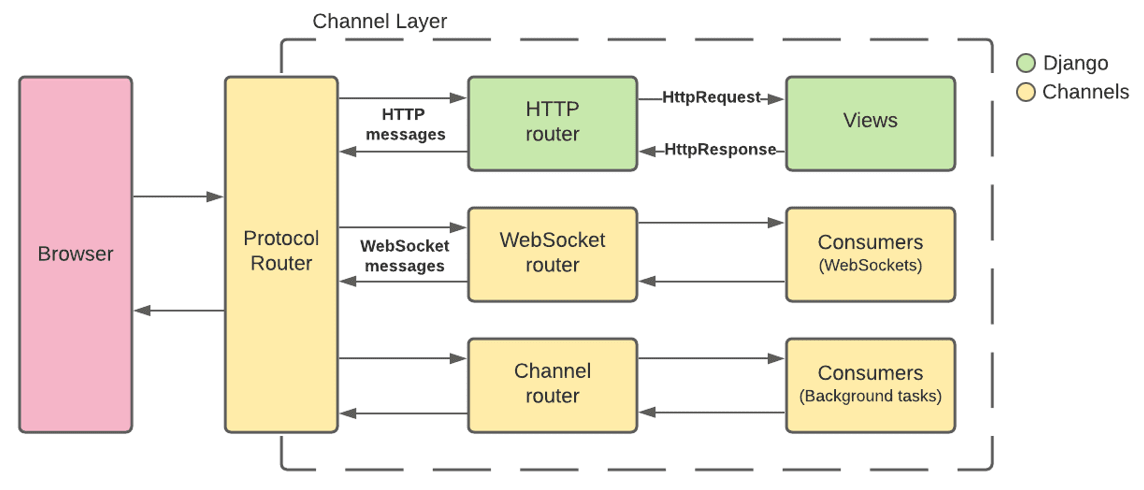
Channels

Channels – give capability to Django to not only work with http request but supports protocols that require long running connections. Like websocket, mqtt(iot), chatbots, radios and other real time applications.

Also it provides authentication & sessions.



Basically Django database needs to be accessed with synchronous code but if we add channels to Django we need to convert sync code into async code.

What is sync and what is async?

Sync - Think of it like **standing in a queue** at a food stall.  
You **wait your turn with your partner and cant spend time**, and only after you're served, the next person is served and you didn’t get time to chat.

Async - You place your order and **continue doing spending time with your partner** — no need to wait in line for order. You two task is working parallely.

1. sync\_to\_async - takes a sync function and returns an async function that wraps it
2. async\_to\_sync - takes an async function and returns a sync function

* Introduction to channel
* Basic Steps to implement
* Install dependencies
* Setup in settings.py
* asgi.py + rounting.py + consumer.py
* trigger them via views.py
* create frontend

[**https://testdriven.io/blog/django-channels/**](https://testdriven.io/blog/django-channels/) **go through this tutorial for guideness.**

**Basic Steps to implement**

**Install dependencies**

pip install channels==3.0.4

docker run -p 6379:6379 -d redis:5

pip install channels\_redis==3.3.1

**Setup in settings.py**

**-- add app & channels to installed app**

INSTALLED\_APPS = [

'django.contrib.admin',

'django.contrib.auth',

'django.contrib.contenttypes',

'django.contrib.sessions',

'django.contrib.messages',

'django.contrib.staticfiles',

'chat.apps.ChatConfig', *# new*

'channels', *# new*

]

**-- convert wsgi->asgi**

WSGI\_APPLICATION = 'core.wsgi.application'

ASGI\_APPLICATION = 'core.asgi.application' *# new*

**-- add configuration for redis**

CHANNEL\_LAYERS = {

'default': {

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

'CONFIG': {

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

},

},

}

**asgi.py + rounting.py + consumer.py**

**asgi.py**

Since we'll be using WebSockets instead of HTTP to communicate from the client to the server, we need to wrap our ASGI config with [ProtocolTypeRouter](https://channels.readthedocs.io/en/latest/topics/routing.html#protocoltyperouter) in *core/asgi.py*:

**import** **os**

**from** **channels.routing** **import** ProtocolTypeRouter

**from** **django.core.asgi** **import** get\_asgi\_application

os.environ.setdefault('DJANGO\_SETTINGS\_MODULE', 'core.settings')

application = ProtocolTypeRouter({

'http': get\_asgi\_application(),

})

**consumer.py**

A [consumer](https://channels.readthedocs.io/en/stable/topics/consumers.html) is the basic unit of Channels code. They are tiny ASGI applications, driven by events.

consumers are long-running by default. A Django project can have multiple consumers that are combined using Channels routing.

Each consumer has it's own scope, which is a set of details about a single incoming connection. They contain pieces of data like protocol type, path, headers, routing arguments, user agent, and more.

Create a new file called *consumers.py* inside "chat":

we will create a ChatConsumer, which inherits from [WebsocketConsumer](https://channels.readthedocs.io/en/latest/topics/consumers.html#websocketconsumer). WebsocketConsumer provides three methods, connect(), disconnect(), and receive():

1. Inside connect() we called accept() in order to accept the connection. After that, we added the user to the channel layer group.
2. Inside disconnect() we removed the user from the channel layer group.
3. Inside receive() we parsed the data to JSON and extracted the message. Then, we forwarded the message using group\_send to chat\_message.

Since WebsocketConsumer is a synchronous consumer, we had to call async\_to\_sync when working with the channel layer

**rounting.py**

Channels provides different [routing](https://channels.readthedocs.io/en/stable/topics/routing.html) classes which allow us to combine and stack consumers

**from** **django.urls** **import** re\_path

**from** **.** **import** consumers

websocket\_urlpatterns = [

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

]

--Register the *routing.py* file inside *core/asgi.py*:

**import** **os**

**from** **channels.routing** **import** ProtocolTypeRouter, URLRouter

**from** **django.core.asgi** **import** get\_asgi\_application

**import** **chat.routing**

os.environ.setdefault('DJANGO\_SETTINGS\_MODULE', 'core.settings')

application = ProtocolTypeRouter({

'http': get\_asgi\_application(),

'websocket': URLRouter(

chat.routing.websocket\_urlpatterns

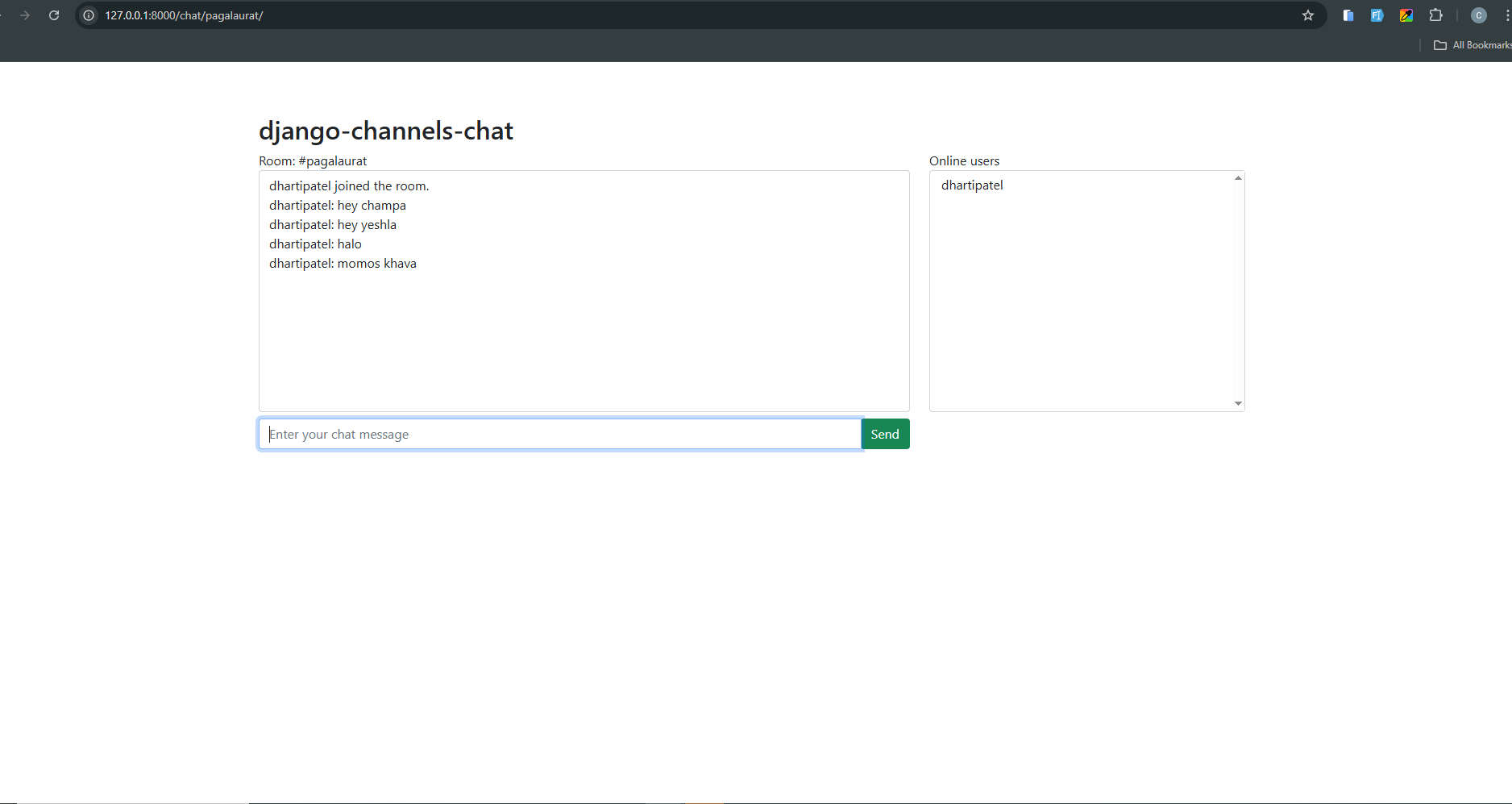
),

})

**create frontend**

To communicate with Channels from the frontend, we'll use the [WebSocket API](https://developer.mozilla.org/en-US/docs/Web/API/WebSocket).

1. onopen - called when a WebSocket connection is established
2. onclose - called when a WebSocket connection is destroyed
3. onmessage - called when a WebSocket receives a message
4. onerror - called when a WebSocket encounters an error



**Note - :**

Channels comes with a built-in class for Django session and [authentication](https://channels.readthedocs.io/en/latest/topics/authentication.html) management called AuthMiddlewareStack.