

View

Вызываемый объект, который на вход получает Request, а возвращает Response.

Между входом и выходом он выполняет какую-то бизнес-логику: валидирует данные, ходит в базу и в сторонние API, выкидывает ошибки, формирует ответ и многое другое. Любая функция или метод класса могут стать View — главное, чтобы они принимали на вход Request, а отдавали Response, о котором мы еще поговорим подробнее. Но, конечно, есть общепринятые средства объявления View.

Function-based View

Объявление View с помощью функции.

Пример из документации aiohttp:

```
async def hello(request):
    return web.Response(text="Hello, world")
```

Class-based View

Объявление View с помощью класса.

С помощью одного такого View можно покрыть все HTTP-методы, объявив их в разных методах с зарезервированными названиями (*get, post, update...*).

Также можно задать дополнительные хранимые данные. Например, создать поле *tags* и написать одинаковые тег *user* для всех View, связанных с пользователями. А потом дописать логику так, чтобы View с одинаковыми тегами группировались в документации.

Пример:

user	user	^
POST	/user.check_auth	∨
POST	/user.logout	∨
POST	/v2.user.login	∨
GET	/v2.user.current	∨

В общем, class-based view позволяют сделать гораздо больше, чем function-based view, поэтому мы будем использовать только их.

Однако не стоит забывать, что каким бы не был View (хоть lambda-функцией), одна из его главных целей — вернуть Response.

Response

Объект, содержащий данные об ответе сервера на запрос клиента. В нем находятся статус ответа, данные, заголовки, новые куки.

Пример. Скриншот с атрибутами Response:

```
resp = (Response: 0) <Response OK not prepared>
  ATTRS = (frozenset: 3) frozenset({'_stored_content_type', '_content_type', '_content_dict'})
  body = (bytes: 160) b '{"status": "ok", "data": {"filename": "782.jpeg", "upload_id": "d0854d26-4b3b-4cfd-b7fd-775acd52cba7", "created": "2021-11-05T23:55:23.113255", "size": 156450}}'
  body_length = (int) 0
  charset = (str) 'utf-8'
  chunked = (bool) False
  compression = (bool) False
  content_length = (int) 160
  content_type = (str) 'application/json'
  cookies = (SimpleCookie: 0)
  headers = (CIMultiDict: 1) <CIMultiDict('Content-Type': 'application/json; charset=utf-8')>
  keep_alive = (NoneType) None
  last_modified = (NoneType) None
  output_length = (str) 'Traceback (most recent call last):\n File "/Users/artembakulev/Library/Application Support/JetBrains/Toolbox/apps/PyCharm-P/ch-0/211.7628.24/PyChar... View'
  prepared = (bool) False
  reason = (str) 'OK'
  status = (int) 200
  task = (NoneType) None
  text = (str) '{"status": "ok", "data": {"filename": "782.jpeg", "upload_id": "d0854d26-4b3b-4cfd-b7fd-775acd52cba7", "created": "2021-11-05T23:55:23.113255", "size": 156450}}'
```

Как только мы вызовем метод *.prepare()* у такого Response, его статус и заголовки будут преобразованы в HTTP-вид и записаны в сокет: начнется его отправка обратно клиенту.

Пока не до конца понятно, как же связать все эти компоненты и объекты — не хватает какой-то общей точки, которая будет создавать Request, передавать его в Router и передавать Response обратно пользователю.

Этой точкой можно назвать *App*, который мы рассмотрим в следующей карточке.