DJANGO CHANNELS

Qu’est-ce que c’est ?

QUI SUIS-JE ?

Paul Guichon

Co-Fondateur de Biru

- pguichon@biru.sh
- https://biru.sh

BIRU

DJANGO CHANNEL : QU'EST-CE QU'IL FAUT POUR LE FAIRE FONCTIONNER

- Un server ASGI
- La librairie django-channel
- Un redis [Optionnel]
- Un reverse proxy

ASGI ?

User ASGI is structured as a single, asynchronous callable. It takes a scope, which is a dict containing details about the specific connection, send, an asynchronous callable, that lets the application send event messages to the client, and receive, an asynchronous callable which lets the application receive event messages from the client.

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

https://asgi.readthedocs.io/en/latest/introduction.html

ET DJANGO-CHANNEL ?

Channels wraps Django's native asynchronous view support, allowing Django projects to handle not only HTTP, but protocols that require long-running connections too - WebSockets, MQTT, chatbots, amateur radio, and more.
https://channels.readthedocs.io/en/latest/introduction.html

MAIS COMMENT ÇA MARCHE ?

- Routing
- Consumers
- Channel Layers

ROUTING

```
application = ProtocolTypeRouter({
# Django's ASGI application to handle traditional HTTP requests
"http": django_asgi_app,

# WebSocket chat handler
"websocket":
    URLRouter([
        path("chat/<username>/", AdminChatConsumer.as_asgi()),
        path("chat/", PublicChatConsumer.as_asgi()),
    ])
})
```

CONSUMERS

```
from channels.consumer import SyncConsumer

class EchoConsumer(SyncConsumer):

    def websocket_connect(self, event):
        self.send({
            "type": "websocket.accept",
        })

    def websocket_receive(self, event):
        self.send({
            "type": "websocket.send",
            "text": event["text"],
        })
```

C'EST TOUT

Hormis les outils pour décomplexifier l'ASGI, il n'y a rien d'autre
Dur d'écrire des gros programmes à l'intérieur de django-channel

CHANNEL LAYERS

- Il faut redis
- L'application devient distribuée
- On peut communiquer avec django-channel hors websocket en passant par redis

```python
from asgiref.sync import async_to_sync

class ChatConsumer(WebsocketConsumer):

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

    def disconnect(self, close_code):
        async_to_sync(self.channel_layer.group_discard)("chat", self.channel_name)

    def receive(self, text_data):
        async_to_sync(self.channel_layer.group_send)(
            "chat",
            {
                "type": "chat.message",
                "text": text_data,
            },
        )

    def chat_message(self, event):
        self.send(text_data=event["text"])
```

CELERY

Gestionnaire de tache asynchrone

- Fonctionne clé en main avec django
- Fonctionne avec Redis

```
@app.task
def coucou(username: str):
    async_to_sync(channel_layer.group_send)(
        f"{username}",
        {"type": "coucou.message", "message": "coucou"},
    )
```

```
@router.post("/coucou")
def coucou_url(request):
    coucou.delay(generated_file.id, request.user.username)
    return 'ok'
```

DEMO

MERCI

- https://gitlab.biru.sh/biru/presentation/meetup

Questions ?

Speaker notes