AIOHTTP INTRODUCTION

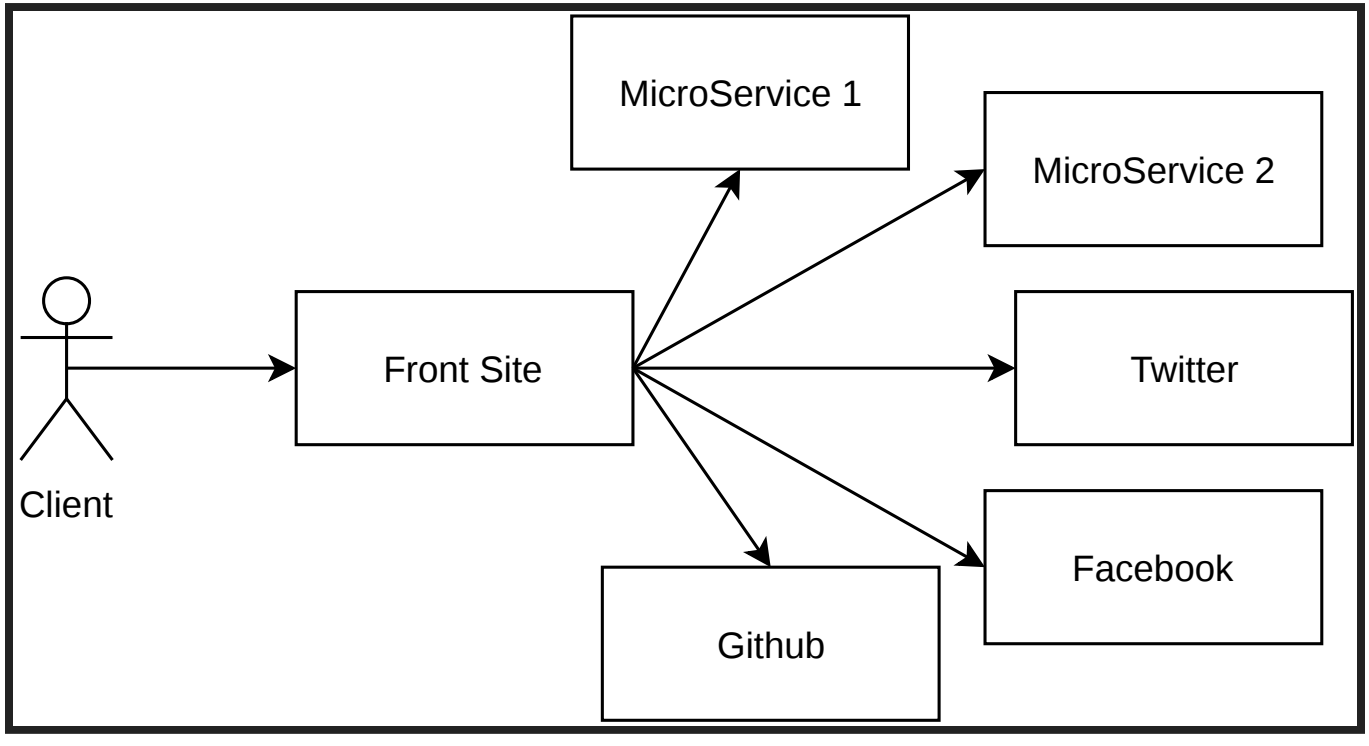
ANDREW SVETLOV

andrew.svetlov@gmail.com

BIO

- Use Python for more than 16 years
- Python Core Developer since 2012
- *asyncio* committer
- *aiohttp* maintainer
- Author of a dozen libraries under *aio-libs* umbrella

WHY?



- 1,000 OS native threads
- 1,000,000 lightweight tasks

AIOHTTP -- ASYNCIO-BASED WEB

- Client API
- Server
- Persistent connections
- Websockets

3 YEARS LONG HISTORY

- Extracted from asyncio (former tulip)
- 22 releases so far
- 3300+ commits
- 150+ contributors
- 98% code coverage

CLIENT API

REQUESTS

```
import requests

r = requests.get('https://api.github.com/user',
                 auth=('user', 'pass'))
print(r.status_code)
print(r.text)
```

AIOHTTP

NO WAY!!!

BARE FUNCTIONS ARE DEPRECATED

REQUESTS WITH SESSION

```
session = requests.Session()

r = session.get(url)
print(r.status_code)
print(r.headers['content-type'])
print(r.text)
```

THINK ABOUT KEEP-ALIVES

AND COOKIES

AIOHTTP WITH SESSION

```
async def coro():
    async with aiohttp.ClientSession() as session:
        async with session.get(url) as r:
            print(r.status)
            print(r.headers['content-type'])
            print(await r.text())
```

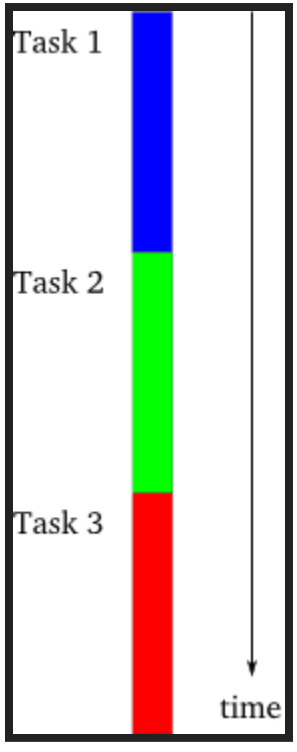
RULE OF THUMB FOR COROUTINES

1. Coroutine is an **async def** function
2. Call a coroutine with **await**
3. If a function contains **awaits** -- make it coroutine

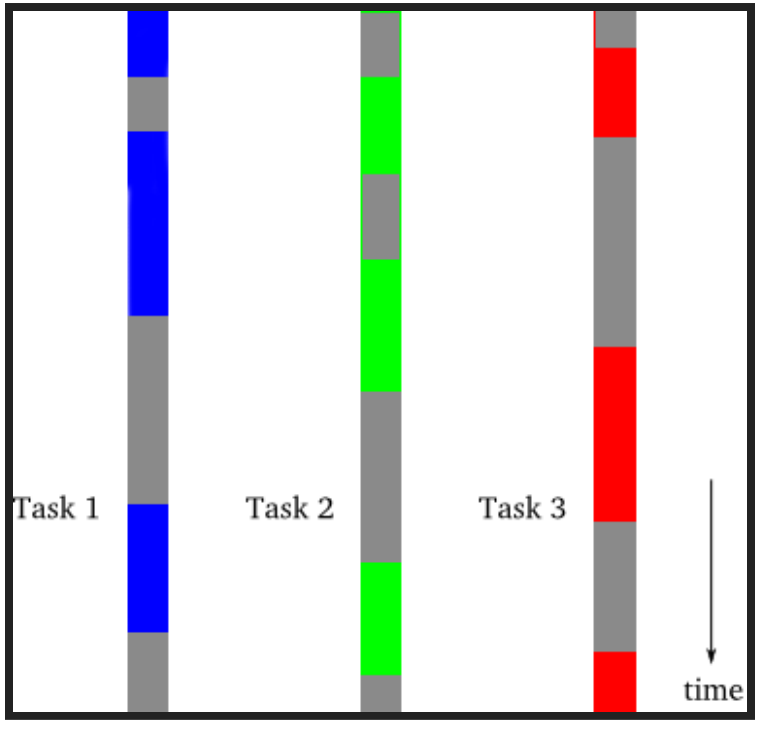
```
async def func():
    await asyncio.sleep(1)

async def other():
    await func()
```

MULTIPLE CUNCURRENT TASKS

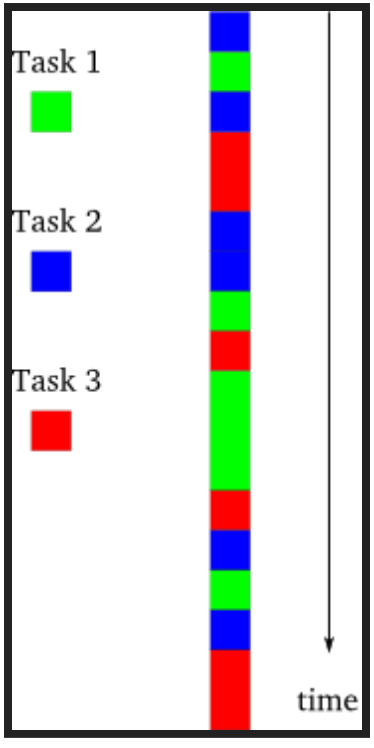
```
async def fetch(session, url):
    async with session.get(url) as r:
        assert r.status == 200
        return await r.text()

tasks = [loop.create_task(fetch(session, url)
    for url in ['http://google.com', 'http://python.org']]
res = await asyncio.gather(*tasks)
```



Sync



Threaded



Async

TIMEOUTS

```
async def coro(session):
    with aiohttp.Timeout(1.5):
        async with session.get(url) as r:
            ...
```

WEBSOCKETS

```
async with client.ws_connect(
        'http://websocket-server.org/endpoint') as ws:

    async for msg in ws:
        if msg.data == 'close':
            await ws.close()
            break
        else:
            ws.send_str("Answer on " + msg.data)
```

SERVER

DJANGO

```
from django.conf.urls import url
from django.http import HttpResponse


def index(request):
    return HttpResponse("Hello, world")

urlpatterns = [
    url(r'^$', index),
]
```

AIOHTTP

```
from aiohttp import web

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

app = web.Application(loop=loop)
app.router.add_route('GET', '/', index)
web.run_app(app)
```

TORNADO

```
import tornado.ioloop
import tornado.web

class MainHandler(tornado.web.RequestHandler):
    def get(self):
        self.write("Hello, world")

app = tornado.web.Application([
    (r"/", MainHandler)])

app.listen(8888)
tornado.ioloop.IOLoop.current().start()
```

SERVERSIDE WEBSOCKETS

```
async def handler(request):
    ws = web.WebSocketResponse()
    await ws.prepare(request)

    async for msg in ws:
        if msg.data == 'close':
            await ws.close()
            break
    else:
        ws.send_str(msg.data + '/answer')

    return ws
```

TIPS AND TRICKS

DEVELOPMENT CYCLE

- Use single process for dev environment
- Make test run easy
- Deploy separately in different processes/containers/nodes

SAY NO TO CELERY

```
async def long_running_operation():
    ...

loop.create_task(long_running_operation())
```

DEBUG MODE: PROBLEM

```
async def f():
    fut = asyncio.Future()
    fut.set_exception(RuntimeError())
    del fut
```

```
...
ERROR:asyncio:Future exception was never retrieved
future: Future finished exception=RuntimeError()
RuntimeError
```

PYTHONASYNCIODEBUG=1

```
$ PYTHONASYNCIODEBUG=x python myapp.py
```

```
ERROR:asyncio:Future exception was never retrieved
future: Future finished exception=RuntimeError() created at filename
source_traceback: Object created at (most recent call last):
...
File "filename.py", line 10, in f
fut = asyncio.Future()
RuntimeError
```

EXPLICIT LOOP

```
async def fetch_all(urls, *, loop):
    async with aiohttp.ClientSession(loop=loop):
        ...

loop = asyncio.get_event_loop()
asyncio.set_event_loop(None)  # !!!
await fetch_all(urls, loop=loop)
```

UTILIZE KEEP-ALIVES

```
async def fetch_all(urls, *, loop):
    coros = []
    async with aiohttp.ClientSession(loop=loop):
        for url in urls:
            coros.append(fetch(url))
        await asyncio.gather(*tasks, loop=loop)
```

TESTING

```
class Test(unittest.TestCase):

    def setUp(self):
        self.loop = asyncio.new_event_loop()
        asyncio.set_event_loop(None)

    def tearDown(self):
        self.loop.close()

    def test_func(self):
        async def go():
            self.assertEqual(1, await func(loop=self.loop))

        self.loop.run_until_complete(go())
```

TESTING WITH PYTEST-AIOHTTP

```
def create_app(loop, path, handler):
    app = web.Application(loop=loop)
    app.router.add_route('GET', path, handler)
    return app

async def test_hello(test_client):
    async def hello(request):
        return web.Response(body=b'Hello, world')
    client = await test_client(create_app, '/', handler)

    resp = await client.get('/')
    assert resp.status == 200
    text = await resp.text()
    assert 'Hello, world' in text
```

NO GLOBAL OBJECTS!!!

```
from motor.motor_asyncio import AsyncIOMotorClient
DBNAME = 'testdb'
db = AsyncIOMotorClient()[DBNAME]

async def register(request):
    post_data = await request.post()
    login, password = post_data['login'], post_data['password']
    matches = await db.users.find({'login': login}).count()
    ...
```

APPLICATION AS A STORAGE

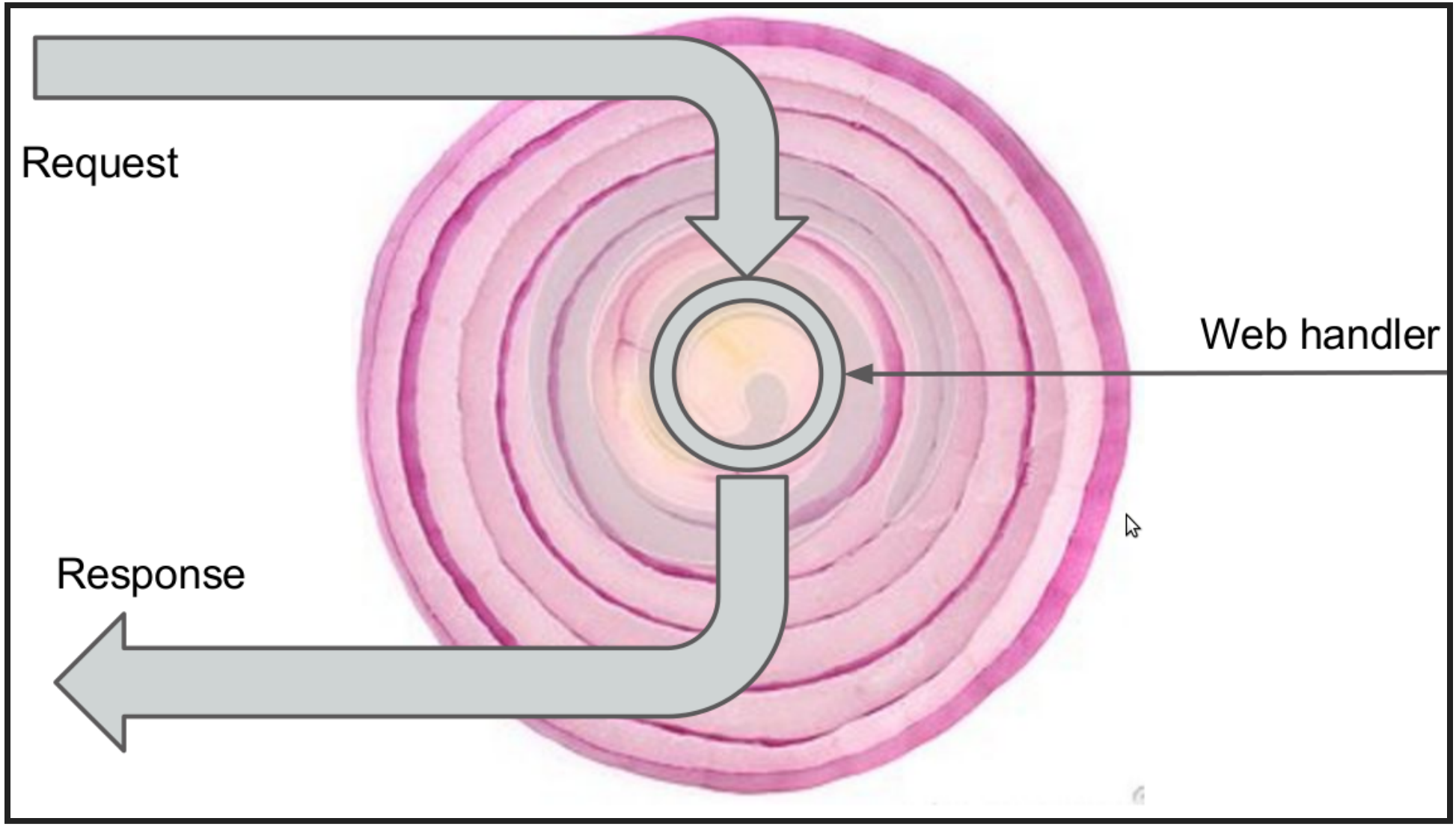
```
async def register(request):
    post_data = await request.post()
    login, password = post_data['login'], post_data['password']
    matches = await request.app['db'].users.find({'login': login}).c
    ...
```

DB INIT AND SHUTDOWN

```
def make_app(loop=None):
    app = web.Application(loop=loop)
    mongo = AsyncIOMotorClient(io_loop=loop)
    db = mongo['testdb']
    app['db'] = db

    async def cleanup(app):
        mongo.close()

    app.on_cleanup.append(cleanup)
    ...
    return app
```

MIDDLEWARES

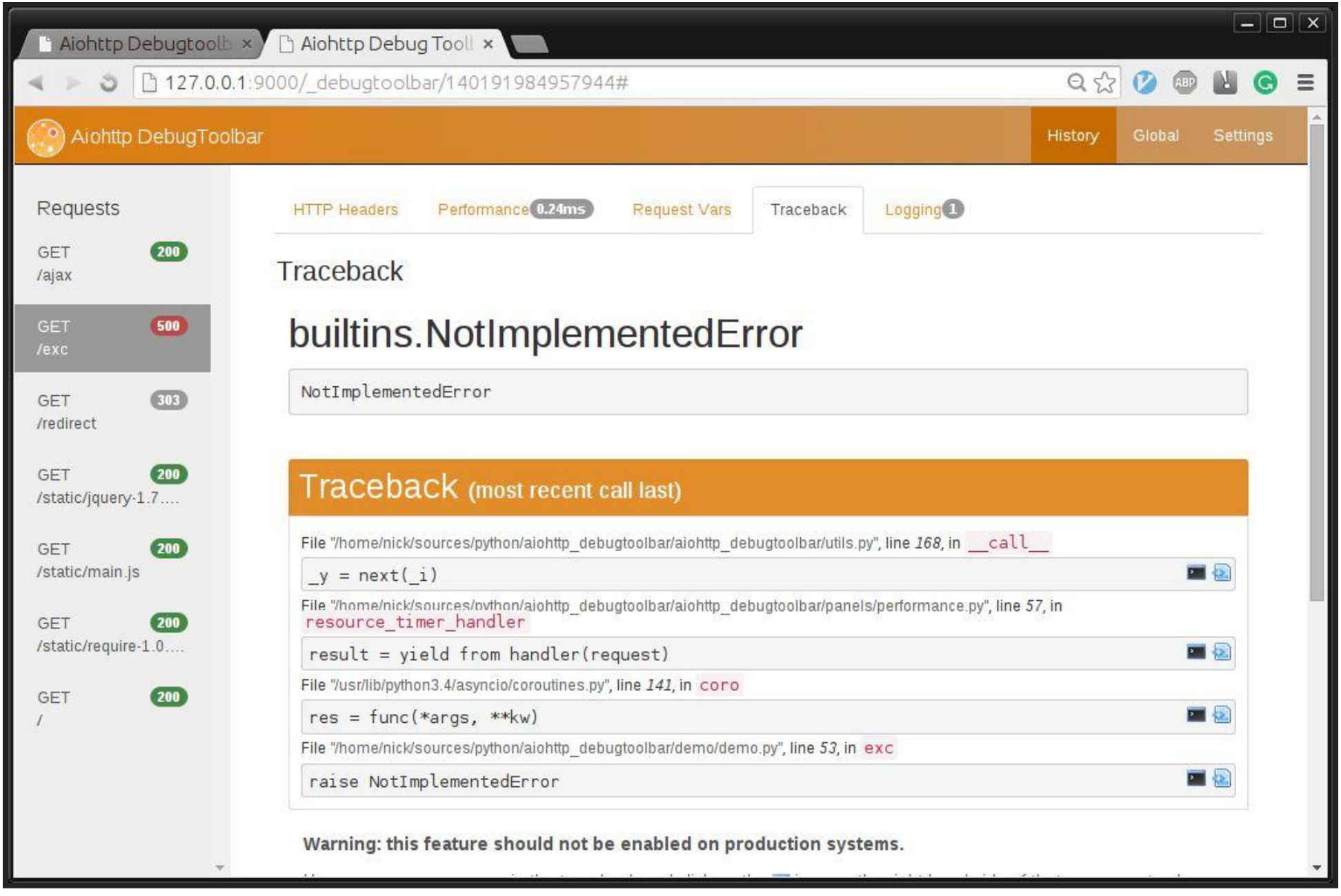
REQUEST LIFECICLE AND MIDDLEWARES



SERVER-SIDE SESSIONS

```
from aiohttp_session import get_session

async def hander(request):
    session = await get_session(request)
    session['key'] = 'value'
    return web.Response()
```

DEBUG TOOLBAR


Aiohttp Debugtoolb
Aiohttp Debug Tool
127.0.0.1:9000/_debugtoolbar/140191984957944#
Aiohttp DebugToolbar
History
Global
Settings
Requests
GET 200 /ajax
GET 500 /exc
GET 303 /redirect
GET 200 /static/jquery-1.7....
GET 200 /static/main.js
GET 200 /static/require-1.0....
GET 200 /
HTTP Headers
Performance 0.24ms
Request Vars
Traceback
Logging 1
Traceback
builtins.NotImplementedError
NotImplementedError
Traceback (most recent call last)
File "/home/nick/sources/python/aiohttp_debugtoolbar/aiohttp_debugtoolbar/utils.py", line 168, in __call__
_y = next(_i)
File "/home/nick/sources/python/aiohttp_debugtoolbar/aiohttp_debugtoolbar/panels/performance.py", line 57, in resource_timer_handler
result = yield from handler(request)
File "/usr/lib/python3.4/asyncio/coroutines.py", line 141, in coro
res = func(*args, **kw)
File "/home/nick/sources/python/aiohttp_debugtoolbar/demo/demo.py", line 53, in exc
raise NotImplementedError
Warning: this feature should not be enabled on production systems.

QUESTIONS?

ANDREW SVETLOV

andrew.svetlov@gmail.com

@andrew_svetlov