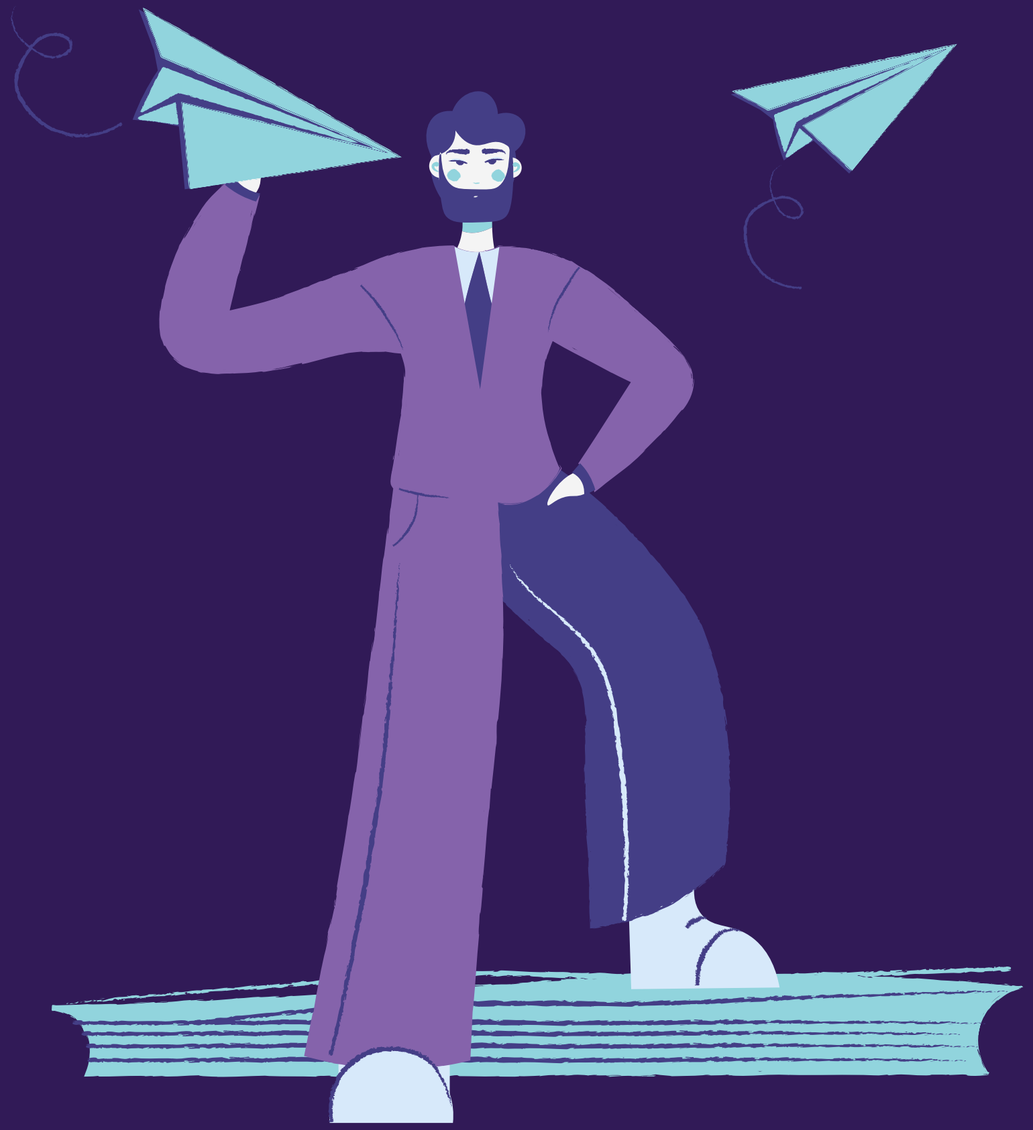
AT70.15 Advanced Topics in Internet Technology

Chatstream

Real-Time Livestream Chat System Using WebRTC, ActionCable, and WebSockets

By Tatiya Seehatrakul | 124875 | 8 May 2025

Table of Contents



Introduction

- Objectives
- Target Users
- Related Work

Objectives

ChatStream aims to deliver a seamless, browser-based livestream and chat system for academic use. It supports secure, one-way video streaming with real-time messaging which is ideal for lectures, talks, and announcements.

By integrating modern web technologies, the project demonstrates real-time communication aligned with core networking and web architecture principles.

1. VIDEO STREAMING

Enable secure, one-way video streaming from a verified presenter (streamer) to multiple authenticated viewers using WebRTC.

2. REAL-TIME CHAT

Provide real-time messaging with WebSocket (ActionCable) support, allowing viewers to interact live during sessions.

3. BROWSER ACCESS

Ensure fast access from any modern browser with no additional software installation, powered by Dockerized deployment and responsive design.

4. NETWORK LAYERS

Demonstrate real-world use of the five-layer Internet Protocol model which are application, transport, network, data link, and physical layers to support live web communication.

Target Users

ChatStream is developed for the academic community.

University students, faculty, and staff can livestream lectures, seminars, and study sessions with ease. The platform ensures fast access, secure authentication, and smooth communication, even in resource-limited settings, using only a browser.

TARGET USERS

- For students, faculty, and academic staff
- For live instruction and academic communication

USE CASES

- Lecture, seminar broadcasting, live announcements
- Interactive study groups with real-time messaging

ACCESSIBILITY

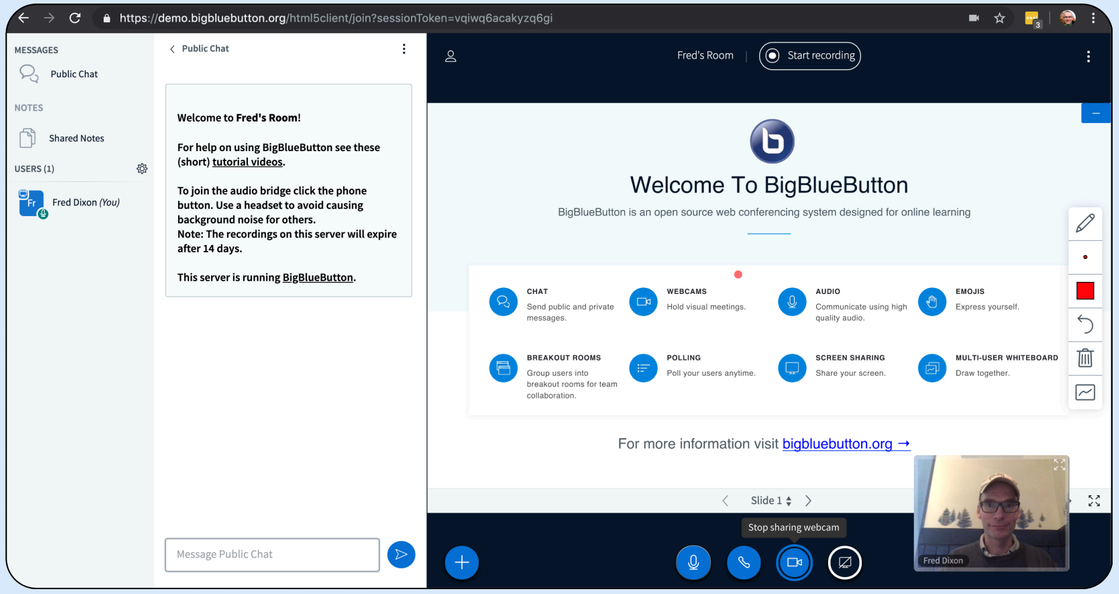
- No installation needed for all modern browsers
- Lightweight system ideal for low-resource or remote environments

Related System – BigBlueButton

- Open-source web conferencing system
- Designed for online education and virtual classrooms
- Uses WebRTC for audio/video streaming
- Supports real-time messaging and screen sharing
- Scalable backend with TURN fallback and bandwidth adaptation

🧠 Inspired ChatStream's academic use-case alignment and media delivery structure

BigBlueButton™



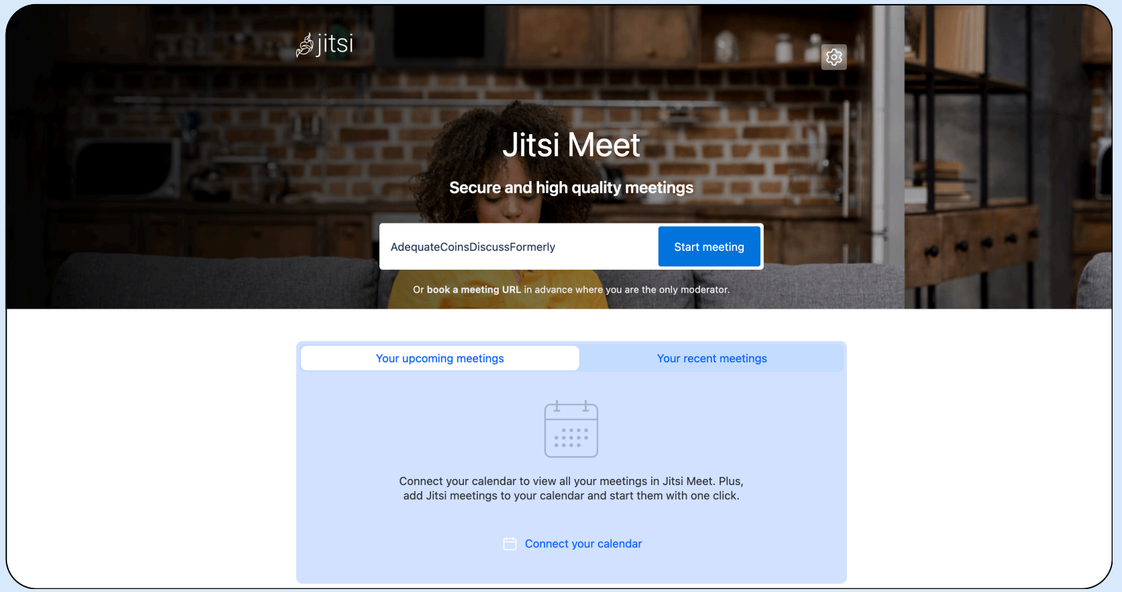
https://bigbluebutton.org/

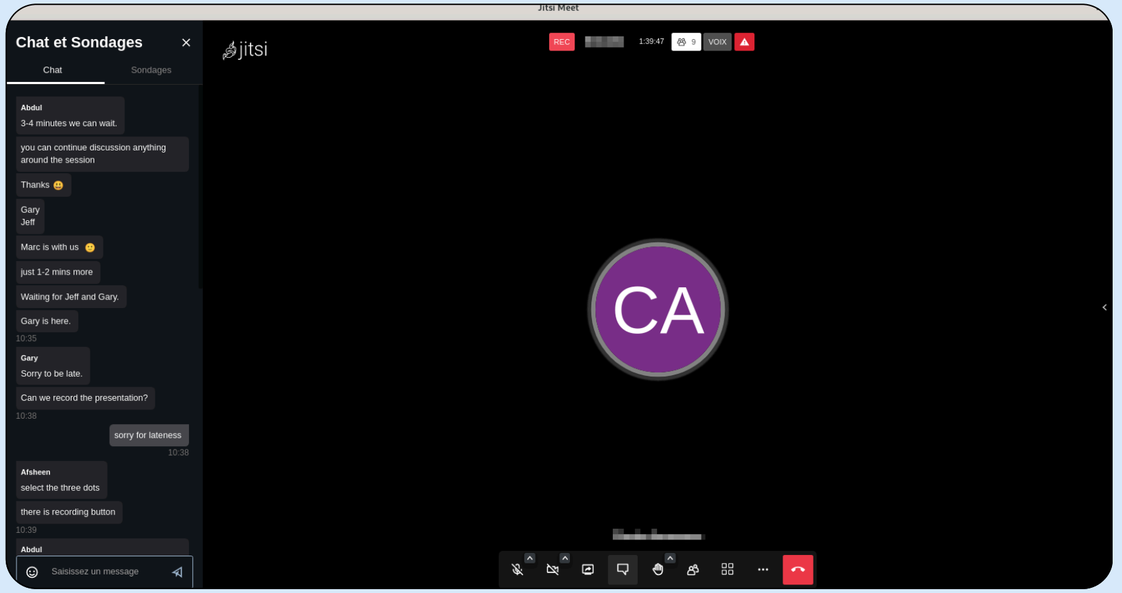
Related System – Jitsi Meet

- Free and open-source video conferencing platform
- Peer-to-peer WebRTC communication
- Signaling via XMPP or WebSocket
- Emphasizes security with encrypted media
- Highly modular and browser-native

🧠 Reinforced our use of peer-to-peer WebRTC with secure signaling

Jitsi Meet



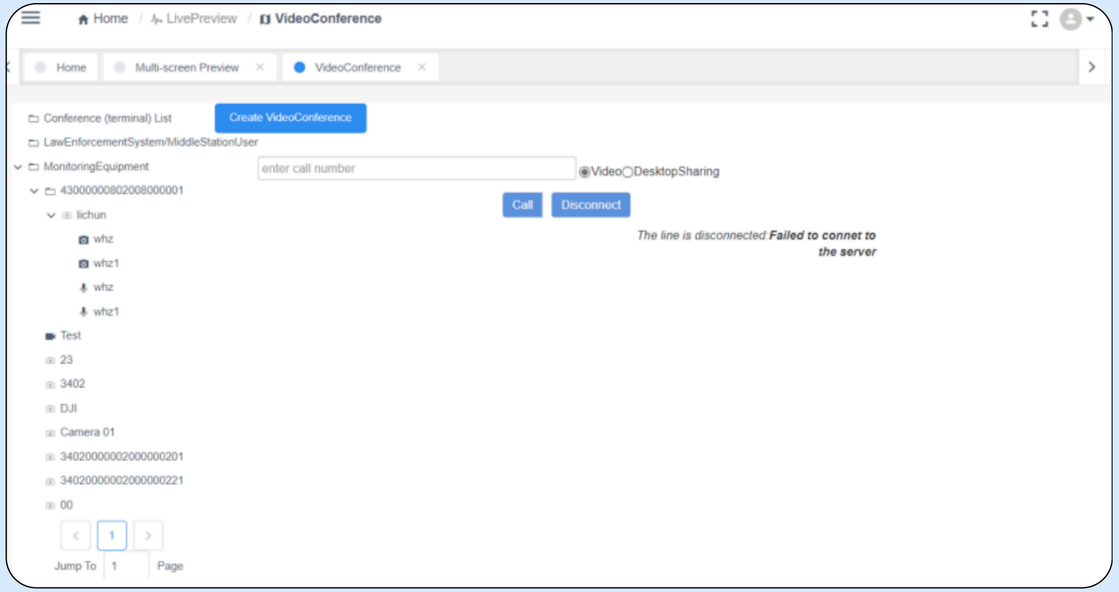


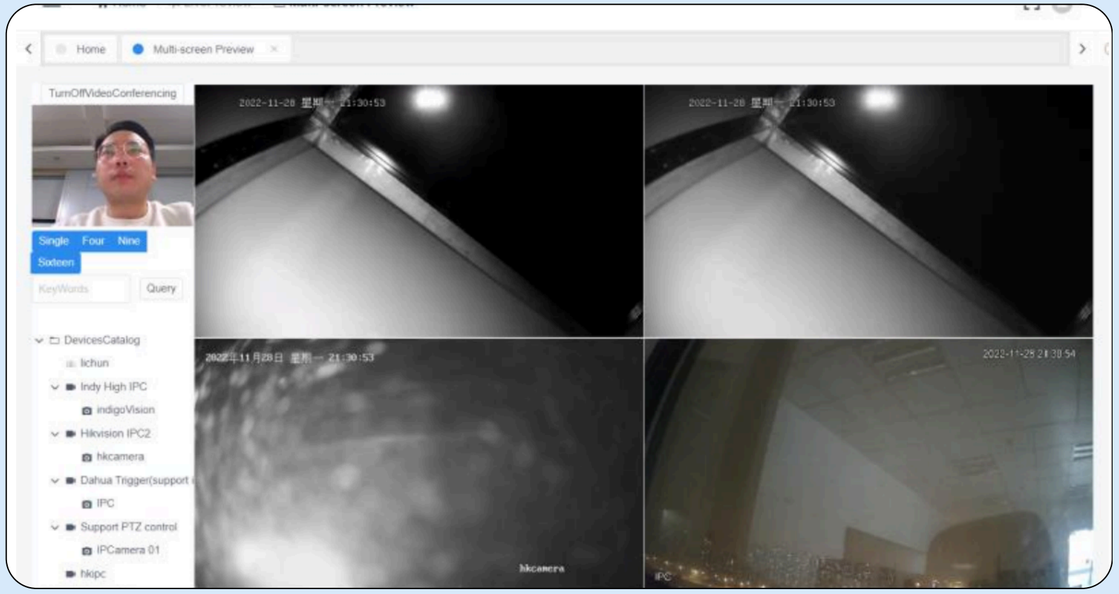
https://meet.jit.si/

Related System – Jain & Gupta

- Developed real-time video conferencing using WebRTC
- Discussed layered architecture and performance optimization
- Used WebSocket for signaling and UDP for media transmission
- Emphasized signaling reliability and low-latency delivery

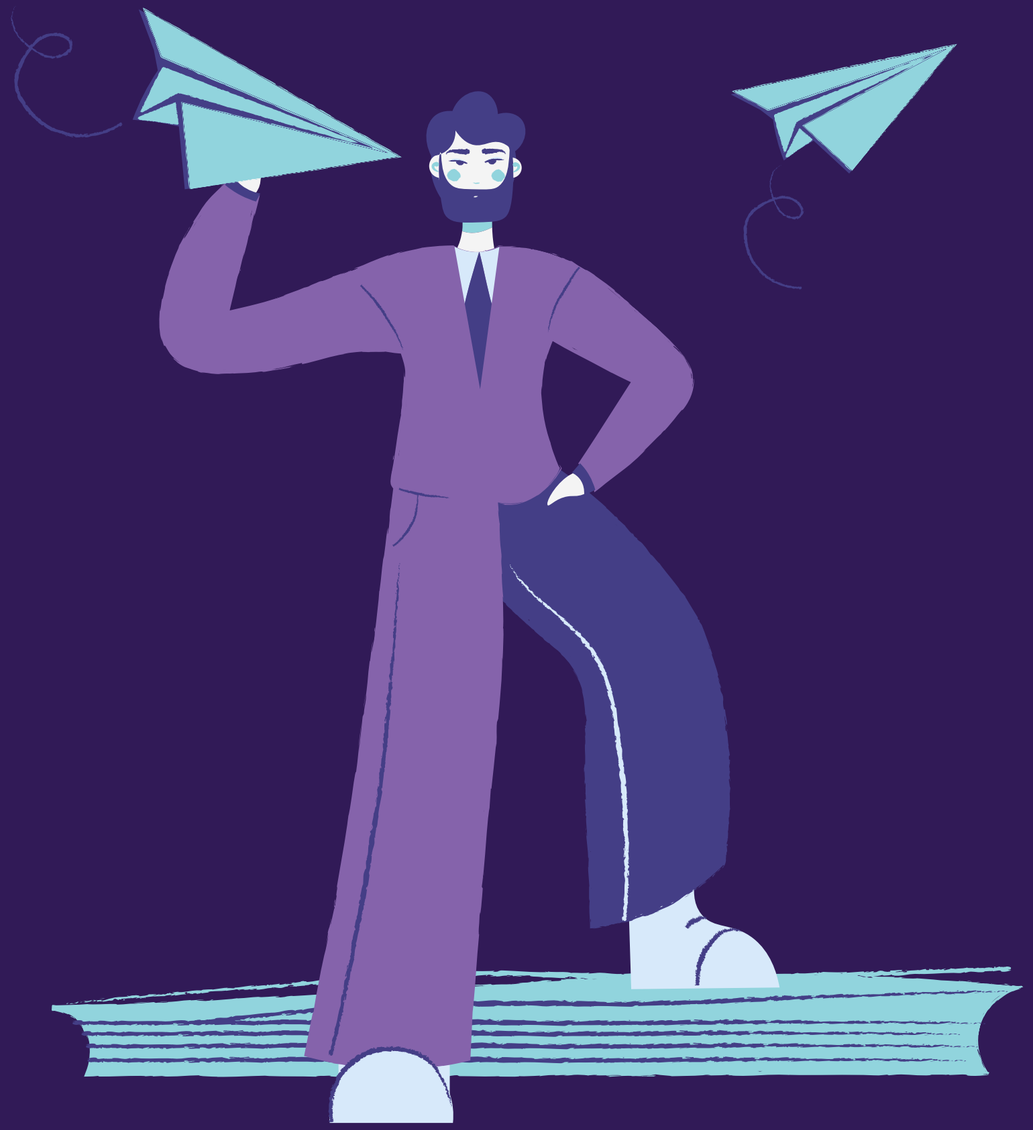
🧠 Supported our use of ActionCable + WebRTC stack





Design and Implementation of WebRTC Video Conference System Structure Compatible with GB/T28181 Devices*

https://www.researchgate.net/publication/367536295_Design_and_Implementation_of_WebRTC_Video_Conference_System_Structure_Compatible_with_GBT28181_Devices



Methodology

- System Overview
  - WebRTC
  - ActionCable & WebSocket
  - Rails + JavaScript
  - Docker Container
- Methodology

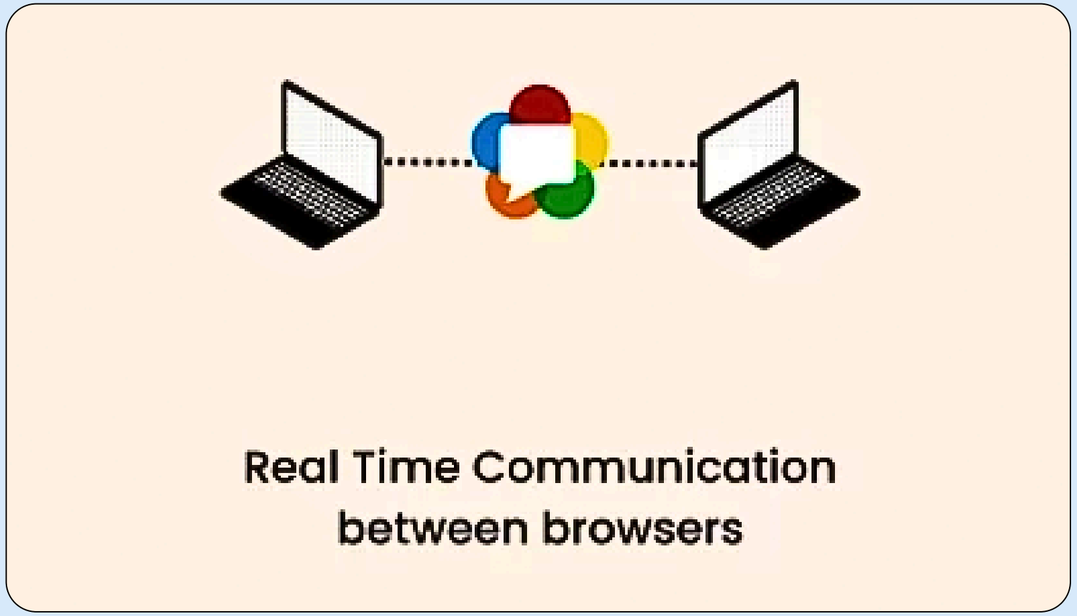
System Overview

- Real-time livestream and chat system built with Ruby on Rails
- Backend:
  - One-way WebRTC video streaming (streamer to viewers)
  - ActionCable WebSocket signaling and real-time chat
- Frontend:
  - Rails views for server-rendered UI
  - JavaScript for media control and interactivity
  - Bootstrap for responsive design
- Fully containerized using Docker with:
  - PostgreSQL (database)
  - Redis (WebSocket and session support)
  - Ensures consistent development and deployment
- ActionCable handles:
  - Session signaling (SDP offers/answers, ICE)
  - Low-latency chat messaging
- WebRTC enables peer-to-peer media transmission

WebRTC – P2P Streaming

- Browser-based API for real-time audio, video, and data sharing
- Streamer and viewers connect using peer-to-peer (P2P) links
- ICE (Interactive Connectivity Establishment) finds best connection path
- STUN helps devices discover their public IP address behind NAT
- TURN acts as a relay server when direct connection fails
- Signaling (using WebSocket) exchanges:
- SDP Offers/Answers – describe media and capabilities
- ICE Candidates – describe network paths

🧠 Delivers media directly browser-to-browser with no central media server

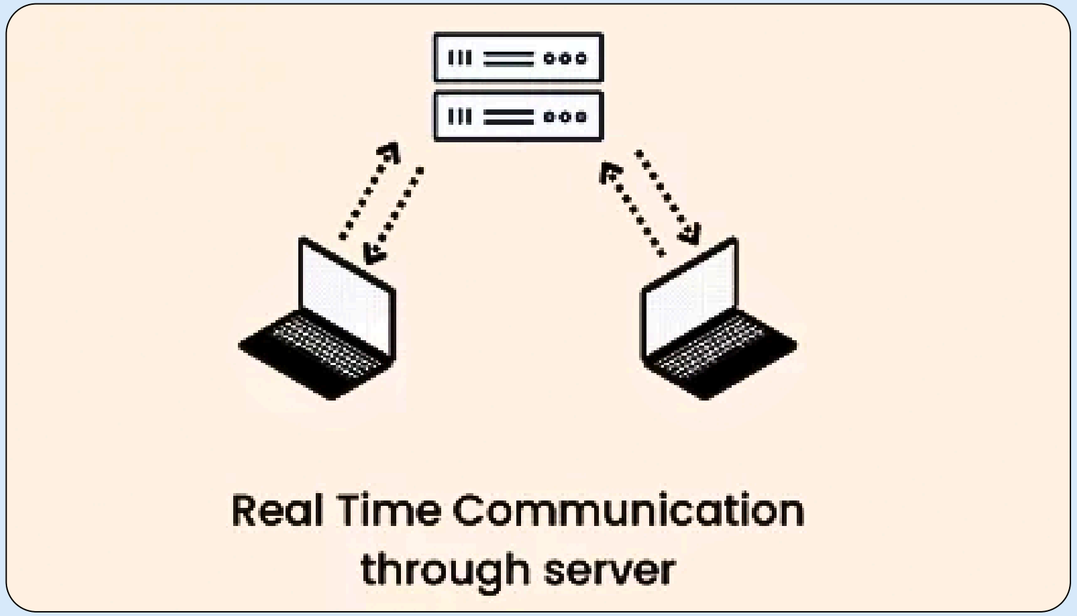


https://www.apizee.com/what-is-webrtc.php

ActionCable & WebSocket

- Built-in WebSocket framework in Ruby on Rails
- Keeps a persistent connection between client and server
- Handles WebRTC signaling (SDP, ICE candidate exchange)
- Enables real-time chat via publish-subscribe channels
- Uses Channels to group users (like chat rooms or streams)
- Sends/receives data instantly — no need to refresh

🧠 Allows chat and signaling to happen instantly over the same live connection



https://www.apizee.com/what-is-webrtc.php

Rails + JavaScript

- Frontend rendered using Rails views (ERB)
- Embedded JavaScript for interactivity and control
- Bootstrap used for responsive layout
- Simple, unified monolithic experience

🧠 Combines logic, layout, and interaction in one framework

Docker Container

- Entire stack containerized with Docker
- Includes PostgreSQL, Redis, and Rails app
- Ensures consistent behavior across systems

- Uses docker-compose.yml to run multiple services
  - Includes:
  - app_web: Rails app container
  - app_db: PostgreSQL database
  - redis: for ActionCable messaging
  - app_pgadmin: DB admin tool for PostgreSQL
- Shared network and volumes for connectivity and persistence


```yaml
services:
  app_web:
    build: .
    ports:
      - "3003:3000"
    depends_on:
      - app_db
      - redis
    volumes:
      - .:/rails

  app_db:
    image: postgres
    environment:
      - POSTGRES_USER=admin
      - POSTGRES_PASSWORD=password

  redis:
    image: redis:alpine
```

Methodology

ChatStream was developed using a modular, incremental approach focused on delivering a reliable, interactive academic livestream system.

The backend was built with Ruby on Rails, while the frontend used JavaScript and WebRTC for real-time video communication. The system was containerized with Docker Compose, enabling consistent development and deployment.

Key features include user authentication, chat messaging with ActionCable, and peer-to-peer video streaming supported by STUN, TURN, and ICE protocols. Rigorous LAN testing ensured robustness and performance.

1. DOCKER SETUP

Used Docker Compose to orchestrate services like Rails, PostgreSQL, Redis, and PgAdmin, ensuring reproducibility across environments.

2. AUTHENTICATION

Implemented Devise for secure sign-up/login, with access filters to restrict chat and streaming to authenticated users only.

3. REAL-TIME CHAT

Configured ActionCable (WebSocket) and Redis for responsive, real-time chat within each chat room using a publish-subscribe model.

4. VIDEO STREAMING

Streamed video using getUserMedia() and WebRTC peer-to-peer connections, with ICE, STUN, and TURN to handle NAT traversal.

5. USER INTERFACE

Used Bootstrap for layout, Stimulus + Turbo Stream for live updates, and tested performance over LAN with multiple devices to ensure responsiveness and robustness.

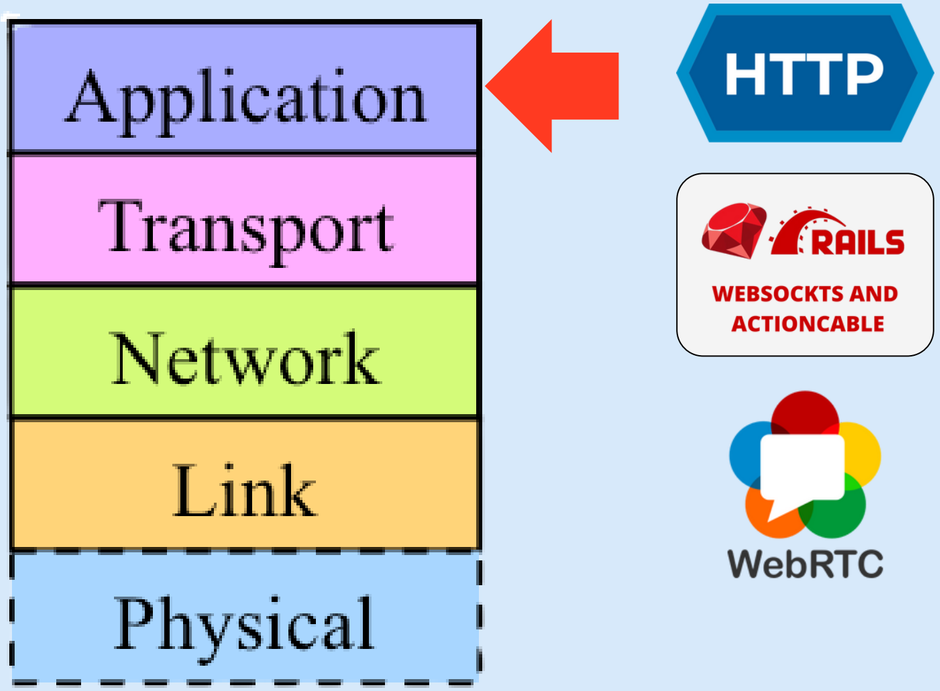
Internet Protocol Stack

- Application Layer
- Transport Layer
- Network Layer
- Link Layer
- Physical Layer

1.Application Layer – Chat & Stream Control

- HTTP/HTTPS for page requests and asset delivery
- WebSocket (via ActionCable) for real-time messaging and signaling
- WebRTC for peer-to-peer video/audio
  - SDP for media negotiation
  - ICE for path selection
  - STUN for public IP discovery
  - TURN for relay fallback
- Runs in browser without plugins

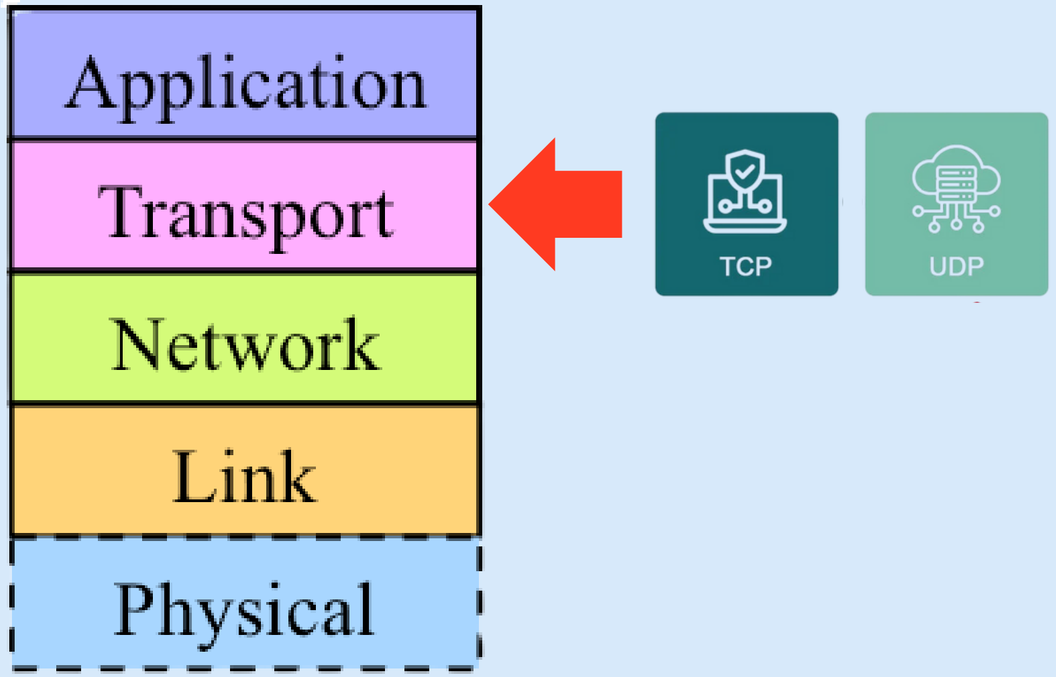
🧠 Handles user interaction, media setup, and real-time communication



2.Transport Layer – Reliable vs Real-Time Transfer

- TCP (chat & signaling)
  - Used by WebSocket/ActionCable
  - Reliable, ordered, error-checked
- UDP (video & audio)
  - Used by WebRTC
  - Faster, no retransmission delays
  - Optional FEC and retransmit support

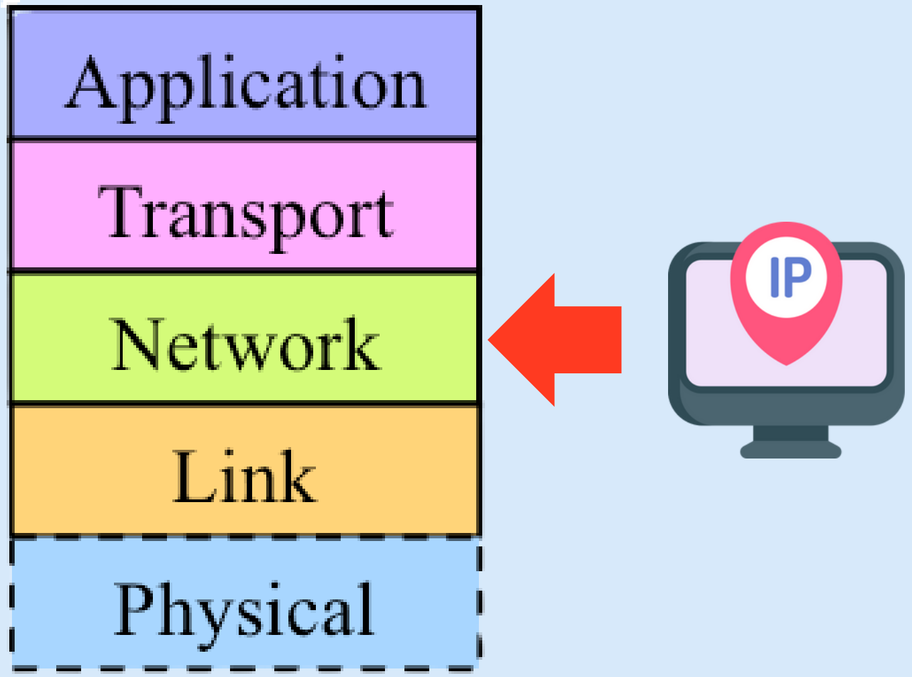
🧠 Chat requires accuracy, video prioritizes speed



3.Network Layer – Addressing & Routing

- Uses IPv4 for addressing devices
- Private IPs for local testing (e.g., 192.168.x.x)
- Public IPs in production (via Ngrok or cloud)
- ICE gathers local, reflexive, and relay IPs

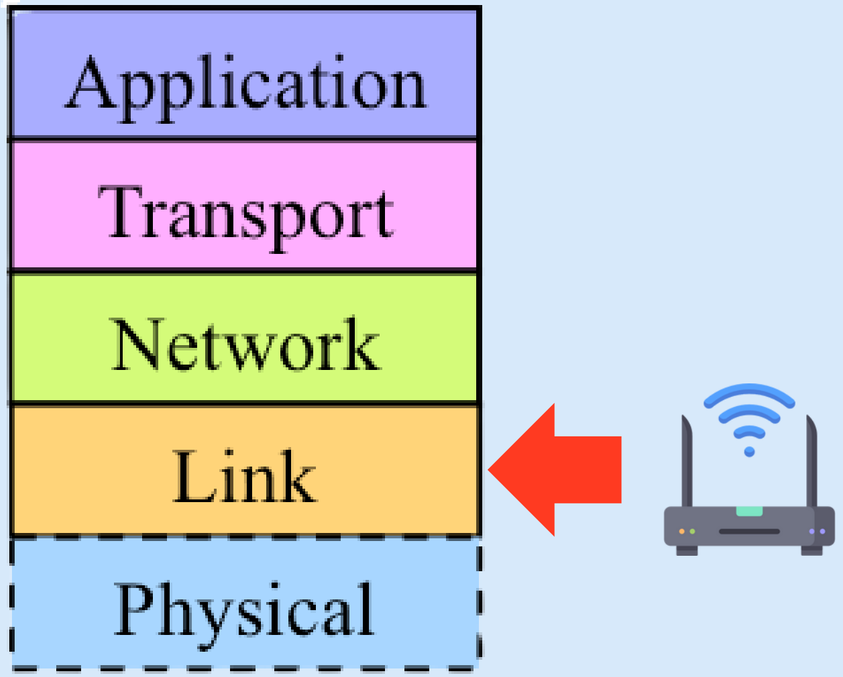
🧠 Connects devices across LAN and internet for real-time delivery



4.Link Layer – Local Network Communication

- Uses Wi-Fi or Ethernet interfaces
- Responsible for frame-level transmission
- MAC addresses identify devices
- LAN testing: multiple devices on same network

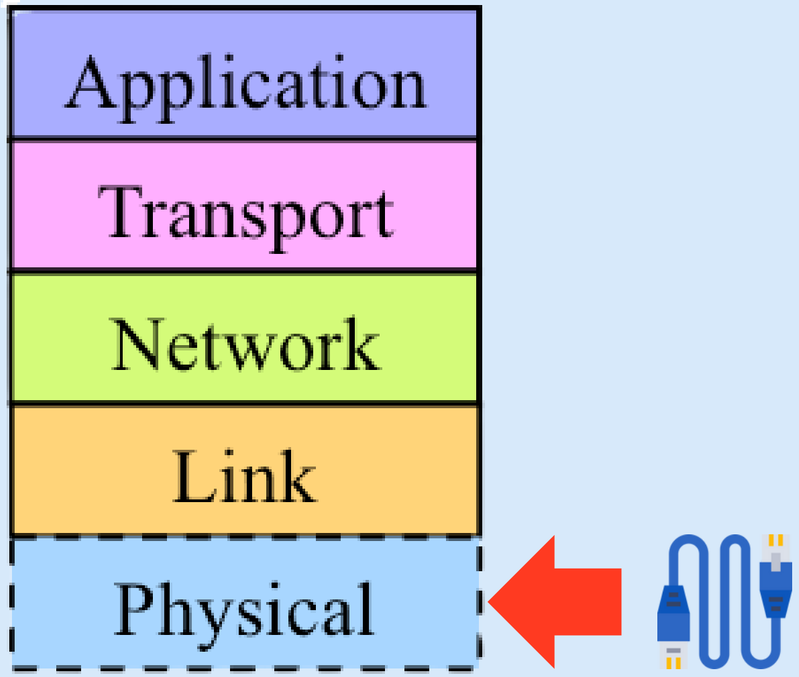
🧠 Makes sure data gets to the right device in the local area



5.Physical Layer – Transmission Medium

- Uses Wi-Fi signals or Ethernet cables
- Carries binary data (0s and 1s)
- Managed by OS and hardware (e.g., network card)
- No direct development here, but foundational to all layers

🧠 Invisible but critical — enables all higher communication layers



Results & Conclusion

- Application Layer
- Transport Layer
- Network Layer
- Link Layer
- Physical Layer

Results

ChatStream successfully delivers a modular and functional livestream system with real-time messaging and user authentication.

The application follows Rails conventions with clear separation of concerns using views and partials. This architecture keeps the system maintainable, secure, and ready for future features like multi-user streaming or moderation.

1.STABLE STREAMING

Peer-to-peer video works reliably using WebRTC in the livestream interface.

2.REAL-TIME CHAT

Messages update instantly via Turbo Stream and ActionCable.

3.SECURE AUTHENTICATION

Users must log in via Devise to access chat and streaming.

4.LOCAL NETWORK TESTED

System runs smoothly on LAN across web browsers.

5.CLEAN VIEW STRUCTURE

Rails views are separated: chat, video, auth which make it easy to manage.

6.NO PAGE RELOADS

Chat form uses AJAX-like updates with Turbo with smooth UX.

Challenges

During development, we faced browser security restrictions that blocked access to camera and mic over local IPs due to lack of HTTPS. These limitations, enforced at the Application Layer, required creative workarounds to enable peer-to-peer streaming during testing.

1. **HTTPS REQUIRED**

Browsers block getUserMedia() unless using secure HTTPS.

2. **APP LAYER BLOCK**

Even though TCP/IP works, restrictions are enforced by the browser layer.

3. **CHROMIUM LIMIT**

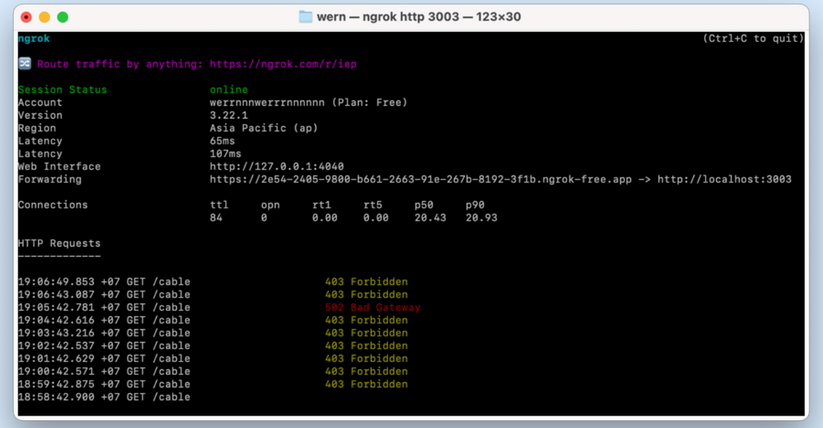
Chrome, Edge, and Brave do not allow insecure local IP access.

4. **LOCALHOST**

Localhost is always treated as secure, used for most testing.

Solutions

To overcome these restrictions, we used secure tunneling and developer configurations. These steps enabled testing across devices while reinforcing the importance of secure contexts in modern web APIs



1. **FIREFOX FLAGS**

Enabled about:config flags to allow insecure media access.

2. **NGROK TUNNELING**

Created HTTPS tunnels from localhost to allow LAN testing.

3. **SECURE WEB API**

Learned how security policies protect users by default.

4. **FUTURE TESTING**

These techniques can support future development in similar scenarios.

Discussion

ChatStream’s design and architecture were validated through LAN-based testing.

WebRTC and ActionCable ensured real-time performance.

The monolithic Rails structure and Docker containerization simplified development and deployment.
While effective for academic livestreaming, the chosen stack also introduced limitations in scalability and frontend flexibility.

1.STABLE PERFORMANCE

Stable peer-to-peer video & chat in LAN tests

2.RAILS STRUCTURE

Rails MPA simplified integration of real-time features

3.REAL-TIME MESSAGING

ActionCable enabled low-latency chat with tight Rails coupling

4.VIDEO STREAMING

WebRTC handled UDP-based streaming with ICE/STUN/TURN

5.CONTAINERIZED

Docker ensured consistent environments

6.LIMITATIONS & TRADE-OFFS

- SPA frameworks offer richer frontend
- ActionCable has scalability limits

Future Work

To enhance ChatStream’s academic functionality, future work will focus on expanding interactivity, moderation, and classroom integration.
These improvements aim to transform the platform into a full-featured academic livestreaming tool suitable for large lectures and learning environments.

SCHEDULED CLASSROOM EVENTS

Add a calendar to schedule livestreams and send reminders

TA-ONLY CHAT MODE

Enable chat moderation so only teaching assistants can reply

INTERACTIVE QUIZZES & POLLS

Introduce real-time polls, Q&A, or mini quizzes during streams

EXPORTABLE CHAT LOGS

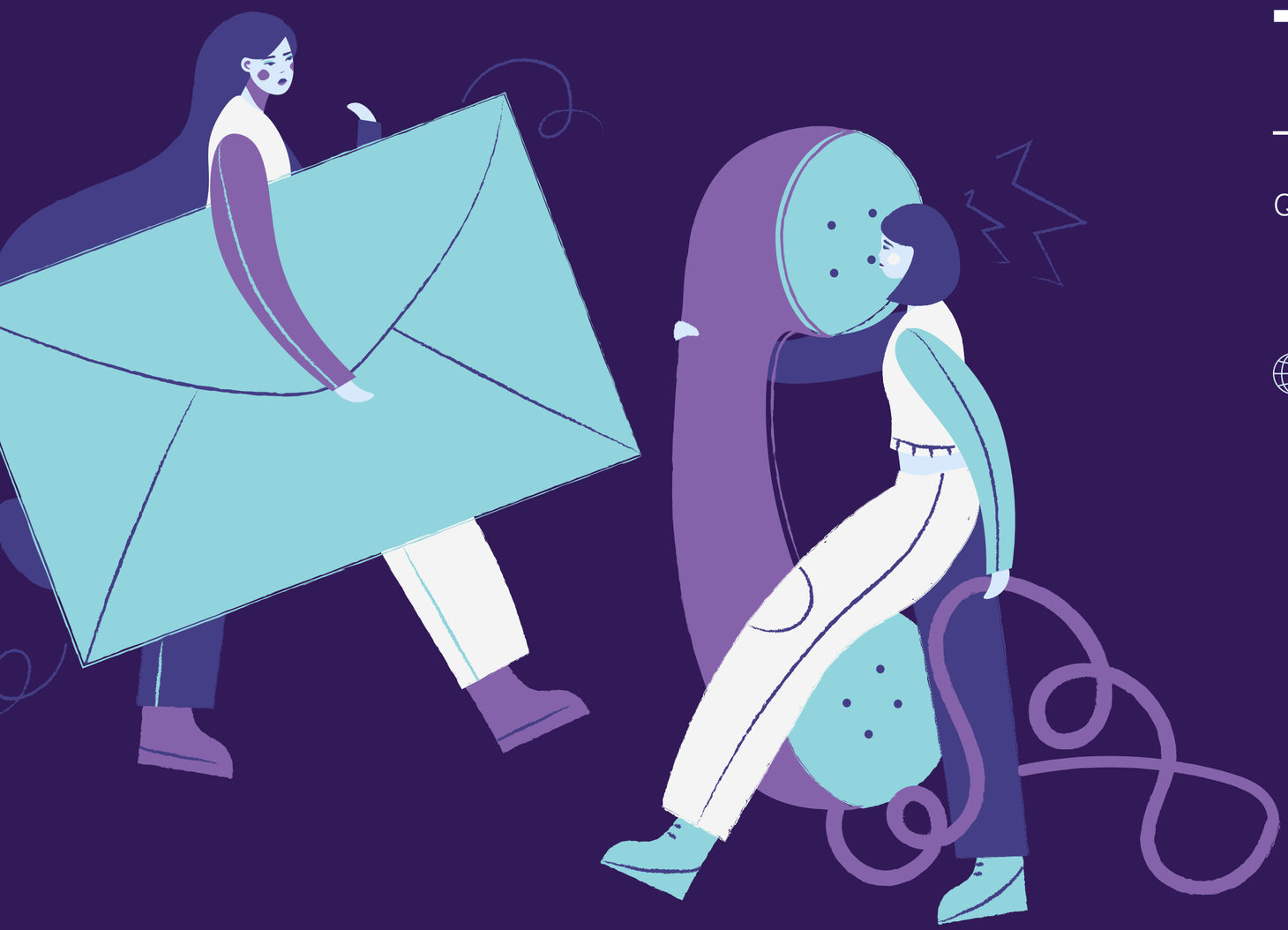
Allow chat history to be downloaded for reviews or attendance

Conclusion

ChatStream successfully delivered a real-time livestream chat system tailored for academic use.

By integrating modern web technologies in a monolithic Rails app, it achieved secure, low-latency video and messaging, tested effectively across local networks.

- Real-time video + chat from one verified streamer
- Integrated WebRTC + ActionCable + Devise for a full-stack solution
- Seamless browser-based use with no external installs
- Focused on security and responsive performance
- Validated through LAN testing and five-layer protocol model
- Ready for future enhancements: screen sharing, multi-streaming, recordings



Thank You!

Q&As

github.com/werrnnnwerrrnnnnnn/chatstream

To access the website

1. **Connect to the same Wi-Fi network**
2. **Open browser with url:**
    - **http://192.168.x.x:3003**