pytest-djangoapp Documentation

Release 1.2.0

Igor ‘idle sign‘ Starikov

Jul 06, 2023

Contents

https://github.com/idlesign/pytest-djangoapp

CHAPTER 1

Description

Nice pytest plugin to help you with Django pluggable application testing.

This exposes some useful tools for Django applications developers to facilitate tests authoring, including:

- Settings overriding
- Template tags testing
- User creation
- Request object creation
- Management command calls
- Mailing
- Migrations
- Messages
- DB queries audit
- Live server & client UI testing
- etc.

Suitable for testing apps for Django 1.8+.

CHAPTER 2

Requirements

1. Python 3.6+

CHAPTER 3

Table of Contents

3.1 Quickstart

3.1.1 Application structuring

Let's say you have classical tests placing (inside application directory):

```
package_dir
|__ myapp
|   |__ __init__.py
|   |__ tests
|   |   |__ __init__.py
|   |   |__ conftest.py  <- Configure djangoapp here.
|
|__ setup.py
```

3.1.2 Configuring djangoapp

Add the following lines into *conftest.py* to configure *djangoapp* and start using it:

```
# conftest.py
from pytest_djangoapp import configure_djangoapp_plugin

pytest_plugins = configure_djangoapp_plugin()
```

You can override default settings:

```
pytest_plugins = configure_djangoapp_plugin({
    'DEBUG': False,
    'SOMETHING': 'else,
})
```

Sometimes you may want to extend default settings, such as *INSTALLED_APPS* or *DATABASES*. To do this you can pass *extend_*-prefixed arguments:

```
pytest_plugins = configure_djangoapp_plugin(
    extend_INSTALLED_APPS=[
        'django.contrib.sites',
    ],
    extend_MIDDLEWARE=[
        'django.contrib.sessions.middleware.SessionMiddleware',
    ],
    extend_DATABASES={
        'dummy': {
            'ENGINE': 'django.db.backends.sqlite3',
            'NAME': ':memory:',
        },
    },
)
```

If you test an application providing some integration with Django Admin contrib, there is a convenient *admin_contrib* argument you can pass to activate this contrib:

```
pytest_plugins = configure_djangoapp_plugin(admin_contrib=True)
```

By default all DB migrations are applied on test runs. Since that can be unnecessary and take a long time, there is a *migrate* argument that you can always set to `False`:

```
pytest_plugins = configure_djangoapp_plugin(migrate=False)
```

You can further change or altogether replace generated settings using *settings_hook* argument:

```
def hook(settings):
    # Do something with settings.
    return settings

pytest_plugins = configure_djangoapp_plugin(
    settings_hook=hook,
)
```

3.1.3 Testing an entire project

Note: Requires Python 3.

Despite the fact that *djangoapp* is primarily aimed to reusable Django applications testing one can use it also to test a project (a set of apps). For that, pass a dotted settings module path into *settings* argument:

```
pytest_plugins = configure_djangoapp_plugin(
    settings='myproject.settings.settings_testing',
)
```

3.1.4 Using fixtures

Use them just as you usually do with *pytest*:

```
# test_some.py

def test_this(settings, request_client):

    # We use `settings` fixture to temporarily override
    # project settings.
    with settings(DEBUG=True, MYVAR='someval'):
        # Now do some testing, with settings overridden.
        ...

    # And we use `request_client` fixture
    # to test our [AJAX] view.
    client = request_client(ajax=True)

    # We pass a tuple with a view name with arguments to not to bother with URL.
    response = client.get(('someview', {'somearg': 'one', 'otherarg': 33}))

    ...

    # See fixtures documentation for more fixtures.
```

3.1.5 Additional app for testing

Sometimes your app may provide tooling for other apps (say it automatically imports modules from them, or provides some urlpatterns). If so, you may want to simulate that other application in your tests.

You can easily do that by adding `testapp` package under your test directory (this will be automatically added to `INSTALLED_APPS` and treated by Django just as any application package):

```
package_dir
|__ myapp
|   |__ __init__.py
|   |__ tests
|   |   |__ __init__.py
|   |   |__ testapp  <- Thirdparty app simulation package.
|   |   |   |__ __init__.py
|   |   |   |__ admin.py  <- This module uses primitives provided by your app.
|   |   |   |__ models.py  <- This module uses base models provided by your app.
|   |   |   |__ urls.py  <- And this module uses urlpatterns provided by your app.
|   |   |__ conftest.py
|
|__ setup.py
```

3.2 Fixtures

`djangoapp` offers you various `pytest` fixtures falling into the categories below:

3.2.1 Commands

These fixtures are related to Django management commands.

`pytest_djangoapp.fixtures.commands.`**`command_makemigrations`**(*conf_app_name*)
Allows to run makemigrations command.

Note: This command can be useful to generate migrations for your application (without a project creation).

Example:

```
def test_makemigrations(command_makemigrations):
    command_makemigrations()
```

Parameters

- **app** – Application name to run 'makemigrations' for. * By default, a name from 'conf_app_name' fixture is used. * If empty string, command is run for any application.
- **args** – Additional arguments to pass to the command.

pytest_djangoapp.fixtures.commands.**command_run**()
Allows management command run.

Example:

```
def test_this(command_run, capsys):
    result = command_run('my_command', args=['one'], options={'two': 'three'})
    out, err = capsys.readouterr()
```

Warning: Django < 1.10 will always return *None*, no matter what command returns.

Parameters

- **command_name** – Command name to run.
- **args** – Required arguments to pass to a command.
- **options** – Optional arguments to pass to a command.

Returns Command output.

3.2.2 Database

These fixtures are related to Django database operation, including issued by ORM.

class pytest_djangoapp.fixtures.db.**Queries**
Allows access to executed DB queries.

clear(*db_alias: str = None*)
Clear queries for the given or default DB.

Parameters **db_alias** – Database alias. Default is used if not given.

clear_all()
Clears all queries logged for all DBs.

get_log(*db_alias: str = None*) → deque

Parameters **db_alias** –

sql(*db_alias: str = None*, *, *drop_auxiliary: bool = True*) → List[str]
Returns a list of queries executed using the given or default DB.

Parameters

- **db_alias** – Database alias. Default is used if not given.
- **drop_auxiliary** – Filter out auxiliary SQL like: * BEGIN * COMMIT * END

time(*db_alias: str = None*) → decimal.Decimal
Returns total time executing queries (in seconds) using the given or default DB.

Parameters **db_alias** – Database alias. Default is used if not given.

pytest_djangoapp.fixtures.db.**db_queries**(*settings*) → pytest_djangoapp.fixtures.db.Queries
Allows access to executed DB queries.

Example:

```
def test_db(db_queries):

    # Previous queries cleared at the beginning.
    assert len(db_queries) == 0

    ...  # Do some DB-related stuff.

    # Assert total queries on all DBs.
    assert len(db_queries) == 10

    # Default DBs SQLs with auxiliary commands filtered out by default.
    sqls = db_queries.sql()

    # Assert total execution time is less than a second.
    assert db_queries.time() < 1

    # Drop SQL gathered so far on default DB.
    db_queries.clear()
```

Warning: Requires Django 1.9+ to work.

3.2.3 Settings

These fixtures are related to Django project configuration.

pytest_djangoapp.fixtures.settings.**settings**() → pytest_djangoapp.fixtures.settings.SettingsProxy
Fixture allowing to temporarily override project settings.

Example:

```
def test_this(settings):
    # By one.
    settings.DEBUG = True

    # Batch.
    settings.update({
        'DEBUG': True,
        'ROOT_URLCONF': 'module.urls',
    })

    # Context manager.
```

(continues on next page)

(continued from previous page)

```
    with settings(DEBUG=True):
        ...
```

3.2.4 Request

These fixtures are related to Django request objects.

class pytest_djangoapp.fixtures.request.**DjagoappClient**(*, *ajax: bool = False*, *user: AbstractUser = None*, *enforce_csrf_checks: bool = False*, *raise_exceptions: bool = True*, *json: bool = False*, ***defaults*)

store_exc_info(***kwargs*)
Store exceptions when they are generated by a view.

class pytest_djangoapp.fixtures.request.**DjangoappRequestFactory**(*ajax=False*, *json=False*, ***defaults*)

generic(*method*, *path*, **args*, ***kwargs*)
Construct an arbitrary HTTP request.

pytest_djangoapp.fixtures.request.**request_client**()
Fixture allowing test client object generation.

Example:

```
def test_this(request_client):

    client = request_client()

    response = client.get(
        ('someview', {'somearg': 'one', 'otherarg': 33})
    ).content

    ...

    ajax_client = request_client(ajax=True)
    ...
```

Parameters

- **ajax** – Make AJAX (XMLHttpRequest) requests.
- **user** – User to perform queries from.
- **raise_exceptions** – Do not allow Django technical exception handlers to catch exceptions issued by views, propagate them instead.
- **json** – Encode data as JSON.

> **Warning:** To be used with Django 2.1+

- **kwargs** – Additional arguments for test client initialization.

pytest_djangoapp.fixtures.request.**request_factory**()
Fixture allowing request object generation.

Example:

```
def test_this(request_factory):
    factory = request_factory()
```

Parameters **kwargs** –

pytest_djangoapp.fixtures.request.**request_get**(*request_factory*)
Fixture allowing GET request object generation.

Example:

```
def test_this(request_get):
    request = request_get('/some')
```

Parameters

- **path** –
- **user** – User making this request.
- **ajax** – Make AJAX (XMLHttpRequest) request.
- **kwargs** – Additional arguments for .get() method.

pytest_djangoapp.fixtures.request.**request_post**(*request_factory*)
Fixture allowing POST request object generation.

Example:

```
def test_this(request_post):
    request = request_post('/some', {'a': 'b'})
```

Parameters

- **path** –
- **data** – Data to post.
- **user** – User making this request.
- **ajax** – Make AJAX (XMLHttpRequest) request.
- **kwargs** – Additional arguments for .post() method.

3.2.5 Templates

These fixtures are related to Django templates system.

pytest_djangoapp.fixtures.templates.**template_context**(*request_get*, *user_create*)
Creates template context object.

To be used with *template_render_tag* fixture.

Example:

```
def test_this(template_context):
    context = template_context({'somevar': 'someval'})
```

Parameters

- **context_dict** – Template context. If not set empty context is used.
- **request** – Expects HttpRequest or string. String is used as a path for GET-request.
- **current_app** –
- **user** – User instance to associate request with. Defaults to a new anonymous user. If string is passed, it is considered to be

pytest_djangoapp.fixtures.templates.**template_render_tag**()

Renders a template tag from a given library by its name.

Example:

```
def test_this(template_render_tag):
    rendered = template_render_tag('library_name', 'mytag arg1 arg2')
```

Parameters

- **library** – Template tags library name to load tag from.
- **tag_str** – Tag string itself. As used in templates, but without {% %}.
- **context** – Template context object. If not set, empty context object is used.

pytest_djangoapp.fixtures.templates.**template_strip_tags**()

Allows HTML tags strip from string.

To be used with *template_render_tag* fixture to easy result assertions.

Example:

```
def test_this(template_strip_tags):
    stripped = template_strip_tags('<b>some</b>')
```

Parameters

- **html** – HTML to strin tags from
- **joiner** – String to join tags contents. Default: |

3.2.6 Users

These fixtures are related to Django users.

pytest_djangoapp.fixtures.users.**user**(*user_create*) → Union[AbstractUser, AnonymousUser]

Exposes Django user object.

Shortcut for *user_create* fixture.

Example:

```
def test_this(user):
    username = user.username
```

Note: User password is accessible via *password_plain* attribute.

pytest_djangoapp.fixtures.users.**user_create**(*user_model*)
Allows Django user object generation.

Example:

```
def test_this(user_create):
    user = user_create()
```

Note: User password is accessible via *password_plain* attribute.

Parameters

- **superuser** – Whether to create a superuser.
- **anonymous** – Whether to create an anonymous user.
- **attributes** – Additional user object attributes to initialize.

pytest_djangoapp.fixtures.users.**user_model**() → Union[AbstractUser, AnonymousUser]
Returns user model class.

Example:

```
def test_this(user_model):
    model_cls = user_model
```

3.2.7 Mail

These fixtures are related to Django mailing system.

pytest_djangoapp.fixtures.mail.**mail_outbox**() → List[django.core.mail.message.EmailMessage]
Returns mail outbox: list of sent message objects.

Example:

```
def test_this(mail_outbox):
    first_subject = mail_outbox[0].subject
```

3.2.8 Messages

Fixtures related to Django Messages framework.

pytest_djangoapp.fixtures.messages.**messages**(*monkeypatch*) → pytest_djangoapp.fixtures.messages.DjangoappMessageStorage
Holds messages sent by Django Messages framework.

Attributes:

- all – a list of all messages
- tags – messages dictionary indexed by tags

Example:

```
def test_this(messages):
    assert len(messages.all)
    assert len(messages.tags['error'])
    assert 'another error' in messages
```

3.2.9 Migrations

These fixtures are related to Django migrations system.

pytest_djangoapp.fixtures.migrations.**check_migrations**(*command_makemigrations*)
Check if migrations are up to date (migration files exist for current models' state).

Example:

```
def test_this(check_migrations):
    result = check_migrations()
```

Parameters **app** – Application name to check migrations for.

3.2.10 Utils

Offers various utility fixtures.

pytest_djangoapp.fixtures.utils.**conf_app_name**() → str
Returns your application name (deduced or provided).

pytest_djangoapp.fixtures.utils.**get_stamp**() → str
Returns current timestamp as a string.

3.2.11 Live server & client

These instruments allows you to lunch a live servers and live clients, e.g. for UI testing purposes.

class pytest_djangoapp.fixtures.live.**LiveClient**(*, *browser: str*)
Base class for live clients.

class pytest_djangoapp.fixtures.live.**SeleniumLiveClient**(*, *browser: str*)
This live client wraps Selenium.

https://selenium-python.readthedocs.io/

pytest_djangoapp.fixtures.live.**liveclient**()
Runs a live client. Available as a context manager.

Example:

```
def test_live(liveserver, liveclient):

    with liveserver() as server:

        # Let's run Firefox using Selenium.
        with liveclient('selenium', browser='firefox') as client:
            selenium = client.handle  # Selenium driver is available in .handle

            # Let's open server's root URL and check heading 1 on that page
```

(continues on next page)

(continued from previous page)

```
            selenium.get(server.url)
            assert selenium.find_element('tag name', 'h1').text == 'Not Found'
```

pytest_djangoapp.fixtures.live.**liveserver**() → Type[pytest_djangoapp.fixtures.live.LiveServer]

Runs a live server. Available as a context manager.

> **Warning:** For Django >= 4.0

Example:

```
def test_live(liveserver):

    with liveserver() as server:
        print(f'Live server is available at: {server.url}')
```

Python Module Index

p

Index

S

T

U