

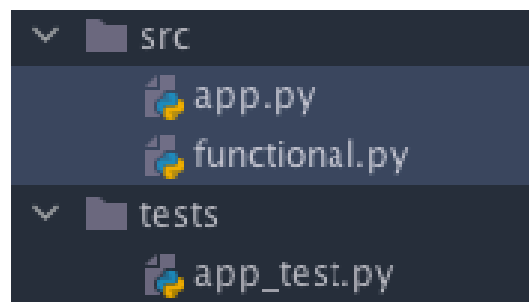
Введение

В этой работе можно будет познакомиться с созданием ci пайплайнов, а также с автоматическим запуском тестов, которые мы написали ранее, применением линтеров и тайпчекеров.

Если ваши результаты при некоторых действиях будут отличаться от скриншотов, это нормально, т.к. используемые нами веб фреймворки достаточно сильно отличаются.

Реструктуризация проекта

Для более удобной работы изменим структуру проекта:



файлы:

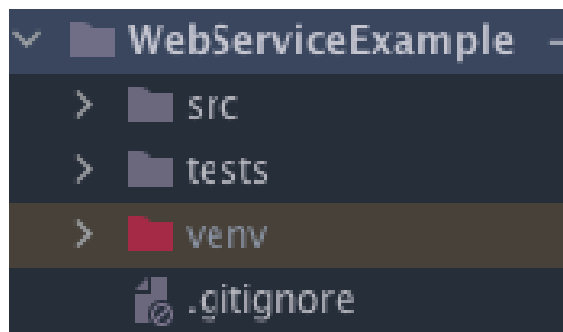
1. *src/app.py* - web сервис
2. *src/functional.py* - файл с функцией, решающей поставленную задачу (калькулятор, форматирование текста или числа Фибоначчи)
3. *tests/app_test.py* - написанные ранее тесты

Тем самым мы отделили файлы нашего проекта от тестов. На самом деле такое разделение крайне условно и очень сильно зависит от практик языка, на котором мы пишем, договоренностей в команде и т.д.

Важный момент

Обратите внимание на то, что после тестов, у вас скорее всего появилась папка с названием *.pytest_cache*. Как следует из названия - это папка, созданная pytest, с кэшированными данными для повышения скорости тестирования. В нашей репозитории на gitlab это явно лишнее (большое количество файлов, изменения в которых нас совсем не

интересуют). Это же относится и к виртуальному окружению python (обычно venv, [почитать](#)). Добавим .gitignore файл. Таким образом наш проект принимает вид:



Для тех, кто еще не сталкивался с данным файлом ([ссылка](#), [ссылка](#))

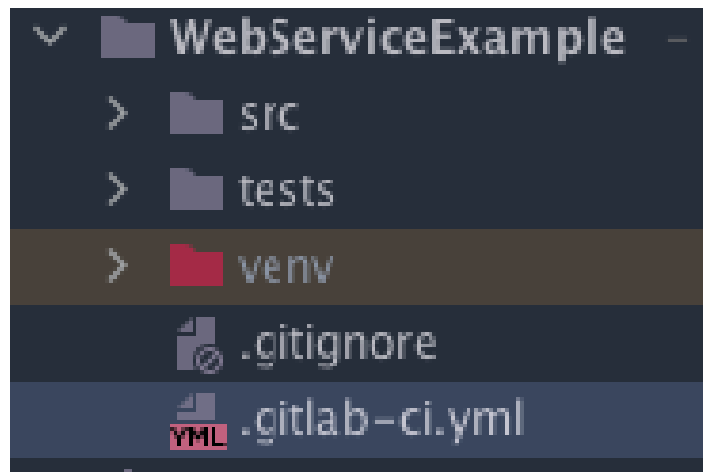
Чем заполнить данный файл?

Тут на помощь приходит гугл и сообщество. В [данном репозитории](#) есть примеры для огромного количества популярных и не очень языков программирования. В случае python - [.gitignore](#), копируем его содержимое и вставляем в ранее созданный файл. Помимо особенностей языка, различные файлы может генерировать операционная система, например, MacOS создает скрытый .DS_Store файл, который нам тоже не нужен в нашем репозитории. Добавим в инор файл в зависимости от ОС: [linux](#), [windows](#), [macos](#). Помимо этого можно добавить к игнорированию папки, создаваемые IDE (.idea для продукции JetBrains, .vscode - Visual Studio Code и т.д)

Делаем коммит и выполняем пуш в свой репозиторий!

Создаем файл с пайплайном

Создаем в корне нашего проекта файл *gitlab-ci.yml*



Содержимое файла

```
# Образ контейнера, в котором будет выполняться ci пайплайн
# Можно использовать другие версии python
image: python:3.8

# Данная настройка позволяет сохранить кеш установки пакетов между разными запусками пайплайна (экономим время на установку)
#variables:
#  PIP_CACHE_DIR: "$CI_PROJECT_DIR/.cache/pip"

#cache:
#  paths:
#    - .cache/pip
#    - venv/

# Действия, которые необходимо выполнить до запуска основных действий пайплайна
# В данном случае, по стандартной практике питона, создаем виртуальное окружение и активируем его
before_script:
  - python --version # For debugging
  - pip install virtualenv
  - virtualenv venv
  - source venv/bin/activate

test:
  script:
    - python -m unittest discover -s "./tests" -p "*_test.py"
```

Используемая команда для запуска тестов применима для tornado. В случае использования других фреймворков используйте команду, которую вы выполняли во 2 л.р.

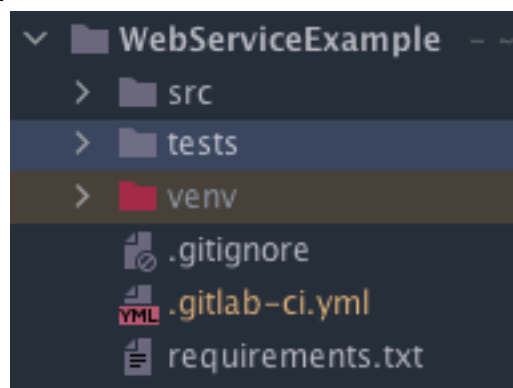
Однако, если мы сейчас запустим пайплайн, все практически мгновенно упадет, т.к. мы не установили зависимости (FastAPI, Tornado и т.д.). Самым простым способом будет захардкодить эти зависимости, как это сделано с модулем `virtualenv` в `ci` файле, однако есть более удобный способ - `requirements` файл.

Выполним в терминале команду:

```
pip freeze > requirements.txt
```

После чего в корне нашего проекта создается одноименный файл, где прописаны наши зависимости с их версиями.

Текущая структура проекта:



Используем сгенерированный файл в CI:

```

# Образ контейнера, в котором будет выполняться ci пайплайн
# Можно использовать другие версии python
image: python:3.8

# Данная настройка позволяет сохранить кеш установки пакетов между разными запусками пайплайна (экономим время на установку)
#variables:
#  PIP_CACHE_DIR: "${CI_PROJECT_DIR}/.cache/pip"

#cache:
#  paths:
#    - .cache/pip
#    - venv/

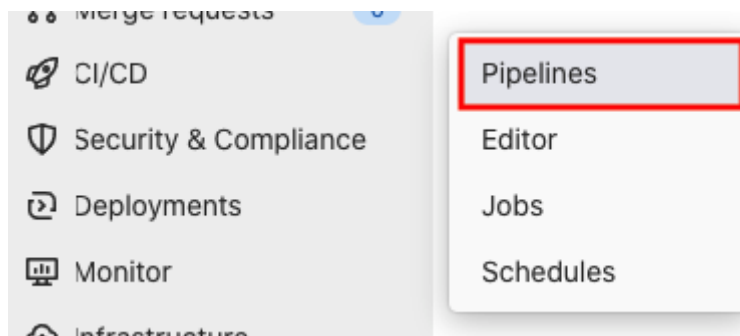
# Действия, которые необходимо выполнить до запуска основных действий пайплайна
# В данном случае, по стандартной практике питона, создаем виртуальное окружение и активируем его
before_script:
  - python --version # For debugging
  - pip install virtualenv
  - virtualenv venv
  - source venv/bin/activate
  - pip install -r requirements.txt

test:
  script:
    - python -m unittest discover -s "./tests" -p "*_test.py"

```

Делаем коммит и выполняем пуш в свой репозиторий!

Далее переходим в раздел Pipelines в CI/CD



Если все было сделано верно, вы увидим результат об успешном завершении пайплайна.

All1

Finished

Branches

Tags

Clear runner caches

CI lint

Run pipeline

Filter pipelines

Q

Show Pipeline ID

Status	Pipeline ID	Triggerer	Commit	Stages	Duration	
<div>passed</div>	<div>#13</div> <div>latest</div>	<div></div>	<div>master → 30f50014</div> <div>rename .gitlab-ci.yml</div>	<div></div>	<div>00:01:10</div> <div>41 minutes ago</div>	<div></div>

Код-чекеры

Кроме использования написанных тестов, часто применяются код-чекеры они же [линтеры](#). Одним из самых популярных в python является flake8. Мы воспользуемся его базовым функционалом (на самом деле есть много submodule, которые позволяют жестко контролировать стиль написания кода).

Установим его с помощью команды:

```
pip install flake8
```

Далее добавим его в тестирование:

```
# Действия, которые необходимо выполнить до запуска основных действий пайплайна
# В данном случае, по стандартной практике питона, создаем виртуальное окружение и активируем его
before_script:
  - python --version # For debugging
  - pip install virtualenv
  - virtualenv venv
  - source venv/bin/activate
  - pip install -r requirements.txt






test:
  script:
    - python -m unittest discover -s "./tests" -p "*_test.py"
    - flake8 .
```

ВАЖНО!





Наш **список зависимостей изменился**, не забудьте опять **выполнить команду `pip freeze`**. Чтобы упростить эту процедуру можно воспользоваться [poetry](#), который делает это автоматически.

Делаем коммит и выполняем пуш в свой репозиторий!

В этот раз пайплайн завершился с ошибкой.

Status	Pipeline ID	Triggerer	Commit	Stages	Duration	
 failed	#14 latest		master -> a15cd5bb Add flake8 to CI		00:01:07 5 minutes ago	 

Разберемся с возникшей проблемой. Переходим в упавший пайплайн.

Status	Pipeline ID	Triggerer	Commit
 failed	#14 latest		master Add
 passed	#13		master ren



failed

Pipeline #14 triggered 9 minutes ago by



Nikolay Vedernikov

Add flake8 to CI

🕒 1 job for `master` in 1 minute and 7 seconds (queued for 2 seconds)

🚩 `latest`

🔑 `a15cd5bb`

🔗 No related merge requests found.

Pipeline

Needs

Jobs 1

Failed Jobs 1

Tests 0

Test

❌ test



Pipeline

Needs

Jobs 1

Failed Jobs 1

Tests 0

Test

❌ test

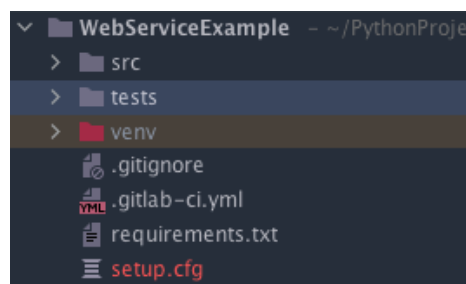


test - failed - (script failure)

Далее нам открываются логи нашего пайплайна. Видим большое количество ошибок от flake8 в коде, который находится в библиотеках, которые мы использовали.

```
Showing last 499.91 KiB of log - Complete Raw
4667 ./env/lib/python3.8/site-packages/pip/_vendor/chardet/jisfreq.py:149:45: E231 missing whitespace after ','
4668 ./env/lib/python3.8/site-packages/pip/_vendor/chardet/jisfreq.py:149:50: E231 missing whitespace after ','
4669 ./env/lib/python3.8/site-packages/pip/_vendor/chardet/jisfreq.py:149:75: E231 missing whitespace after ','
4670 ./env/lib/python3.8/site-packages/pip/_vendor/chardet/jisfreq.py:149:80: E501 line too long (87 > 79 characters)
4671 ./env/lib/python3.8/site-packages/pip/_vendor/chardet/jisfreq.py:149:81: E261 at least two spaces before inline comment
4672 ./env/lib/python3.8/site-packages/pip/_vendor/chardet/jisfreq.py:150:2: E131 continuation line unaligned for hanging indent
4673 ./env/lib/python3.8/site-packages/pip/_vendor/chardet/jisfreq.py:150:5: E231 missing whitespace after ','
4674 ./env/lib/python3.8/site-packages/pip/_vendor/chardet/jisfreq.py:150:10: E231 missing whitespace after ','
4675 ./env/lib/python3.8/site-packages/pip/_vendor/chardet/jisfreq.py:150:15: E231 missing whitespace after ','
4676 ./env/lib/python3.8/site-packages/pip/_vendor/chardet/jisfreq.py:150:20: E231 missing whitespace after ','
4677 ./env/lib/python3.8/site-packages/pip/_vendor/chardet/jisfreq.py:150:25: E231 missing whitespace after ','
4678 ./env/lib/python3.8/site-packages/pip/_vendor/chardet/jisfreq.py:150:30: E231 missing whitespace after ','
4679 ./env/lib/python3.8/site-packages/pip/_vendor/chardet/jisfreq.py:150:35: E231 missing whitespace after ','
4680 ./env/lib/python3.8/site-packages/pip/_vendor/chardet/jisfreq.py:150:40: E231 missing whitespace after ','
4681 ./env/lib/python3.8/site-packages/pip/_vendor/chardet/jisfreq.py:150:50: E231 missing whitespace after ','
4682 ./env/lib/python3.8/site-packages/pip/_vendor/chardet/jisfreq.py:150:55: E231 missing whitespace after ','
4683 ./env/lib/python3.8/site-packages/pip/_vendor/chardet/jisfreq.py:150:60: E231 missing whitespace after ','
4684 ./env/lib/python3.8/site-packages/pip/_vendor/chardet/jisfreq.py:150:75: E231 missing whitespace after ','
4685 ./env/lib/python3.8/site-packages/pip/_vendor/chardet/jisfreq.py:150:80: E501 line too long (87 > 79 characters)
4686 ./env/lib/python3.8/site-packages/pip/_vendor/chardet/jisfreq.py:150:81: E261 at least two spaces before inline comment
4687 ./env/lib/python3.8/site-packages/pip/_vendor/chardet/jisfreq.py:151:1: E122 continuation line missing indentation or outdented
4688 ./env/lib/python3.8/site-packages/pip/_vendor/chardet/jisfreq.py:151:5: E231 missing whitespace after ','
4689 ./env/lib/python3.8/site-packages/pip/_vendor/chardet/jisfreq.py:151:10: E231 missing whitespace after ','
4690 ./env/lib/python3.8/site-packages/pip/_vendor/chardet/jisfreq.py:151:20: E231 missing whitespace after ','
4691 ./env/lib/python3.8/site-packages/pip/_vendor/chardet/jisfreq.py:151:25: E231 missing whitespace after ','
4692 ./env/lib/python3.8/site-packages/pip/_vendor/chardet/jisfreq.py:151:30: E231 missing whitespace after ','
4693 ./env/lib/python3.8/site-packages/pip/_vendor/chardet/jisfreq.py:151:35: E231 missing whitespace after ','
4694 ./env/lib/python3.8/site-packages/pip/_vendor/chardet/jisfreq.py:151:65: E231 missing whitespace after ','
4695 ./env/lib/python3.8/site-packages/pip/_vendor/chardet/jisfreq.py:151:75: E231 missing whitespace after ','
4696 ./env/lib/python3.8/site-packages/pip/_vendor/chardet/jisfreq.py:151:80: E501 line too long (87 > 79 characters)
4697 ./env/lib/python3.8/site-packages/pip/_vendor/chardet/jisfreq.py:151:81: E261 at least two spaces before inline comment
4698 ./env/lib/python3.8/site-packages/pip/_vendor/chardet/jisfreq.py:152:2: E131 continuation line unaligned for hanging indent
4699 ./env/lib/python3.8/site-packages/pip/_vendor/chardet/jisfreq.py:152:10: E231 missing whitespace after ','
4700 ./env/lib/python3.8/site-packages/pip/_vendor/chardet/jisfreq.py:152:15: E231 missing whitespace after ','
4701 ./env/lib/python3.8/site-packages/pip/_vendor/chardet/jisfreq.py:152:20: E231 missing whitespace after ','
4702 ./env/lib/python3.8/site-packages/pip/_vendor/chardet/jisfreq.py:152:30: E231 missing whitespace after ','
4703 ./env/lib/python3.8/site-packages/pip/_vendor/chardet/jisfreq.py:152:35: E231 missing whitespace after ','
4704 ./env/lib/python3.8/site-packages/pip/_vendor/chardet/jisfreq.py:152:40: E231 missing whitespace after ','
4705 ./env/lib/python3.8/site-packages/pip/_vendor/chardet/jisfreq.py:152:50: E231 missing whitespace after ','
4706 ./env/lib/python3.8/site-packages/pip/_vendor/chardet/jisfreq.py:152:70: E231 missing whitespace after ','
4707 ./env/lib/python3.8/site-packages/pip/_vendor/chardet/jisfreq.py:152:75: E231 missing whitespace after ','
4708 ./env/lib/python3.8/site-packages/pip/_vendor/chardet/jisfreq.py:152:80: E501 line too long (87 > 79 characters)
4709 ./env/lib/python3.8/site-packages/pip/_vendor/c
4710 Job's log exceeded limit of 4194304 bytes.
4711 Job execution will continue but no more output will be collected.
```





Чтобы исключить эти файлы создадим файл *setup.cfg*.



Укажем используемую версию python и папки, которые не надо проверять.

```
1  [flake8]
2  python_version = 3.8
3  exclude = .git, __pycache__, venv
```

Делаем коммит и выполняем пуш в свой репозиторий!

Status	Pipeline ID	Triggerer	Commit	Stages	Duration
 failed	#15 latest		master -> f875b496  Add setup file		00:00:26 1 minute ago

Повторно смотрим логи и исправляем все несоответствия PEP правилам.

```
73 $ python -m unittest discover -s "./tests" -p "*_test.py"
74 WARNING:tornado.access:400 GET /?n=o (127.0.0.1) 0.34ms
75 WARNING:tornado.access:400 GET /?n=&* (127.0.0.1) 0.28ms
76 WARNING:tornado.access:400 GET /?n=1io2 (127.0.0.1) 0.27ms
77 ..
78 -----
79 Ran 2 tests in 0.015s
80 OK
81 $ flake8 .
82 ./tests/app_test.py:27:80: E501 line too long (85 > 79 characters)
84 ERROR: Job failed: exit code 1
```

Делаем коммит и выполняем пуш в свой репозиторий!

All 4	Finished	Branches	Tags	Clear runner caches	CI lint	Run pipeline
Filter pipelines						
Q Show Pipeline ID						
Status	Pipeline ID	Triggerer	Commit	Stages	Duration	
passed	#16 latest		master 4816755e Refactor line length		00:00:25 20 seconds ago	

Проверка типов данных

Для языков с динамической типизацией существует проблема, который лишит ЯП со статической типизацией, ошибки с типами данных. Например, у C++/Go/Rust и многих других статически типизированных языков, проверки типов производятся компилятором. В случае python мы лишены данной возможности “из коробки”, однако воспользуемся аннотацией типов и модулем `myru`.

Установим `myru`:

`pip install myru`

<code>@lru_cache(maxsize=100)</code>	4 4	<code>@lru_cache(maxsize=100)</code>
<code>def calculate_fib(n):</code>	>> 5 5	<code>def calculate_fib(n: int) -> int:</code>
<code>if n == 1 or n == 0:</code>	6 6	<code>if n == 1 or n == 0:</code>
<code>return 1</code>	7 7	<code>return 1</code>
<code>return calculate_fib(n - 1) + calculate_fib(n - 2)</code>	8 8	<code>return calculate_fib(n - 1) + calculate_fib(n - 2)</code>

Если функция ничего не возвращает (аналог `void` в c-like языках), необходимо указать `None`.

<code>def run_service():</code>	>> 29 29	<code>def run_service() -> None:</code>
<code>app = make_app()</code>	31 31	<code>app = make_app()</code>
<code>app.listen(PORT, HOST)</code>	32 32	<code>app.listen(PORT, HOST)</code>
<code>tornado.ioloop.IOLoop.current().start()</code>	33 33	<code>tornado.ioloop.IOLoop.current().start()</code>

Питон все также остается динамически типизированным языком (интерпретатор полностью игнорирует аннотации), однако становится проще не ошибиться в большом проекте, помимо этого улучшается работа подсказок в IDE.

Таким образом анатируем все функции и методы, которые есть у нас в коде.

Также нам необходимо добавить базовые настройки в setup файл.



```
[mypy]
python_version = 3.8
exclude = venv
disallow_untyped_defs = True
ignore_missing_imports = True
```

Добавим запуск туру в нашем пайплайне.

```
before_script:
  - python --version # For debugging
  - pip install virtualenv
  - virtualenv venv
  - source venv/bin/activate
  - pip install -r requirements.txt

test:
  script:
    - python -m unittest discover -s "./tests" -p "*_test.py"
    - flake8 .
    - mypy .
```

Делаем коммит и выполняем пуш в свой репозиторий!

Status	Pipeline ID	Triggerer	Commit	Stages	Duration
passed	#19 latest		master - 6c1993e7 Add mypy		00:00:28 21 seconds ago