
Pomodorr

Kamil Daniewski

Jun 20, 2020

CONTENTS:

1	Basic setup	1
1.1	Configure environment variables	1
1.2	Building and running containers	3
2	Docker Remote Debugging	5
2.1	Configure Remote Python Interpreter	6
2.2	Known issues	10
3	Indices and tables	13

BASIC SETUP

In order to recreate the structure of `.envs` directory that the project uses, start with creating the `.envs` directory in the root of the project, as it is done below.

```
pomodorr
├── .envs
│   ├── .local
│   │   ├── .django
│   │   ├── .postgres
│   └── .production
│       ├── .django
│       └── .postgres
```

1.1 Configure environment variables

The configurations listed below resemble the default environment setting files needed to set the project up:

- **local envs:**

pomodorr/.envs/.local/.django

General

USE_DOCKER=yes

IPYTHONDIR=/app/.ipython

Redis

REDIS_URL=redis://redis:6379/0

Celery / Flower

CELERY_FLOWER_USER=<your_celery_flower_user>

CELERY_FLOWER_PASSWORD=<your_celery_flower_password>

pomodorr/.envs/.local/.postgres

PostgreSQL

POSTGRES_HOST=postgres

POSTGRES_PORT=5432

POSTGRES_DB=pomodorr

```
POSTGRES_USER=<your_postgres_user>
POSTGRES_PASSWORD=<your_postgres_password>
```

- **production envs:**

pomodorr/.envs/.production/.django

```
# General
# DJANGO_READ_DOT_ENV_FILE=True
DJANGO_SETTINGS_MODULE=config.settings.production
DJANGO_SECRET_KEY=<your_secret_key>
DJANGO_ADMIN_URL=<your_admin_url>
DJANGO_ALLOWED_HOSTS=<your_hosting_domain>

# Security
# TIP: better off using DNS, however, redirect is OK too
DJANGO_SECURE_SSL_REDIRECT=False

# Email
MAILGUN_API_KEY=<your_mailgun_api_key>
DJANGO_SERVER_EMAIL=<your_server_email>
MAILGUN_DOMAIN=<your_mailgun_domain>

# AWS
DJANGO_AWS_ACCESS_KEY_ID=<your_aws_access_key_id>
DJANGO_AWS_SECRET_ACCESS_KEY=<your_aws_secret_access_key>
DJANGO_AWS_STORAGE_BUCKET_NAME=<your_aws_storage_bucket_name>

# Gunicorn
WEB_CONCURRENCY=4

# Sentry
SENTRY_DSN=<your_sentry_dsn>

# Redis
REDIS_URL=redis://redis:6379/0

# Celery / Flower
CELERY_FLOWER_USER=<your_celery_flower_user>
CELERY_FLOWER_PASSWORD=<your_celery_flower_password>
```

pomodorr/.envs/.production/.postgres

```
# PostgreSQL
POSTGRES_HOST=postgres
```

```
POSTGRES_PORT=5432
POSTGRES_DB=pomodorr
POSTGRES_USER=<your_postgres_user>
POSTGRES_PASSWORD=<your_postgres_password>
```

1.2 Building and running containers

The details about getting the project up and running is described on the official django-cookiecutter [documentation](#).

Having configured the environment variables all there is left to do is to build the docker containers and run them. Assuming your current directory is the root directory of the project, type:

```
$ docker-compose -f local.yml build
```

The last step is to get the containers up.

```
$ docker-compose -f local.yml up
```

Note:

You may encounter some problems with already used ports. In that situation, check the `pomodorr/local.yml` configuration file and change the clashing ports.

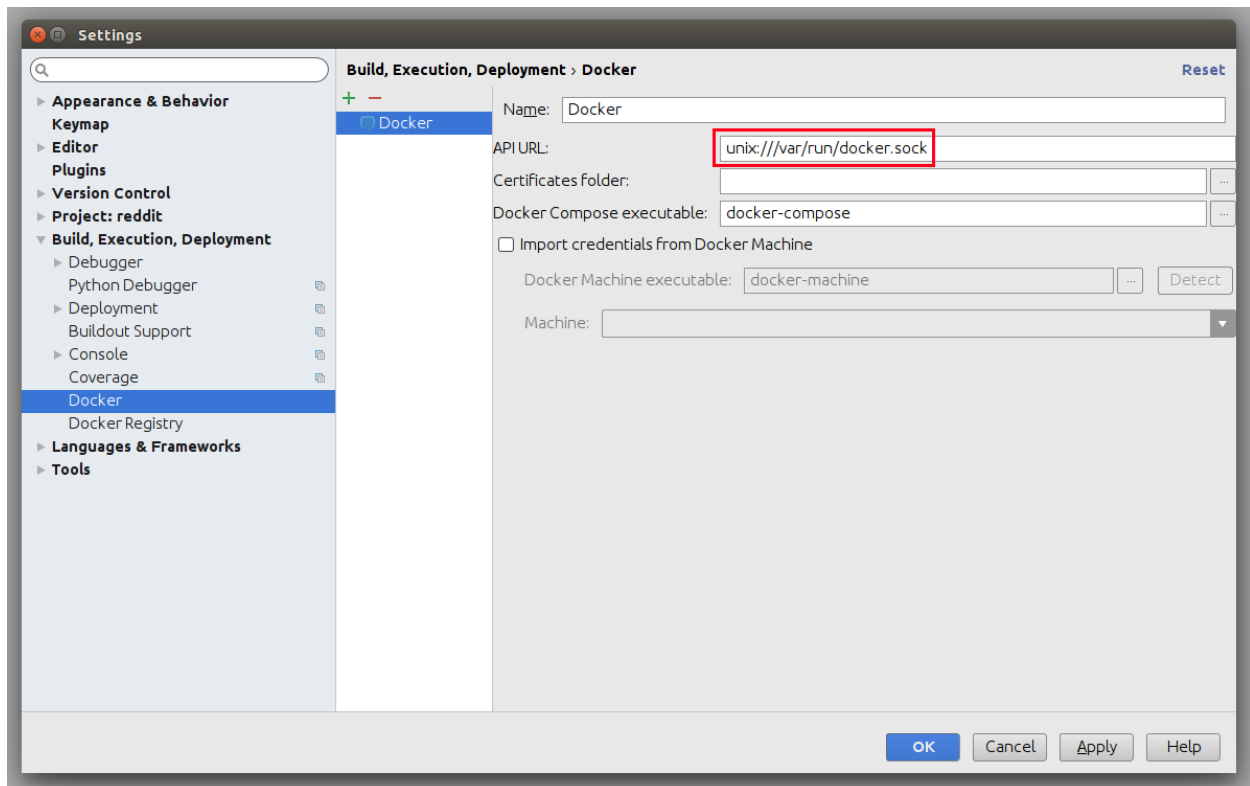
Likewise, in case of having troubles with setting the project up, please consider having a look at the [troubleshooting](#) page of the official django-cookiecutter documentation.

Otherwise feel free to send an email message or report an issue on the [github](#) if there is an evidence of a bug.

DOCKER REMOTE DEBUGGING

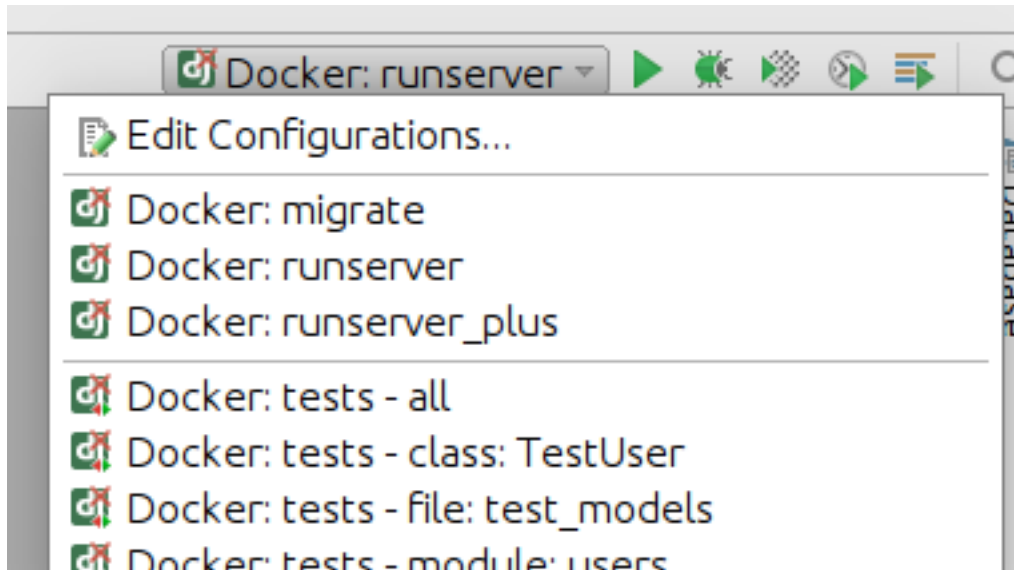
To connect to python remote interpreter inside docker, you have to make sure first, that Pycharm is aware of your docker.

Go to *Settings > Build, Execution, Deployment > Docker*. If you are on linux, you can use docker directly using its socket `unix:///var/run/docker.sock`, if you are on Windows or Mac, make sure that you have docker-machine installed, then you can simply *Import credentials from Docker Machine*.



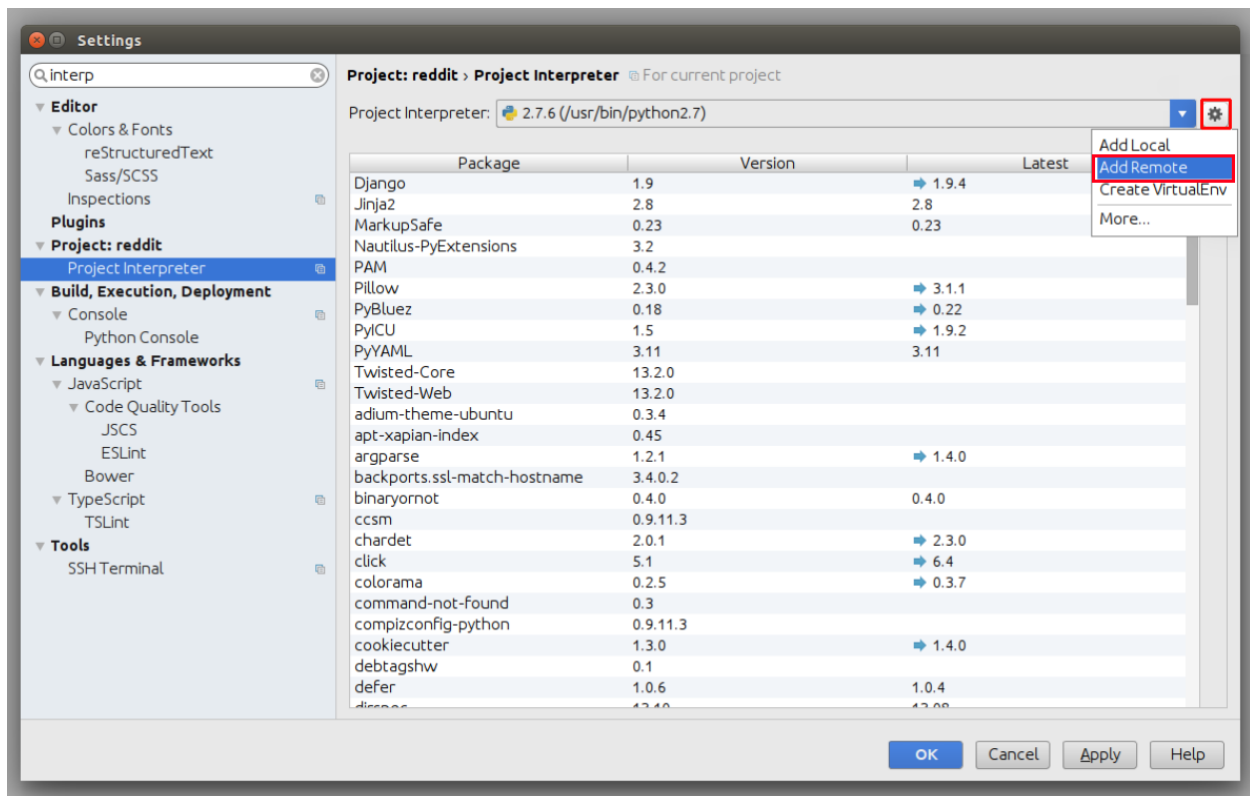
2.1 Configure Remote Python Interpreter

This repository comes with already prepared “Run/Debug Configurations” for docker.

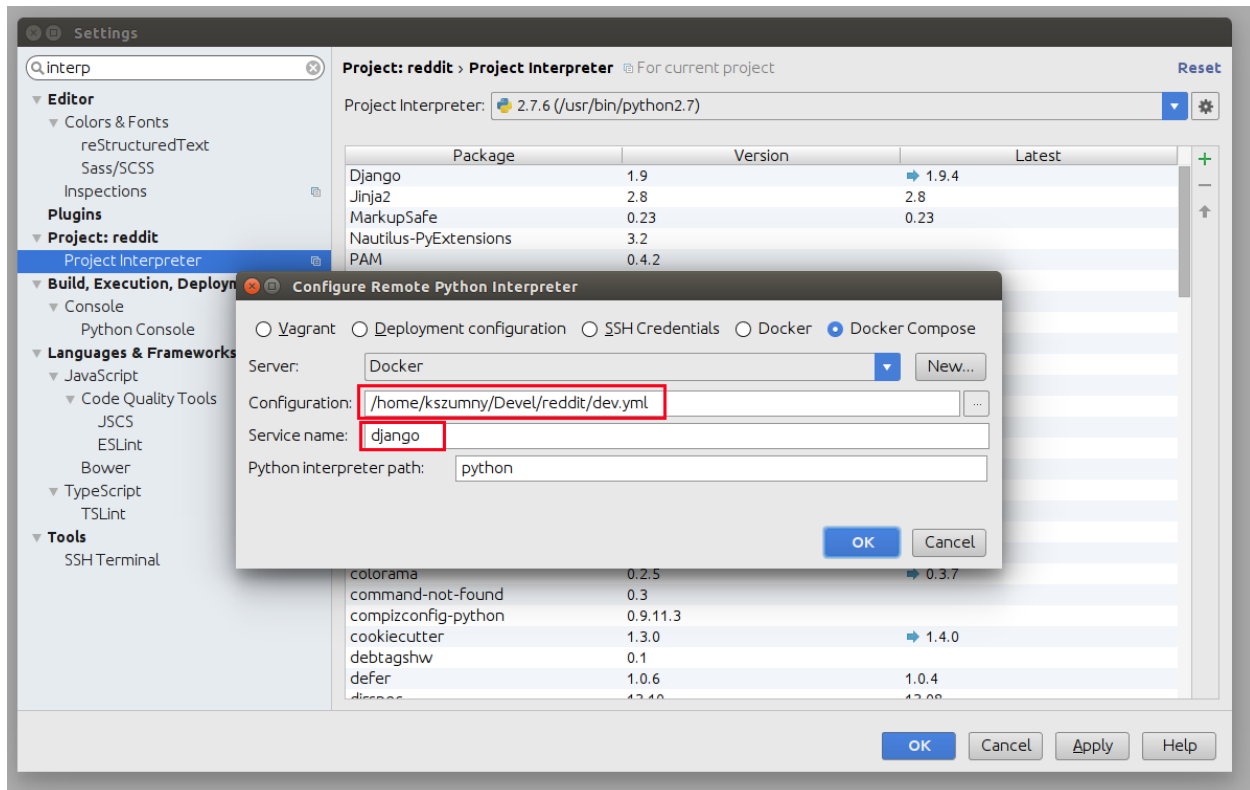


But as you can see, at the beginning there is something wrong with them. They have red X on django icon, and they cannot be used, without configuring remote python interpreter. To do that, you have to go to *Settings > Build, Execution, Deployment* first.

Next, you have to add new remote python interpreter, based on already tested deployment settings. Go to *Settings > Project > Project Interpreter*. Click on the cog icon, and click *Add Remote*.



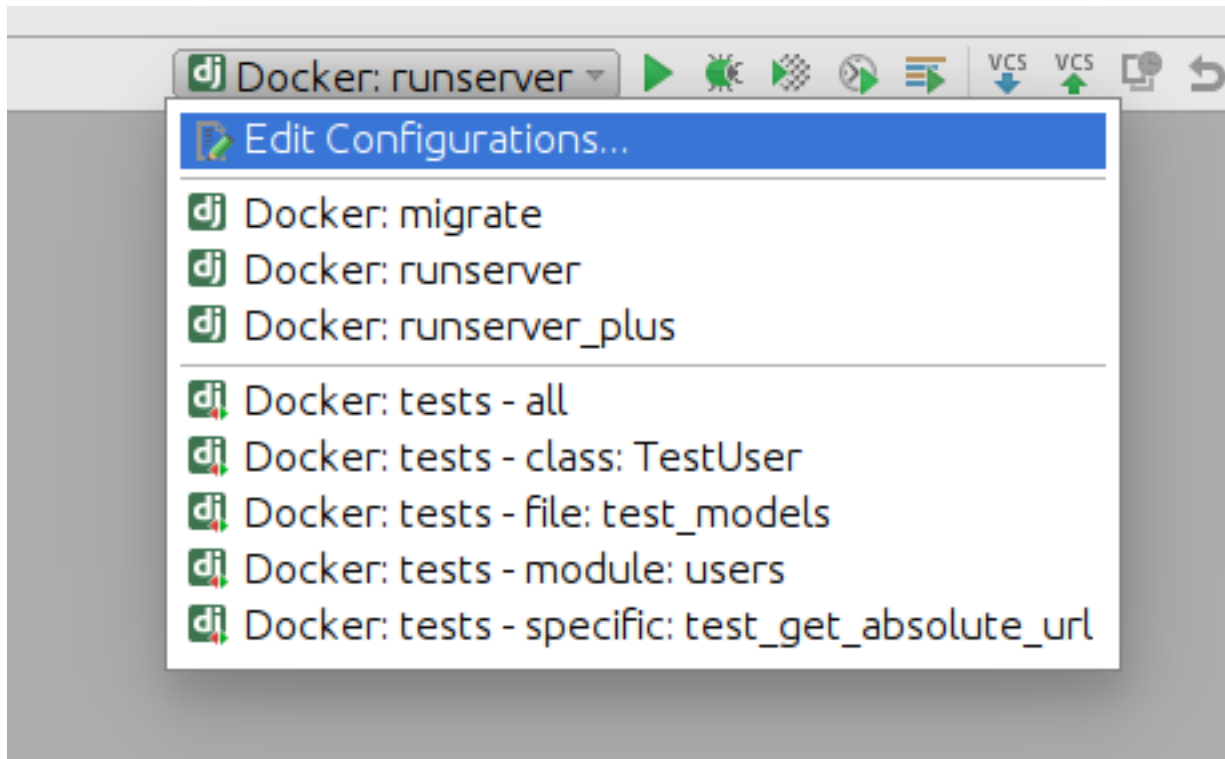
Switch to *Docker Compose* and select *local.yml* file from directory of your project, next set *Service name* to *django*



Having that, click *OK*. Close *Settings* panel, and wait few seconds. ...

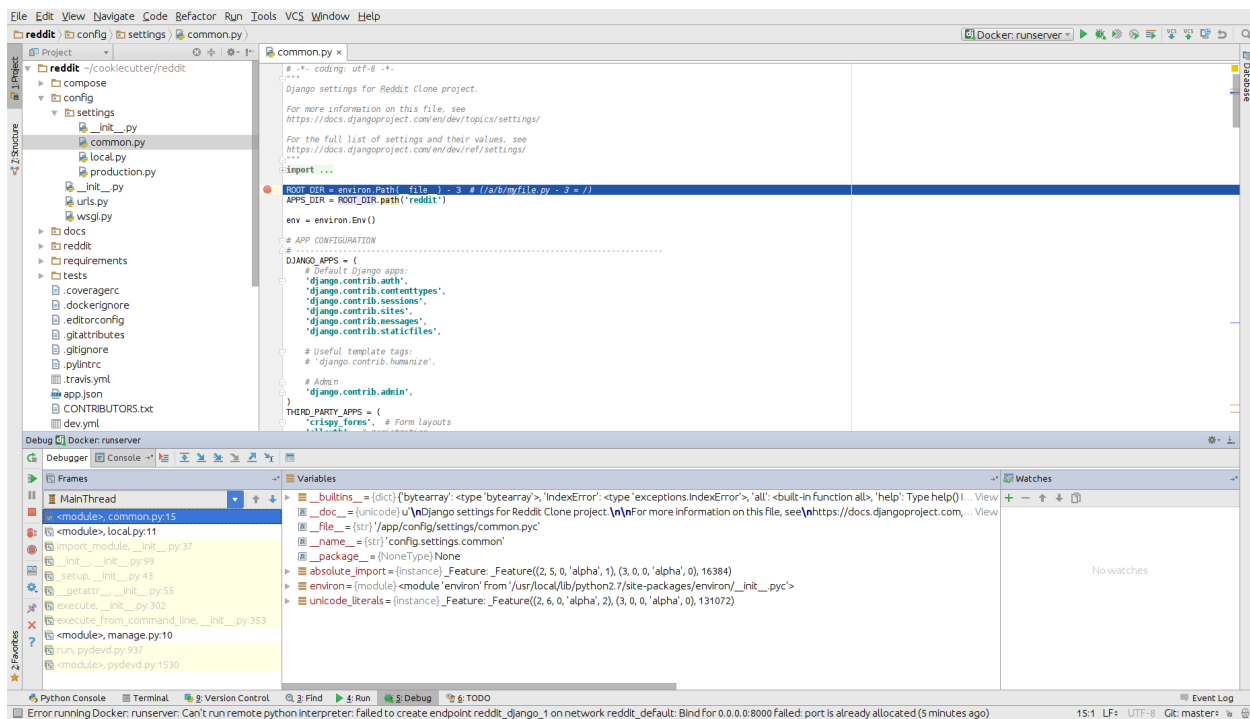


After few seconds, all *Run/Debug Configurations* should be ready to use.



Things you can do with provided configuration:

- run and debug python code



- run and debug tests

The top screenshot shows the PyCharm IDE with the 'test_views.py' file open. The code defines two test classes: `TestUserRedirectView` and `TestUserUpdateView`. The `TestUserRedirectView` class has a `test_get_redirect_url` method that tests the `get_redirect_url` method of the `UserRedirectView` class. The `TestUserUpdateView` class has a `test_get_object` method that tests the `get_object` method of the `UserUpdateView` class. The bottom screenshot shows the 'Run' tool window with the test results. All 7 tests passed. The 'Debug' tool window shows the current frame is `test_get_redirect_url` in `test_views.py`.

File Edit View Navigate Code Refactor Run Tools VCS Window Help

reddit / cookiecutter/reddit / compose / config / docs / reddit / contrib / static / taskapp / templates / users / migrations / tests / test_views.py

```

from django.test import RequestFactory
from test_plus.test import TestCase

from .views import (
    UserRedirectView,
    UserUpdateView
)

class BaseUserTestCase(TestCase):
    def setUp(self):
        self.user = self.make_user()
        self.factory = RequestFactory()

class TestUserRedirectView(BaseUserTestCase):
    def test_get_redirect_url(self):
        # Instantiate the view directly. Never do this outside a test!
        view = UserRedirectView()
        # Generate a fake request
        request = self.factory.get('/fake-url')
        # Attach the user to the request
        request.user = self.user
        # Attach the request to the view
        view.request = request
        # Expect: '/users/testuser/', as that is the default username for
        # self.make_user()
        self.assertEqual(
            view.get_redirect_url(),
            '/users/testuser/'
        )

class TestUserUpdateView(BaseUserTestCase):
    def setUp(self):
        # call BaseUserTestCase.setUp()
        super(TestUserUpdateView, self).setUp()
        # Instantiate the view directly. Never do this outside a test!
        self.view = UserUpdateView()
        # Generate a fake request
        request = self.factory.get('/fake-url')
        # Attach the user to the request
        request.user = self.user
        # Attach the request to the view
        self.view.request = request

```

Run Docker: tests - all All 7 tests passed - 328ms

Test Results	Duration
reddit.users.tests.test_models.TestUser	89ms
test_str	45ms
test_get_absolute_url	44ms
reddit.users.tests.test_views.TestUserRedirectView	40ms
test_get_redirect_url	40ms
reddit.users.tests.test_views.TestUserUpdateView	96ms
test_get_object	42ms
test_get_success_url	54ms
reddit.users.tests.test_admin.TestMyUserCreation	103ms
test_clean_username_false	53ms
test_clean_username_success	50ms

Python Console Terminal Version Control Find Run Debug TODO

Tests Passed: 7 passed (a minute ago)

9.1 LF: UTF-8 GC: master

File Edit View Navigate Code Refactor Run Tools VCS Window Help

reddit / cookiecutter/reddit / compose / config / docs / reddit / contrib / static / taskapp / templates / users / migrations / tests / test_views.py

```

def setUp(self):
    self.user = self.make_user()
    self.factory = RequestFactory()

class TestUserRedirectView(BaseUserTestCase):
    def test_get_redirect_url(self):
        self: test_get_redirect_url (reddit.users.tests.test_views.TestUserRedirectView)
        # Instantiate the view directly. Never do this outside a test!
        view = UserRedirectView()
        # Generate a fake request
        request = self.factory.get('/fake-url')
        # Attach the user to the request
        request.user = self.user
        # Attach the request to the view
        view.request = request
        # Expect: '/users/testuser/', as that is the default username for
        # self.make_user()
        self.assertEqual(
            view.get_redirect_url(),
            '/users/testuser/'
        )

class TestUserUpdateView(BaseUserTestCase):
    def setUp(self):
        # call BaseUserTestCase.setUp()
        super(TestUserUpdateView, self).setUp()
        # Instantiate the view directly. Never do this outside a test!
        self.view = UserUpdateView()
        # Generate a fake request
        request = self.factory.get('/fake-url')
        # Attach the user to the request
        request.user = self.user
        # Attach the request to the view
        self.view.request = request

```

Debug Docker: tests - all

Frames

- MainThread
- test_get_redirect_url test_views.py:22
- run_case.py:329
- __call__ case.py:393
- __call__ test_cases.py:214
- run_suite.py:108
- __call__ suite.py:70
- run_tcounttest.py:259
- run_suite, django_test_runner.py:151
- run_tests, runner.py:533
- run_tests, django_test_runner.py:156
- run_tests, django_test_runner.py:256
- handle, django_test_manage.py:93

Variables

self = (TestUserRedirectView) test_get_redirect_url (reddit.users.tests.test_views.TestUserRedirectView)

Watches

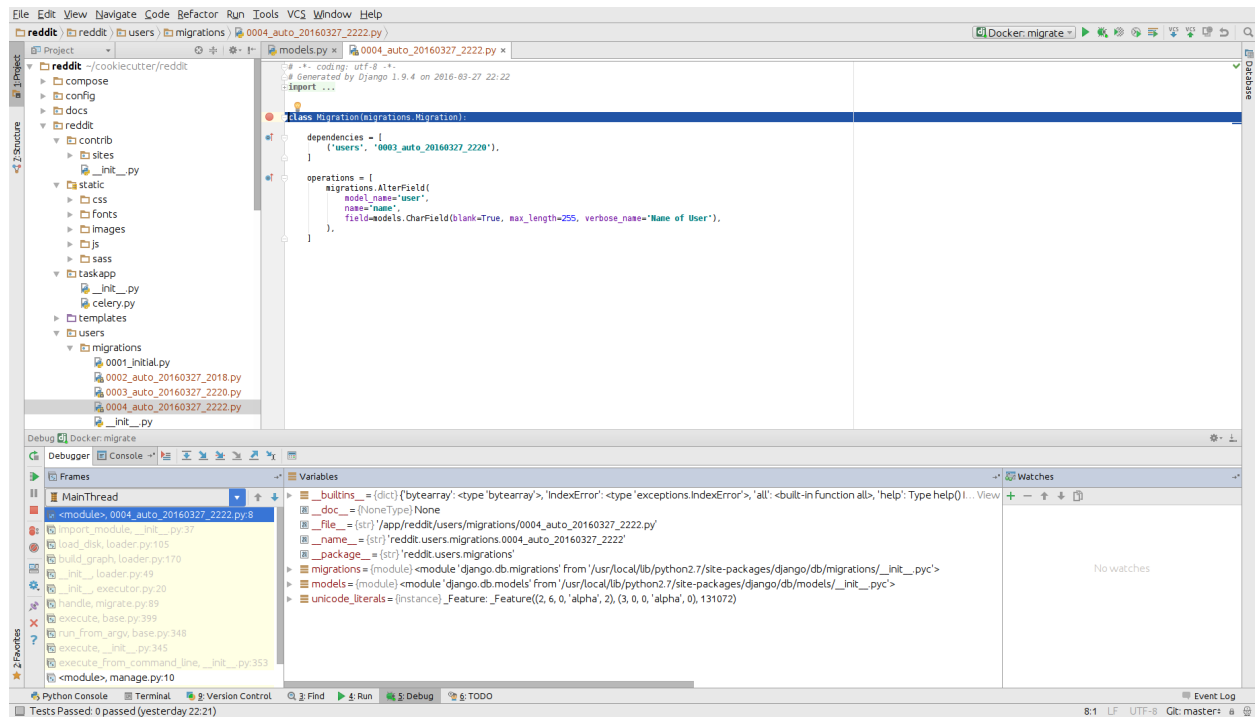
No watches

Python Console Terminal Version Control Find Run Debug TODO

Tests Passed: 7 passed (a minute ago)

22.1 LF: UTF-8 GC: master

- run and debug migrations or different django management commands



- and many others..

2.2 Known issues

- Pycharm hangs on “Connecting to Debugger”



This might be fault of your firewall. Take a look on this ticket - <https://youtrack.jetbrains.com/issue/PY-18913>

- Modified files in `.idea` directory

Most of the files from `.idea/` were added to `.gitignore` with a few exceptions, which were made, to provide “ready to go” configuration. After adding remote interpreter some of these files are altered by PyCharm:

```
$ git status
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

        modified:   .idea/project_name.iml

no changes added to commit (use "git add" and/or "git commit -a")
```

In theory you can remove them from repository, but then, other people will lose a ability to initialize a project from provided configurations as you did. To get rid of this annoying state, you can run command:

```
$ git update-index --assume-unchanged pomodorr.iml
```


INDICES AND TABLES

- `genindex`
- `modindex`
- `search`