**Project configures**

This guide walks through configuring Django to support WebSockets using Daphne, Channels, and Redis.

**Daphne** - Daphne is an HTTP, HTTP2, and WebSocket protocol server for ASGI and Channels.

**Channels** - Extends Django to handle WebSocket connections and asynchronous protocols.

**Redis** - Used as the message broker for Channels.

pip install django==2.2.5 daphne channels channels-redis

**Add Required Dependencies to INSTALLED\_APPS**

INSTALLED\_APPS = [

'daphne',

'chat',

'echo'

]

**Add this line in settings.py file –**

CHANNEL\_LAYERS = {

'default': {

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

'CONFIG': {

"hosts": [('localhost', 6379)], # Ensure Redis is running on this port

},

},

}

This configuration ensures that Django Channels uses Redis to handle communication between different WebSocket connections.

**The asgi.py file setting -**

The asgi.py file is responsible for configuring the ASGI (Asynchronous Server Gateway Interface) application in Django. It plays a key role in enabling asynchronous communication, such as WebSocket connections, in Django projects. Specifically, it tells Django how to handle HTTP and WebSocket requests.

**Routing.py setup –**

The routing.py file in a Django project is responsible for defining the routing for WebSocket connections. Just like the urls.py file maps HTTP URLs to views, routing.py maps WebSocket URLs to consumers, which handle WebSocket connections, messages, and disconnections.

Code –

websocket\_urlpatterns = [

path('ws/chat/<str:room\_name>/', consumers.ChatConsumer),

]

path('ws/chat/<str:room\_name>/'): This pattern defines a WebSocket URL that includes a dynamic room\_name parameter.

**Consumer.py –**

The consumers.py file in Django Channels is where you define your **WebSocket consumers**. A consumer is a class-based handler that manages WebSocket connections, receives messages, and sends messages back to the client. It's similar to Django views, but instead of handling HTTP requests, it handles WebSocket events such as connections, messages, and disconnections.

**websocket\_connect(self, event)**: This method is called when a client connects to the WebSocket.

**websocket\_disconnect(self, event)**: This method is called when the WebSocket connection is disconnected.

**websocket\_receive(self, event)**: This method is called when the server receives a message from the WebSocket connection.

**chat\_message(self, event)**: This method handles incoming messages that were broadcasted to the chat group.

**chat\_activity(self, event):** This method is another handler for WebSocket events, but it is specifically used for broadcasting non-message activity, like notifications or typing indicators.

**get\_chat(self):** This method asynchronously retrieves the chat room from the database using self.chat\_id. If the chat does not exist, it returns None.

**create\_message(self, text)**: This method creates a new message in the Message model, associating it with the chat room and the user who sent it.