

<https://chatgpt.com/c/6986bd77-6244-8326-b14f-86b49f19ad27>

Vous : bjr

Bjr 

Tu veux qu'on parle de quoi ? (Tu peux m'écrire en français ou en "mode texto".)

Pouvez vous réaliser un système de chat en ligne en utilisant une API django côté backend. Pour le côté frontend vous pouvez utiliser JAVASCRIPT et HTML et CSS. Mais pour le backend, utilisez seulement une API django. Django==5.2.8

Oui. Voici un **mini-système de chat en ligne** avec :

- **Backend : Django 5.2.8 + Django REST Framework**
- **Temps réel : WebSocket via Django Channels** (c'est la manière "propre" de faire du chat en ligne)
- **API Django** pour l'historique / listes de salons / messages
- **Frontend : HTML + CSS + JavaScript** (WebSocket + fetch API)

Remarque : pour un "chat en ligne" (temps réel), **les WebSockets sont indispensables**. L'API REST seule implique du polling (moins bon).

1) Backend (Django API + WebSocket)

A. Installation

```
python -m venv venv
source venv/bin/activate # (Windows: venv\Scripts\activate)
pip install "Django==5.2.8" djangorestframework channels channels_redis
django-admin startproject config .
python manage.py startapp chat
```

B. config/settings.py

Ajoute les apps + Channels :

```
INSTALLED_APPS = [
```

```

# ...
"rest_framework",
"channels",
"chat",
]

ASGI_APPLICATION = "config.asgi.application"

# Pour dev simple (sans Redis), tu peux utiliser l'InMemory layer.
# Pour prod, préfère Redis (channels_redis).
CHANNEL_LAYERS = {
    "default": {
        "BACKEND": "channels.layers.InMemoryChannelLayer",
    }
}

```

Si tu veux Redis :

```

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

```

C. Modèles chat/models.py

```

from django.db import models

class Room(models.Model):
    name = models.CharField(max_length=100, unique=True)

    def __str__(self):
        return self.name

class Message(models.Model):
    room = models.ForeignKey(Room, on_delete=models.CASCADE, related_name="messages")
    author = models.CharField(max_length=80)
    content = models.TextField()
    created_at = models.DateTimeField(auto_now_add=True)

    class Meta:
        ordering = ["created_at"]

```

Migration :

```
python manage.py makemigrations
python manage.py migrate
```

D. Sérialiseurs `chat/serializers.py`

```
from rest_framework import serializers
from .models import Room, Message

class RoomSerializer(serializers.ModelSerializer):
    class Meta:
        model = Room
        fields = ["id", "name"]

class MessageSerializer(serializers.ModelSerializer):
    class Meta:
        model = Message
        fields = ["id", "room", "author", "content", "created_at"]
        read_only_fields = ["id", "created_at"]
```

E. API Views `chat/views.py`

```
from rest_framework import generics
from .models import Room, Message
from .serializers import RoomSerializer, MessageSerializer

class RoomListCreateView(generics.ListCreateAPIView):
    queryset = Room.objects.all()
    serializer_class = RoomSerializer

class MessageListView(generics.ListAPIView):
    serializer_class = MessageSerializer

    def get_queryset(self):
        room_id = self.kwargs["room_id"]
        return Message.objects.filter(room_id=room_id).order_by("created_at")[:200]
```

F. URLs API `chat/urls.py`

```
from django.urls import path
from .views import RoomListCreateView, MessageListView
```

```
urlpatterns = [
    path("rooms/", RoomListCreateView.as_view(), name="rooms"),
    path("rooms/<int:room_id>/messages/", MessageListView.as_view(), name="room-messages"),
]
```

Inclure dans config/urls.py :

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

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

2) WebSocket (Channels)

A. Consumer chat/consumers.py

```
import json
from channels.generic.websocket import AsyncWebsocketConsumer
from channels.db import database_sync_to_async
from .models import Room, Message

class ChatConsumer(AsyncWebsocketConsumer):
    async def connect(self):
        self.room_name = self.scope["url_route"]["kwargs"]["room_name"]
        self.group_name = f"chat_{self.room_name}"

        await self.channel_layer.group_add(self.group_name, self.channel_name)
        await self.accept()

        # Assure que la room existe
        await self.get_or_create_room(self.room_name)

    async def disconnect(self, close_code):
        await self.channel_layer.group_discard(self.group_name, self.channel_name)

    async def receive(self, text_data):
        data = json.loads(text_data)
        author = data.get("author", "anonymous")
        content = data.get("content", "").strip()
```

```

if not content:
    return

msg = await self.create_message(self.room_name, author, content)

await self.channel_layer.group_send(
    self.group_name,
    {
        "type": "chat_message",
        "id": msg["id"],
        "author": msg["author"],
        "content": msg["content"],
        "created_at": msg["created_at"],
    }
)

async def chat_message(self, event):
    await self.send(text_data=json.dumps(event))

@database_sync_to_async
def get_or_create_room(self, name):
    return Room.objects.get_or_create(name=name)

@database_sync_to_async
def create_message(self, room_name, author, content):
    room = Room.objects.get(name=room_name)
    m = Message.objects.create(room=room, author=author, content=content)
    return {
        "id": m.id,
        "author": m.author,
        "content": m.content,
        "created_at": m.created_at.isoformat(),
    }

```

B. Routing WS `chat/routing.py`

```

from django.urls import re_path
from .consumers import ChatConsumer

websocket_urlpatterns = [
    re_path(r"ws/chat/(?P<room_name>[^/]+)/$", ChatConsumer.as_asgi()),
]

```

C. config/asgi.py

Remplace par une config Channels :

```
import os
from django.core.asgi import get_asgi_application
from channels.routing import ProtocolTypeRouter, URLRouter
from channels.auth import AuthMiddlewareStack
import chat.routing

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

django_asgi_app = get_asgi_application()

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

Lancer :

```
python manage.py runserver
```

3) Frontend (HTML + CSS + JS)

Crée un fichier `index.html` (tu peux le servir via n'importe quel serveur statique, ou même ouvrir en local pour tester si CORS OK).

index.html

```
<!doctype html>
<html lang="fr">
<head>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1" />
    <title>Chat Django</title>
    <style>
        body { font-family: Arial, sans-serif; background:#f5f5f5; margin:0; }
        .wrap { max-width:900px; margin: 30px auto; padding: 16px; }
        .card { background:#fff; border-radius:12px; padding:16px; box-shadow:0 6px 20px rgba(0,0,0,.08); }
```

```
.row { display:flex; gap:12px; flex-wrap:wrap; }
input, button { padding:10px; border-radius:10px; border:1px solid #ddd; }
button { cursor:pointer; background:#222; color:#fff; border:none; }
#messages { height: 420px; overflow:auto; border:1px solid #eee; border-radius:12px; padding:12px; background:#fafafa; }
.msg { padding:10px 12px; margin:8px 0; border-radius:12px; background:#fff; border:1px solid #eee; }
.meta { font-size:12px; color:#666; margin-bottom:6px; }
</style>
</head>
<body>
<div class="wrap">
  <div class="card">
    <h2>Chat en ligne</h2>

    <div class="row">
      <input id="author" placeholder="Votre nom" value="Mario" />
      <input id="room" placeholder="Salon (ex: general)" value="general" />
      <button id="join">Rejoindre</button>
      <button id="load">Charger l'historique</button>
    </div>

    <div style="margin-top:14px" id="messages"></div>

    <div class="row" style="margin-top:14px">
      <input id="text" placeholder="Écrire un message..." style="flex:1" />
      <button id="send">Envoyer</button>
    </div>
  </div>
</div>

<script>
const messagesEl = document.getElementById("messages");
const authorEl = document.getElementById("author");
const roomEl = document.getElementById("room");
const textEl = document.getElementById("text");

let socket = null;

function addMessage({author, content, created_at}) {
  const div = document.createElement("div");
  div.className = "msg";
  div.innerHTML = `
    <div class="meta"><b>${escapeHtml(author)}</b> • ${new Date(created_at).toLocaleString()}</div>
    <div>${escapeHtml(content)}</div>
  `;
  messagesEl.appendChild(div);
  messagesEl.scrollTop = messagesEl.scrollHeight;
}

// Function to escape HTML characters
function escapeHtml(text) {
  return text.replace(/</g, "<").replace(/>/g, ">").replace(/"/g, """).replace(/'/g, "&apos;").replace(/\n/g, "<br>").replace(/\r/g, "<br>").replace(/\t/g, "<span style='white-space: pre; margin-left: 2em;'>").replace(/\</g, "&lt;").replace(/\>/g, "&gt;").replace(/\&/g, "&amp;");
```

```

}

function escapeHtml(s) {
  return (s ?? "").replace(/([<>"'])/g, m => ({
    "&:&#038;": "&lt;",
    "<:&lt;",
    ">:&gt;",
    "\":&quot;",
    "'':&#039;" 
  })[m]);
}

async function loadHistory() {
  const roomName = roomEl.value.trim();
  if (!roomName) return;

  // 1) récupérer la liste des rooms pour obtenir l'id
  const rooms = await fetch("http://127.0.0.1:8000/api/rooms/").then(r => r.json());
  const room = rooms.find(x => x.name === roomName);
  if (!room) {
    alert("Salon inexistant. Clique 'Rejoindre' d'abord (WS crée le salon).");
    return;
  }

  // 2) récupérer les messages
  const msgs = await fetch(`http://127.0.0.1:8000/api/rooms/${room.id}/messages/`).then(r => r.json());
  messagesEl.innerHTML = "";
  msgs.forEach(addMessage);
}

function connectWS() {
  const roomName = roomEl.value.trim();
  const author = authorEl.value.trim() || "anonymous";
  if (!roomName) return alert("Choisis un salon");

  if (socket) socket.close();

  socket = new WebSocket(`ws://127.0.0.1:8000/ws/chat/${encodeURIComponent(roomName)}/`);

  socket.onopen = () => addMessage({author:"system", content:`Connecté au salon "${roomName}"`, created_at:new Date().toISOString()});
  socket.onmessage = (e) => {
    const data = JSON.parse(e.data);
    if (data.type === "chat_message") addMessage(data);
  };
  socket.onclose = () => addMessage({author:"system", content:`Déconnecté`, created_at:new Date().toISOString()});

  document.getElementById("send").onclick = () => {
    const content = textEl.value.trim();
    if (!content || !socket || socket.readyState !== 1) return;
  }
}

```

```
socket.send(JSON.stringify({ author, content }));
textEl.value = "";
textEl.focus();
};

textEl.onkeydown = (ev) => {
  if (ev.key === "Enter") document.getElementById("send").click();
};

document.getElementById("join").onclick = connectWS;
document.getElementById("load").onclick = loadHistory;
</script>
</body>
</html>
```

4) Test rapide

1. Lance Django :

```
python manage.py runserver
```

2. Ouvre `index.html` dans 2 onglets, même salon (`general`), deux noms différents.
 3. Clique **Rejoindre** puis envoie des messages.
 4. Clique **Charger l'historique** pour recharger via l'API REST.
-

5) Améliorations faciles (si tu veux)

- Auth (JWT) au lieu du champ `author` libre
 - “typing...” / présence en ligne
 - Pagination / recherche messages
 - CORS (si ton frontend est sur un autre domaine)
 - Redis + déploiement (Daphne/Uvicorn + Nginx)
-

Si tu veux, je peux aussi te donner une **structure de projet complète** (arborescence + tous les fichiers) ou ajouter **authentification JWT** et **CORS** proprement.

Oui je voudrais une structure de projet complète (arborescence + tous les fichiers) au très grand complet. Pouvez vous également mettre le tout dans un fichier winrar ?

Voici une **structure de projet complète** (arborescence + tous les fichiers) prête à l'emploi.

Je ne peux pas générer un **.rar WinRAR** directement ici, mais je t'ai mis le projet dans une archive **.zip** (WinRAR ouvre parfaitement les .zip).

Téléchargement : [django_chat_api_full.zip](#)

Arborescence du projet

```
django_chat_api/
├── .gitignore
├── README.md
├── manage.py
├── requirements.txt
└── config/
    ├── __init__.py
    ├── asgi.py
    ├── settings.py
    ├── urls.py
    └── wsgi.py
└── chat/
    ├── __init__.py
    ├── admin.py
    ├── apps.py
    ├── consumers.py
    ├── models.py
    ├── routing.py
    ├── serializers.py
    ├── urls.py
    ├── views.py
    └── migrations/
        └── __init__.py
└── frontend/
    └── index.html
```

Lancer le projet

```
python -m venv venv
# Windows:
venv\Scripts\activate
# Linux/macOS:
# source venv/bin/activate

pip install -r requirements.txt
python manage.py migrate
python manage.py runserver
```

Puis ouvre :

- frontend/index.html (double-clic)
- Backend : <http://127.0.0.1:8000/>
- API : <http://127.0.0.1:8000/api/rooms/>
- WebSocket : <ws://127.0.0.1:8000/ws/chat/<room>/>

Si tu veux absolument un .rar, dis-moi juste si tu préfères :

- **RAR4 ou RAR5**,
et je te donne la commande WinRAR (GUI ou ligne de commande) pour convertir le .zip en .rar en 10 secondes.

Est ce que vous m'avez donné tous les fichier au grand complet dans le fichier zip ?

Oui — **le .zip contient bien tout le projet “au complet”** pour ce chat (backend Django + API + WebSocket + frontend), c'est-à-dire :

- Tous les fichiers Django nécessaires : manage.py, config/*, chat/*
- L'API REST (rooms + messages)
- Le WebSocket (Channels) : consumers.py + routing.py + asgi.py
- Le frontend : frontend/index.html
- requirements.txt, README.md, .gitignore

Ce que **le zip ne peut pas contenir** (normal) tant que tu n'as pas exécuté le projet chez toi :

- db.sqlite3 (la base n'existe qu'après migrate)
- Le dossier venv/ (environnement virtuel)
- Les fichiers de migrations générés (0001_initial.py, etc.) : ils apparaissent après makemigrations.
→ Mais c'est prévu : Django les génère localement.

Pour obtenir exactement ces fichiers "générés", fais :

```
python manage.py makemigrations  
python manage.py migrate
```

Si tu veux, je peux aussi te fournir une version avec une migration 0001_initial.py déjà incluse (donc *encore plus "prête"*), mais celle-ci est déjà correcte et standard.

Pouvez vous dire quelle est la version de Django utilisée dans ce projet ?

Dans le projet que je t'ai donné, la version de Django prévue est **Django 5.2.8**.

Tu peux le vérifier dans le fichier **requirements.txt** de l'archive, où il y a :

- Django==5.2.8

Ok je vais l'essayer

Parfait. 🎉

Si tu tombes sur un souci, voici les 3 erreurs les plus fréquentes et quoi faire :

1. WebSocket ne se connecte pas

- Vérifie que tu lances bien :

```
python manage.py runserver
```

- Et que l'URL WS est bien : ws://127.0.0.1:8000/ws/chat/general/

2. Erreur CORS (si tu ouvres le HTML en local)

- Dans le projet, CORS_ALLOW_ALL_ORIGINS = True est déjà activé, donc l'API REST devrait passer.

- Si ton navigateur bloque quand même, essaye de servir le frontend via un petit serveur :

```
python -m http.server 5500
```

Puis ouvre `http://127.0.0.1:5500/frontend/index.html`

3. Channels/Redis

- En dev, ça marche déjà avec `InMemoryChannelLayer`.
- Si tu passes à Redis plus tard, lance Redis et active la config Redis dans `settings.py`.

Si tu me copies/colles l'erreur exacte (console Django + console navigateur), je te dis précisément quoi corriger.