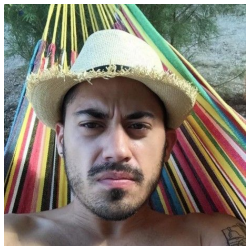
Real time sweetness with Django Channels

Python Athens User Group — Thursday 13 June 2019

Paris Kasidiaris / @pariskasid

Who am I



Paris Kasidiaris / @pariskasid

Co-founder of SourceLair (https://lair.io)

Co-organizer of Docker Athens User Group (https://docker.gr)

In love with Python

I love Django. Let's talk Django.

So, what did you think about Jamie Foxx’ performance?

Let's pretend this never happened.

Django

The web framework for perfectionists with deadlines.

- High level Python web framework
- End-to-end solution; handles databases, authentication, user management, HTTP and template rendering (that’s a lot)

Paris Kasidiaris / @pariskasid

HTTP in Django

- Almost all business logic of the request/response cycle is being implemented in Python functions called **views**
- HTTP requests are being mapped to *views* according to the requested path in the **urls** configuration
- All these are being orchestrated in a **WSGI** entrypoint

Paris Kasidiaris / @pariskasid

Example view

```
from django.shortcuts import render

def home(request):
    return render(request, "echo/index.html")
```

Example urls configuration

```
from django.urls import path

from echo import views as echo_views


urlpatterns = [
    path('', echo_views.home),
]
```

Paris Kasidiaris / @pariskasid

WSGI

What the hell is WSGI ?!

WSGI

WSGI is a very rare **Sexually Transmitted Disease** exclusive to Python web developers.

Paris Kasidiaris / @pariskasid

For real now...

WSGI

WSGI is the Web Server Gateway Interface.

It is a specification that describes how a web server communicates with web applications, and how web applications can be chained together to process one request.

WSGI is a Python standard described in detail in **PEP 3333**.

Paris Kasidiaris / @pariskasid

Example WSGI application

```
def hello_world(environ, start_response):
    data = b"Hello, World!\n"
    start_response("200 OK", [
        ("Content-Type", "text/plain"),
    ])
    return [data]
```

Paris Kasidiaris / @pariskasid

Making Django real-time

Limitations of Django for real-time use cases

1. WSGI does not support real-time
2. Real-time semantics are missing from Django

Paris Kasidiaris / @pariskasid

WSGI design limitations for real-time

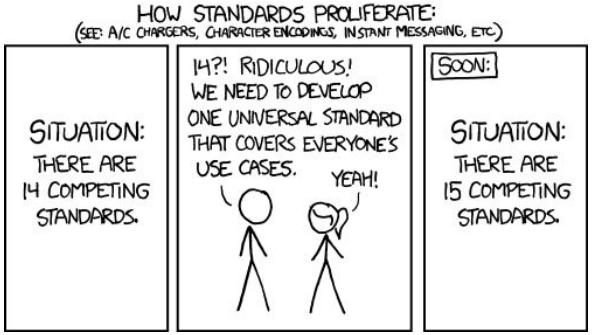
- Synchronous implementation
- Does not support background tasks
- Strictly supports only one protocol: HTTP (not WebSocket)

Paris Kasidiaris / @pariskasid

How do we overcome the limitations of a standard?

...we create another one.

Paris Kasidiaris / @pariskasid



https://xkcd.com/927/

So, how did we overcome the limitations of WSGI?

With **ASGI**.

Paris Kasidiaris / @pariskasid

ASGI ?!

What the hell is ASGI ?!

ASGI in a nutshell

ASGI is a another **Sexually Transmitted...**

Paris Kasidiaris / @pariskasid

Enough already with these “jokes”...

ASGI in a nutshell

ASGI is the Asynchronous Server Gateway Interface.

It is a spiritual successor to WSGI, intended to provide a standard interface between async-capable Python web servers, frameworks, and applications.

The ASGI is authored by the Django team: https://github.com/django/asgiref.

Paris Kasidiaris / @pariskasid

ASGI vs. WSGI

- Multiple protocols: HTTP, HTTP/2, WebSocket
- Asynchronous messaging lifecycle
- In-process background tasks

- Only HTTP
- Synchronous request/response lifecycle
- No background tasks at all

Paris Kasidiaris / @pariskasid

How does ASGI work?

ASGI is a single, **asynchronous callable**, accepting three arguments:

1. **scope**: Contains details about the incoming connection
2. **send**: An awaitable that lets you send events to the client
3. **receive**: Lets you receive events from the client.

What makes ASGI special?

The design of ASGI:

1. Allows multiple incoming and outgoing events for each application

2. Allows for a background coroutine (e.g. listen for events on an external trigger, like a Redis queue)

Paris Kasidiaris / @pariskasid

Example ASGI application

```
async def application(scope, receive, send):
    assert scope["type"] == "http"
    await send({
        "type": "http.response.start",
        "status": 200,
        "headers": [
            [b"content-type", b"text/plain"],
        ]
    })
    await send({
        "type": "http.response.body",
        "body": b"Hello, world!",
    })
```

Paris Kasidiaris / @pariskasid

Another example ASGI application

```
async def application(scope, receive, send):
    event = await receive()
    await send({
        "type": "websocket.send",
        "text": "Hello world!"
    })
```

Paris Kasidiaris / @pariskasid

Example ASGI message

```
{
    "type": "http.response.start",
    "status": 200,
    "headers": [
        ["content-type",  "text/plain"],
    ]
}
```

Paris Kasidiaris / @pariskasid

What about real time semantics?

Enter Django Channels

Django Channels (or simply... Channels)

Channels is a project that takes Django and extends its abilities beyond HTTP - to handle WebSockets, chat protocols, IoT protocols, and more.

Paris Kasidiaris / @pariskasid

Channels semantics for real-time use cases

1. Scopes and Events
2. Consumers
3. Multi-protocol routing

Paris Kasidiaris / @pariskasid

Scopes

The **scope** is a set of details about a single incoming connection, like:

- The *path* a web request was made from
- The originating *IP address* of a WebSocket
- The *user* messaging a chatbot

The scope persists throughout the connection (HTTP → request/response cycle, WebSocket → socket lifetime)

Paris Kasidiaris / @pariskasid

Events

Channels applications are instantiated once per scope.

Then, they are fed the stream of events happening within that scope to decide what to do with.

Within the lifetime of a scope the application instance will handle any events from it and persist any event data desired onto the application instance.

Paris Kasidiaris / @pariskasid

Consumers

A consumer is the basic unit of Channels code.

It is being called a *consumer* as it consumes *events*.

Consumers are valid ASGI applications.

Example consumer

```
from channels.generic.websocket import AsyncJsonWebsocketConsumer

class EchoChamberConsumer(AsyncJsonWebsocketConsumer):
    async def connect(self):
        await self.accept()

    async def receive_json(self, content):
        await self.send_json(content)

    async def disconnect(self, close_code):
        await self.close()
```

Paris Kasidiaris / @pariskasid

Multi protocol routing

Channels provides routing classes that allow you to combine and stack your consumers (and any other valid ASGI application) to dispatch based on what the connection is.

Paris Kasidiaris / @pariskasid

Example of multi-protocol routing

```
from channels.routing import ProtocolTypeRouter, URLRouter
from django.conf.urls import url

from echo.consumers import EchoChamberConsumer

application = ProtocolTypeRouter({
    # (http->django views is added by default)
    "websocket": URLRouter([
        url(r"^echo/$", EchoChamberConsumer),
    ]),
})
```

Paris Kasidiaris / @pariskasid

Running a Channels app

1. Add *ASGI_APPLICATION* to your Django settings (e.g. "myproject.routing.application")
2. Run it in development with.. *python manage.py runserver* (Channels overwrite the Django command)
3. Run it in production with *uvicorn myproject.routing:app*

Paris Kasidiaris / @pariskasid

Let's get our hands dirty

Let’s create an echo chamber

1. Open a WebSocket
2. Type in a text box
3. Send the letter typed to the server using the WebSocket
4. Send the letter back to the client using the WebSocket

Paris Kasidiaris / @pariskasid

github.com/parisk/echo-chamber

Okay. I lied.

import antigravity

Thank you!