kmmbvnr) The purpose of the fork is to fix most annoying bugs and add *some features*. To remain compatible with original package `django-whatever` retains same namespace: `django_any`.

CHAPTER 1

Introduction

`django-whatever` is explicit replacement for old-style, big and error-prone implicit fixture files.

`django-whatever` allows you to specify only fields important for tests and fills the rest randomly with acceptable values.

It makes tests clean and easy to understand, without reading fixture files.

```
from django_any import any_model

class TestMyShop(TestCase):
    def test_order_updates_user_account(self):
        account = any_model(Account, amount=25, user__is_active=True)
        order = any_model(Order, user=account.user, amount=10)
        order.proceed()

        account = Account.objects.get(pk=account.pk)
        self.assertEquals(15, account.amount)
```

CHAPTER 2

Contents

2.1 Installation

Warning: To retain backwards compatibility with code written for `django-any`, `django-whatever` package shares the same namespace, so make sure to remove previous `django_any` installations if you have any.

- Install the package with your python package manager:

```
pip install django-whatever
```

- Add `'django_any'` to your `INSTALLED_APPS` setting (it's not a typo, we use same app name as `django_any`):

```
INSTALLED_APPS = (
    ...
    "django_any",
    ...
)
```

2.2 Models creation

You can get model instance saved in database without secifying any model fields

Given you have a simple polls model from django tutorial:

```
#models.py
class Poll(models.Model):
    question = models.CharField(max_length=200)
    pub_date = models.DateTimeField('date published')
```

You can create random poll instance not having to provide question text or pub_date, if they're not relevant for your test case:

```
#tests.py
from django_any import any_model
user = any_model(Poll)
```

It will create a poll instance with both fields filled with meaningless but valid data:

```
'question': 'KStKMESUXDjnlDNcJLsAiLcZQGnVXhORIKxWYtPwiqgVXgFvgpmMajwbGFRkoCo'
'pub_date': datetime.datetime(2002, 10, 1, 6, 1, 16)
```

django-any will fill all required foreign keys and create related model instances as well.

2.2.1 Specifying values

If you need an instance with specific values you can provide them this way:

```
user = any_model(Poll, question='To be or not ot be?')
```

Note, that `pub_date` will still be generated randomly for you.:

```
'question': 'To be or not ot be?'
'pub_date': datetime.datetime(2004, 5, 11, 16, 48, 35)
```

2.2.2 Constraints

`django-whatever` will preserve all field constraints, such as max_length, and choices when filling models with random data. It also tries to honor model custom validation by making model instances until `full_clean()` returns `True`.

2.2.3 Relations

`django-whatever` supports Django ORM-like *double-underscore* syntax for setting values for related instances:

```
#models.py
class Choice(models.Model):
    poll = models.ForeignKey(Poll, on_delete=models.CASCADE)
    choice = models.CharField(max_length=200)
    votes = models.IntegerField()

#tests.py
order = any_model(Choice, poll__pub_date = datetime.now())
```

2.2.4 Fixtures

In case you need data from fixtures for creation of your models you can use `Q` objects to select values for fields from db:

```
order = any_model(Order, customer__location=Q(country='US'))
```

It will create an order for existing customer, whose country is US.

2.2.5 Custom model fields

It's quite common to create custom model fields to store data. To let `django-whatever` know how to generate random data for this filed you should register it explicitly:

```
@any_field.register(model_utils.field.AutoCreatedField)
def any_auto_created_field(field, **kwargs):
    return datetime.datetime.now()
```

Note: When project defines a lot of custom fields, it can be a hassle to register all of them manually. Quite often the registered sample data for the field the custom field was inherited from will work just as well, so django-whatever tries to use parent generator if no function is registered for a custom field.

2.3 Forms creation

You can get data for form submissions without secifying form fields (it acs very much like *any_model*).

Given you have a simple contact form:

```
#forms.py
class ContactForm(forms.Form):
    subject = forms.CharField(max_length=100)
    message = forms.CharField(max_length=10)
    sender = forms.EmailField()
    cc_myself = forms.BooleanField(required=False)
```

You can create a random data to test that form this way:

```
from django_any.forms import any_form
post, files = any_form(ContactForm)
```

It will create a tuple dictionaries with `POST` and `FILES` data:

```
>> post
{'cc_myself': 'False',
 'message': 'wEqhESoSOO',
 'sender': 'bbfGIXwKrd@KPXRDCAyJD.xyR',
 'subject': 'hVocAZRuZPRHuVyWCV'
}
>> files
{}
```

Now we can pass the data to the form and check if it validates:

```
test_form = ContactForm(post, files)
self.assertTrue(test_form.is_vlaid())
```

For advances usage see *post_any_data*

2.4 Additional features

django-any provides clean API for creating users and authenticating as users.

2.4.1 Creating users

`django_any.contrib.`**`any_user`**(*password=None*, *permissions=[]*, *groups=[]*, ***kwargs*)

Without arguments it creates non-superuser and non-staff active user. Function can simultaneusly grant permissions to user and assign him to specified groups (groups and permissons are not created and should exist prior to function call). You can also provide any key-valued arguments, which `User` model takes.

```
from django_any.contrib import any_user
# create random user
foo = any_user()

# create inactive superuser
any_user(is_superuser=True, is_active=False, is_staff=True)

# create user with permissions 'can_add' and 'can_delete' in app books
any_user(permissions=['books.can_add', 'books.can_delete'])

# create user in group 'Special users'
any_user(groups=['Special users'])
```

2.4.2 Custom test client

Django-whatever has custom test clent, that extends default django client. It provides two useful methods for authorization and forms posting.

`login_as(self, **kwargs):`

Log into site as random user. Key-valued arguments are the same as for `any_user` function. To log in as specific user, provide argument `user` (note, that user password will be reset).

```
from django_any.test import Client
self.client = Client()
# log in as admin
self.client.login_as(is_superuser=True)

# log in as specific user
user = User.objects.create_user('john', 'lennon@thebeatles.com', 'johnpassword')
self.client.login_as(user=user)
```

`post_any_data(self, url, extra=None, context_forms=_request_context_forms, **kwargs):`

Posts random froms data to `url`. Additional data can be passed in dictionary format to `extra` argument. By default, it's assumed that forms are rendered by same url, their names and default values are taken from context and can be overriden by `context_forms` argument.

2.4.3 Creating models with default values

Basic `any_model` provides totally random values that pass validation and meet requiremnts of creation, but sometimes it's useful to keep defaults. In those cases it's recommended to use `any_model_with_defaults`.

`any_model_with_defaults(modelClass, **kwargs):`

```
from django_any.contrib import any_model_with_defaults

#models.py
```

(continues on next page)

(continued from previous page)

```
class Poll(models.Model):
    question = models.CharField(max_length=200)
    pub_date = models.DateTimeField('date published', defalut=datetime.
↪datetime(2000, 12, 10))

#tests.py
poll = any_model_with_defaults(Poll)
```

Note, that `question` value is random, but `pub_date` is taken from `default` attribute:

```
'question': 'HJ:34KW<DGdfSgfL67KVRD:'
'pub_date': datetime.datetime(2000, 12, 10)
```

2.5 Debugging

It is recomended to specify `django_any.WithTestDataSeed` as metaclass for your TestCase:

```
from django_any import any_model, WithTestDataSeed

class SiteTests(TestCase):
    __metaclass__ = WithTestDataSeed

    def test_something(self):
        ....
```

If you test sometimes fails, in error log, you could found used random seed.:

```
======================================================================
FAIL: test__something (mysite.SiteTests) With seed 1434556623
```

You could use this seed, to repeat and debug you tests, with exactly the same random data:

```
from django_any import any_model, WithTestDataSeed, with_seed, without_random_seed

class SiteTests(TestCase):
    __metaclass__ = WithTestDataSeed

    @without_random_seed
    @with_seed(1434556623)
    def test_something(self):
        ....
```

`without_random_seed` decorator disables test run with random seed, and `with_seed` runs test with selected seed.

2.6 Xunit reference

The python basic types generators

`django_any.xunit.`**`any_boolean`**`()`
Returns True or False

```
>>> result = any_boolean()
>>> type(result)
<type 'bool'>
```

django_any.xunit.**any_date**(*from_date=None*, *to_date=None*)
Return random date from the [from_date, to_date] interval

Parameters

- **from_date** – default value: date(1990, 1, 1)
- **to_date** – default value: date.today()

```
>>> result = any_date(from_date=date(1990, 1, 1), to_date=date(1990, 1, 3))
>>> type(result)
<type 'datetime.date'>
>>> result >= date(1990, 1, 1) and result <= date(1990, 1, 3)
True
```

django_any.xunit.**any_datetime**(*from_date=None*, *to_date=None*)
Return random datetime from the [from_date, to_date] interval

Parameters

- **from_date** – default value: datetime(1990, 1, 1)
- **to_date** – default value: datetime.now()

```
>>> result = any_datetime(from_date=datetime(1990, 1, 1), to_date=datetime(1990,␣
↪1, 3))
>>> type(result)
<type 'datetime.datetime'>
>>> result >= datetime(1990, 1, 1) and result <= datetime(1990, 1, 3)
True
```

django_any.xunit.**any_decimal**(*min_value=Decimal('0')*, *max_value=Decimal('99.99')*, *decimal_places=2*)
Return random decimal from the [min_value, max_value] interval

```
>>> result = any_decimal(min_value=0.999, max_value=3, decimal_places=3)
>>> type(result)
<class 'decimal.Decimal'>
>>> result >= Decimal('0.999') and result <= Decimal(3)
True
```

django_any.xunit.**any_email**()
Return random email

```
>>> import re
>>> result = any_email()
>>> type(result)
<type 'str'>
>>> from django.core.validators import EmailValidator
>>> EmailValidator(result)
None
```

django_any.xunit.**any_float**(*min_value=0*, *max_value=100*, *precision=2*)
Returns random float

```
>>> result = any_float(min_value=0, max_value=100, precision=2)
>>> type(result)
<type 'float'>
>>> result >=0 and result <= 100
True
```

django_any.xunit.**any_int**(*min_value=0*, *max_value=100*, ***kwargs*)

Return random integer from the selected range

```
>>> result = any_int(min_value=0, max_value=100)
>>> type(result)
<type 'int'>
>>> result in range(0,101)
True
```

django_any.xunit.**any_letter**(*letters='abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ'*, ***kwargs*)

Return random letter

```
>>> result = any_letter(letters=ascii_letters)
>>> type(result)
<type 'str'>
>>> len(result)
1
>>> result in ascii_letters
True
```

django_any.xunit.**any_string**(*letters='abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ'*, *min_length=3*, *max_length=100*)

Return string with random content

```
>>> result = any_string(letters=ascii_letters, min_length=3, max_length=100)
>>> type(result)
<type 'str'>
>>> len(result) in range(3,101)
True
>>> any([c in ascii_letters for c in result])
True
```

django_any.xunit.**weighted_choice**(*choices*)

Supposes that choices is sequence of two elements items, where first one is the probability and second is the result object or callable

```
>>> result = weighted_choice([(20,'x'), (100, 'y')])
>>> result in ['x', 'y']
True
```

2.7 Changelog

2.7.1 dev

- run tests on Python 3.9
- add Django 3.1 to the tests matrix

- move from Travis to GitHub actions for automated testing and linting

2.7.2 0.3.1

- automated tests in Travis CI
- fixed docs builds
- fix "Invalid sequence" python warnings

2.7.3 0.3.0

- dropped Django 1.1, 2.0, 2.2 and Python 2.7, 3.4 support
- added Django 3.0 and Python 3.8 suport
- added automated codestyle checks

2.7.4 0.2.4

- A bunch of maintenance updates that happened during the "unmaintained" mode.
- The last version with Django 1.11, 2.0, and 2.1 support

2.7.5 0.2.3

- Values for custom fields are created with their parent generator if the field is not registered (#22)

2.7.6 0.2.2

- Fixed pip installation
- Added ability to login_as arbitrary user

2.7.7 0.2.1

- Added xunit reference docs
- Fixed `any_text_field` return value
- Updated setup.py to use versiontools
- Minor updates and bugfixes in docs and LICENCE

2.7.8 0.2.0

- Fixed `ImportError: cannot import name _strclass` for python 2.7
- Added `any_model_with_defaults` function to generate models with default values where acceptible
- Fixed tests for django 1.4 compatibility
- Added support for GenericIPAddressField in both model and forms (django 1.4 and above)
- Multiple minor updates and bugfixes in docs

2.7.9 0.1.0

- Forked django-any and renamed to django-whatever
- Created complete documentation for package
- Models with `GenericForeignKey` can be created with `any_model(MyModel, content_object=object)`
- Self-referencing models no longer produce `"RuntimeError: maximum recursion depth exceeded"`
- `ImageField` and naive callable `upload_to` support.

2.7.10 django-any changelog

Not maintained

CHAPTER 3

Development

You can grab latest code on Github.

Feel free to submit issues, pull requests are also welcome.

CHAPTER 4

Authors & contributors

- kmmbvnr (`django-any` author)
- Vitaa
- Coagulant (`django-whatever` maintainer)

Python Module Index

d

Index

A

D

W