Beverage Tasting

Jan Balaz

Jul 25, 2020

CONTENTS:

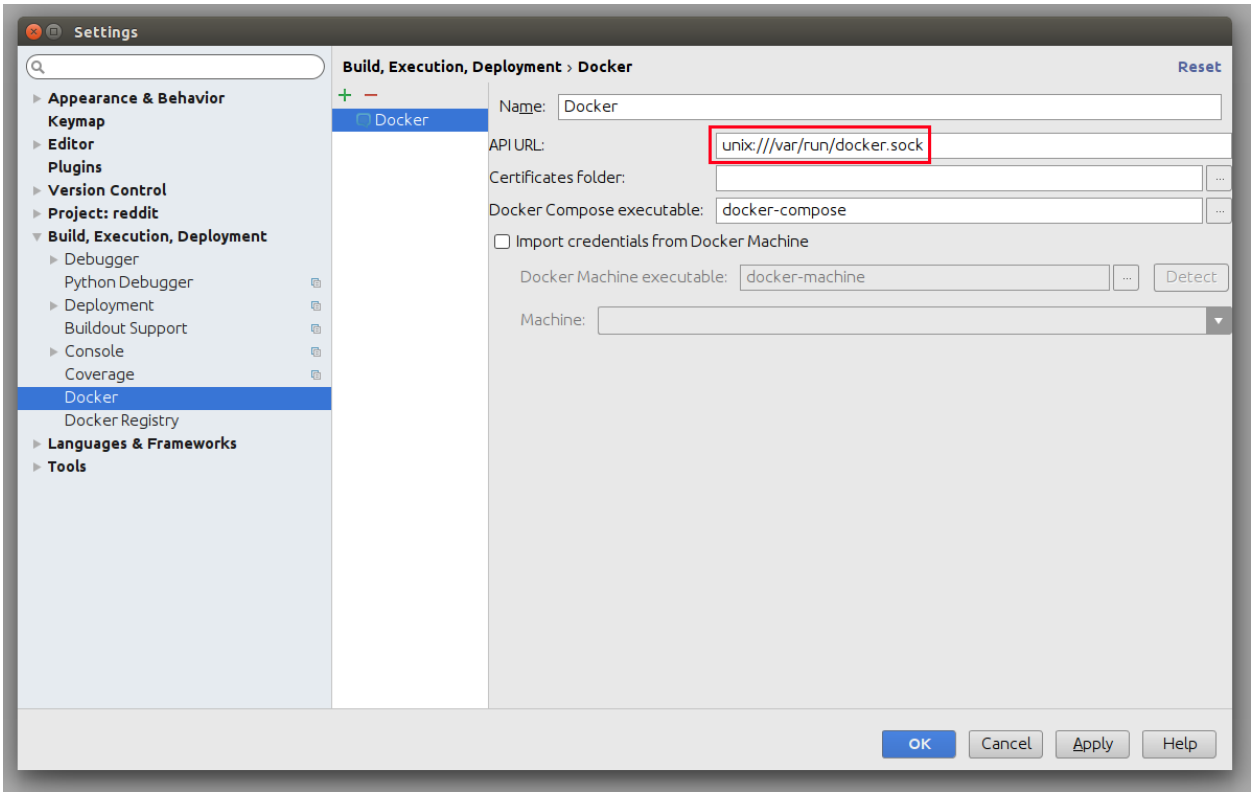
CHAPTER

ONE

DOCKER REMOTE DEBUGGING

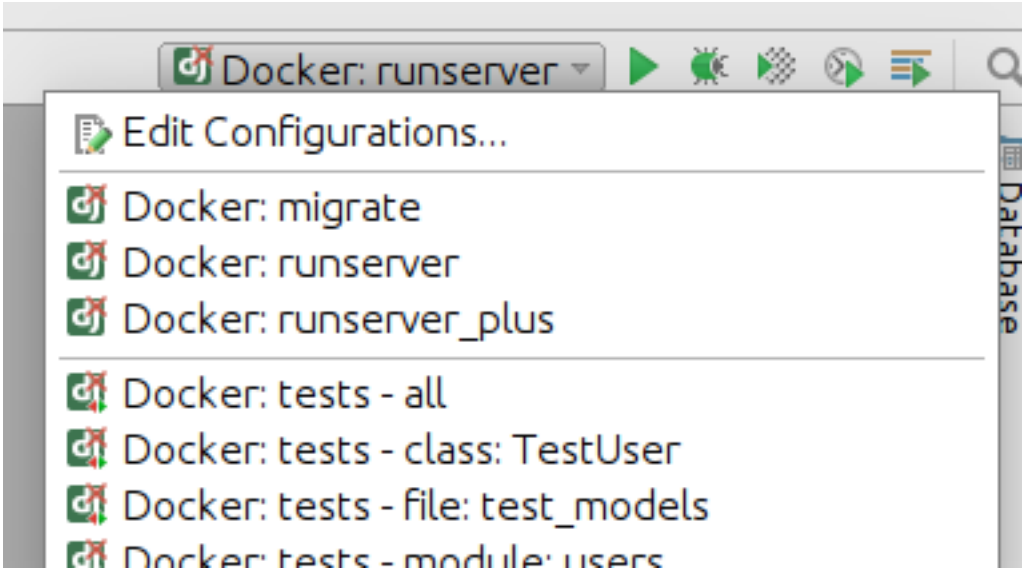
To connect to python remote interpreter inside docker, you have to make sure first, that Pycharm is aware of your docker.

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*.



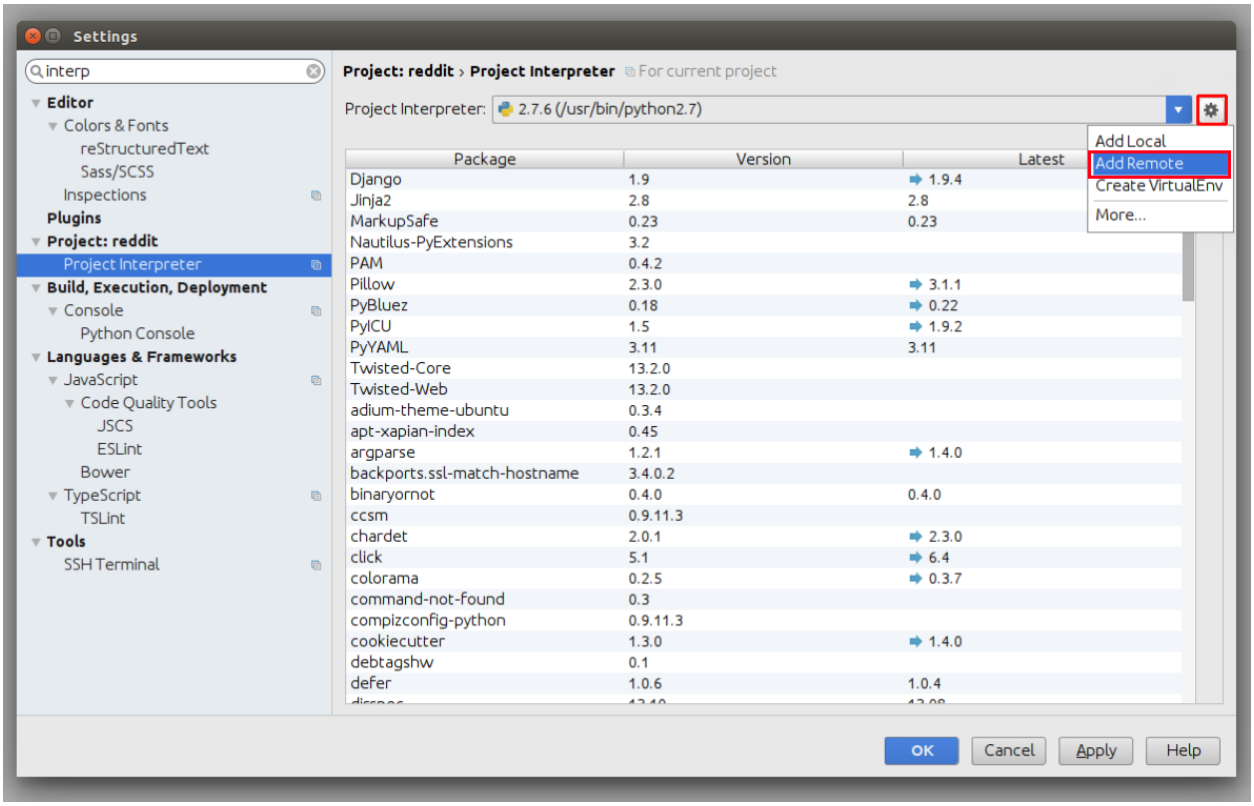
1.1 Configure Remote Python Interpreter

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

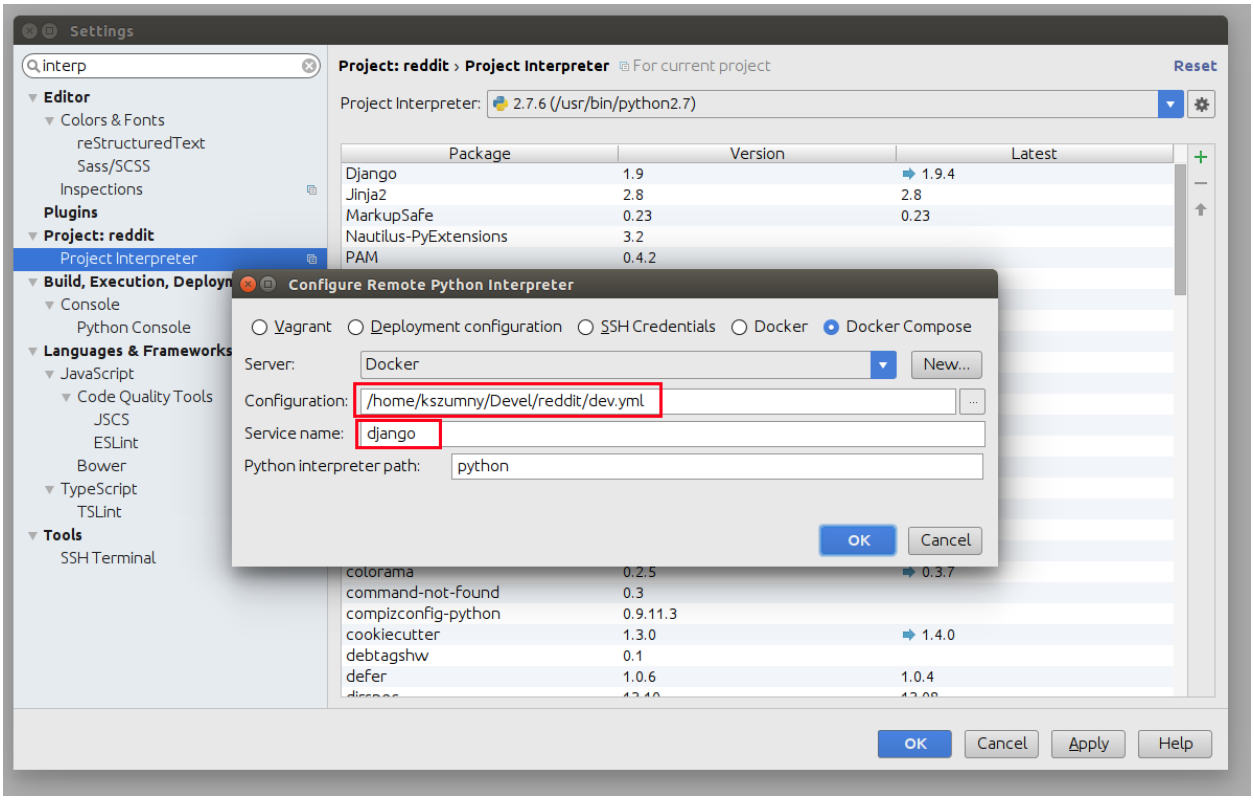


But as you can see, at the beginning there is something wrong with them. They have red X on django icon, and they cannot be used, without configuring remote python interpreter. To do that, you have to go to *Settings* > *Build, Execution, Deployment* first.

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*.

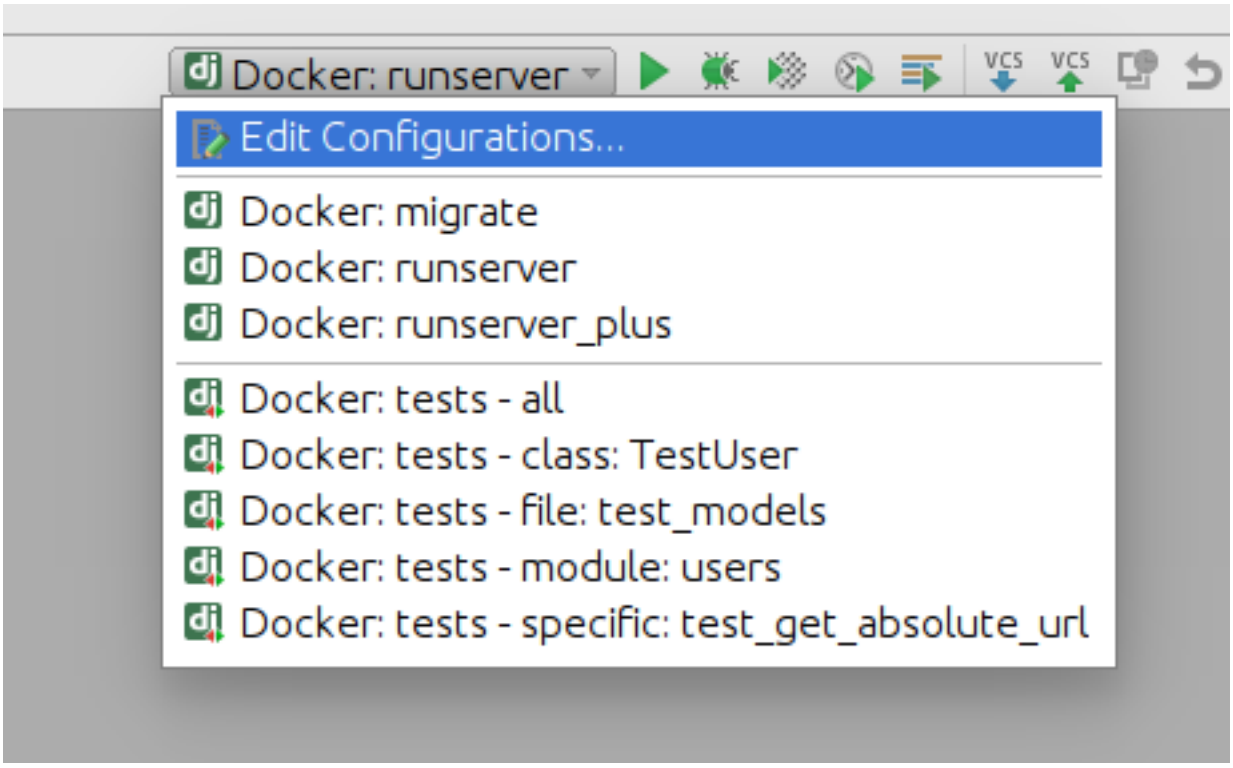


Switch to *Docker Compose* and select *local.yml* file from directory of your project, next set *Service name* to *django*



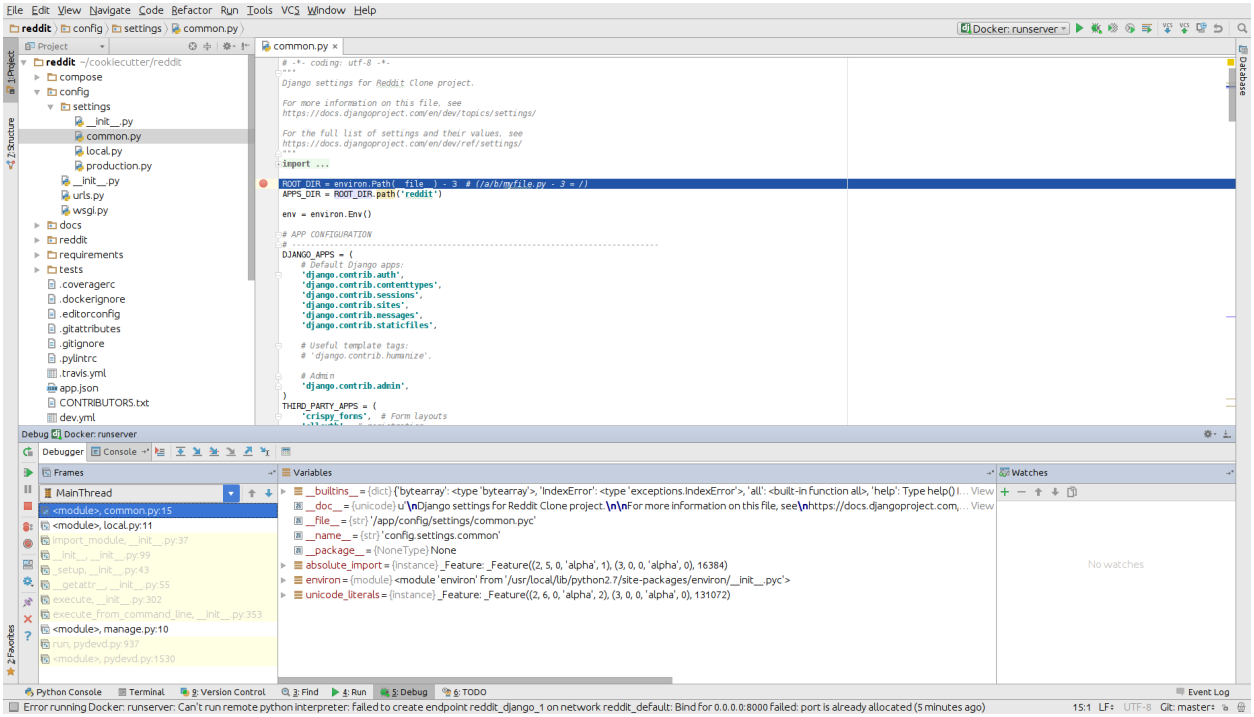
Having that, click *OK*. Close *Settings* panel, and wait few seconds...

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

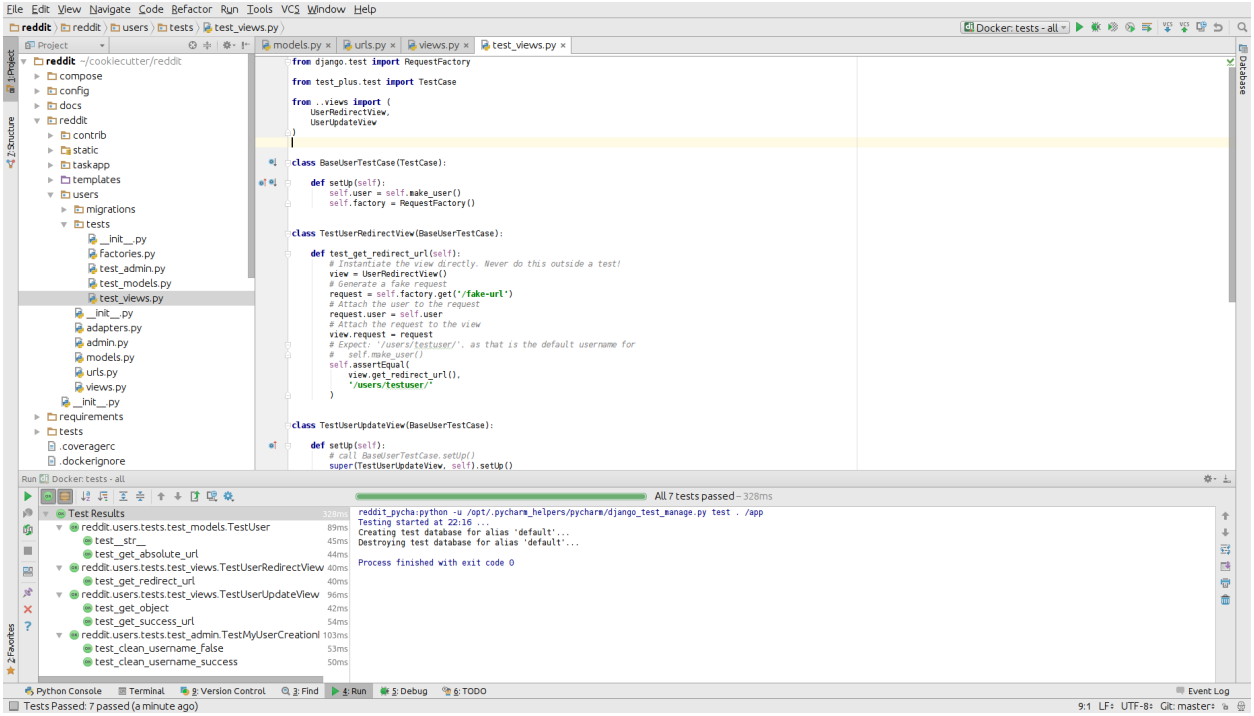


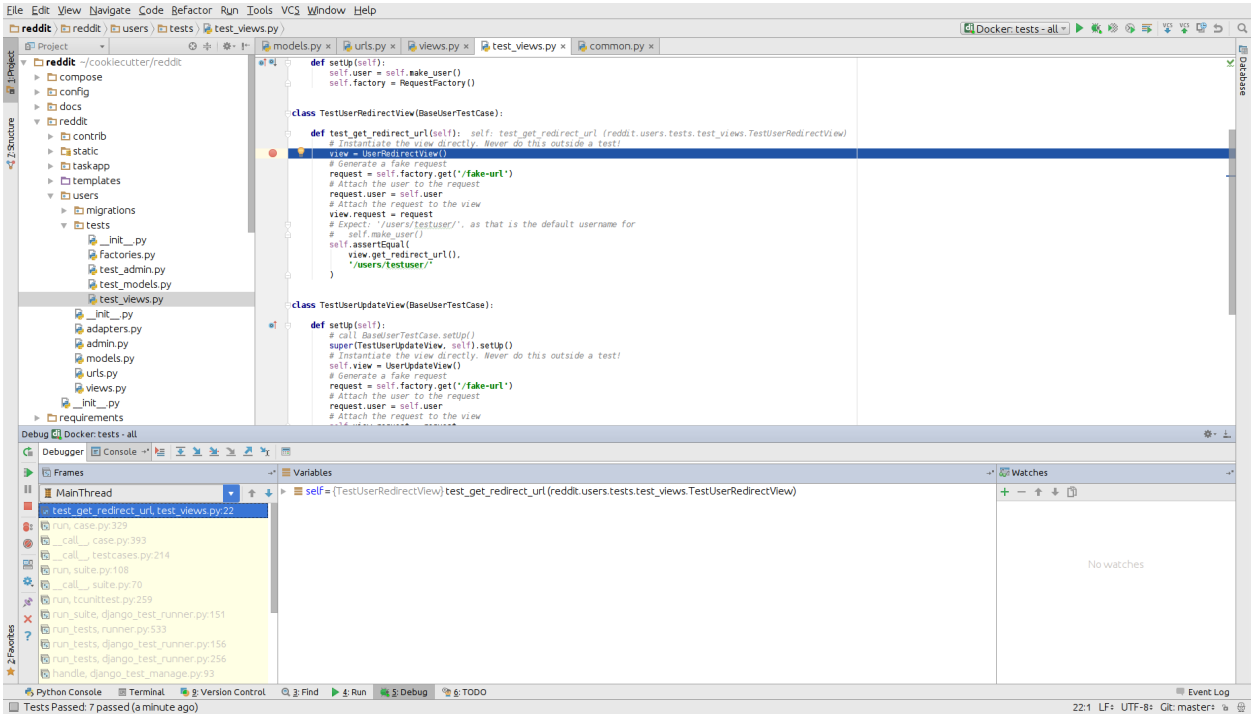
Things you can do with provided configuration:

- run and debug python code

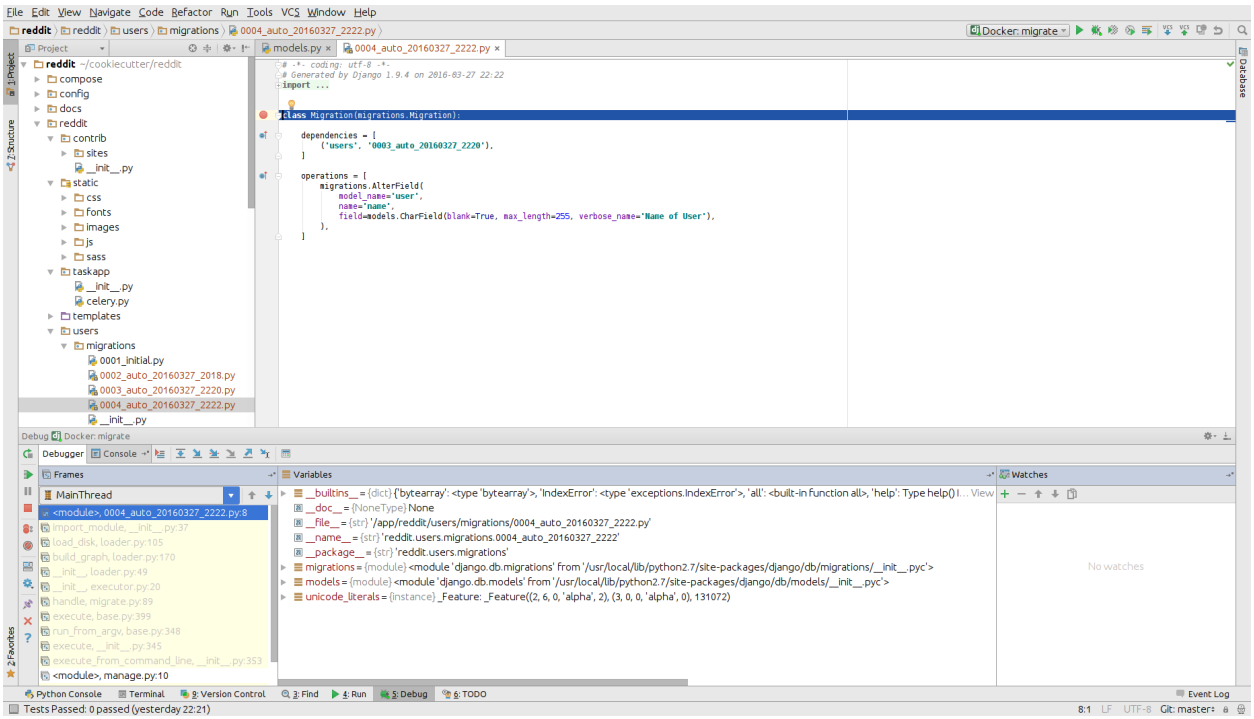


- run and debug tests





- run and debug migrations or different django management commands



- and many others..

1.2 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 beverage_tasting.iml
```

CHAPTER

TWO

RUNNING LOCALLY WITH DOCKER

This project relies heavily on Docker and Docker Compose.

2.1 Prerequisites

- Docker
- Docker Compose

2.2 Build the App

This command should always be run when something changes either in Dockerfile or in app requirements:

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

2.3 Run the App

At first build the frontend static files:

```
docker-compose -f web.yml up --build
```

Command will build the frontend container and use it to generate static files to `/beverage_tasting/static`. Django later collects and serves static files from this directory.

Next step is to actually run Django and PostgreSQL:

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

It might be handy to always start services with build options. It is fast when cached and only takes longer when changes occurred. Following command is therefore recommended:

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

Usually only *local.yml* file will be used with Docker Compose. You can set environment variable pointing to this file:

```
export COMPOSE_FILE=local.yml
```

Then you can run only:

```
docker-compose up --build
```

2.4 Executing Management Commands

Run one-off container to perform common Django commands:

```
docker-compose run --rm django python manage.py createsuperuser
```

Command will create container with only the Django application and destroy itself afterwards.

2.5 Environment Configuration

Configuration files for local environment are located in `.envs/.local`. Username and password for PostgreSQL in `.envs/.local/.postgres` might come handy when setting up remote access, e.g. for DBeaver.

CHAPTER

THREE

PASSWORDLESS AUTHENTICATION

This app uses passwordless authentication. Advantage of this approach is no need to use any user passwords. This guide will show you the authentication flow.

3.1 Obtain a Callback Token

The first step is to obtain a callback token. Callback token is later replaced for the long-lived token. Use following command to obtain it:

```
curl -X POST -d "email=info@tastebeer.org" https://tastebeer.org/auth/email/
```

Command above will trigger email sending to the address provided.

Note: Local development server does not send real e-mails. Instead e-mail HTML is printed to standard output. Search for login code there.

Warning: Authentication library currently uses only 6-digit callback tokens.

3.2 Obtain an Authorization Token

Callback token is short lived (15 minutes) and should be exchanged for authorization token:

```
curl -X POST -d "email=info@tastebeer.org&token=531680" https://tastebeer.org/auth/
↪token/

# returns token
{"token":"89ae6b76a9ec140a16ff369ef2f16e77f9b2919b"}
```

Note: This is a moment when user registration took place. User e-mail is connected with given token in database.

3.3 Using the Authorization Token

Most of API calls are private, sometimes limited only to the owner user. Provide obtained authorization token to get access to such resource:

```
curl -i -H "Accept: application/json" \
-H "Content-Type: application/json" \
-H "Authorization: Token 89ae6b76a9ec140a16ff369ef2f16e77f9b2919b" \
https://tastebeer.org/api/users/me/
```

CHAPTER

FOUR

API

API of this application was created with Django REST framework. It is browsable locally at localhost:8000/api/.

Alternatively use cURL if you have authorization token:

```
curl -i -H "Accept: application/json" \
-H "Content-Type: application/json" \
-H "Authorization: Token 89ae6b76a9ec140a16ff369ef2f16e77f9b2919b" \
localhost:8000/api/
```

API endpoints won't be listed here as they are easily accessible on their own.

4.1 List All URLs

Listing all API URLs might be helpful and it can be done easily:

```
docker-compose -f local.yml run --rm django python manage.py show_urls
```

CHAPTER

FIVE

TESTING

This project aims to high coverage and quality of testing.

Run the following command to test the application:

```
docker-compose -f local.yml run --rm django pytest
```

5.1 Coverage

Run tests with code coverage first:

```
docker-compose -f local.yml run --rm django coverage run -m pytest
```

Once finished either run `report` to see coverage immediately or generate browsable `html` files:

```
docker-compose -f local.yml run --rm django coverage report
docker-compose -f local.yml run --rm django coverage html
```

Generated HTML report can be found in **coverage_html_report**.

CHAPTER

SIX

CODE STYLE

No specific code style is enforced, but the code must adhere to the rules set by following tools:

- mypy
- Black
- Flake8

6.1 Run the Lint Check

Recommended order of running tools mentioned above is as follows:

```
docker-compose -f local.yml run --rm django mypy
docker-compose -f local.yml run --rm django black .
docker-compose -f local.yml run --rm django flake8
```

This order makes the most sense because `mypy` fail requires code adjustments. These are then reformatted with `Black` if needed and `Flake8` confirms change validity according to PEP8 recommendations.

CHAPTER

SEVEN

CONTRIBUTIONS

Your help with development, testing or documentation is very welcomed. Please, follow this short guide to make the most of it.

7.1 Development Contributions

For now it is just a matter of a pull request in our repository.

7.2 Documentation Contributions

Any contributions should be either in a form of an issue or a pull request in our repository.

Use following command to generate the documentation locally:

```
docker-compose -f local.yml run --rm django sphinx-build docs/ docs/_build/html/
```

7.3 Bug Reporting

Please use issues in our repository to report bugs you found.

Warning: Report any security related bugs to contact@tastebeer.org directly.

CHAPTER

EIGHT

INDICES AND TABLES

- genindex
- modindex
- search