Realtime chat app using Django

Last Updated : 06 Sep, 2024

Chat Room has been the most basic step toward creating real-time and live projects. The chat page that we will create will be a simple HTML boilerplate with a simple h1 text with the name of the current user and a link to log out to the user who is just logged in. You may need to comment on the line until we create auth system for this. This ensures that when two users are chatting, one can log out and it will not affect the other user. The other will be still logged in and he can send and receive messages.

Prerequisites:

- Django
- Django Migrations
- Django Channel

Introduction to Django Channels and Asynchronous Programming

Channels are the python project which was created to extend the ability of Django to the next level. We were working in standard Django which did not support asynchronous and channels and connection via WebSockets to create real-time applications. Channels extend the ability of Django beyond HTTP and make it work with WebSockets, chat protocols, IoT protocols, and more. It is built on ASGI support which stands for Asynchronous Server Gateway Interface. ASGI is the successor of WSGI which provides an interface between async and python. Channels provide the functionality of ASGI, by extending WSGI to it, and it provides ASGI support with WSGI. Channels also bundle the event-driven architecture with the channel layers, a system that allows you to easily communicate between processes and separate your project into different processes.

Steps for creating the chat application

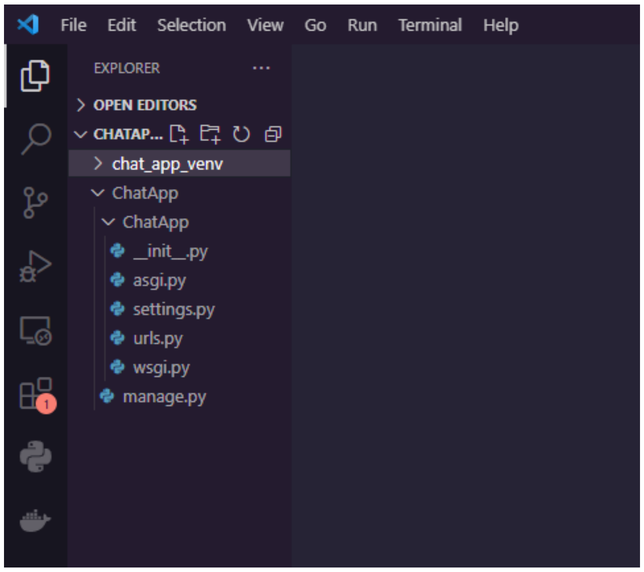
Step 1: Install and setup Django

Step 2: Create your virtual environment.

Step 3: Then create a Django project named **ChatApp.** For creating the project write the command in your terminal.

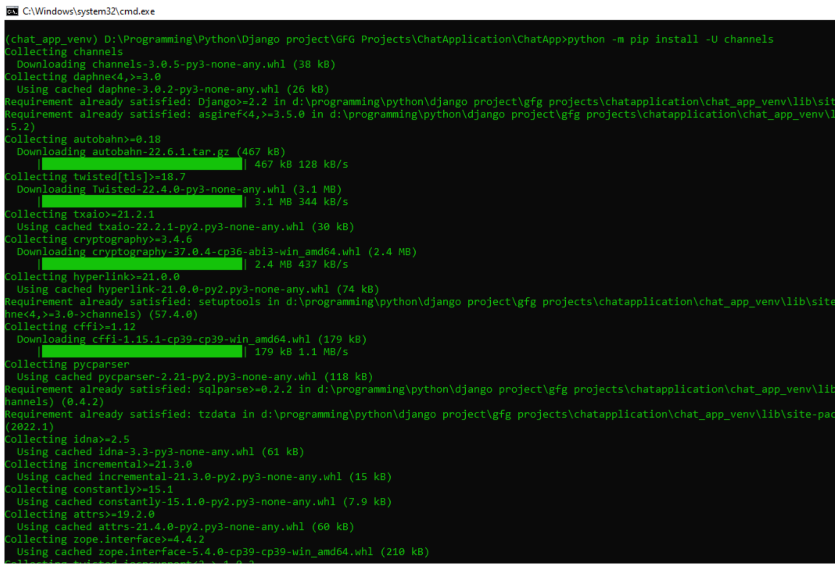
```
django-admin startproject ChatApp
```

Step 4: After the Creation of the Project, the folder structure should look like this, and then open your favorite IDE for working further.



Step 5: Install django-channels for working with the chat app. This will install channels to your environment.

```
python -m pip install -U channels
```



```
C:\Windows\system32\cmd.exe
(chat_app_venv) D:\Programming\Python\Django project\GFG Projects\ChatApplication\ChatApp>python -m pip install -U channels
Collecting channels
  Downloading channels-3.0.5-py3-none-any.whl (38 kB)
Collecting daphne<4,>=3.0
  Using cached daphne-3.0.2-py3-none-any.whl (26 kB)
Requirement already satisfied: Django>=2.2 in d:\programming\python\django project\gfg projects\chatapplication\chat_app_venv\lib\sit
Requirement already satisfied: asgiref<4,>=3.5.0 in d:\programming\python\django project\gfg projects\chatapplication\chat_app_venv\l
.5.2)
Collecting autobahn>=0.18
  Downloading autobahn-22.6.1.tar.gz (467 kB)
     |████████████████████████████████| 467 kB 128 kB/s
Collecting twisted[tls]>=18.7
  Downloading Twisted-22.4.0-py3-none-any.whl (3.1 MB)
     |████████████████████████████████| 3.1 MB 344 kB/s
Collecting txaio>=21.2.1
  Using cached txaio-22.2.1-py2.py3-none-any.whl (30 kB)
Collecting cryptography>=3.4.6
  Downloading cryptography-37.0.4-cp36-abi3-win_amd64.whl (2.4 MB)
     |████████████████████████████████| 2.4 MB 437 kB/s
Collecting hyperlink>=21.0.0
  Using cached hyperlink-21.0.0-py2.py3-none-any.whl (74 kB)
Requirement already satisfied: setuptools in d:\programming\python\django project\gfg projects\chatapplication\chat_app_venv\lib\site
hne<4,>=3.0->channels) (57.4.0)
Collecting cffi>=1.12
  Downloading cffi-1.15.1-cp39-cp39-win_amd64.whl (179 kB)
     |████████████████████████████████| 179 kB 1.1 MB/s
Collecting pycparser
  Using cached pycparser-2.21-py2.py3-none-any.whl (118 kB)
Requirement already satisfied: sqlparse>=0.2.2 in d:\programming\python\django project\gfg projects\chatapplication\chat_app_venv\lib
hannels) (0.4.2)
Requirement already satisfied: tzdata in d:\programming\python\django project\gfg projects\chatapplication\chat_app_venv\lib\site-pac
(2022.1)
Collecting idna>=2.5
  Using cached idna-3.3-py3-none-any.whl (61 kB)
Collecting incremental>=21.3.0
  Using cached incremental-21.3.0-py2.py3-none-any.whl (15 kB)
Collecting constantly>=15.1
  Using cached constantly-15.1.0-py2.py3-none-any.whl (7.9 kB)
Collecting attrs>=19.2.0
  Using cached attrs-21.4.0-py2.py3-none-any.whl (60 kB)
Collecting zope.interface>=4.4.2
  Using cached zope.interface-5.4.0-cp39-cp39-win_amd64.whl (210 kB)
```

Note : Starting from version 4.0.0 of channels, ASGI runserver in development mode does not work anymore. You will have to install daphne as well.

python -m pip install -U daphne

After installing daphne, add daphne to your installed apps.

Python

```python
# code

INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',

    # add django daphne
    'daphne' ,
]
```

Step 6: After installing channels, add channels to your installed apps. This will let Django know that channels had been introduced in the project and we can work further.

Python

```python
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',

    # add django channels
    'channels' ,
]
```

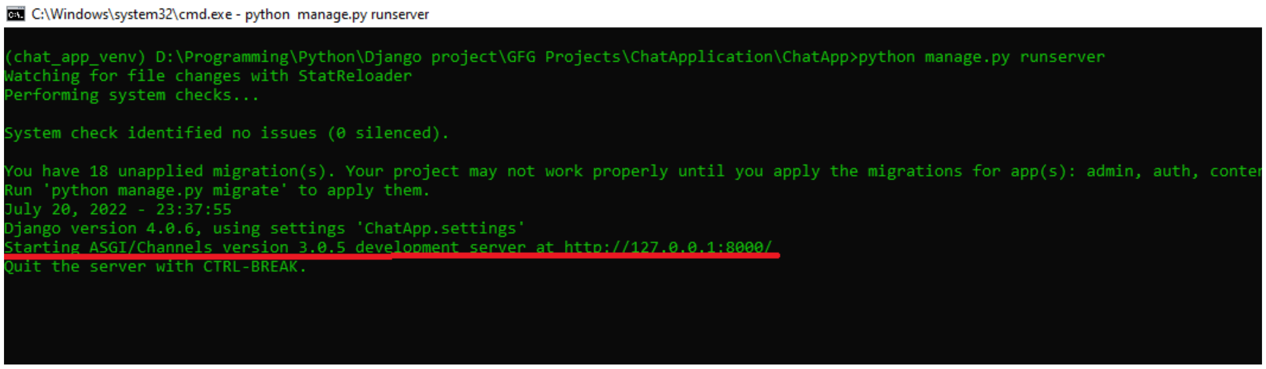
Step 7: Set the ASGI application to your default ASGI file in the project. Now run the server, you will notice that the ASGI server will take place over the Django server and it will support ASGI now And finally, set your ASGI_APPLICATION setting to point to that routing object as your root application:

Python

```python
import os

from channels.routing import ProtocolTypeRouter
from django.core.asgi import get_asgi_application

django_asgi_app = get_asgi_application()

os.environ.setdefault("DJANGO_SETTINGS_MODULE",
"ChatApp.settings")
application = ProtocolTypeRouter({
    "http": django_asgi_app,
    # Just HTTP for now. (We can add other protocols later.)
})

ASGI_APPLICATION = 'ChatApp.asgi.application'
```

To run the server, write the following command in the terminal.

```
python3 manage.py runserver
```

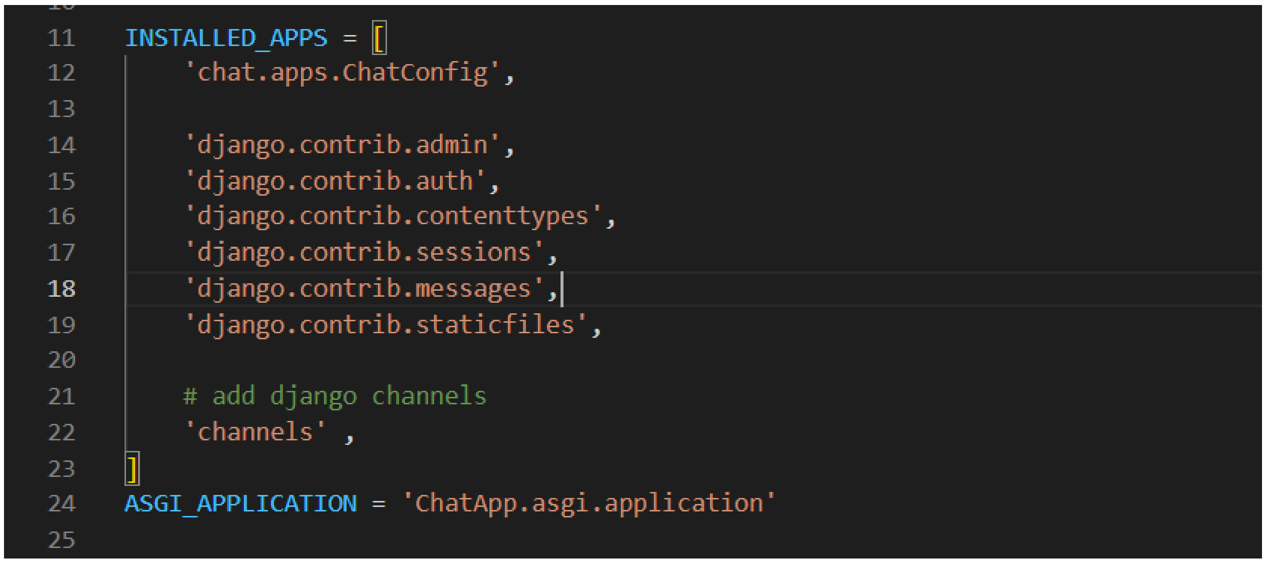


```
(chat_app_venv) D:\Programming\Python\Django project\GFG Projects\ChatApplication\ChatApp>python manage.py runserver
Watching for file changes with StatReloader
Performing system checks...

System check identified no issues (0 silenced).

You have 18 unapplied migration(s). Your project may not work properly until you apply the migrations for app(s): admin, auth, conter
Run 'python manage.py migrate' to apply them.
July 20, 2022 - 23:37:55
Django version 4.0.6, using settings 'ChatApp.settings'
Starting ASGI/Channels version 3.0.5 development server at http://127.0.0.1:8000/
Quit the server with CTRL-BREAK.
```

Step 8: Create a new app that will have all the chat functionality. To create an app write a command in the terminal.

```
python manage.py startapp chat
```

And add your app to the installed apps in settings.py.



```
INSTALLED_APPS = [
    'chat.apps.ChatConfig',

    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',

    # add django channels
    'channels' ,
]
ASGI_APPLICATION = 'ChatApp.asgi.application'
```

Step 9: Create a few files in your chat app

- **chat/urls.py:** This will route the Django application to different views in the app.
- **Create a templates folder:** Inside your app, create two files inside the template/chat named chat.Page.html, and LoginPage.html.
- **routing.py:** This will route the WebSocket connections to the consumers.
- **consumers.py:** This is the file where all the asynchronous functionality will take place

Paste this code into your ChatApp/urls.py. This will route you to your chat app.

Python

```python
from django.contrib import admin
from django.urls import path, include

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

Paste this code into your chat/urls.py. This will route you toward views.

Python

```python
from django.urls import path, include
from chat import views as chat_views
from django.contrib.auth.views import LoginView, LogoutView


urlpatterns = [
    path("", chat_views.chatPage, name="chat-page"),

    # login-section
    path("auth/login/", LoginView.as_view
         (template_name="chat/LoginPage.html"), name="login-user"),
    path("auth/logout/", LogoutView.as_view(), name="logout-user"),
]
```

Paste this code into your views.py. This will route your views to the chatPage.html that had been created in the templates folder of the chat app.

Python

```python
from django.shortcuts import render, redirect


def chatPage(request, *args, **kwargs):
    if not request.user.is_authenticated:
        return redirect("login-user")
    context = {}
    return render(request, "chat/chatPage.html", context)
```

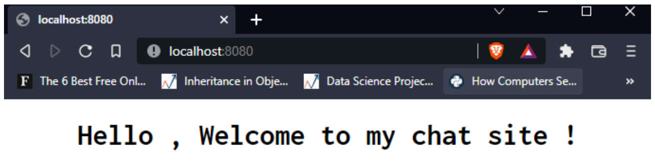
The URL is in Django format, this is Django syntax to map to a URL. We will create a URL named “logout-user”, then Django will map this URL name to the URL from the template. Django provides a few pythonic syntaxes to deal with the control statement. Here we have provided {% if request.user.is_authenticated %} line in the HTML, this is given by Django which ensures that if there is any user who is logged in, then only displays the logout link.

- templates/chat/chatPage.html

HTML

```html
<!DOCTYPE html>
<html>
  <body>
    <center><h1>Hello , Welcome to my chat site ! {{request.user}}</h1></center>
    <br>
    {% if request.user.is_authenticated  %}
    <center> Logout the chat Page <a href = "{% url 'logout-user' %}">Logout</a></center>
    {% endif %}
    <div
      class="chat__item__container"
      id="id_chat_item_container"
      style="font-size: 20px"
    >
      <br />
      <input type="text" id="id_message_send_input" />
      <button type="submit" id="id_message_send_button">Send Message</button>
      <br />
      <br />
    </div>
    <script>
      const chatSocket = new WebSocket("ws://" + window.location.host + "/");
      chatSocket.onopen = function (e) {
        console.log("The connection was setup successfully !");
      };
      chatSocket.onclose = function (e) {
        console.log("Something unexpected happened !");
      };
      document.querySelector("#id_message_send_input").focus();
      document.querySelector("#id_message_send_input").onkeyup = function (e) {
        if (e.keyCode == 13) {
          document.querySelector("#id_message_send_button").click();
        }
      };
      document.querySelector("#id_message_send_button").onclick = function (e) {
        var messageInput = document.querySelector(
          "#id_message_send_input"
        ).value;
        chatSocket.send(JSON.stringify({ message: messageInput, username : "{{request.user.username}}"}));
      };
      chatSocket.onmessage = function (e) {
        const data = JSON.parse(e.data);
        var div = document.createElement("div");
        div.innerHTML = data.username + " : " + data.message;
        document.querySelector("#id_message_send_input").value = "";
        document.querySelector("#id_chat_item_container").appendChild(div);
      };
    </script>
  </body>
</html>
```

{{request.user.userrname}} tells the username of the currently logged-in user. If the user is logged in, it will give its username; if it's not logged in, it will print nothing. The chat page looks like this now, because there is no current logged-in user and {{request.user.username}} prints out nothing.



Hello , Welcome to my chat site !

- templates/chat/LoginPage.html

HTML

```html
<!DOCTYPE html>
<html>
<body>
    <form method ="post">
        {% csrf_token %}
        {{form.as_p}}
        <br>
        <button type = "submit">Login</button>
    </form>
</body>
</html>
```

Implement the WebSockets, asynchronous, and Django channels

Till now we have set up our standard Django project. After implementing the Django application. Now is the time to create the ASGI application.

Step 10: Firstly migrate your database.

```
python manage.py makemigrations
```

```
python manage.py migrate
```

Step 11: Open routing.py and create a route for ChatConsumer (which we will be creating in the next step). Now we have two types of routings in the project. First is urls.py which is for the native Django routing of URLs, and another is for the WebSockets for ASGI support of Django.

chat/routing.py

Python

```python
from django.urls import path , include
from chat.consumers import ChatConsumer

# Here, "" is routing to the URL ChatConsumer which
# will handle the chat functionality.
websocket_urlpatterns = [
    path("" , ChatConsumer.as_asgi()) ,
]
```

Step 12. Open consumers.py will handle the events, like onmessage event, onopen event, etc, We will see these events in **chatPage.html** where we have created the socket connection.

Code explanation:

- **class ChatConsumer(AsyncWebsocketConsumer):** Here we are creating a class named ChatConsumer which inherits from AsyncWebsocketConsumer and is used to create, destroy and do a few more things with WebSockets. And here we are creating ChatSocket for the required purpose.
- **async def connect(self):** This function works on the websocket instance which has been created and when the connection is open or created, it connects and accepts the connection. It creates a group name for the chatroom and adds the group to the channel layer group.
- **async def disconnect():** This just removes the instance from the group.
- **async def receive():** This function is triggered when we send data from the WebSocket (the event for this to work is: send), this receives the text data which has been converted into the JSON format (as it is suitable for the javascript) after the text_data has been received, then it needs to be spread out to the other instances which are active in the group. we retrieve the

message parameter which holds the message and the username parameter which was sent by the socket via HTML or js. This message which is received will be spread to other instances via the channel_layer.group_send() method which takes the first argument as the roomGroupName that to which group this instance belongs and where the data needs to be sent. then the second argument is the dictionary which defines the function which will handle the sending of the data (“type”: “sendMessage”) and also dictionary has the variable message which holds the message data.

- **async def sendMessage(self, event):** This function takes the instance which is sending the data and the event, basically event holds the data which was sent via the group_send() method of the receive() function. Then it sends the message and the username parameter to all the instances which are active in the group. And it is dumped in JSON format so that js can understand the notation. JSON is the format (Javascript object notation)

chat/consumers.py

Python

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

class ChatConsumer(AsyncWebsocketConsumer):
    async def connect(self):
        self.roomGroupName = "group_chat_gfg"
        await self.channel_layer.group_add(
            self.roomGroupName ,
            self.channel_name
        )
        await self.accept()
    async def disconnect(self , close_code):
        await self.channel.name.group_discard(
            self.roomGroupName ,
            self.channel_name
        )
    async def receive(self, text_data):
        text_data_json = json.loads(text_data)
        message = text_data_json["message"]
        username = text_data_json["username"]
        await self.channel_layer.group_send(
            self.roomGroupName,{
                "type" : "sendMessage" ,
                "message" : message ,
                "username" : username ,
            })
    async def sendMessage(self , event) :
        message = event["message"]
        username = event["username"]
        await self.send(text_data = 
json.dumps({"message":message ,"username":username}))
```

The ChatConsumer which we are mapping in the routing.py is the same in this consumers.py which we have just created. These scripts are on asynchronous mode hence we are working with async and await here.

Step 13: Write the below code in your asgi.py for making it work with sockets and creating routings.

ChatApp/asgi.py

Python

```python
import os
from django.core.asgi import get_asgi_application

os.environ.setdefault('DJANGO_SETTINGS_MODULE', 
'ChatApp.settings')

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

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

We usually work with wsgi.py which is in the standard Django without any asynchronous support. But here we are using asynchronous channels. So we have to define the routings in a different way than URLs. For HTTP we define that use the normal application which we were already using, now we have introduced another protocol, that is ws (WebSocket) for which you have to route. The ProtocolTypeRouter creates routes for different types of protocols used in the application. AuthMiddlewareStack authenticates the routes and instances for the Authentication and URLRouter routes the ws (WebSocket connections). The protocol for WebSockets is known as “ws”. For different requests we use HTTP.

Here the router routes the WebSocket URL to a variable in the chat app that is “websocket_urlpatterns” and this variable holds the routes for the WebSocket connections.

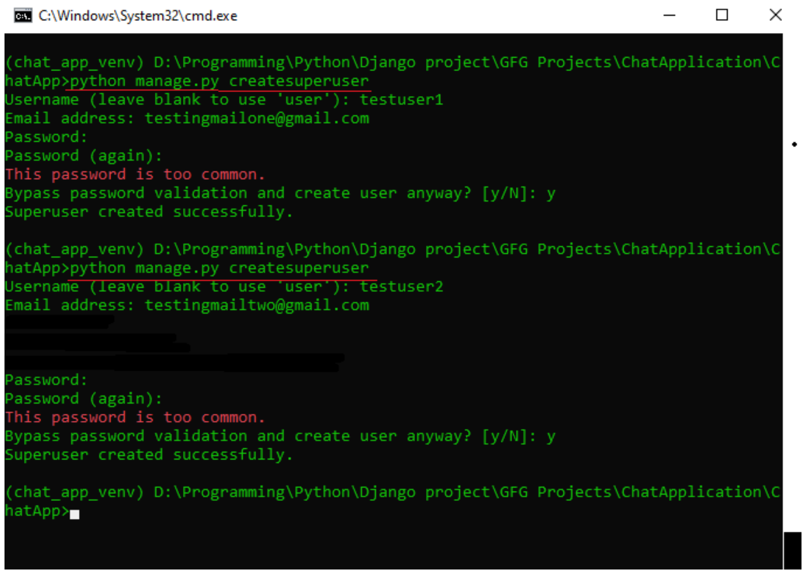
Step 14: This code defines the channel layer in which we will be working and sharing data. For the deployment and production level, don’t use InMemoryChannelLayer, because there are huge chances for your data leakage. This is not good for production. For production use the Redis channel.

ChatApp/settings.py

Python

```python
CHANNEL_LAYERS = {
    "default": {
        "BACKEND": "channels.layers.InMemoryChannelLayer"
    }
}
```

Step 15: Now, we need to create 2 users for that we will use “**python manage.py createsuperuser**” command which creates a superuser in the system.

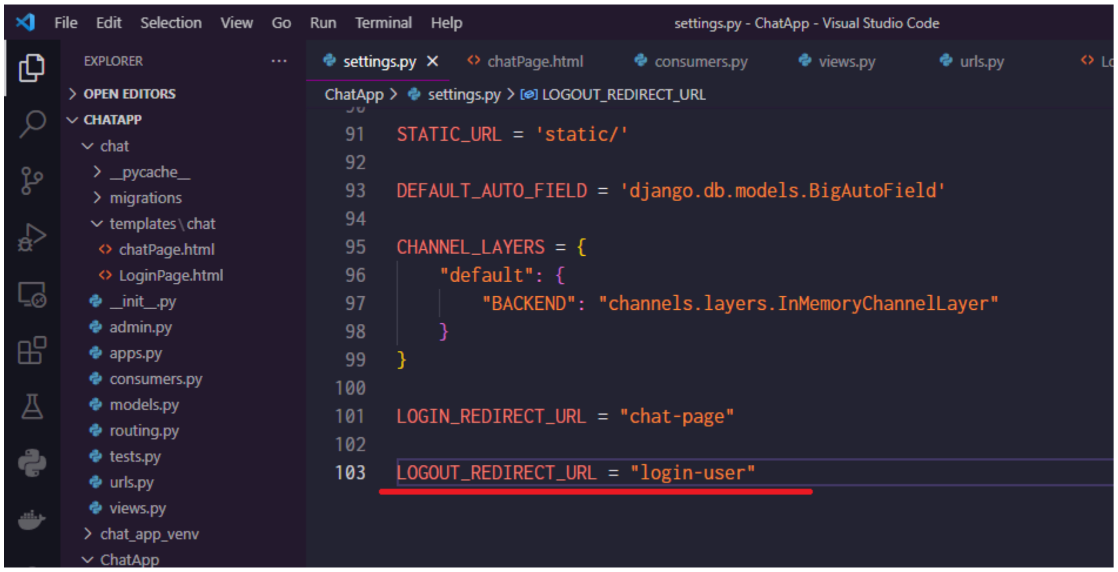


C:\Windows\System32\cmd.exe

```
(chat_app_venv) D:\Programming\Python\Django project\GFG Projects\ChatApplication\C
hatApp>python manage.py createsuperuser
Username (leave blank to use 'user'): testuser1
Email address: testingmailone@gmail.com
Password:
Password (again):
This password is too common.
Bypass password validation and create user anyway? [y/N]: y
Superuser created successfully.

(chat_app_venv) D:\Programming\Python\Django project\GFG Projects\ChatApplication\C
hatApp>python manage.py createsuperuser
Username (leave blank to use 'user'): testuser2
Email address: testingmailtwo@gmail.com

[redacted]

Password:
Password (again):
This password is too common.
Bypass password validation and create user anyway? [y/N]: y
Superuser created successfully.

(chat_app_venv) D:\Programming\Python\Django project\GFG Projects\ChatApplication\C
hatApp>
```

Step 16: We have set the parameter LOGIN_REDIRECT_URL = “chat-page”, this is the name of our landing page URL. This means that whenever the user gets logged in, he will be sent to the chatPage as a verified user and he is eligible to chat through. Now similarly we need to set up the LOGOUT_REDIRECT_URL for the site.

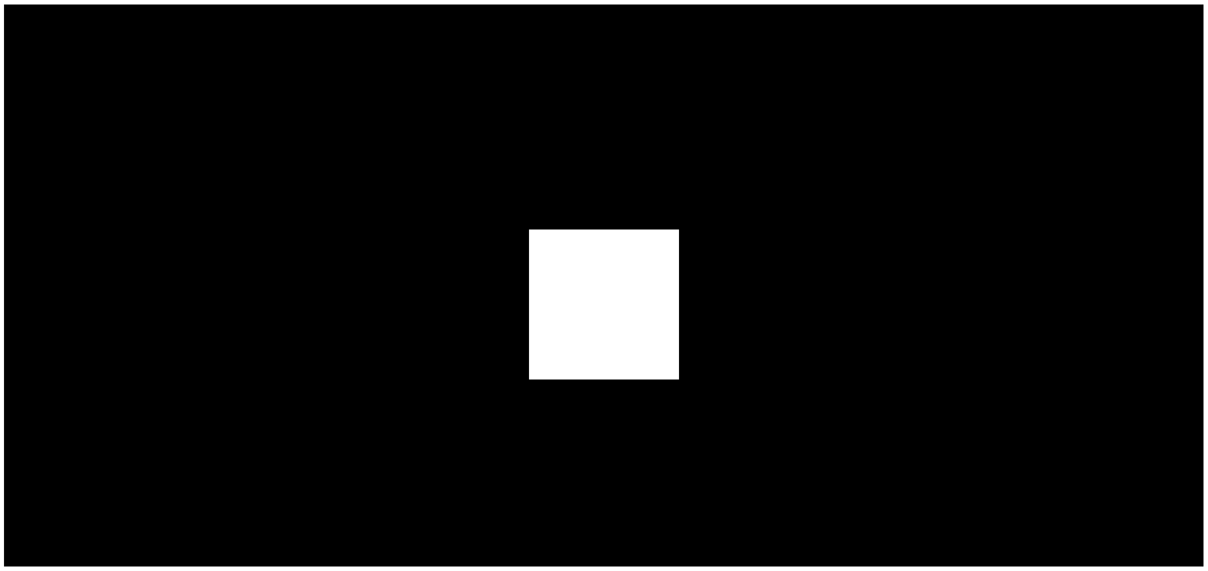
ChatApp/settings.py



```
STATIC_URL = 'static/'

DEFAULT_AUTO_FIELD = 'django.db.models.BigAutoField'

CHANNEL_LAYERS = {
    "default": {
        "BACKEND": "channels.layers.InMemoryChannelLayer"
    }
}

LOGIN_REDIRECT_URL = "chat-page"

LOGOUT_REDIRECT_URL = "login-user"
```

Deployment

Now, run your server and move to the site and start two different browsers to log into two other users. It is because if you have logged in with first user credentials, the login details are stored in the cookies, then if you log in from second user details in the same browser even with different tabs, So, you cannot chat with two other users in the same browser, that's why to use two different browsers.



00:00 00:00

omka... 25

Previous Article **Next Article**

Create URL Bookmark Manager Using Django

Similar Reads

Realtime Distance Estimation Using OpenCV - Python

Prerequisite: Introduction to OpenCV In this article, we are going to see how to calculate the distance with a webcam using OpenCV in Python. By using it, one...

5 min read

Adding Tags Using Django-Taggit in Django Project

Django-Taggit is a Django application which is used to add tags to blogs, articles etc. It makes very easy for us to make adding the tags functionality to our djang...

2 min read

How to customize Django forms using Django Widget Tweaks ?

Django forms are a great feature to create usable forms with just few lines of code. But Django doesn't easily let us edit the form for good designs. Here, we...

3 min read

Integrating Django with Reactjs using Django REST Framework

In this article, we will learn the process of communicating between the Django Backend and React js frontend using the Django REST Framework. For the sake...

15+ min read

Converting WhatsApp chat data into a Word Cloud using Python

Let us see how to create a word cloud using a WhatsApp chat file. Convert the WhatsApp chat file from .txt format to .csv file. This can be done using Pandas....

4 min read

GUI chat application using Tkinter in Python

Chatbots are computer program that allows user to interact using input methods. The Application of chatbots is to interact with customers in big enterprises or...

7 min read

Export WhatsApp Chat History to Excel Using Python

In this article, we will discuss how to export a specific user's chats to an Excel sheet. To export the chats, we will use several Python modules and libraries. In...

5 min read

Simple chat Application using Websockets with FastAPI

In this article, we'll delve into the concept of WebSockets and their role in facilitating bidirectional communication between clients and servers....

6 min read

Simple Chat Room using Python

This article demonstrates - How to set up a simple Chat Room server and allow multiple clients to connect to it using a client-side script. The code uses the...

8 min read

Weather app using Django | Python

In this tutorial, we will learn how to create a Weather app that uses Django as backend. Django provides a Python Web framework based web framework that...

2 min read

Article Tags : Python Django-Projects python Python Django +1 More

Practice Tags : python python

DSA for Beginners
Basic DSA Problems
DSA Roadmap
Top 100 DSA Interview Problems
DSA Roadmap by Sandeep Jain
All Cheat Sheets

Machine Learning
ML Maths
Data Visualisation
Pandas
NumPy
NLP
Deep Learning

Web Technologies

HTML
CSS
JavaScript
TypeScript
ReactJS
NextJS
Bootstrap
Web Design

Python Tutorial

Python Programming Examples
Python Projects
Python Tkinter
Web Scraping
OpenCV Tutorial
Python Interview Question
Django

Computer Science

Operating Systems
Computer Network
Database Management System
Software Engineering
Digital Logic Design
Engineering Maths
Software Development
Software Testing

DevOps

Git
Linux
AWS
Docker
Kubernetes
Azure
GCP
DevOps Roadmap

System Design

High Level Design
Low Level Design
UML Diagrams
Interview Guide
Design Patterns
OOAD
System Design Bootcamp
Interview Questions

Inteview Preparation

Competitive Programming
Top DS or Algo for CP
Company-Wise Recruitment Process
Company-Wise Preparation
Aptitude Preparation
Puzzles

School Subjects

Mathematics
Physics
Chemistry
Biology
Social Science
English Grammar
Commerce
World GK

GeeksforGeeks Videos

DSA
Python
Java
C++
Web Development
Data Science
CS Subjects