
My Awesome Project

Andrew Chen Wang

Aug 15, 2020

CONTENTS:

1	How To - Project Documentation	1
1.1	Get Started	1
1.2	Docstrings to Documentation	1
2	Docker Remote Debugging	3
2.1	Configure Remote Python Interpreter	4
2.2	Known issues	8
3	Users	11
4	Indices and tables	13

HOW TO - PROJECT DOCUMENTATION

1.1 Get Started

Documentation can be written as rst files in the *my_awesome_project/docs/_source*.

To build and serve docs, use the commands:

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

Changes to files in *docs/_source* will be picked up and reloaded automatically.

[Sphinx](#) is the tool used to build documentation.

1.2 Docstrings to Documentation

The sphinx extension [apidoc](#) is used to automatically document code using signatures and docstrings.

Numpy or Google style docstrings will be picked up from project files and available for documentation. See the [Napoleon](#) extension for details.

For an in-use example, see the [page source](#) for *Users*.

To compile all docstrings automatically into documentation source files, use the command:

```
make apidocs
```

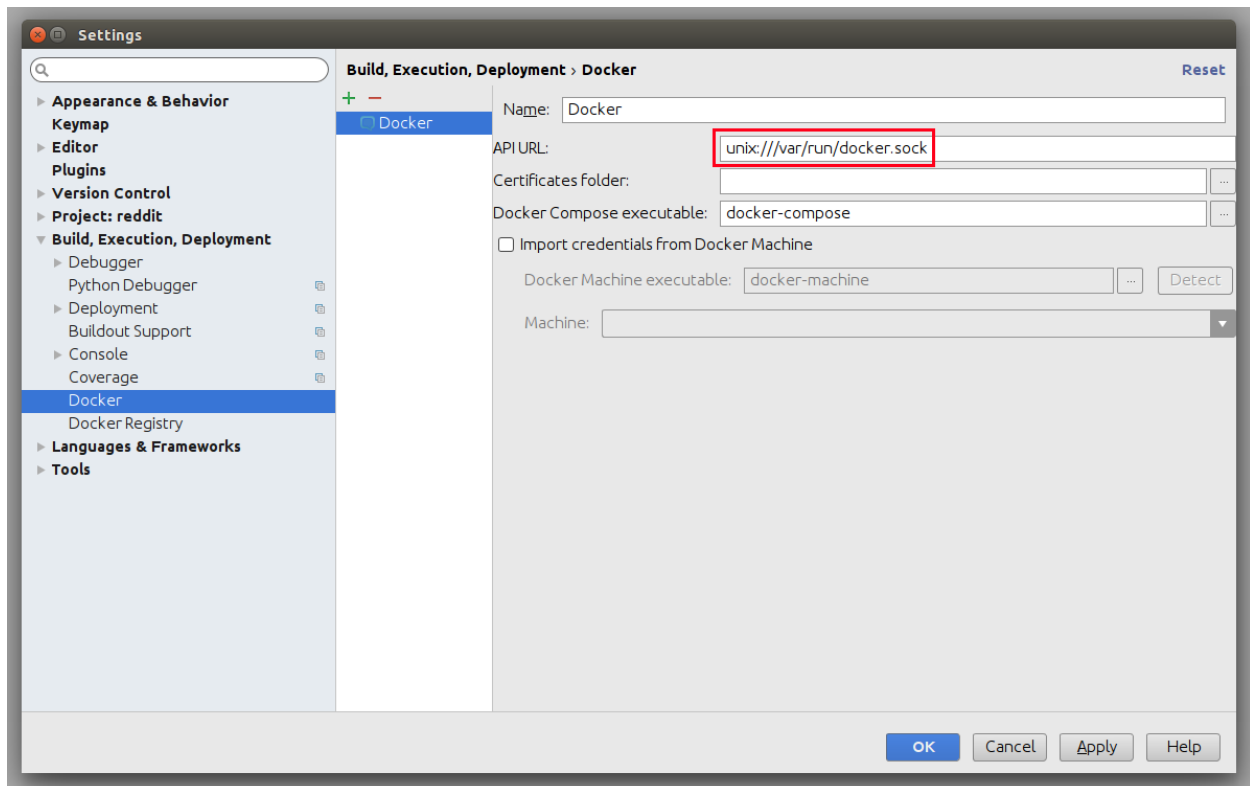
This can be done in the docker container:

```
docker run --rm docs make apidocs
```


DOCKER REMOTE DEBUGGING

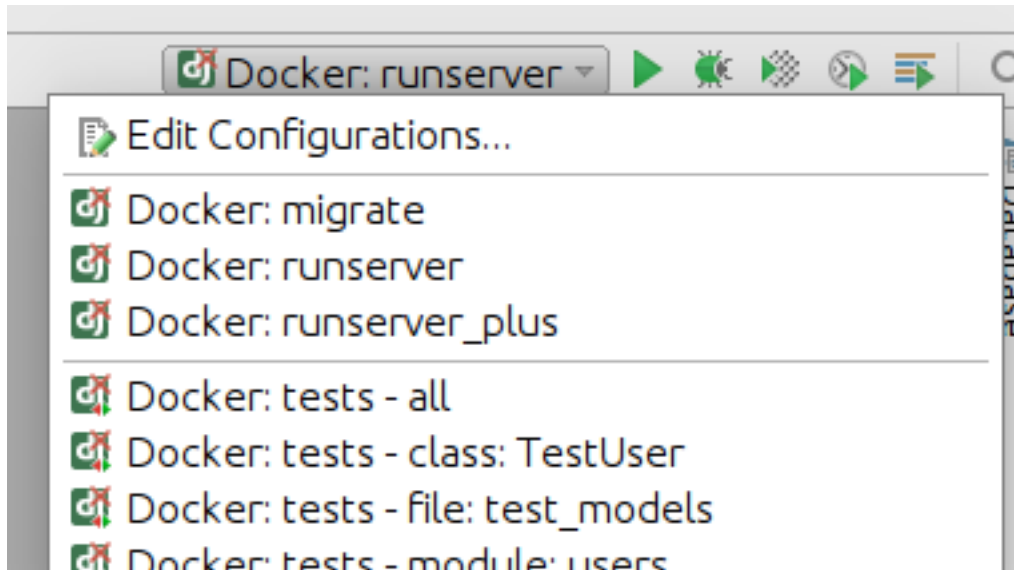
To connect to python remote interpreter inside docker, you have to make sure first, that Pycharm is aware of your docker.

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*.



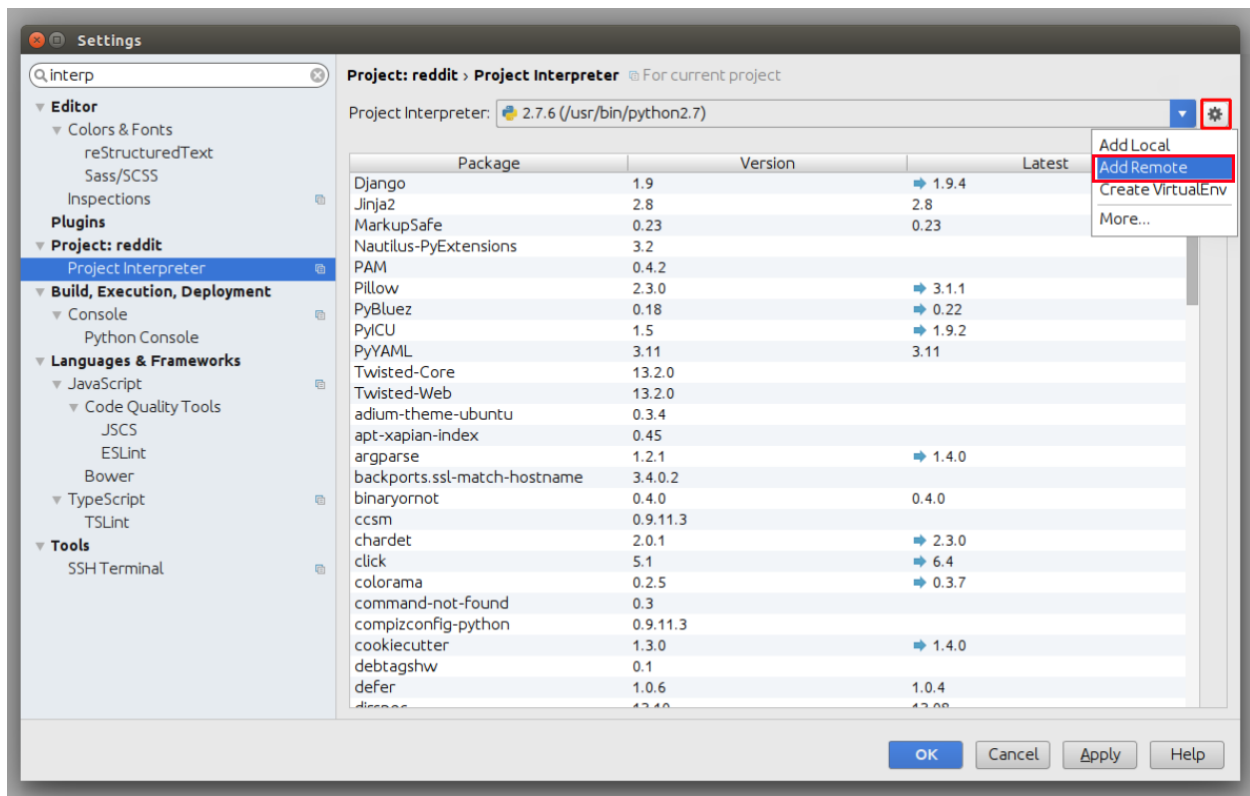
2.1 Configure Remote Python Interpreter

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

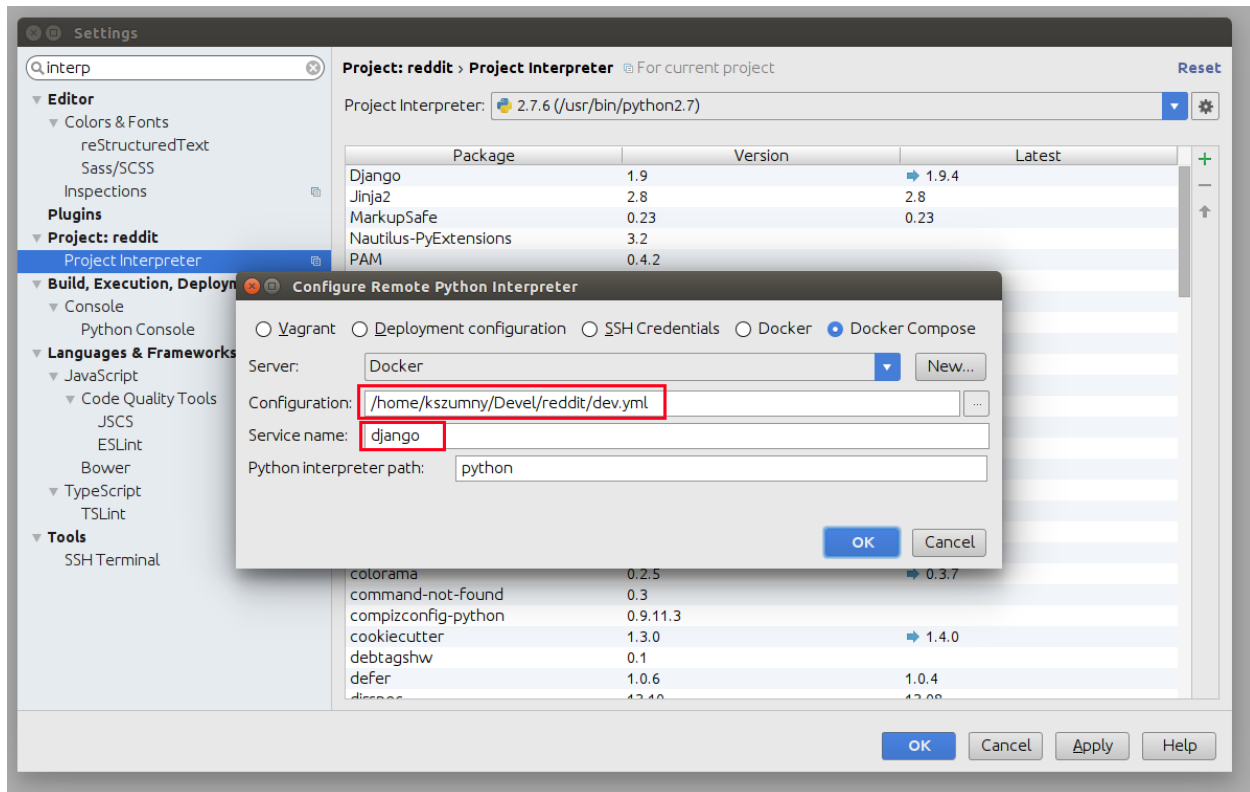


But as you can see, at the beginning there is something wrong with them. They have red X on django icon, and they cannot be used, without configuring remote python interpreter. To do that, you have to go to *Settings > Build, Execution, Deployment* first.

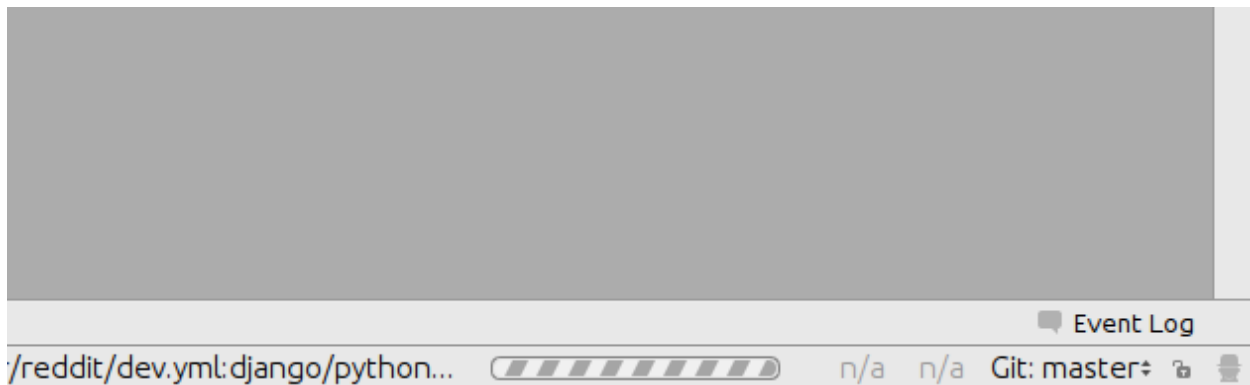
Next, you have to add new remote python interpreter, based on already tested deployment settings. Go to *Settings > Project > Project Interpreter*. Click on the cog icon, and click *Add Remote*.



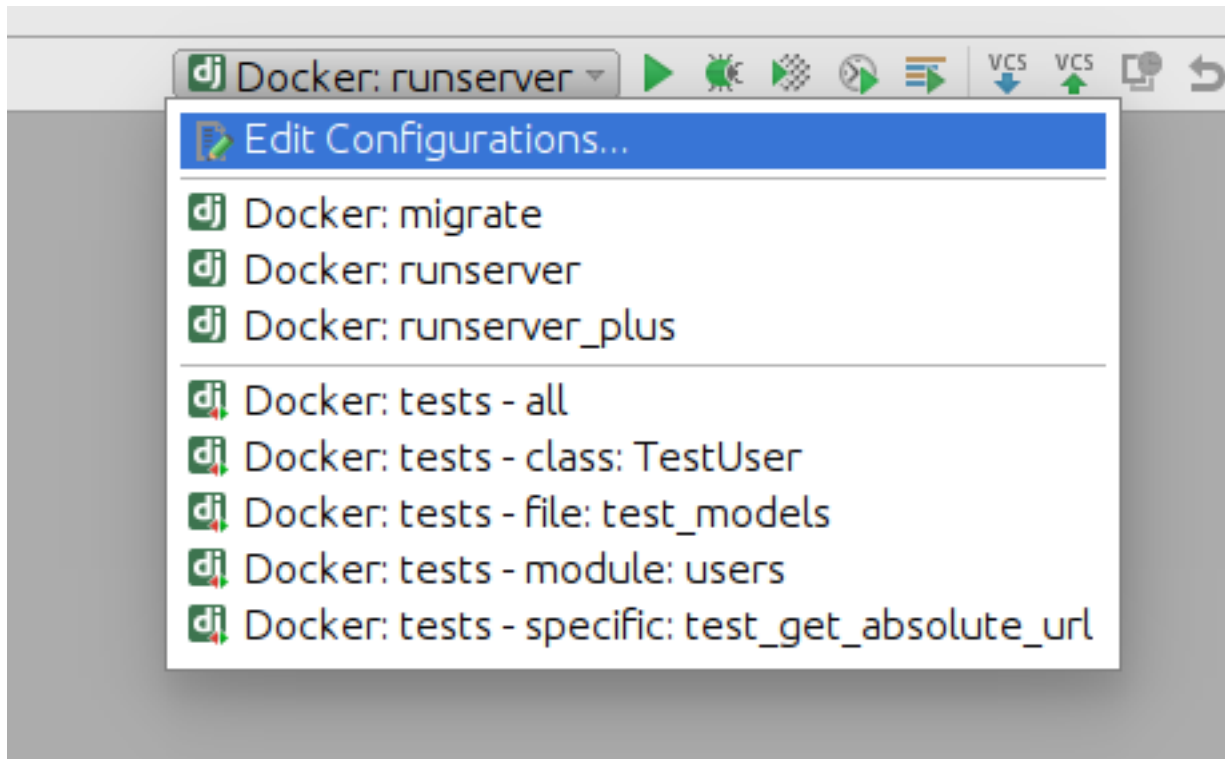
Switch to *Docker Compose* and select *local.yml* file from directory of your project, next set *Service name* to *django*



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

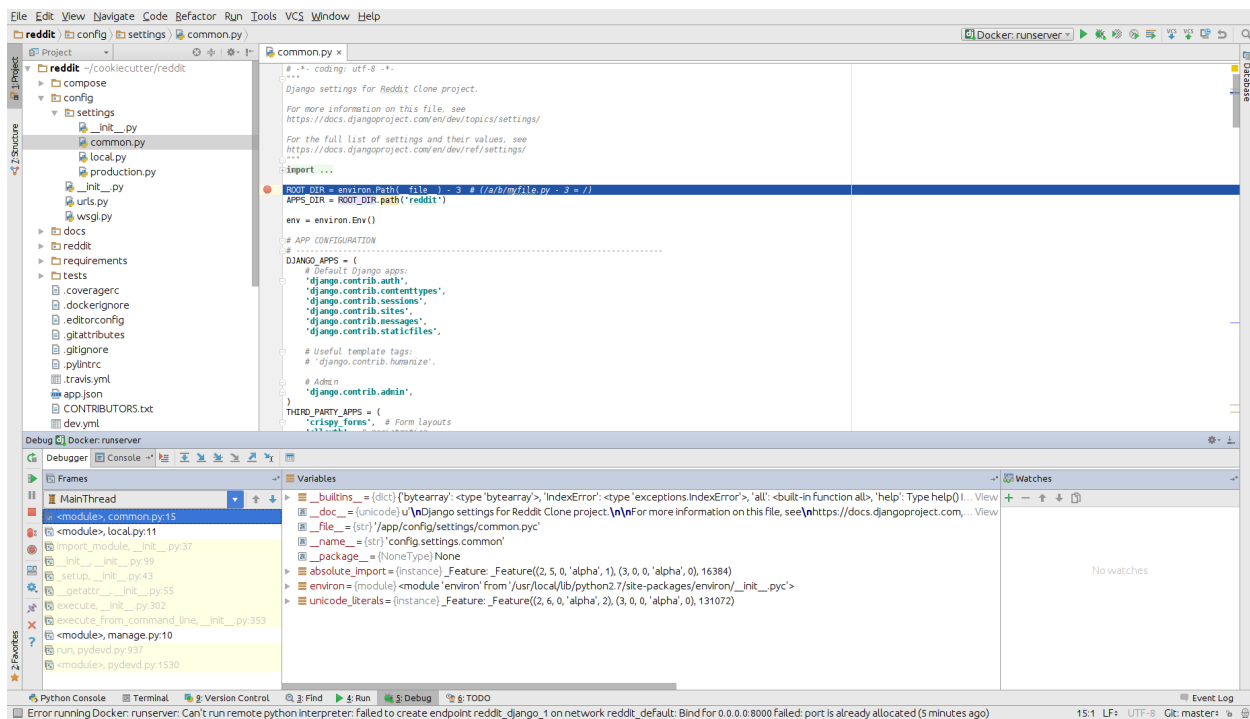


After few seconds, all *Run/Debug Configurations* should be ready to use.



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

Project: reddit / cookiecutter/reddit

models.py | urls.py | views.py | 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

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

Python Console | Terminal | Version Control | Find | Run | Debug | TODO

Tests Passed: 7 passed (a minute ago)

9.1 LF: UTF-8: GC: master: b

File Edit View Navigate Code Refactor Run Tools VCS Window Help

Project: reddit / cookiecutter/reddit

models.py | urls.py | views.py | test_views.py | common.py

```

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

```

Debug Docker: tests - all

Debugger | Console | Variables | Watches

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_tunittest.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

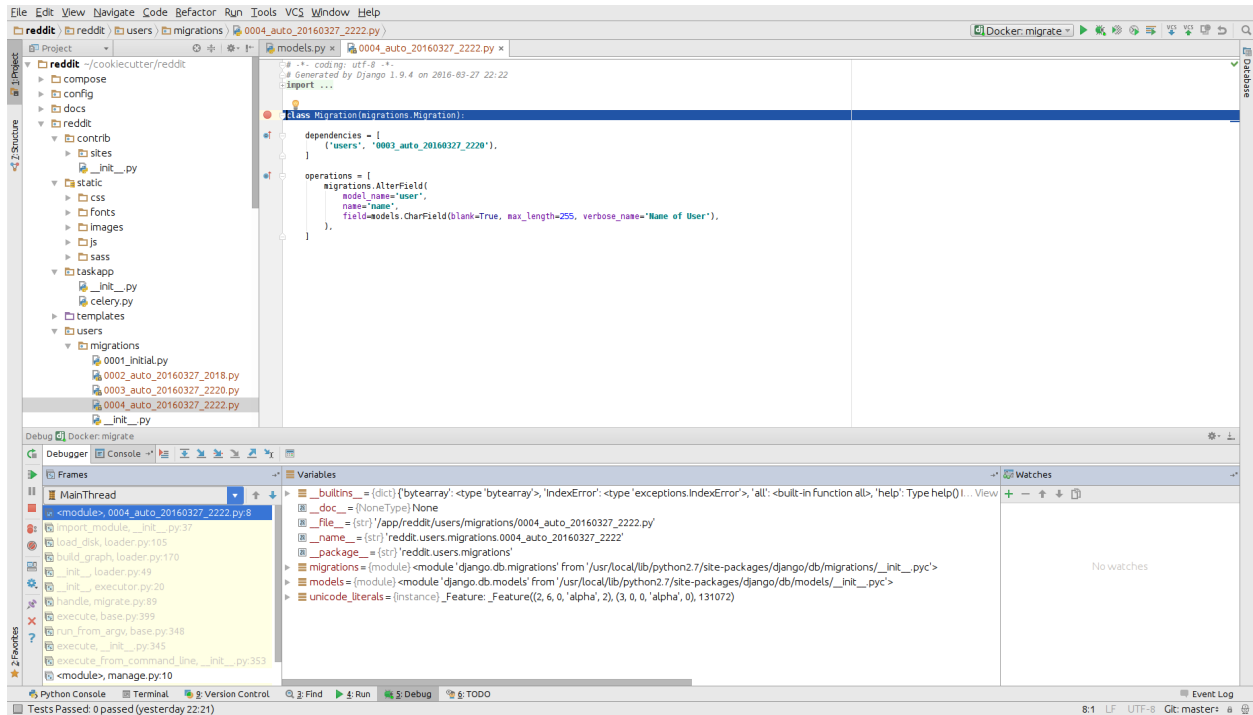
handle, django_test_manage.py:93

Python Console | Terminal | Version Control | Find | Run | Debug | TODO

Tests Passed: 7 passed (a minute ago)

22.1 LF: UTF-8: GC: master: b

- 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 my_awesome_project.iml
```


USERS

Starting a new project, it's highly recommended to set up a custom user model, even if the default User model is sufficient for you.

This model behaves identically to the default user model, but you'll be able to customize it in the future if the need arises.

```
class my_awesome_project.users.models.User (*args, **kwargs)
    Default user for My Awesome Project.

    exception DoesNotExist

    exception MultipleObjectsReturned

    get_absolute_url ()
        Get url for user's detail view.

        Returns URL for user detail.

        Return type str

    name
        First and last name do not cover name patterns around the globe
```


INDICES AND TABLES

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