GameStream

Platformă de Live Streaming pentru Jocuri

Sisteme Distribuite

Student: Tuns Andrei
Grupa: CAL IV

18 ianuarie 2026

Cuprins

1 Introducere

1.1 Contextul Proiectului

În era digitală actuală, platformele de streaming live au devenit o componentă esențială a ecosistemului de divertisment online. Platforme precum Twitch, YouTube Live și Facebook Gaming au revoluționat modul în care conținutul video este creat și consumat în timp real.

Proiectul **GameStream** reprezintă o implementare simplificată a unei platforme de live streaming, dezvoltată cu scopul de a demonstra conceptele fundamentale ale sistemelor distribuite într-un context practic și relevant.

1.2 Obiectivele Proiectului

Obiectivele principale ale acestui proiect sunt:

1. **Implementarea unei arhitecturi de microservicii** - Demonstrarea modului în care servicii independente pot colabora pentru a livra o funcționalitate complexă.
2. **Utilizarea unui Message Broker** - Implementarea comunicării asincrone între servicii folosind RabbitMQ.
3. **Procesarea video în timp real** - Captarea și transcodarea stream-urilor video folosind FFmpeg.
4. **Stocare distribuită** - Utilizarea MinIO ca soluție de object storage compatibilă S3.
5. **Containerizare și orchestrare** - Deployment-ul întregii aplicații folosind Docker și Docker Compose.

1.3 Funcționalități Implementate

Platforma oferă următoarele funcționalități:

- Captura ecranului direct din browser folosind API-ul `getDisplayMedia()`
- Transmisie video în timp real prin WebSocket
- Conversie automată în format HLS (HTTP Live Streaming)
- Vizualizare stream-uri de pe orice dispozitiv din rețea
- Interfață web modernă și responsivă
- Statistici în timp real pentru streamer

2 Arhitectura Sistemului

2.1 Prezentare Generală

Sistemul este construit pe o arhitectură de microservicii, unde fiecare componentă are o responsabilitate bine definită și comunică cu celelalte prin intermediul unui message broker (RabbitMQ) sau prin apeluri HTTP/WebSocket directe.

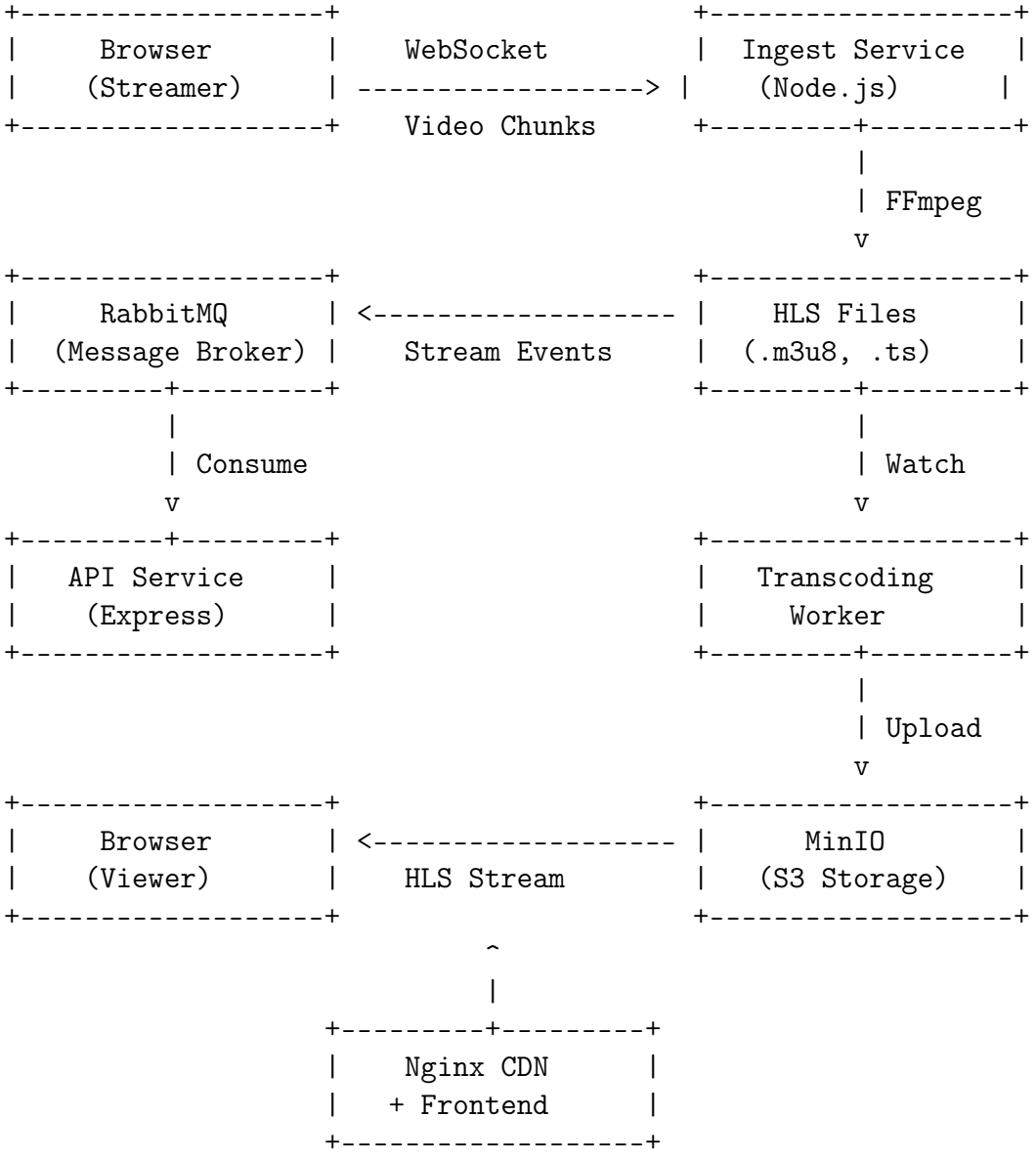

Figura 1: Diagrama arhitecturii sistemului GameStream

2.2 Fluxul de Date

Procesul de streaming urmează următorii pași:

1. **Captură Video** - Streamer-ul inițiază partajarea ecranului folosind API-ul navigator.mediaDevices.getDisplayMedia(). Browser-ul capturează conținutul ecranului și îl comprimă în format WebM folosind MediaRecorder.

2. **Transmisie WebSocket** - Chunk-urile video sunt trimise în timp real către Ingest Service prin conexiune WebSocket securizată (WSS).

3. **Transcodare FFmpeg** - Ingest Service primește datele și le pipe-ează către procesul FFmpeg, care convertește stream-ul WebM în format HLS, generând fișiere `.m3u8` (playlist) și `.ts` (segmente video).

4. **Publicare Evenimente** - La pornirea stream-ului, un eveniment este publicat pe exchange-ul RabbitMQ de tip fanout, notificând toate serviciile interesate.

5. **Stocare și Distribuție** - Transcoding Worker monitorizează directorul HLS și încarcă fișierele în MinIO. Nginx CDN servește fișierele către spectatori.

6. **Vizualizare** - Spectatorii accesează stream-ul prin player-ul Video.js care consumă playlist-ul HLS.

3 Componente și Servicii

3.1 Ingest Service

Responsabilitate: Primirea stream-ului video de la browser și conversia în format HLS.
Tehnologii: Node.js, WebSocket (ws), FFmpeg
Port: 3000 (WSS - WebSocket Secure)

3.1.1 Funcționalitate

```
const https = require('https');
const WebSocket = require('ws');
const fs = require('fs');

const server = https.createServer({
    cert: fs.readFileSync('/app/ssl/server.crt'),
    key: fs.readFileSync('/app/ssl/server.key')
});

const wss = new WebSocket.Server({ server });
```

Listing 1: Configurare WebSocket Server cu SSL

Serviciul ascultă conexiuni WebSocket și pentru fiecare client:

- Creează un proces FFmpeg dedicat
- Pipe-ează datele primite către FFmpeg stdin
- Publică evenimente pe RabbitMQ la start/stop stream

3.1.2 Configurare FFmpeg

```
const ffmpegArgs = [
    '-i', 'pipe:0',              // Input din stdin
    '-c:v', 'libx264',           // Codec video H.264
    '-preset', 'ultrafast',      // Vitez  maxim  de encoding
    '-tune', 'zerolatency',      // Optimizat pentru laten    minim
    '-f', 'hls',                 // Format output HLS
    '-hls_time', '2',            // Durata segment: 2 secunde
    '-hls_list_size', '10',      // Num r segmente  n  playlist
    '-hls_flags', 'delete_segments+append_list',
    `${outputPath}/playlist.m3u8`
];
```

Listing 2: Parametrii FFmpeg pentru conversie HLS

3.2 Transcoding Worker

Responsabilitate: Monitorizarea fișierelor HLS și încărcarea în MinIO.
Tehnologii: Node.js, Chokidar, MinIO Client

3.2.1 Funcționalitate

Utilizează biblioteca `chokidar` pentru a monitoriza directorul de output HLS și încarcă automat fișierele noi în bucket-ul MinIO.

```
const chokidar = require('chokidar');

const watcher = chokidar.watch('/app/hls_output', {
    persistent: true,
    ignoreInitial: true
});

watcher.on('add', async (filePath) => {
    // Upload fi ier  n  MinIO
    await minioClient.fPutObject(
        bucketName,
        objectName,
        filePath
    );
});
```

Listing 3: Monitorizare fișiere cu Chokidar

3.3 API Service

Responsabilitate: Gestionarea stream-urilor și expunerea API-ului REST.
Tehnologii: Node.js, Express.js, amqplib
Port: 4000

3.3.1 Endpoints

Metodă	Endpoint	Descriere
GET	/api/streams	Returnează lista stream-urilor active
GET	/api/streams/:key	Returnează detaliile unui stream specific
GET	/health	Health check pentru monitorizare

Tabela 1: Endpoints API disponibile

3.3.2 Consumare Evenimente RabbitMQ

API Service consumă evenimente de la RabbitMQ folosind o coadă exclusivă (auto-delete) legată la exchange-ul fanout:

```
const EXCHANGE_NAME = 'stream_events_fanout';

// Declarare exchange fanout
await channel.assertExchange(EXCHANGE_NAME, 'fanout', {
    durable: true
});

// Coad  exclusiv  pentru acest consumer
const q = await channel.assertQueue('', { exclusive: true });
await channel.bindQueue(q.queue, EXCHANGE_NAME, '');

```

```
channel.consume(q.queue, (msg) => {
    const event = JSON.parse(msg.content.toString());
    if (event.type === 'STREAM_STARTED') {
        activeStreams.set(event.streamKey, event);
    }
});
```

Listing 4: Configurare consumer RabbitMQ

3.4 CDN (Nginx)

Responsabilitate: Servirea frontend-ului și a fișierelor HLS, plus reverse proxy pentru API.

Tehnologii: Nginx, SSL/TLS

Port: 8443 (HTTPS)

3.4.1 Configurare Nginx

```
server {
    listen 443 ssl;

    ssl_certificate /etc/nginx/ssl/server.crt;
    ssl_certificate_key /etc/nginx/ssl/server.key;

    # Frontend static
    location / {
        root /usr/share/nginx/html;
        index index.html;
    }

    # Proxy pentru API
    location /api/ {
        proxy_pass http://api-service:4000/api/;
    }

    # Servire HLS cu CORS
    location /hls/ {
        alias /usr/share/nginx/html/hls/;
        add_header Access-Control-Allow-Origin *;
        add_header Cache-Control no-cache;

        types {
            application/vnd.apple.mpegurl m3u8;
            video/mp2t ts;
        }
    }
}
```

Listing 5: Configurare Nginx pentru HLS și API proxy

3.5 RabbitMQ

Responsabilitate: Message broker pentru comunicare asincronă între servicii.

Port: 5672 (AMQP), 15672 (Management UI)

3.5.1 Topologie

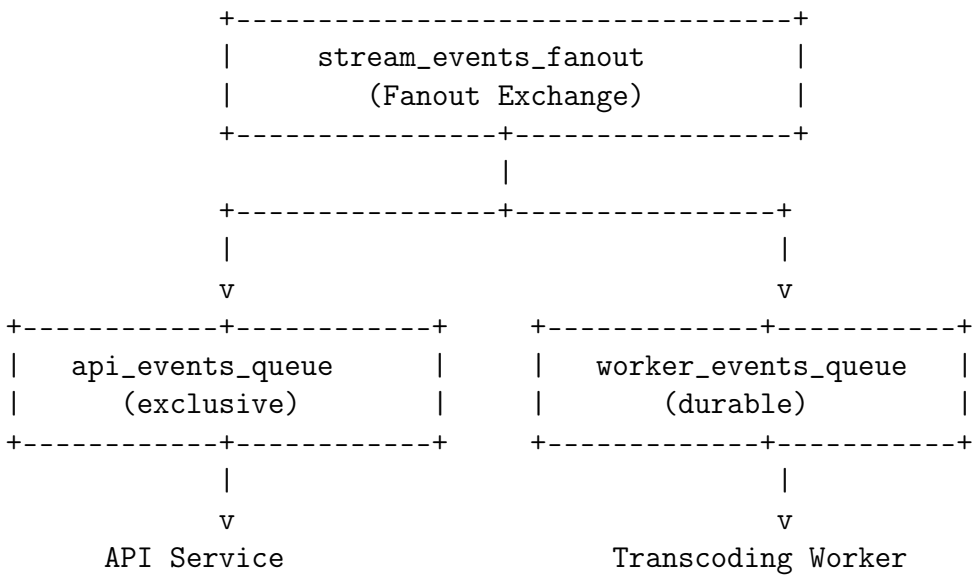

Figura 2: Topologia RabbitMQ cu Fanout Exchange

De ce Fanout Exchange?

Inițial, sistemul folosea o coadă simplă, dar aceasta avea problema că mesajele erau consumate de un singur consumer. Folosind un exchange de tip `fanout`, fiecare mesaj este distribuit către **toate** cozile legate, permițând atât API Service cât și Transcoding Worker să primească același eveniment.

3.6 MinIO

Responsabilitate: Object storage distribuită pentru fișierele HLS.

Port: 9000 (API), 9001 (Console)

MinIO oferă o interfață compatibilă 100% cu Amazon S3, permițând utilizarea acelorași SDK-uri și unelte. În acest proiect, segmentele video HLS sunt stocate în bucket-uri și servite prin Nginx.

4 Tehnologii Utilizate

4.1 Backend și Runtime

Tehnologie	Versiune	Utilizare
Node.js	18-alpine	Runtime JavaScript pentru toate serviciile
Express.js	4.18.2	Framework REST API pentru api-service
WebSocket (ws)	8.14.2	Comunicare bidirecțională real-time
amqplib	0.10.3	Client RabbitMQ pentru Node.js
minio	7.1.3	Client S3 pentru upload în MinIO
chokidar	3.5.3	File system watcher pentru Node.js
cors	2.8.5	Middleware CORS pentru Express

Tabela 2: Tehnologii backend

4.2 Procesare Video

Tehnologie	Utilizare
FFmpeg	Transcodare video din WebM în HLS
HLS (HTTP Live Streaming)	Protocol de streaming adaptiv dezvoltat de Apple
MediaRecorder API	API browser pentru înregistrare media
getDisplayMedia API	API browser pentru screen sharing
Video.js	Player video open-source cu suport HLS

Tabela 3: Tehnologii procesare video

4.3 Infrastructură și DevOps

Tehnologie	Versiune	Utilizare
Docker	Latest	Containerizare servicii
Docker Compose	Latest	Orchestrare multi-container
Nginx	Alpine	Reverse proxy, CDN, HTTPS termination
RabbitMQ	3-management	Message broker cu UI de management
MinIO	Latest	Object storage compatibil S3

Tabela 4: Tehnologii infrastructură

4.4 Frontend

Tehnologie	Versiune	Utilizare
HTML5/CSS3/JS	ES6+	Interfață utilizator
Bootstrap	5.3.2	Framework CSS responsive
Video.js	8.6.1	Player video HLS
Inter Font	Google Fonts	Tipografie modernă

Tabela 5: Tehnologii frontend

5 Docker și Containerizare

5.1 Docker Compose

Întreaga aplicație este orchestrată folosind Docker Compose, care definește toate serviciile, rețelele și volumele necesare.

```
version: '3.8'

services:
  rabbitmq:
    image: rabbitmq:3-management
    ports:
      - "5672:5672"
      - "15672:15672"
    healthcheck:
      test: rabbitmq-diagnostics -q ping

  minio:
    image: minio/minio
    command: server /data --console-address ":9001"
    ports:
      - "9000:9000"
      - "9001:9001"

  ingest-service:
    build: ./ingest-service
    ports:
      - "3000:3000"
    volumes:
      - hls_output:/app/hls_output
      - ./ssl:/app/ssl:ro
    depends_on:
      rabbitmq:
        condition: service_healthy

  transcoding-worker:
    build: ./transcoding-worker
    volumes:
      - hls_output:/app/hls_output
    depends_on:
      - minio
      - rabbitmq

  api-service:
    build: ./api-service
    ports:
      - "4000:4000"
    depends_on:
      rabbitmq:
        condition: service_healthy

  cdn:
    build: ./cdn
    ports:
      - "8443:443"
    volumes:
```

```
      - hls_output:/usr/share/nginx/html/hls:ro

volumes:
  hls_output:
```

Listing 6: Docker Compose - Structura serviciilor

5.2 Dockerfile-uri

5.2.1 Ingest Service

```
FROM node:18-alpine
RUN apk add --no-cache ffmpeg
WORKDIR /app
COPY package*.json ./
RUN npm install --production
COPY src/ ./src/
RUN mkdir -p /app/hls_output /app/ssl
EXPOSE 3000
CMD ["node", "src/index.js"]
```

Listing 7: Dockerfile Ingest Service

5.2.2 CDN (Nginx)

```
FROM nginx:alpine
COPY nginx.conf /etc/nginx/conf.d/default.conf
COPY ssl/ /etc/nginx/ssl/
COPY public/ /usr/share/nginx/html/
RUN mkdir -p /usr/share/nginx/html/hls
EXPOSE 443
CMD ["nginx", "-g", "daemon off;"]
```

Listing 8: Dockerfile CDN

6 Securitate

6.1 HTTPS și WSS

API-ul `getDisplayMedia()` pentru captură de ecran necesită un context securizat (HTTPS). Din acest motiv, întreaga aplicație folosește:

- **HTTPS** pentru frontend și API (port 8443)
- **WSS** (WebSocket Secure) pentru comunicarea cu Ingest Service (port 3000)

6.2 Certificate SSL Self-Signed

Pentru dezvoltare și testare, se folosesc certificate auto-semnate generate cu OpenSSL:

```
openssl req -x509 -nodes -days 365 \
    -newkey rsa:2048 \
    -keyout ssl/server.key \
    -out ssl/server.crt \
    -subj "/CN=localhost" \
    -addext "subjectAltName=IP:192.168.1.100,DNS:localhost"
```

Listing 9: Generare certificate SSL

Parametrul `subjectAltName` permite accesarea aplicației atât prin `localhost` cât și prin adresa IP locală.

7 Concepte de Sisteme Distribuite

Proiectul demonstrează următoarele concepte fundamentale ale sistemelor distribuite:

7.1 Message Queue Pattern

RabbitMQ implementează pattern-ul message queue, permițând:

- Decuplarea producătorilor de consumatori
- Procesare asincronă a evenimentelor
- Persistența mesajelor în caz de eșec

7.2 Microservices Architecture

Fiecare serviciu:

- Are o singură responsabilitate
- Poate fi dezvoltat și deployat independent
- Comunică prin interfețe bine definite (HTTP, WebSocket, AMQP)

7.3 Horizontal Scaling

Arhitectura permite scalarea orizontală a worker-ilor:

```
docker-compose up -d --scale transcoding-worker=3
```

7.4 Fault Tolerance

- Mesajele RabbitMQ persistă în caz de eșec al consumer-ului
- Serviciile se reconectează automat la RabbitMQ
- Health checks asigură pornirea în ordinea corectă

7.5 Service Discovery

Docker Compose oferă DNS intern automat, permițând serviciilor să se adreseze prin nume (ex: `rabbitmq`, `minio`, `api-service`).

8 Ghid de Instalare

8.1 Cerințe

- Docker Desktop instalat și pornit
- Git
- OpenSSL (pentru generarea certificatelor)
- Browser modern (Chrome, Firefox, Edge)

8.2 Pași de Instalare

1. Clonarea repository-ului

```
git clone https://github.com/markoqaq/proiect-sdi.git
cd proiect-sdi
```

2. Aflarea adresei IP locale (Windows PowerShell)

```
(Get-NetIPAddress -AddressFamily IPv4 |
    Where-Object { $_.InterfaceAlias -notlike "*Loopback*" }
).IPAddress
```

3. Generarea certificatelor SSL

```
mkdir ssl
openssl req -x509 -nodes -days 365 -newkey rsa:2048 \
    -keyout ssl/server.key -out ssl/server.crt \
    -subj "/CN=localhost" \
    -addext "subjectAltName=IP:YOUR_IP,DNS:localhost"
```

4. Copierea certificatelor

```
cp -r ssl cdn/
cp -r ssl ingest-service/
```

5. Pornirea serviciilor

```
docker-compose up --build -d
```

6. Verificare

```
docker-compose ps
```

8.3 Accesare

- **Frontend**: `https://YOUR_IP:8443`
- **RabbitMQ Console**: `http://localhost:15672` (admin/admin123)
- **MinIO Console**: `http://localhost:9001` (minioadmin/minioadmin123)

9 Utilizare

9.1 Pentru Streamer

1. Accesează `https://YOUR_IP:8443`
2. Acceptă certificatul SSL self-signed
3. Click pe **"Transmite Live"**
4. Setează titlul stream-ului
5. Click **"Începe Stream-ul"**
6. Selectează ecranul sau fereastra de partajat
7. Stream-ul este acum LIVE!

9.2 Pentru Spectator

1. Accesează aceeași adresă din browser (poate fi de pe alt dispozitiv)
2. În pagina principală vezi lista stream-urilor active
3. Click pe un stream pentru a-l viziona
4. Player-ul HLS pornește automat

10 Structura Proiectului

```
proiect-sdi/
|-- docker-compose.yml          # Orchestrare containere
|-- README.md                   # Documentatie scurta
|-- documentatie.tex            # Aceasta documentatie
|
|-- ingest-service/             # Serviciu primire stream
|   |-- Dockerfile
|   |-- package.json
|   |-- ssl/                    # Certificate SSL
|   +-- src/
|       +-- index.js
|
|-- transcoding-worker/         # Worker procesare video
|   |-- Dockerfile
|   |-- package.json
|   +-- src/
|       +-- index.js
|
|-- api-service/                # API REST
|   |-- Dockerfile
|   |-- package.json
|   +-- src/
|       +-- index.js
|
+-- cdn/                        # Frontend + CDN
    |-- Dockerfile
    |-- nginx.conf
    |-- ssl/                    # Certificate SSL
    +-- public/
        +-- index.html
```

Listing 10: Structura directoarelor

11 Concluzii

11.1 Obiective Atinse

Proiectul GameStream demonstrează cu succes implementarea unei platforme de live streaming folosind concepte moderne de sisteme distribuite:

- **Arhitectură de microservicii** - 4 servicii independente care colaborează
- **Message broker** - RabbitMQ cu fanout exchange pentru distribuție evenimente
- **Containerizare** - Docker și Docker Compose pentru deployment simplu
- **Procesare video** - FFmpeg pentru transcodare în timp real
- **Stocare distribuită** - MinIO ca soluție S3-compatibilă

11.2 Provocări Întâmpinate

1. **HTTPS pentru getDisplayMedia** - API-ul de screen capture necesită context securizat, ceea ce a impus generarea de certificate SSL self-signed.
2. **Distribuție mesaje RabbitMQ** - Inițial, folosind o coadă simplă, mesajele erau consumate de un singur serviciu. Soluția a fost migrarea la exchange fanout.
3. **Sincronizare servicii** - Health checks în Docker Compose pentru a asigura pornirea în ordinea corectă.

11.3 Îmbunătățiri Viitoare

- Autentificare și autorizare utilizatori
- Chat în timp real pentru spectatori
- Transcoding adaptiv (multiple rezoluții)
- Deployment în cloud (Kubernetes)
- Înregistrare și replay stream-uri

12 Bibliografie

1. Docker Documentation - https://docs.docker.com/
2. RabbitMQ Tutorials - https://www.rabbitmq.com/getstarted.html
3. MinIO Documentation - https://min.io/docs/
4. FFmpeg Documentation - https://ffmpeg.org/documentation.html
5. HTTP Live Streaming - https://developer.apple.com/streaming/
6. WebSocket API - https://developer.mozilla.org/en-US/docs/Web/API/WebSocket
7. MediaRecorder API - https://developer.mozilla.org/en-US/docs/Web/API/MediaRecorder
8. Video.js - https://videojs.com/
9. Node.js Documentation - https://nodejs.org/docs/
10. Nginx Documentation - https://nginx.org/en/docs/