

Web asíncrona con Django Channels



Acerca de mí

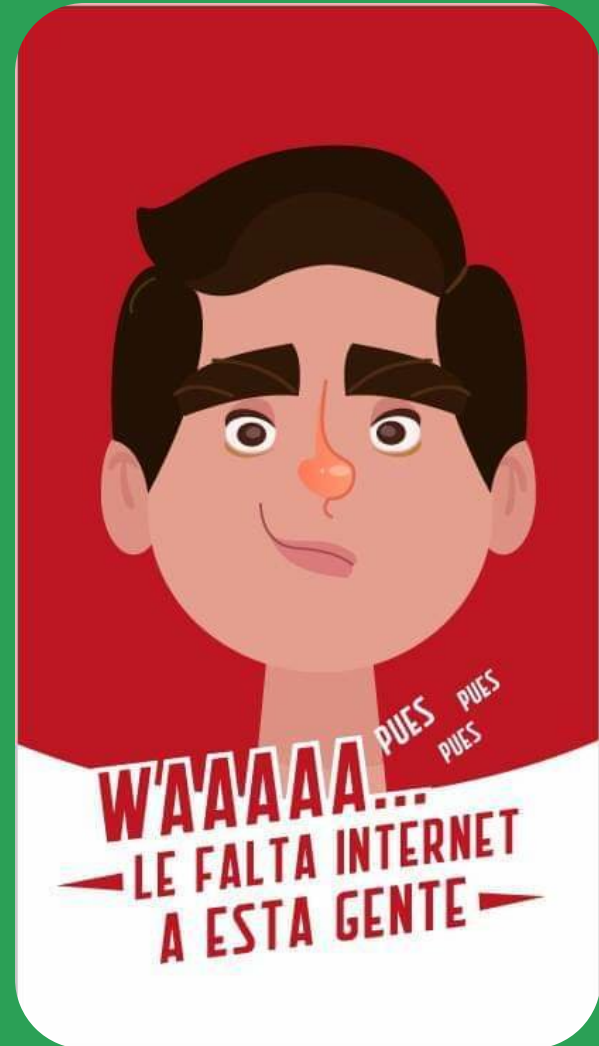
David Fernando Zuluaga A



@David_fza 

@dafer154 

zuluaaristi@gmail.com 



Temas a tocar

- Django channels y sus conceptos
- Ejemplo de Django Channels
- Desglosando el código
- Preguntas

¿Qué es Django Channels?



- Es un proyecto creado en **Sep 2016** que contó con un apoyo de 150000 Euros de fundación mozilla
- Permite la utilización de **WebSockets** y ejecutar **tareas en segundo plano** de forma asíncrona.
- Permite la comunicación **Asíncrona** dentro de lo síncrono de Django.
- Permite que el modelo **WSGI** (Response-request) se pueda modificar como **ASGI** (Consumers)
- También combina una arquitectura por **eventos** con capas de canal
- Implementa push, Http2, auth

Versiones de Django Channels

V1.0

- Fue creada hace 3 años
- Se utilizaba con **Python 3.0** hacia atras
- Se instalaba la **capa de canal**(Redis) a parte
- No era completamente Asíncrono

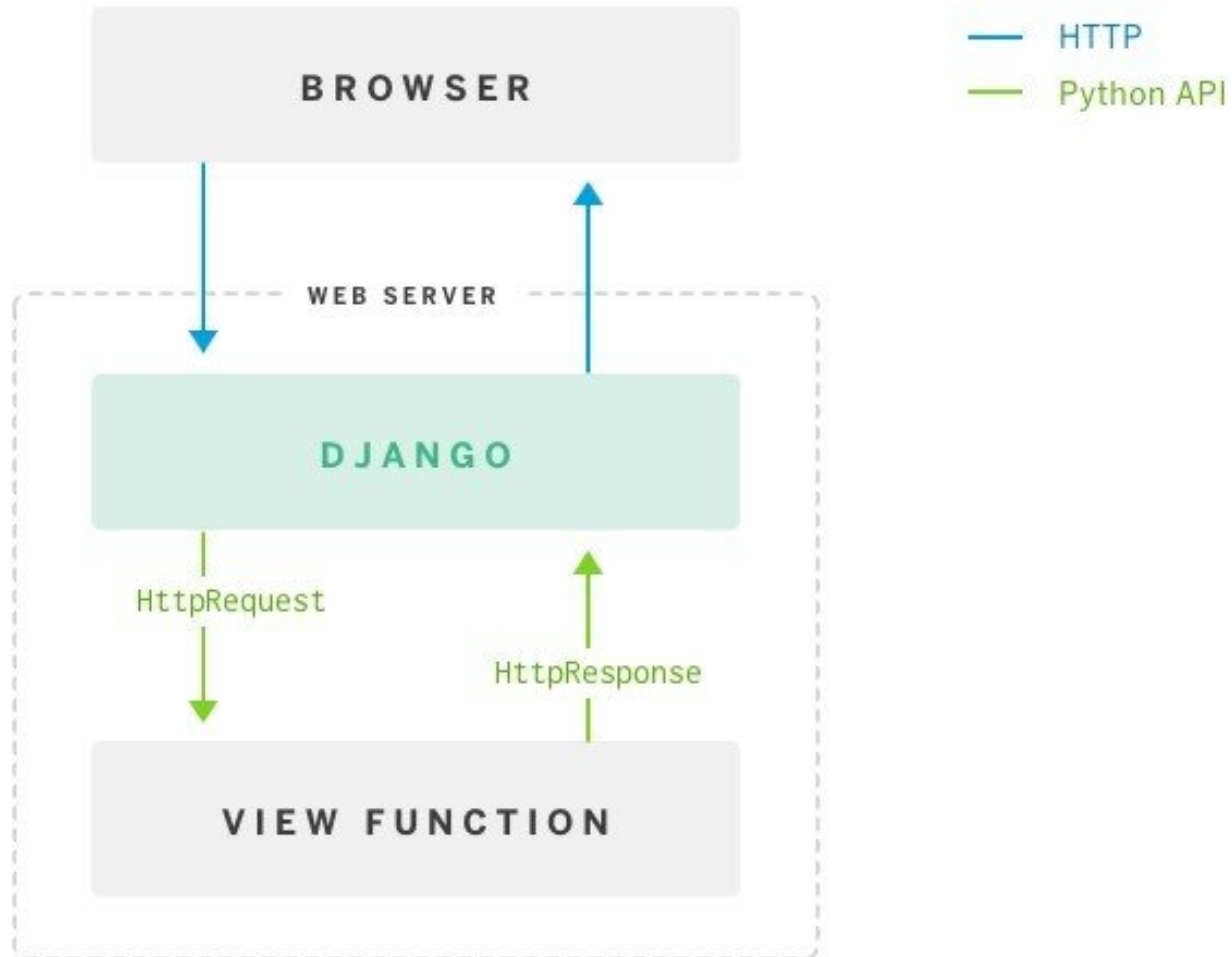
V2.0

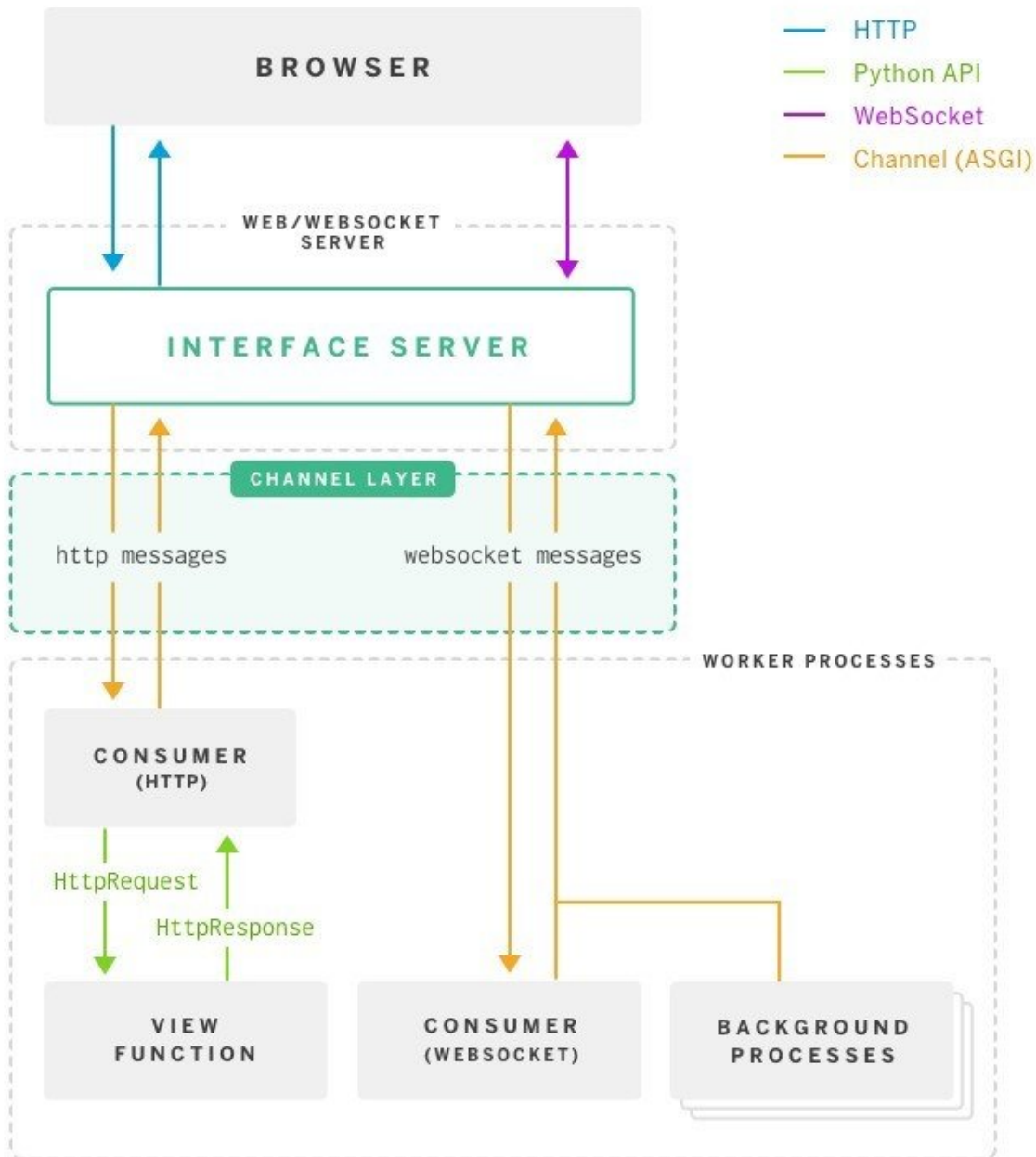
- Es su última versión
- es utilizado a partir de **python 3.5** en adelante
- Nuevos **consumers**
- Asíncrono completo
- Consumers basados en **funciones** y **enrutamiento**
- HTTP **sessions** y Django auth

Documentación

<https://channels.readthedocs.io/en/stable/index.html>

Las entrañas de Django





CARACTERÍSTICAS

- Consumer
- Channel Layer
- WebSocket

Síncrono Vs Asíncrono

Síncrono

Un código síncrono es aquel código donde **cada instrucción espera a la anterior**

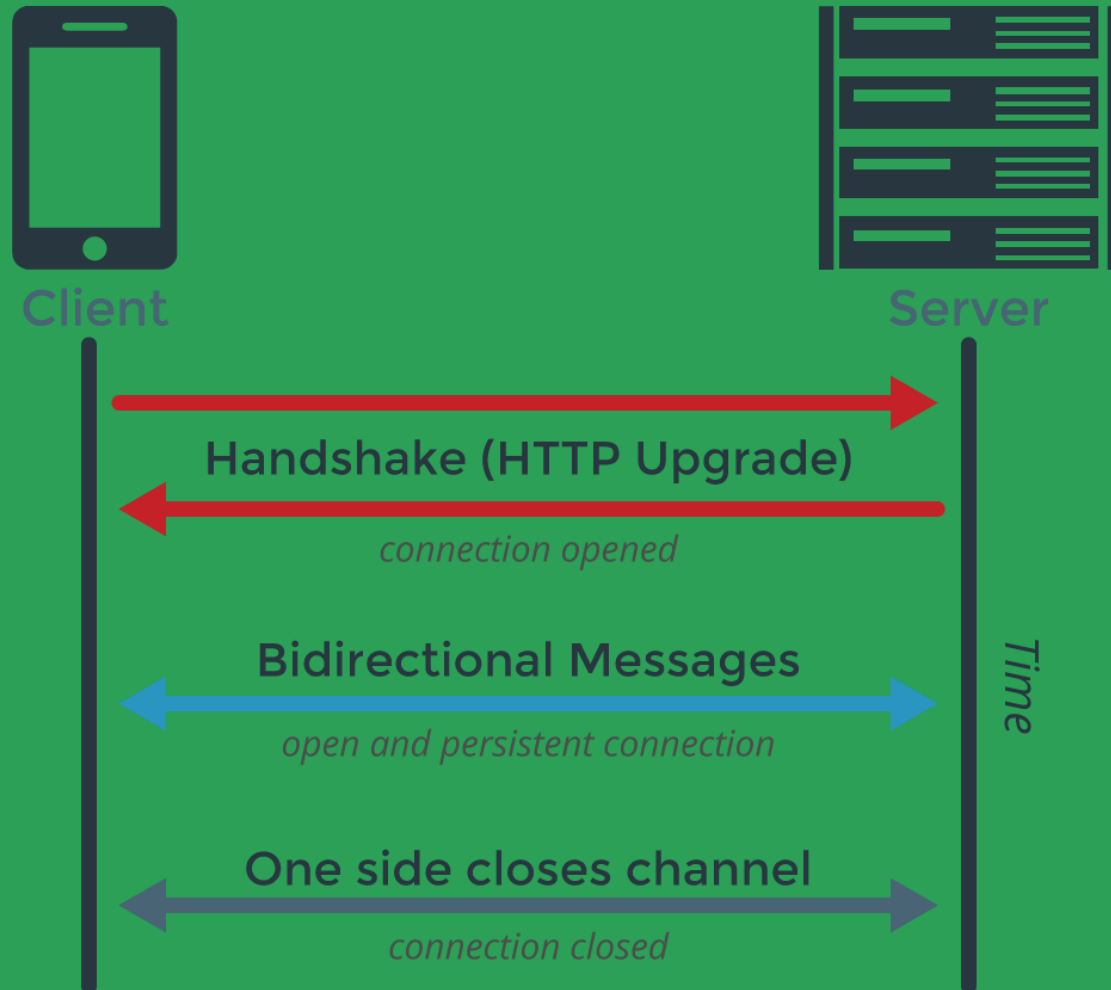
Asíncrono

- un código asíncrono **no espera a las instrucciones** diferidas y continúa con su ejecución
- permite tener una mejor respuesta en las aplicaciones y **reduce el tiempo** de espera del cliente.

Entendiendo los WebSockets
















- Permiten abrir una sesión de **comunicación** entre el navegador del usuario y un servidor.
- Se puede enviar mensajes a un servidor y recibir respuestas por **eventos** sin tener que consultar al servidor para una respuesta
- En el momento que hay un cambio en un evento todos los que están **suscritos** pueden recibir inmediatamente esa información
- Cuenta con diferentes **propiedades** como:
WebSocket.onclose, WebSocket.onmessage,
WebSocket.send(data)

Entendiendo los WebSockets



Navegadores que lo soportan

🔗 Take this quick survey to help us improve our browser compatibility tables

													
	 Chrome	 Edge	 Firefox	 Internet Explorer	 Opera	 Safari	 Android webview	 Chrome for Android	 Edge Mobile	 Firefox for Android	 Opera for Android	 Safari on iOS	 Samsung Internet
Basic support	Yes	Yes	11 ★ ▼	?	Yes	Yes	?	?	Yes	14 ★ ▼	?	Yes	?

Documentación

<https://developer.mozilla.org/en-US/docs/Web/API/WebSocket>

Ejemplo de WebSocket

```
var exampleSocket = new WebSocket("ws://www.example.com/socketserver", "protocolOne");

exampleSocket.onopen = function (event) {
  exampleSocket.send("Here's some text that the server is urgently awaiting!");
};

//Se recibe el mensaje desde el servidor

exampleSocket.onmessage = function (event) {
  console.log(event.data);
}

exampleSocket.close();
```

Explicacion

https://developer.mozilla.org/es/docs/WebSockets-840092-dup/Writing_WebSocket_client_applications

WSGI vs ASGI



WSGI

- WSGI es la **arquitectura** entre los servidores web y las aplicaciones web o frameworks en Python
- Del lado del servidor, se invoca la aplicación por cada pedido que recibe del cliente **HTTP**
- Las aplicaciones WSGI son un único y se pueden llamar de forma **síncrona** que toma una solicitud y devuelve una respuesta
- esto no permite conexiones de larga duración, como las conexiones HTTP o WebSocket

ASGI

- ASGI es el nombre de la especificación del servidor **asíncrono** en el que se basan los canales
- Está estructurado como una **doble llamada**
- Cada **evento** que envíe o reciba es un dict de Python, con un formato predefinido.
- ASGI también está diseñado para ser un **superconjunto** de WSGI, permitiendo que las aplicaciones WSGI se ejecuten dentro de los servidores ASGI

Routing como protocolo

- Este es usado para el **direccionamiento** de las url que permiten tener conexión con los **websockets**
- Puede combinar múltiples consumers (que son sus propias aplicaciones ASGI) en una aplicación más grande que represente su proyecto mediante **routing**
- Un **router** solo puede tener un **consumer** para una conexión determinada.

Ejemplo de Routing

Routing.py

```
from django.conf.urls import url
from channels.routing import ProtocolTypeRouter, URLRouter
from channels.auth import AuthMiddlewareStack
from chat.consumers import AdminChatConsumer, PublicChatConsumer
from aprs_news.consumers import APRSNewsConsumer

application = ProtocolTypeRouter({

    # WebSocket chat handler
    "websocket": AuthMiddlewareStack(
        URLRouter([
            url(r"^chat/$", PublicChatConsumer),
        ])
    ),
})
```

¿En qué consiste los Channel Layer?

- Permiten hablar entre las diferentes **instancias** de la aplicación
- Es el responsable del **transporte** de los mensajes
- Son una parte útil de hacer una aplicación distribuida en **tiempo real**
- Son controladas por **Redis**

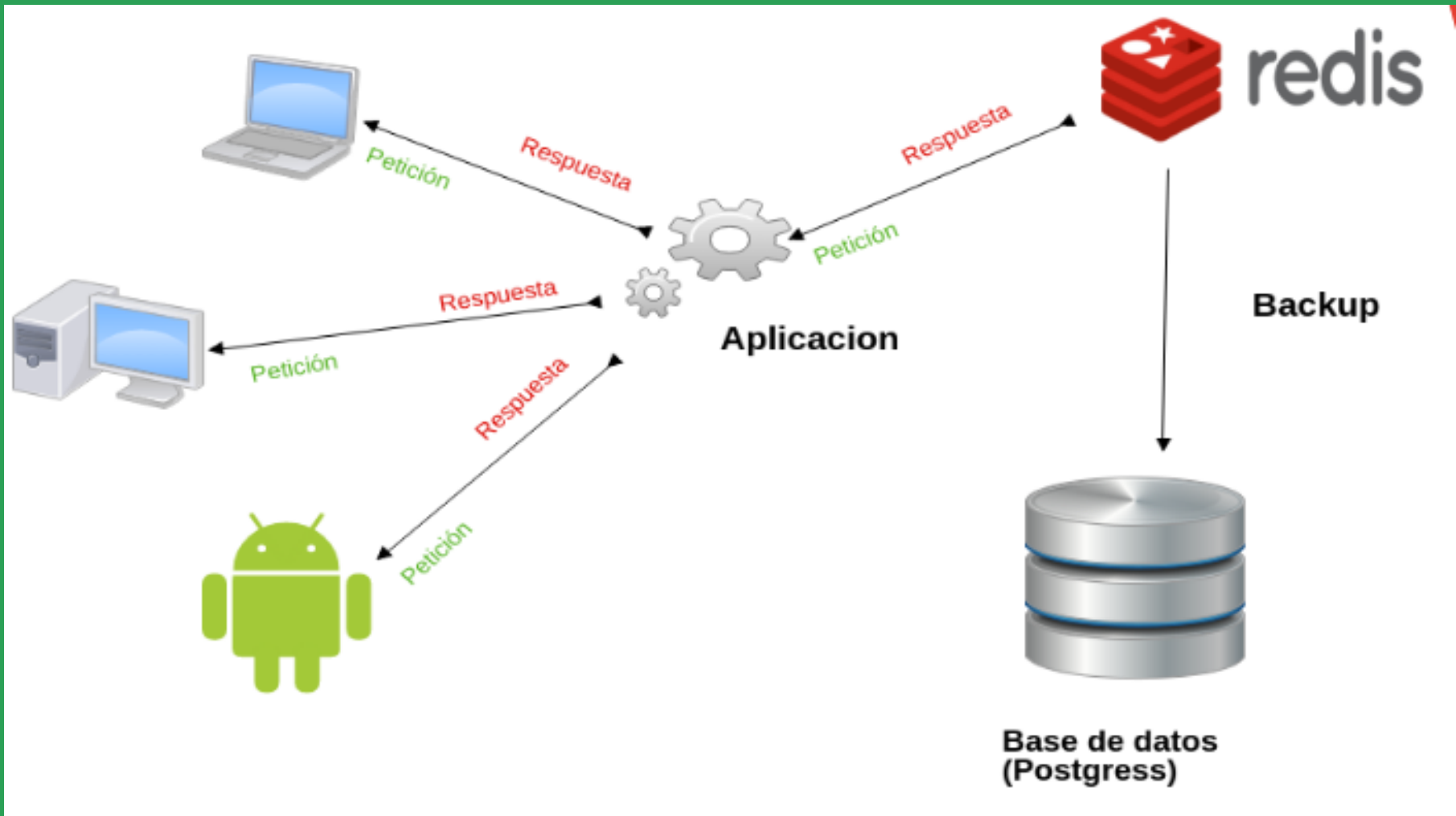
settings.py

```
CHANNEL_LAYERS = {  
    "default": {  
        "BACKEND": "channels_redis.core.RedisChannelLayer",  
        "CONFIG": {  
            "hosts": [("redis-server-name", 6379)],  
        },  
    },  
}
```

Redis como capa de canal

- Redis es la **capa de canal** que utiliza Django-channels para la transmisión de los **mensajes en tiempo real**.
- Las capas de canal tienen una interfaz puramente **asíncrona**.
- Mantiene la **información en memoria**
- Permite **reducir las cargas** de los servidores web

Redis como capa de canal



¿Qué es un consumer?



¿Qué es un consumer?

- Es paralelo a la vista en Django, donde se controla los **WebSockets** a través de Python
- A diferencia de las vistas de Django, los consumidores son de **larga duración**.
- Permite estructurar su código como una serie de funciones a las que llamar cuando suceda un **evento**, en lugar de hacer que escriba un bucle de eventos
- Permite escribir **código síncrono** o asíncrono
- Se manejan distintos Consumers: **SyncConsumer**, **WebsocketConsumer**, **AsyncWebsocketConsumer**, **JsonWebsocketConsumer** y otros mas

Ejemplo de un consumer

consumer.py

```
from channels.generic.websocket import WebsocketConsumer

class MyConsumer(WebsocketConsumer):
    groups = ["broadcast"]

    def connect(self):
        self.accept()
        self.accept("subprotocol")
        # To reject the connection, call:
        self.close()

    def receive(self, text_data=None, bytes_data=None):
        # Called with either text_data or bytes_data for each frame
        # You can call:
        self.send(text_data="Hello world!")
        # Want to force-close the connection? Call:
        self.close()

    def disconnect(self, close_code):
        # Called when the socket closes
```

Ejemplo de aplicación

<https://github.com/dafer154/questions-realTime>



Características

- Logueo de usuarios
- Registro de preguntas de geografía
- Envío de preguntas en tiempo real
- Resultados de los estudiantes

Demo time



Conociendo a fondo la aplicación

- Settings y Channel layer
- Consumer
- Routing
- Websocket-Template

Settings y Channel layer

Settings.py

```
...

INSTALLED_APPS = [
    'channels',
    'bootstrap3',
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'usuarios',
    'preguntas'
]

...

WSGI_APPLICATION = 'QuestionsRT.wsgi.application'
ASGI_APPLICATION = 'QuestionsRT.routing.application'
.....

CHANNEL_LAYERS = {
    "default": {
        "BACKEND": "channels_redis.core.RedisChannelLayer",
        "CONFIG": {
            "hosts": ['redis://localhost:6379/4']
        },
    },
}
```

Channel Layer Corriendo

```

david@david-Lenovo-ideapad-300-14ISK: ~/Escritorio/questions-realTime/QuestionsRT
(pyday) david@david-Lenovo-ideapad-300-14ISK:~/Escritorio/questions-realTime/Que
stionsRT$ python manage.py runserver
Performing system checks...

System check identified no issues (0 silenced).
November 27, 2018 - 16:11:17
Django version 2.1.1, using settings 'QuestionsRT.settings'
Starting ASGI/Channels version 2.1.3 development server at http://127.0.0.1:8000
/
Quit the server with CONTROL-C.
2018-11-27 16:11:17,252 - INFO - server - HTTP/2 support not enabled (install th
e http2 and tls Twisted extras)
2018-11-27 16:11:17,253 - INFO - server - Configuring endpoint tcp:port=8000:int
erface=127.0.0.1
2018-11-27 16:11:17,253 - INFO - server - Listening on TCP address 127.0.0.1:800
0
[2018/11/27 16:11:18] WebSocket HANDSHAKING /classroom/stream [127.0.0.1:42876]
[2018/11/27 16:11:18] WebSocket CONNECT /classroom/stream [127.0.0.1:42876]
```

Routing

Routing.py

```
from django.urls import path
from channels.routing import ProtocolTypeRouter, URLRouter
from channels.auth import AuthMiddlewareStack
from preguntas.consumers import ClassroomVirtual

websocket_urlpatterns = [
    path("classroom/stream", ClassroomVirtual),
]

application = ProtocolTypeRouter({
    "websocket": AuthMiddlewareStack(
        URLRouter(
            websocket_urlpatterns
        ),
    ),
})
```

Consumer

Consumer Connect

Consumer.py

```
from asgiref.sync import async_to_sync
from channels.generic.websocket import WebsocketConsumer
import json
from .models import Pregunta

class ClassroomVirtual(WebsocketConsumer):

    def connect(self):
        async_to_sync(self.channel_layer.group_add)("chat", self.channel_name)
        self.accept()
```


Consumer receive

Consumer.py

```
...  
  
class ClassroomVirtual(WebsocketConsumer):  
  
    def connect(self):  
        ....  
  
    def receive(self, text_data):  
        pregunta = Pregunta.objects.get(pk=text_data)  
  
        data = {  
            'pregunta': str(pregunta.pregunta),  
            'falso': str('Falso'),  
            'verdadero': str('Verdadero'),  
            'id_pregunta': int(pregunta.id),  
            'tipo': str('Pregunta de Falso o Verdadero')  
        }  
  
        async_to_sync(self.channel_layer.group_send)(  
            "chat",  
            {  
                'type': 'chat_message',  
                'message': data  
            }  
        )
```

Consumer message, Disconnect

Consumer.py

```
...  
  
class ClassroomVirtual(WebSocketConsumer):  
  
    def connect(self):  
        ....  
  
    def receive(self, text_data):  
        ....  
  
# @def chat_message: metodo que recoge el Json Data y lo envia de nuevo a la pagina web,  
# para que este sea visualizado a todos los usuarios que estan dentro de la Url  
  
def chat_message(self, event):  
    message = event['message']  
    self.send(text_data=json.dumps(message))  
  
# @def disconnect: metodo que permite desconectarse del websocket  
  
def disconnect(self, close_code):  
    async_to_sync(self.channel_layer.group_discard)("chat", self.channel_name)
```

```
....  
from .models import Pregunta  
  
class ClassroomVirtual(WebsocketConsumer):  
  
    def connect(self):  
        async_to_sync(self.channel_layer.group_add)("chat", self.channel_name)  
        self.accept()  
  
    def receive(self, text_data):  
        pregunta = Pregunta.objects.get(pk=text_data)  
  
        data = {  
            'pregunta': str(pregunta.pregunta),  
            'falso': str('Falso'),  
            'verdadero': str('Verdadero'),  
            'id_pregunta': int(pregunta.id),  
            'tipo': str('Pregunta de Falso o Verdadero')  
        }  
  
        async_to_sync(self.channel_layer.group_send)(  
            "chat",  
            {  
                'type': 'chat_message',  
                'message': data  
            }  
        )  
  
    def chat_message(self, event):  
        message = event['message']  
        self.send(text_data=json.dumps(message))  
  
    def disconnect(self, close_code):  
        async_to_sync(self.channel_layer.group_discard)("chat", self.channel_name)
```

Frontend-Template

pregunta_list.html

```
<script>

    $(function () {

        var ws_scheme = window.location.protocol == "https:" ? "wss" : "ws";
        var ws_path = ws_scheme + '://' + window.location.host + "/classroom/stream";
        var socket = new ReconnectingWebSocket(ws_path);

        socket.onopen = function() {
            console.log("Hello everywhere")
        }

        if (socket.readyState == WebSocket.OPEN) socket.onopen();

        $(".btn-pregunta").on('click', function(event){
            var id_pregunta = $(this).data("id");
            socket.send(id_pregunta);
        });

        socket.onmessage = function (message) {
            var data = JSON.parse(message.data);
        };

        socket.onclose = function () {
            console.log("Desconectado");
        }
    });

</script>
```

Recursos

- <https://channels.readthedocs.io/en/latest/introduction.html>
- <https://asgi.readthedocs.io/en/latest/>
- <https://developer.mozilla.org/es/docs/WebSockets-840092-dup>
- https://developer.mozilla.org/es/docs/WebSockets-840092-dup/Writing_WebSocket_client_applications
- <http://www.python.org.ar/wiki/WSGI>

Gracias!!



¿Preguntas?



- zuluaaristi@gmail.com 
- @dafer154 