Trabalhando com aplicações em tempo real

Raíssa Azevedo

Django Framework

Raíssa Azevedo

Como o Django se comporta em aplicações em Tempo Real ?

Entendendo as aplicações Realtime:

Uma aplicação realtime é a que há o compartilhamento de dados de uma ponta a outra a distância, e o conceito é: “O dado sai de um ponto estará disponível no outro ponto, no menor tempo possível”.

O PROJETO “**realtime”**:

Esse projeto vai contar com o banco de dados Redis.

```
pip3 install django channels django-bootstrap4 channels-redis
pip3 freeze > requirements.txt
```

Porque não usar o Mysql ou o Postgre?

Porque precisa ser um sistema database que suporte atualização de dados em tempo real.

Em seguida, criar o projeto e a aplicação:

```
django-admin startproject realtime .
django-admin startapp chat
```

Fazer todas as configurações necessárias no settings.py.

```
INSTALLED_APPS = [
    'bootstrap4',
    'channels',
    'chat',
```

Por padrão o Django faz uso de uma aplicação WSGI.

Quando é necessário utilizar o realtime o WSGI não suporta. Portanto, é necessário criar uma aplicação ASGI:

```
# Configuração específica do Channels
ASGI_APPLICATION = 'realtime.routing.application'
```

Além disso, é necessário configurar o Channels.

```
CHANNEL_LAYERS = {
    'default': {
        'BACKEND': 'channels_redis.core.RedisChannelLayer',
        'CONFIG': {
            'hosts': [('127.0.0.1', 6379)]
        },
    },
}
```

6379 - É a porta do Redis.

Channels Layers – É preciso configurar o Channels para entender que é o Redis que vai comandar o fluxo de dados.

Configurando as Rotas e preparando o WebSocket:

Websocket é um interação entre o navegador e a sua aplicação para haver um comunicação realtime. Os navegadores atuais, possuem suporte ao Websocket.

Portanto, é possível utilizar esse recurso para criar um interação em tempo real com a aplicação.

Criar o arquivo **urls.py** na aplicação chat.

```
from django.urls import path

from .views import IndexView, SalaView


urlpatterns = [
    path('', IndexView.as_view(), name='index'),
    path('chat/<str:nome_sala>/', SalaView.as_view(), name='sala'),
]
```

Criar um arquivo **routing.py** na aplicação chat:

```
from django.urls import re_path

from .consumers import ChatConsumer

websocket_urlpatterns = [
    re_path(r'ws/chat/(?P<nome_sala>\w+)/$', ChatConsumer.as_asgi()),
]
```

Essa é uma rota específica do Channels.

A rota é iniciada com **ws** que significa WebSocket. O ChatConsumer é quem irá fazer a ponte entre o navegador e aplicação.

No arquivo de urls do Projeto:

```
from django.urls import path, include

urlpatterns = [
    path('', include('chat.urls')),
    path('admin/', admin.site.urls),
]
```

Criar um arquivo **routing.py** dentro da do diretório do Preojeto realtime:

```
from channels.routing import ProtocolTypeRouter, URLRouter
from channels.auth import AuthMiddlewareStack

from chat.routing import websocket_urlpatterns

application = ProtocolTypeRouter({
    'websocket': AuthMiddlewareStack(
        URLRouter(
            websocket_urlpatterns
        )
    ),
})
```

Trabalhando nas Views:

Na views do aplicação chat:

```
from django.views.generic import TemplateView
from django.utils.safestring import mark_safe
import json


class IndexView(TemplateView):
    template_name = 'index.html'


class SalaView(TemplateView):
    template_name = 'sala.html'

    def get_context_data(self, **kwargs):
        context = super(SalaView, self).get_context_data(**kwargs)
        context['nome_sala_json'] = mark_safe(
            json.dumps(self.kwargs['nome_sala'])
        )
        return context
```

mark_safe – Do pacote SafeString é um função pra remover qualquer dado inseguro fornecido pelo usuário do chat.

Definindo os Templates:

No **Index.html**:

```
{% load bootstrap4 %}
<!DOCTYPE html>
<html>
<head>
   <meta charset="utf-8" />
   <link rel="shortcut icon" type="image/x-icon" href="/static/img/favicon.ico">
   <title>Rai Chat</title>
   {% bootstrap_css %}
</head>
<body>
 <div clas="container">
    Qual sala de chat você gostaria de entrar?<br/>
       <input id="nome_sala" name="nome_sala" type="text" size="100" placeholder="Nome da sala..."><br/>
       {% buttons %}
          <input id="botao" class="btn btn-primary" type="button" value="Entrar" />
       {% endbuttons %}
 </div>
 <script>
    document.querySelector('#nome_sala').focus();
    document.querySelector('#nome_sala').onkeyup = function(e){
       if(e.keyCode === 13){
          document.querySelector('#botao').click();
       }
    };

    document.querySelector('#botao').onclick = function(e){
       var nome_sala = document.querySelector('#nome_sala').value;
       if(nome_sala != ""){
          window.location.pathname = '/chat/' + nome_sala + '/';
       }else{
          alert('Você precisa informar o nome da sala.');
          document.querySelector('#nome_sala').focus();
       }
    };

 </script>
{% bootstrap_javascript jquery='full' %}
</body>
</html>
```

No **sala.html**:

```
{% load bootstrap4 %}
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8" />
    <link rel="shortcut icon" type="image/x-icon" href="/static/img/favicon.ico">
    <title>Rai Chat</title>
    {% bootstrap_css %}
</head>
<body>
    <div class="container">
        <textarea id="sala" cols="70" rows="15"></textarea><br/>
        <input id="texto" type="text" size="50"/><br/>
        {% buttons %}
            <input id="botao" type="button" value="Enviar" />
        {% endbuttons %}
    </div>

{% bootstrap_javascript jquery='full' %}
<script>
    var nome_sala = {{ nome_sala_json }};

    var chatSocket = new WebSocket(
        'ws://' + window.location.host +
        '/ws/chat/' + nome_sala + '/');

    chatSocket.onmessage = function(e){
        var dados = JSON.parse(e.data);
        var mensagem = dados['mensagem'];
        document.querySelector('#sala').value += (mensagem + '\n');
    };

    chatSocket.onclose = function(e){
        console.error('O chat encerrou de forma inesperada.');
    };

    document.querySelector('#texto').focus();
    document.querySelector('#texto').onkeyup = function(e){
        if(e.keyCode === 13){
            document.querySelector('#botao').click();
        }
    };

    document.querySelector('#botao').onclick = function(e){
        var mensagemInput = document.querySelector('#texto');
        var mensagem = mensagemInput.value;
        chatSocket.send(JSON.stringify({
            'mensagem': mensagem
        }));
        mensagemInput.value = '';
    };
</script>
</body>
</html>
```

Criando o Consumer:

O elo de ligação entre a aplicação no navegador e o projeto realtime:

Criar o arquivo **consumers.py** dentro da aplicação chat:

```
from channels.generic.websocket import AsyncWebsocketConsumer
import json


class ChatConsumer(AsyncWebsocketConsumer):

    async def connect(self):
        self.room_name = self.scope['url_route']['kwargs']['nome_sala']
        self.room_group_name = f'chat_{self.room_name}'

        # Entrar na sala
        await self.channel_layer.group_add(
            self.room_group_name,
            self.channel_name
        )

        await self.accept()

    async def disconnect(self, code):
        # Quando o usuário sai da sala
        await self.channel_layer.group_discard(
            self.room_group_name,
            self.channel_name
        )

    # Recebe mensagem do WebSocket
    async def receive(self, text_data=None, bytes_data=None):
        text_data_json = json.loads(text_data)
        mensagem = text_data_json['mensagem']

        # Envia a mensagem para a sala
        await self.channel_layer.group_send(
            self.room_group_name,
            {
                'type': 'chat_message',
                'message': mensagem
            }
        )

    # Recebe a mensagem da sala
    async def chat_message(self, event):
        mensagem = event['message']

        # Envia a mensagem para o WebSocket
        await self.send(text_data=json.dumps({
            'mensagem': mensagem
        }))
```

Quando trabalhamos com programação async é importante avisar que é assíncrona. Ela sendo assíncrona, todas as funções e comandos executados que dependem de algum retorno/finalização. É utilizado o **await**.

Rodando a aplicação em tempo real:

Iniciando o servidor Redis:

Vai até onde o servidor Redis está salvo dentro os diretórios:

```
redis-cli
```

Abra o terminal e digite:

```
pip3 manage.py migrate
```

Se houver erro ao executar o migrate:

```
pip uninstall django-channels
pip uninstall channels
pip install channels
```

Em seguida rodar a aplicação no servidor:

```
pip3 manage.py runserver
```