Django Channels – Introduction and Basic Setup

Last Updated : 24 Sep, 2024

Django is a powerful Python framework for web development. It is fast, secure, and reliable. Channels allow Django projects to handle HTTP along with asynchronous protocols like WebSockets, MQTT, chatbots, and more.

Channels:

Channels preserve the synchronous behavior of Django and add a layer of asynchronous protocols allowing users to write the views that are entirely synchronous, asynchronous, or a mixture of both. Channels basically allow the application to support "long-running connections". It replaces Django's default **WSGI** with its **ASGI** .

Django Channels allow for handling WebSockets, and understanding this can elevate your web development skills. The **Complete Django Web Development Course – Basics to Advance** provides a deeper exploration of Django Channels and other advanced topics."

ASGI:

ASGI (Asynchronous Server Gateway Interface) provides an interface between async Python web servers and applications while it supports all the features provided by WSGI.

Consumers:

A **consumer** is a basic unit of Channels. It is an event-driven class that supports both async and sync applications. Consumers can run longer and hence they support web sockets that need persistent connection.

In this post, we will set up a basic example of channels. We will build a calculator app that will allow the user to send multiple expressions to the server and receive the result through a single persistent connection.

Environment Setup:

- It is always a good idea to create a virtual environment for the python apps in order to avoid version conflicts. Run the following commands in the terminal to get started

```
easy-install pip
python3 -m pip install virtualenv
virtualenv venv
source venv/bin/activate
```

- Now install **Django** and **Channels** :


```
pip install django
pip install channels
# On windows, try an unofficial wheel of 'Twisted' in case of dependency
errors
```

LiveCalculator App:

Now start a Django project and create an app named **'liveCalculator'**

```
django-admin startproject sampleProject
cd sampleProject
python3 manage.py startapp liveCalculator
```

In **sampleProject/settings.py** , register **channels** and **liveCalculator** .

settings.py:

```
INSTALLED_APPS = [
   'channels',
   'liveCalculator',
   'django.contrib.admin',
```

```
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
]
```

In sampleProject/asgi.py, add the http protocol.

asgi.py:

Python

```python
import os

import django
from channels.http import AsgiHandler
from channels.routing import ProtocolTypeRouter

os.environ.setdefault('DJANGO_SETTINGS_MODULE', 'sampleProject.settings')
django.setup()

application = ProtocolTypeRouter({
  "http": AsgiHandler(),
  # Just HTTP for now. (We can add other protocols later.)
})
```

Now we need to register this asgi into our application. Add this line in sampleProject/settings.py :

```
ASGI_APPLICATION = "sampleProject.asgi.application"
```

Create a new folder **liveCalculator/templates/liveCalculator** and create a new file **index.html** inside it. It will be the starting page of our app. Add the following code in **index.html:**

index.html:

HTML

```html
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Live Calculator</title>
</head>

<body>
    <textarea name="ta" id="results" cols="30" rows="10">

    </textarea><br>
    Enter the expression: <input type="text" id="exp">
    <input type="button" id="submit" value="Get Results">
    <script>
        const socket = new WebSocket('ws://localhost:8000/ws/livec/');
        socket.onmessage = (e) => {
            result = JSON.parse(e.data).result;
            document.getElementById("results").value += "Server: " + result + "\n";
        }

        socket.onclose = (e) => {
            console.log("Socket closed!");
        }

        document.querySelector('#exp').onkeyup = function (e) {
            if (e.keyCode === 13) {  // enter, return
                document.querySelector('#submit').click();
            }
        };

        document.querySelector("#submit").onclick = (e) => {
            inputfield = document.querySelector("#exp")
            exp = inputfield.value
            socket.send(JSON.stringify(
                {
                    expression: exp
```

```
                    }
            ))
            document.querySelector("#results").value += "You: " + exp + "\n";
            inputfield.value = "";
        }

    </script>
</body>

</html>
```

The above code will render a text area and an input box where the user can enter the expression. It will create a socket connection that we will make later and append the received result in the text area. When the user inputs the expression, it will send the expression through a socket connection.

Now create a view to render this page in **liveCalculator/views.py** :

liveCalculator/views.py:

Python

```python
from django.shortcuts import render

# Create your views here.

def index(request):
    return render(request, 'liveCalculator/index.html', {})
```

Next, we need to create a route for this view. Add a new file **urls.py** in **liveCalculator** directory and add the following code:

liveCalculator/urls.py:

Python

```python
from django.conf.urls import url
from . import views

urlpatterns = [
```

```
    url(r'^$', views.index, name="index"),
]
```

Register this route in **sampleProject/urls.py** :

sampleProject/urls.py:

Python

```python
from django.contrib import admin
from django.urls import path
from django.conf.urls import include, url
urlpatterns = [
    path('admin/', admin.site.urls),
    url(r'^', include('liveCalculator.urls'))
]
```

Now we need to create the **consumer** for our web socket connection. We will use the generic **WebsocketConsumer** class to implement its event-driven methods. Create a new file **consumers.py** in **liveCalculator** folder and add the following code:

consumers.py:

Python

```python
import json
from channels.generic.websocket import WebsocketConsumer

class Calculator(WebsocketConsumer):
    def connect(self):
        self.accept()

    def disconnect(self, close_code):
        self.close()

    def receive(self, text_data):
        text_data_json = json.loads(text_data)
        expression = text_data_json['expression']
        try:
```

```
                result = eval(expression)
            except Exception as e:
                result = "Invalid Expression"
            self.send(text_data=json.dumps({
                'result': result
            }))
```

The **WebsocketConsumer** class supports these user-defined methods:

- **connect()** : We can write the business logic of what should happen when the client sends a connection request.
- **disconnect()** : We can write the business logic of what should happen when the client sends a disconnection request.
- **receive():** We can write the business logic of what should happen when the client sends a message.

It also supports these built-in methods:

- **accept():** It will accept the incoming connection.
- **close():** It will close the current connection.
- **send():** It will send the specified message to the client.

We have simply used the above methods in our **Calculator** class to accept the connection, evaluate the expression when a message a received, and send it to the client.

Next, we also need to define the routing method for this consumer. Create a new file **routing.py** in the same folder and add the following code to it:

routing.py:

Python

```python
from django.urls import re_path

from . import consumers

websocket_urlpatterns = [
    re_path(r'ws/livec/$', consumers.Calculator.as_asgi()),
```

```
]
```

Note that we have used **as_asgi()** method on our Calculator class to use it for our application. This will enable the socket on **ws://<IP:Port>/ws/livec** . Now register **routing.py** into **asgi.py** by declaring the **WebSocket** protocol.

asgi.py:

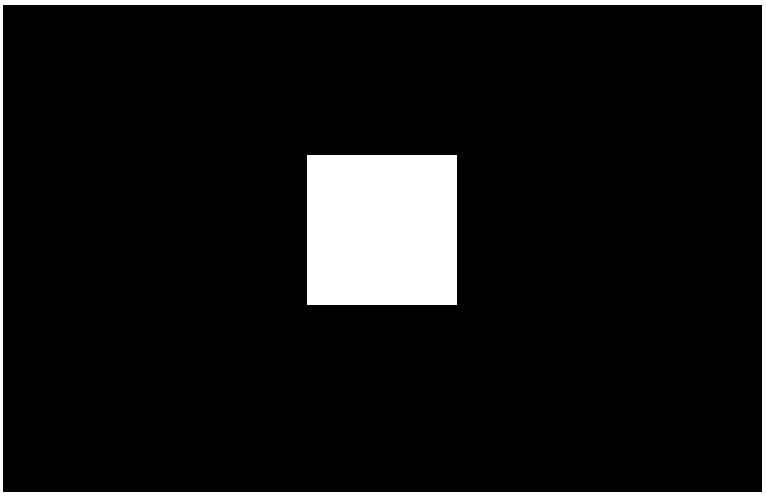
Python

```python
import os

from channels.auth import AuthMiddlewareStack
from channels.routing import ProtocolTypeRouter, URLRouter
from django.core.asgi import get_asgi_application
import liveCalculator.routing

os.environ.setdefault("DJANGO_SETTINGS_MODULE", "sampleProject.settings")

application = ProtocolTypeRouter({
  "http": get_asgi_application(),
  "websocket": AuthMiddlewareStack(
        URLRouter(
            liveCalculator.routing.websocket_urlpatterns
        )
    ),
})
```

We are almost done with our first Channels application. Save all the files and run the following commands in the terminal:

```
python3 manage.py makemigrations
python3 manage.py migrate
python3 manage.py runserver
```

Now open http://localhost:8000 on your browser, and you should see the output like this:



00:00 00:17

See the log of [redacted] ection only once, and we can se [redacted] a new connection.

Are you ready to elevate your web development skills from foundational knowledge to advanced expertise? Explore our Mastering Django Framework - Beginner to Advanced Course on GeeksforGeeks, designed for aspiring developers and experienced programmers. This comprehensive course covers everything you need to know about Django, from the basics to advanced features. Gain practical experience through **hands-on projects** and real-world applications, mastering essential Django principles and techniques. Whether you're just starting or looking to refine your skills, this course will empower you to build sophisticated web applications efficiently. Ready to enhance your web development journey? Enroll now and unlock your potential with Django!

Next Article

Django Introduction | Set 2 (Creating a Project)

Similar Reads

Token Authentication in Django Channels and Websockets

Prerequisites: Django, WebSockets, Django channels, Token authentication The most popular Django topics right now are WebSockets and Django channels...

13 min read

Django Installation and Setup

Installing and setting up Django is a straightforward process. Below are the step-by-step instructions to install Django and set up a new Django project on your...

2 min read

Django+React Full Stack Development Setup using Dact

When we work on a project having Django as our Back-end and having a powerful front-end using React, the development setup takes a significant...

2 min read

Splitting and Merging Channels with Python-OpenCV

In this article, we will learn how to split a multi-channel image into separate channels and combine those separate channels into a multi-channel image usin...

2 min read

How to Find Mean Across the Image Channels in PyTorch?

In this article, we are going to see how to find mean across the image channels in PyTorch. We have to compute the mean of an image across the channels Red,...

2 min read

Django Basic App Model - Makemigrations and Migrate

In this article, we will create a basic model of an app and also learn about what are migrations in Django and migrate in Django also develop some basic...

5 min read

How to Override and Extend Basic Django Admin Templates?

The Django admin interface provides a robust way to manage our application's data, offering a user-friendly and efficient design. By default, it gives a clean and...

10 min read

How to Create a Basic Project using MVT in Django ?

Prerequisite - Django Project MVT Structure Assuming you have gone through the previous article. This article focuses on creating a basic project to render a...

2 min read

How to Create a basic API using Django Rest Framework ?

Django REST Framework is a wrapper over the default Django Framework, basically used to create APIs of various kinds. There are three stages before...

4 min read

Django project - Creating a Basic E-commerce Website for Displaying...

Project Title - Basic Ecommerce Website using Django Django is a powerful framework based on python. Here we will see how to create a basic e-commerc...

3 min read

Article Tags : Django Python Python Django Socket-programming

Practice Tags : python