SE Project

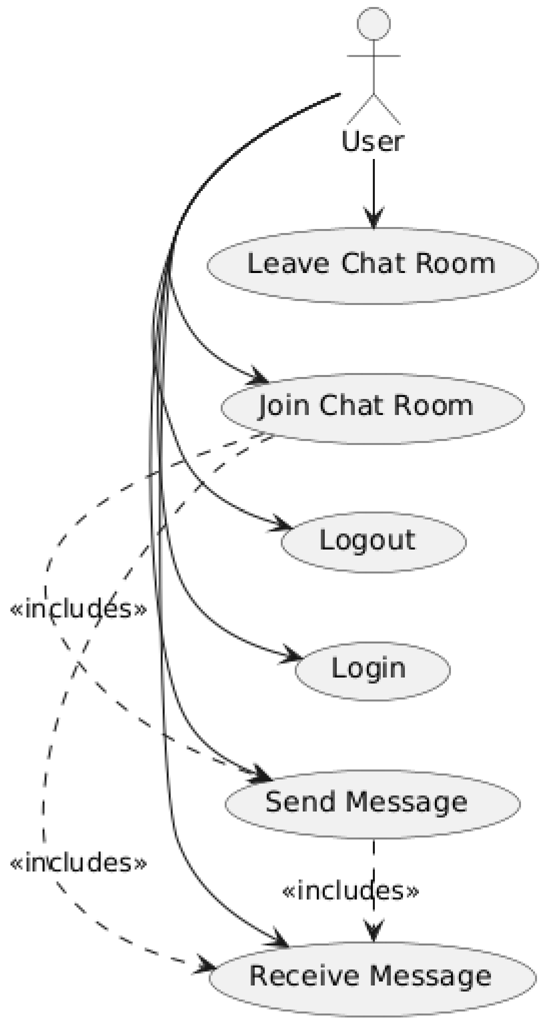
ChApp

Django + WebSocket

Vlad Sminchise
Technical University of Cluj-Napoca

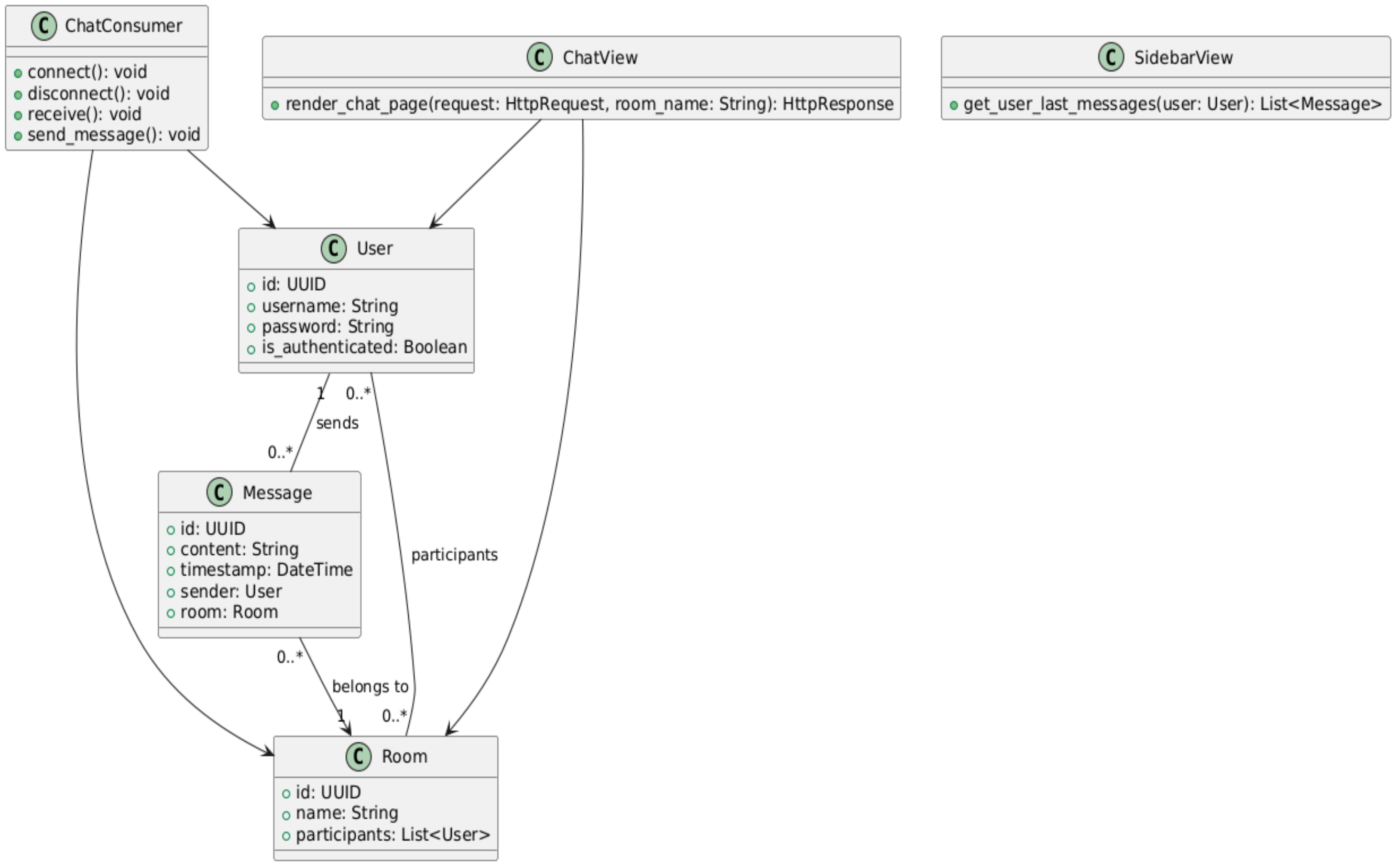
A. Design

Use cases



This use case diagram shows the functionalities of the application of sending and receiving text messages. The user can join a chat room which connects them to another user via a WebSocket server.

Class diagram



The class diagram shows the classes used for this purpose. The User class holds information about the current, connected user, information obtained from a database. The Room represents the server that connects the two users and the ChatConsumer class handles the connection between the client and the server. The class Message is responsible for the sent and received information and data about it.

B. Implementation

Functionality

The chat room and the sent and received messages all meet the WebSocket communication protocol standards.

WebSocket is a duplex communication protocol that allows real-time communication between a server and a client as long as there is a connection established.

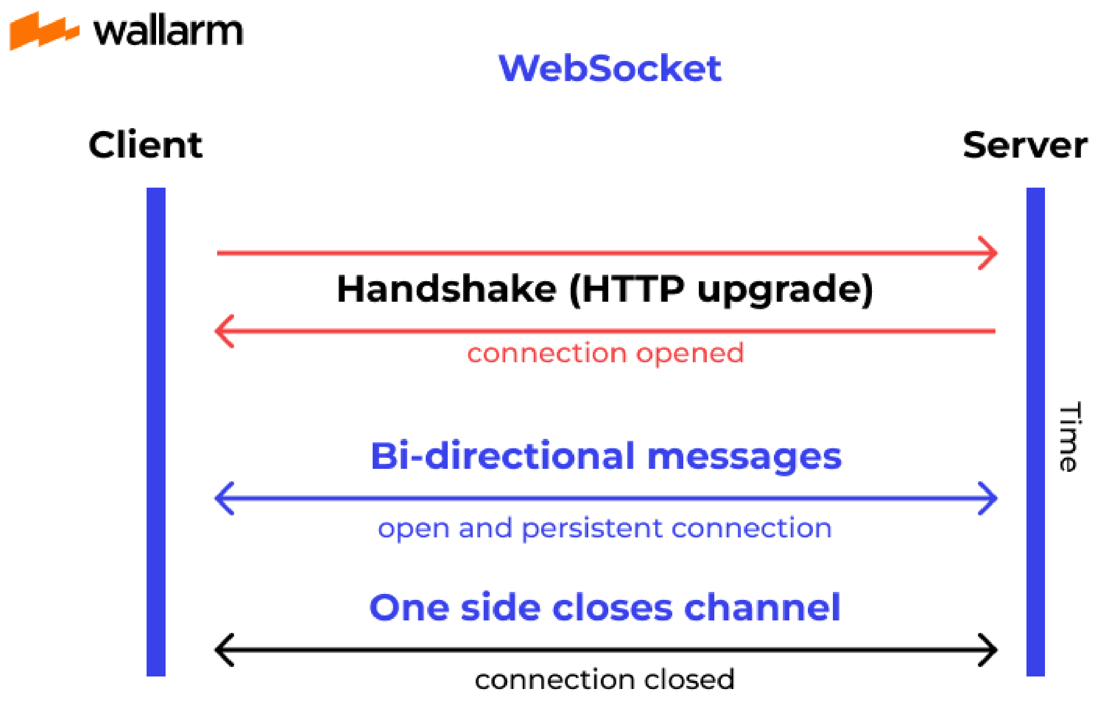


Image from https://www.wallarm.com

This protocol was used for creating this application for its fast and simple way of functioning, but the main reason is the real-time communication feature.

Code

views.py:

```python
from django.shortcuts import render
from django.contrib.auth.decorators import login_required
from django.contrib.auth.models import User
from .models import Message
from django.db.models import Q
from datetime import datetime,timezone


@login_required
def chat_room(request, room_name):
    search_query = request.GET.get('search', '')
    users = User.objects.exclude(id=request.user.id)

    # Filter chats between the logged-in user and the specific room_name user
    chats = Message.objects.filter(
        (Q(sender=request.user) & Q(receiver__username=room_name)) |
        (Q(receiver=request.user) & Q(sender__username=room_name))
    )

    if search_query:
        chats = chats.filter(Q(content__icontains=search_query))

    chats = chats.order_by('timestamp')

    user_last_messages = []

    # Populate the user_last_messages list with the last message for each user
    for user in users:
        last_message = Message.objects.filter(
            (Q(sender=request.user) & Q(receiver=user)) |
            (Q(receiver=request.user) & Q(sender=user))
        ).order_by('-timestamp').first()

        user_last_messages.append({
            'user': user,
            'last_message': last_message
        })

    # Use `datetime.min.replace(tzinfo=timezone.utc)` for offset-aware
consistency
    user_last_messages.sort(
        key=lambda x: x['last_message'].timestamp if x['last_message'] else
datetime.min.replace(tzinfo=timezone.utc),
        reverse=True
    )

    return render(request, 'chat.html', {
        'room_name': room_name,
        'chats': chats,
        'users': users,
        'user_last_messages': user_last_messages,
        'search_query': search_query
    })


@login_required
def start_page(request):
    # Get all users except the current logged-in user
    users = User.objects.exclude(id=request.user.id)

    return render(request, 'home.html', {
        'users': users,
    })
```

models.py:

```
from django.db import models
from django.contrib.auth.models import User

class Message(models.Model):
    sender = models.ForeignKey(User, related_name="sent_messages",
on_delete=models.CASCADE)
    receiver = models.ForeignKey(User, related_name="received_messages",
on_delete=models.CASCADE)
    content = models.TextField()
    timestamp = models.DateTimeField(auto_now_add=True)

    def __str__(self):
        return f"{self.sender} -> {self.receiver}: {self.content[:20]}"
```

consumers.py:

```
# chat/consumers.py
import json
from channels.generic.websocket import AsyncWebsocketConsumer
from django.contrib.auth.models import User
from .models import Message
from asgiref.sync import sync_to_async


class ChatConsumer(AsyncWebsocketConsumer):
    async def connect(self):
        self.room_name = self.scope['url_route']['kwargs']['room_name']
        user1 = self.scope['user'].username
        user2 = self.room_name
        self.room_group_name = f"chat_{''.join(sorted([user1, user2]))}"

        # Join room group
        await self.channel_layer.group_add(self.room_group_name,
self.channel_name)
        await self.accept()

    async def disconnect(self, close_code):
        # Leave room group
        await self.channel_layer.group_discard(self.room_group_name,
self.channel_name)

    async def receive(self, text_data):
        text_data_json = json.loads(text_data)
        message = text_data_json['message']
        sender = self.scope['user']
        receiver = await self.get_receiver_user()

        await self.save_message(sender, receiver, message)

        await self.channel_layer.group_send(
            self.room_group_name,

            {
                'type': 'chat_message',
                'sender': sender.username,
                'receiver': receiver.username,
                'message': message
            }
        )
    async def chat_message(self, event):
        message = event['message']
        sender = event['sender']
        receiver = event['receiver']

        # Send message to WebSocket
        await self.send(text_data=json.dumps({
            'sender': sender,
            'receiver': receiver,
            'message': message
        }))

    @sync_to_async
    def save_message(self, sender, receiver, message):
        Message.objects.create(sender=sender, receiver=receiver, content=message)

    @sync_to_async
    def get_receiver_user(self):
        return User.objects.get(username=self.room_name)
```

asgi.py:

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

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


# Initialize the ASGI application

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

users/views.py:

```python
from django.shortcuts import render, redirect
from django.contrib.auth.decorators import login_required
from django.contrib.auth.models import User
from django.contrib.auth import authenticate, login, logout
from django.contrib import messages



def login_page(request):
    if request.method == 'POST':
        username = request.POST.get('username')
        password = request.POST.get('password')
        print(username)
        print(password)
        user = authenticate(request, username=username, password=password)
        if user is not None:
            login(request, user)
            messages.success(request, 'Login successful!')
            return redirect('/chat/home/')
        else:
            messages.error(request, 'Invalid email or password. Please try
again.')
    if request.user.is_authenticated:
        return redirect('/chat/home/')
    return render(request,'login.html')


@login_required
def logout_page(request):
    logout(request)
    messages.success(request, 'You have been logged out successfully.')
    return redirect('/')


def signup_view(request):
    if request.method == 'POST':
        email = request.POST.get('email')
        username = request.POST.get('username')
        password1 = request.POST.get('password')
        confirm_password = request.POST.get('confirm_password')

        # Check if passwords match
        if password1 != confirm_password:
            messages.error(request, 'Passwords do not match. Please try again.')
            return render(request, 'signup.html')

        # Check if email is already taken
        if User.objects.filter(email=email).exists():
            messages.error(request, 'Email is already in use. Please try
another.')
            return render(request, 'signup.html')

        # Create the new user
        user = User.objects.create_user(username=username,
                                        email=email,
                                        password=password1
                                        )
        user.save()
        messages.success(request, 'Signup successful! You can now log in.')
        return redirect('login')
    if request.user.is_authenticated:
        return redirect('/chat/home/')
    return render(request, 'signup.html')
```

Appendix

Mini project:

views.py:

```
from django.http import HttpResponse
from django.template import loader
from django.views.decorators.csrf import csrf_protect

def members(request):
  template = loader.get_template('my_app/student_entry.html')
  return HttpResponse(template.render())

from django.shortcuts import render, redirect
from .models import Student

@csrf_protect
def student_entry(request):
  return render(request,'my_app/student_entry.html')

@csrf_protect
def process_student_entry(request):
```

```python
    if request.method == 'POST':
        student_name = request.POST.get('student_name')
        classroom = request.POST.get('classroom')

        # Create a new patient entry in the database using the Patient model
        student = Student(student_name=student_name, classroom=classroom)
        student.save()



        return HttpResponse("Data successfully inserted!")
    else:
        return HttpResponse("Invalid request method.")
```

models.py:

```python
from django.db import models

# Create your models here.

class DemoTable(models.Model):
    text = models.CharField(max_length=200)
    class Meta:
        db_table = "demo"


class Student(models.Model):
    student_name = models.CharField(max_length=255)
    classroom = models.CharField(max_length=10)

    def __str__(self):
        return f"{self.student_name} - {self.classroom}"
```

C. References

[1] WebSocket Documentation https://websocket.org [accessed 29.12.2024]

[2] What is WebSocket https://www.wallarm.com/what/a-simple-explanation-of-what-a-websocket-is [acccessed 03.01.2025]