Django Channels

# Installation

python -m pip install -U channels

$ python3 -c 'import channels; print(channels.\_\_version\_\_)'

INSTALLED\_APPS = (

'django.contrib.auth',

'django.contrib.contenttypes',

'django.contrib.sessions',

'django.contrib.sites',

...

'channels',

)

# Basic Setup

**index view** - type the name of a chat room to join.

**room view** - see messages posted in a particular chat room.

The room view will use a WebSocket to communicate with the Django server and listen for any messages that are posted.

**Creating a project**

$ django-admin startproject mysite

**Creating the Chat app**

$ python3 manage.py startapp chat

**Add the index view**

We will now create the first view, an index view that lets you type the name of a chat room to join.

Create a templates directory in your chat directory. Within the templates directory you have just created, create another directory called chat, and within that create a file called index.html to hold the template for the index view.

<!-- chat/templates/chat/index.html -->

<!DOCTYPE html>

<html>

<head>

<meta charset="utf-8"/>

<title>Chat Rooms</title>

</head>

<body>

What chat room would you like to enter?<br>

<input id="room-name-input" type="text" size="100"><br>

<input id="room-name-submit" type="button" value="Enter">

<script>

document.querySelector('#room-name-input').focus();

document.querySelector('#room-name-input').onkeyup = function(e) {

if (e.keyCode === 13) { // enter, return

document.querySelector('#room-name-submit').click();

}

};

document.querySelector('#room-name-submit').onclick = function(e) {

var roomName = document.querySelector('#room-name-input').value;

window.location.pathname = '/chat/' + roomName + '/';

};

</script>

</body>

</html>

**# chat/views.py**

from django.shortcuts import render

def index(request):

return render(request, 'chat/index.html')

**# chat/urls.py**

from django.urls import path

from . import views

urlpatterns = [

path('', views.index, name='index'),

]

**# mysite/urls.py**

from django.conf.urls import include

from django.urls import path

from django.contrib import admin

urlpatterns = [

path('chat/', include('chat.urls')),

path('admin/', admin.site.urls),

]

Let’s verify that the index view works. Run the following command:

$ python3 manage.py runserver

# Integrate the Channels library

A Channel routing configuration is similar to a Django URLconf in that it tells Channels what code to run when an HTTP request is received by the Channels server.

We’ll start with an empty routing configuration. Create a file mysite/routing.py and include the following code:

**# mysite/routing.py**

from channels.routing import ProtocolTypeRouter

application = ProtocolTypeRouter({

# (http->django views is added by default)

})

Now add the Channels library to the list of installed apps.

**# mysite/settings.py**

INSTALLED\_APPS = [

'channels',

]

You’ll also need to point Channels at the root routing configuration. Edit the mysite/settings.py file again and add the following to the bottom of it:

**# mysite/settings.py**

**# Channels**

ASGI\_APPLICATION = 'mysite.routing.application'

With Channels now in the installed apps, it will take control of the runserver command, replacing the standard Django development server with the Channels development server.

Let’s ensure that the Channels development server is working correctly. Run the following command:

$ python3 manage.py runserver

# Implement a Chat Server

Add the room view

We will now create the second view, a room view that lets you see messages posted in a particular chat room.

<!-- chat/templates/chat/room.html -->

<html>

<head>

<meta charset="utf-8"/>

<title>Chat Room</title>

</head>

<body>

<textarea id="chat-log" cols="100" rows="20"></textarea><br>

<input id="chat-message-input" type="text" size="100"><br>

<input id="chat-message-submit" type="button" value="Send">

{{ room\_name|json\_script:"room-name" }}

<script>

const roomName = JSON.parse(document.getElementById('room-name').textContent);

const chatSocket = new WebSocket(

'ws://'

+ window.location.host

+ '/ws/chat/'

+ roomName

+ '/'

);

chatSocket.onmessage = function(e) {

const data = JSON.parse(e.data);

document.querySelector('#chat-log').value += (data.message + '\n');

};

chatSocket.onclose = function(e) {

console.error('Chat socket closed unexpectedly');

};

document.querySelector('#chat-message-input').focus();

document.querySelector('#chat-message-input').onkeyup = function(e) {

if (e.keyCode === 13) { // enter, return

document.querySelector('#chat-message-submit').click();

}

};

document.querySelector('#chat-message-submit').onclick = function(e) {

const messageInputDom = document.querySelector('#chat-message-input');

const message = messageInputDom.value;

chatSocket.send(JSON.stringify({

'message': message

}));

messageInputDom.value = '';

};

</script>

</body>

</html>

**# chat/views.py**

from django.shortcuts import render

def index(request):

return render(request, 'chat/index.html', {})

def room(request, room\_name):

return render(request, 'chat/room.html', {

'room\_name': room\_name

})

**# chat/urls.py**

from django.urls import path

from . import views

urlpatterns = [

path('', views.index, name='index'),

path('<str:room\_name>/', views.room, name='room'),

]

$ python3 manage.py runserver

*WebSocket connection to 'ws://127.0.0.1:8000/ws/chat/lobby/' failed: Unexpected response code: 500*

Write your first consumer

When Django accepts an HTTP request, it consults the root URLconf to lookup a view function, and then calls the view function to handle the request.

Similarly, when Channels accepts a WebSocket connection, it consults the root routing configuration to lookup a consumer, and then calls various functions on the consumer to handle events from the connection.

We will write a basic consumer that accepts WebSocket connections on the path /ws/chat/ROOM\_NAME/ that takes any message it receives on the WebSocket and echos it back to the same WebSocket.

**# chat/consumers.py**

import json

from channels.generic.websocket import WebsocketConsumer

class ChatConsumer(WebsocketConsumer):

def connect(self):

self.accept()

def disconnect(self, close\_code):

pass

def receive(self, text\_data):

text\_data\_json = json.loads(text\_data)

message = text\_data\_json['message']

self.send(text\_data=json.dumps({

'message': message

}))

We need to create a routing configuration for the chat app that has a route to the consumer. Create a new file chat/routing.py.

**# chat/routing.py**

from django.urls import re\_path

from . import consumers

websocket\_urlpatterns = [

re\_path(r'ws/chat/(?P<room\_name>\w+)/$', consumers.ChatConsumer),

]

**# mysite/routing.py**

from channels.auth import AuthMiddlewareStack

from channels.routing import ProtocolTypeRouter, URLRouter

import chat.routing

application = ProtocolTypeRouter({

# (http->django views is added by default)

'websocket': AuthMiddlewareStack(

URLRouter(

chat.routing.websocket\_urlpatterns

)

),

})

# Enable a channel layer

A channel layer is a kind of communication system. It allows multiple consumer instances to talk with each other, and with other parts of Django.

A channel layer provides the following abstractions:

A channel is a mailbox where messages can be sent to. Each channel has a name. Anyone who has the name of a channel can send a message to the channel.

A group is a group of related channels. A group has a name. Anyone who has the name of a group can add/remove a channel to the group by name and send a message to all channels in the group. It is not possible to enumerate what channels are in a particular group.

Every consumer instance has an automatically generated unique channel name, and so can be communicated with via a channel layer.

In our chat application we want to have multiple instances of ChatConsumer in the same room communicate with each other. To do that we will have each ChatConsumer add its channel to a group whose name is based on the room name. That will allow ChatConsumers to transmit messages to all other ChatConsumers in the same room.

We will use a channel layer that uses Redis as its backing store. To start a Redis server on port 6379, run the following command:

$ python3 -m pip install channels\_redis

Before we can use a channel layer, we must configure it.

# mysite/settings.py

# Channels

ASGI\_APPLICATION = 'mysite.routing.application'

CHANNEL\_LAYERS = {

'default': {

'BACKEND': 'channels\_redis.core.RedisChannelLayer',

'CONFIG': {

"hosts": [('127.0.0.1', 6379)],

},

},

}

$ python3 manage.py shell

>>> import channels.layers

>>> channel\_layer = channels.layers.get\_channel\_layer()

>>> from asgiref.sync import async\_to\_sync

>>> async\_to\_sync(channel\_layer.send)('test\_channel', {'type': 'hello'})

>>> async\_to\_sync(channel\_layer.receive)('test\_channel')

{'type': 'hello'}

Type Control-D to exit the Django shell.

**# chat/consumers.py**

import json

from asgiref.sync import async\_to\_sync

from channels.generic.websocket import WebsocketConsumer

class ChatConsumer(WebsocketConsumer):

def connect(self):

self.room\_name = self.scope['url\_route']['kwargs']['room\_name']

self.room\_group\_name = 'chat\_%s' % self.room\_name

# Join room group

async\_to\_sync(self.channel\_layer.group\_add)(

self.room\_group\_name,

self.channel\_name

)

self.accept()

def disconnect(self, close\_code):

# Leave room group

async\_to\_sync(self.channel\_layer.group\_discard)(

self.room\_group\_name,

self.channel\_name

)

# Receive message from WebSocket

def receive(self, text\_data):

text\_data\_json = json.loads(text\_data)

message = text\_data\_json['message']

# Send message to room group

async\_to\_sync(self.channel\_layer.group\_send)(

self.room\_group\_name,

{

'type': 'chat\_message',

'message': message

}

)

# Receive message from room group

def chat\_message(self, event):

message = event['message']

# Send message to WebSocket

self.send(text\_data=json.dumps({

'message': message

}))

When a user posts a message, a JavaScript function will transmit the message over WebSocket to a ChatConsumer. The ChatConsumer will receive that message and forward it to the group corresponding to the room name. Every ChatConsumer in the same group (and thus in the same room) will then receive the message from the group and forward it over WebSocket back to JavaScript, where it will be appended to the chat log.

# Rewrite Chat Server as Asynchronous

Asynchronous consumers can provide a higher level of performance since they don’t need to create additional threads when handling requests.

# chat/consumers.py

import json

from channels.generic.websocket import AsyncWebsocketConsumer

class ChatConsumer(AsyncWebsocketConsumer):

async def connect(self):

self.room\_name = self.scope['url\_route']['kwargs']['room\_name']

self.room\_group\_name = 'chat\_%s' % self.room\_name

# Join room group

await self.channel\_layer.group\_add(

self.room\_group\_name,

self.channel\_name

)

await self.accept()

# Leave room group

async def disconnect(self, close\_code):

await self.channel\_layer.group\_discard(

self.room\_group\_name,

self.channel\_name

)

# Receive message from WebSocket

async def receive(self, text\_data):

text\_data\_json = json.loads(text\_data)

message = text\_data\_json['message']

# Send message to room group

await self.channel\_layer.group\_send(

self.room\_group\_name,

{

'type': 'chat\_message',

'message': message

}

)

# Receive message from room group

async def chat\_message(self, event):

message = event['message']

# Send message to WebSocket

await self.send(text\_data=json.dumps({

'message': message

}))