Universitatea Politehnica din București

Facultatea de Automatică și Calculatoare

Aplicații multimedia

Aplicație Web de Livestreaming

Documentație

Autor: Mihăilescu Paul-Constantin 331AA

An: 2023

Cuprins

1. Introducere

Livestreaming-ul reprezinta difuzarea în timp real a evenimentelor către o audiență prin intermediul internetului. În era digitală, livestreaming-ul a devenit tot mai important deoarece permite oamenilor să aibă acces la evenimente de oriunde din lume, indiferent de locația lor.

În acest proiect, este prezentată o aplicație web de livestreaming, care oferă o soluție ușor de utilizat și accesibilă pentru a transmite și viziona evenimente în direct pe diferite dispozitive. Aplicația folosește o conexiune peer-to-peer, ceea ce înseamnă că utilizatorii pot transmite și recepționa livestream-uri direct între dispozitive, fără a fi necesară o infrastructură costisitoare de servere.

Obiectivul principal al aplicației este de a oferi o platformă ușor de utilizat și accesibilă pentru utilizatori cu ajutorul căreia se pot vizualiza evenimente în direct. Prin intermediul acesteia, utilizatorii pot iniția un livestream și genera un cod unic care le permite altor utilizatori să se alăture livestream-ului fără autentificare. Conexiunea peer-to-peer permite o conexiune directă între cel care transmite evenimentul și cel care îl vizualizează, reducând timpii de încărcare și întârzierile în transmisie.

Un alt obiectiv important al aplicației este de a facilita experiența utilizatorilor prin eliminarea necesității de autentificare pentru a transmite sau a vizualiza un eveniment în direct. Această funcționalitate simplifică procesul de utilizare a aplicației și îi permite utilizatorului să înceapă transmisia în direct într-un timp cât mai scurt posibil.

2. Realizări similare

O aplicație similara este Twitch[7], care este considerată ca fiind una dintre cele mai mari platforme de transmisiune în direct a jocurilor, precum și a altor tipuri de conținut, cum ar fi muzică, talk-show-uri și conținut creativ. Twitch le permite utilizatorilor să își deschidă propriile canale și să transmită în direct către fanii lor, care pot urmări și interacționa cu transmisia în timp real. Twitch oferă, de asemenea, funcții precum chat și abonamente, care pot contribui la crearea unei comunități în jurul unui anumit canal sau creator de conținut.

O altă aplicație care oferă funcționalități similare este Discord[1]. Deși este cunoscută în primul rând ca o aplicație de chat și comunicare, Discord oferă, de asemenea, funcționalități de partajare a ecranului și a camerei, care pot fi utilizate în scopuri de livestreaming. Discord permite utilizatorilor să își creeze propriile servere și să invite alte persoane să se alăture, ceea ce poate fi util pentru organizarea de livestream-uri de grup sau pentru a colabora la proiecte creative.

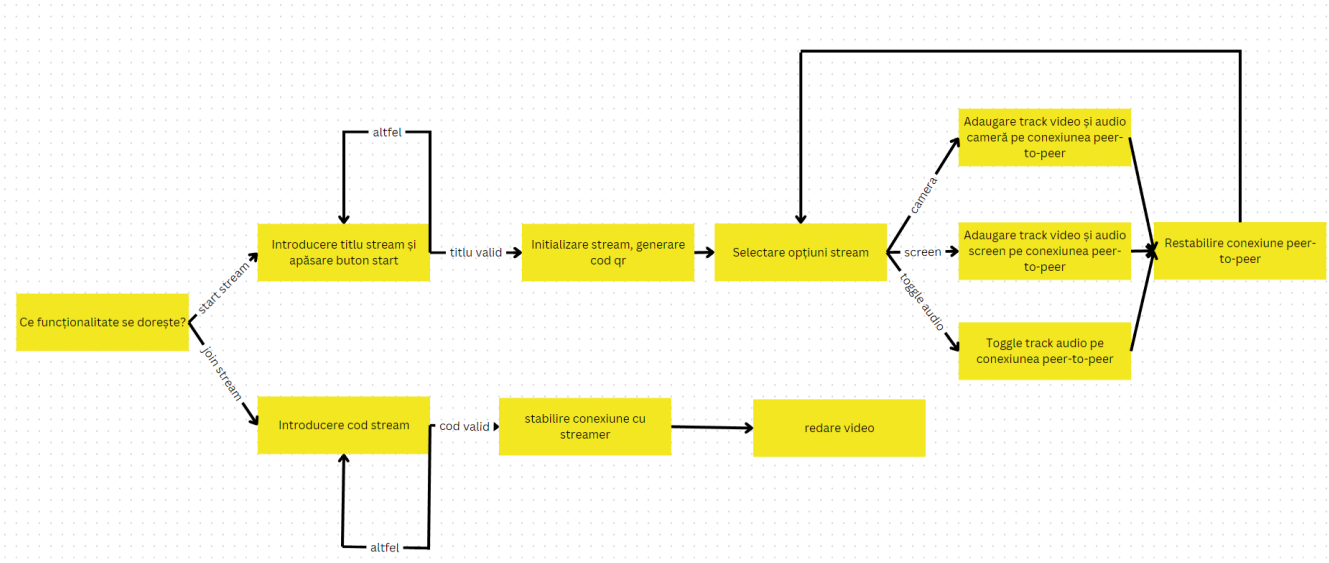
Deși există o serie de aplicații și platforme similare disponibile, aplicația dezvoltată se remarcă prin ușurința de utilizare, lipsa cerințelor de autentificare și conexiunea peer-to-peer.

3. Prezentarea tehnică

Pentru dezvoltarea aplicației de livestreaming am utilizat următoarele librării,tehnologii și framework-uri:

- HTML/CSS: pentru a defini structura paginilor noastre web și pentru a stiliza elementele de pe pagină
- Node.js și npm: pentru dezvoltarea aplicației javascript
- socket.io CDN și npm package: implementarea funcționalității de livestreaming peer-to-peer. Am utilizat atât varianta pentru browser (CDN), cât și varianta pentru Node.js
- Express: creearea serverului HTTP și genstionarea rutelor și cererilor HTTP
- Nodemon: dezvoltarea și testarea aplicației
- WebRTC [8]: api web pentru comunicarea în timp real între două entități
- qr-code-styling.js [3]: generare cod qr pentru livestream

Figură 3-1 Algoritmul general



Detalii conexiune peer-to-peer și modul de implementare

WebRTC (Web Real-Time Communication) este o colecție de API-uri care permit comunicarea în timp real între browsere sau între un browser și un server, permițând aplicațiilor web să ofere transmiterea fluxului audio/video live și partajarea de fișiere peer-to-peer. Una dintre caracteristicile cheie ale WebRTC este capacitatea de a stabili conexiuni peer-to-peer între browsere fără a fi nevoie ca un server să acționeze ca mediator.

Comunicarea peer-to-peer (P2P [4]) se referă la un tip de comunicare în rețea în care două sau mai multe dispozitive comunică direct între ele, fără a fi nevoie de un server central. În WebRTC, comunicarea P2P se realizează prin utilizarea API-ului RTCPeerConnection, care permite ca două browsere să stabilească un canal de date direct între ele.

API-ul RTCPeerConnection permite schimbul de metadate și fluxuri media între două entități, utilizând un **server de semnalizare** pentru a iniția conexiunea. Odată ce serverul de semnalizare a inițiat conexiunea și a făcut schimb de informații necesare, omologii pot stabili o conexiune directă între ei.

Unul dintre principalele avantaje ale comunicării P2P este îmbunătățirea performanțelor și reducerea latenței, deoarece nu este nevoie ca datele să fie transmise către un server înainte de a fi retransmise către destinatar. În plus, comunicarea P2P este mai robustă și mai tolerantă la defecțiuni, deoarece nu există puncte intermediare care pot întrerupe comunicarea.

Pentru a stabili conexiunea peer-to-peer folosind WebRTC, este necesar să se trimită datele necesare între cele trei entități implicate: streamer-ul, viewer-ul și server-ul. Pentru a realiza acest lucru, se poate folosi biblioteca socket.io care permite trimiterea de mesaje bidirecționale în timp real între client și server.

Atunci când streamer-ul pornește un livestream, acesta trimite un mesaj prin socket.io către server pentru a anunța începutul transmisiei. Server-ul (server de semnalizare) primește acest mesaj și creează o cameră virtuală asociată streamer-ului, generând un cod unic de cameră. Acest cod este trimis înapoi prin socket.io către streamer, care îl poate distribui apoi viewer-ilor.

Viewer-ul poate intra apoi în cameră folosind același cod unic, pe care îl trimite prin socket.io către server. Server-ul primește codul și îl folosește pentru a găsi camera virtuală asociată streamer-ului. Server-ul trimite apoi prin socket.io streamerului toate informațiile necesare pentru a se conecta la viewer și a începe transmisia fluxului video către viewer.

În acest proces, socket.io este folosit pentru a transmite mesaje între client și server, inclusiv pentru a trimite datele necesare pentru a stabili conexiunea peer-to-peer între streamer și viewer.

Protocolul WebRTC utilizează protocolul de transport în timp real (RTP) pentru a transmite track-uri audio și video între entități. Calitatea track-urilor la recepție depinde în mare măsură de condițiile de rețea și de lățimea de bandă disponibilă. WebRTC se adaptează la condițiile de rețea disponibile prin ajustarea dinamică a debitului de biți și a calității pieselor audio și video transmise. Astfel, se asigură obținerea celei mai bune calități posibile, având în vedere resursele de rețea disponibile. Cu toate acestea, în cazul în care condițiile de rețea sunt slabe, se poate produce o oarecare degradare a calității la nivelul receptorului. Atunci când un pachet este pierdut în timpul transmiterii, partea care îl primește poate experimenta un “freeze” în fluxul video/audio până când pachetul lipsă este retransmis. Când pachetul lipsă este primit și procesat, fluxul va fi reluat cu următorul cadru așteptat.

Limitări conexiune peer-to-peer

În timp ce conexiunile peer-to-peer au multe avantaje, există unele limitări în ceea ce privește utilizarea lor pentru o aplicație de livestreaming. Una dintre principalele limitări este numărul de vieweri care pot fi conectați. Conexiunile P2P nu sunt ideale pentru streaming-ul pe scară largă,

deoarece fiecare spectator trebuie să stabilească o conexiune separată cu streamerul. Acest lucru poate deveni rapid copleșitor pentru conexiunea la internet și resursele CPU ale streamerului.

În plus, conexiunile P2P să nu sunt potrivite pentru anumite tipuri de conținut, cum ar fi informațiile private sau sensibile care trebuie transmise în siguranță. În astfel de cazuri, un server centralizat poate fi o opțiune mai sigură.

În cele din urmă, pot exista limitări în ceea ce privește tipul de conținut care poate fi transmis prin intermediul conexiunilor P2P. De exemplu, dacă fluxul conține conținut protejat prin drepturi de autor, ar putea fi dificil să se asigure că acesta nu este distribuit ilegal prin intermediul conexiunilor P2P.

Alternativă peer-to-peer

O alternativă la conexiunile peer-to-peer este arhitectura client-server. În această arhitectură, fluxul video este transmis către un server central, care îl transmite apoi vieweri. Această arhitectură are mai multe avantaje față de conexiunile peer-to-peer, cum ar fi o mai bună scalabilitate, fiabilitate și securitate.

Cu o arhitectură client-server, serverul poate gestiona mai multe conexiuni și poate distribui sarcina în mod egal.

În plus, arhitectura client-server este mai sigură decât cea peer-to-peer, deoarece serverul poate valida și autentifica fluxurile de intrare și viewerii și poate preveni orice acces neautorizat sau atac.

Cu toate acestea, un potențial dezavantaj al arhitecturii client-server este că necesită un server mai puternic și mai scump pentru a gestiona traficul, ceea ce poate crește costurile de operare. În plus, ar putea exista o anumită întârziere sau latență între streamer și vieweri din cauza etapei suplimentare de transmitere a videoclipului prin server.

Generarea codului qr

În proiectul nostru, am folosit qr-code-styling.js CDN pentru a genera un cod QR care reprezintă link-ul localhost:5000/watch?code=uniqueroomcode. Codul unic de cameră este generat atunci când un utilizator începe o transmisiune în direct. Prin scanarea codului QR, utilizatorii pot accesa cu ușurință livestream-ul fără a fi nevoiți să tasteze manual codul camerei.

Partajare ecran/cameră

API-ul WebRTC oferă două metode principale de accesare a dispozitivelor media de pe calculatorul unui utilizator: navigator.mediaDevices.getDisplayMedia() și navigator..getUserMedia(). Aceste două metode permit unei aplicații web să capteze imagini video și audio de la camera și microfonul unui utilizator sau de pe ecranul acestuia.

Metoda navigator.getUserMedia() solicită utilizatorului permisiunea de a accesa camera și microfonul. Odată ce permisiunea este acordată, metoda returnează un obiect MediaStream care poate fi utilizat pentru a accesa dispozitivele media ale utilizatorului. Obiectul MediaStream conține unul sau mai multe obiecte MediaStreamTrack, care reprezintă track-urile audio și video individuale.

Metoda navigator.mediaDevices.getDisplayMedia() permite utilizatorului să partajeze ecranul său sau o anumită fereastr. Această metodă solicită, de asemenea, utilizatorului permisiunea de a accesa ecranul său. Odată ce permisiunea este acordată, metoda returnează un obiect MediaStream care reprezintă ecranul utilizatorului.

Ambele metode returnează o promisiune care se rezolvă cu un obiect MediaStream dacă utilizatorul acordă permisiunea sau o respinge dacă permisiunea este refuzată. Odată obținut obiectul MediaStream, acesta poate fi utilizat pentru a transmite conținutul media către alți omologi utilizând API-ul RTCPeerConnection.

Afișare video și aspect ratio

În mod implicit, video-ul este afișat în format 16:9.În cazul în care videoclipul are un alt aspect ratio, acesta este redimensionat pentru a se încadra în raportul de aspect 16:9. Acest lucru înseamnă că, dacă videoclipul are un raport de aspect mai înalt sau mai lat, vor apărea bare negre de o parte și de alta sau în partea de sus și de jos a videoclipului pentru a umple spațiul gol.

Pentru ca videoclipul să fie afișat în mod corespunzător, tag-ul <video> din html este încadrat într-un div cu fundalul negru. Acest lucru ajută ca barele negre să se îmbine cu videoclipul.

Pentru a menține un aspect consistent, videoclipul are o dimensiune fixă de 16:9. Astfel, toate videoclipurile sunt afișate într-un format standard. În plus, acest lucru ajută la prevenirea distorsiunilor sau a întinderii videoclipului, care pot apărea dacă videoclipul este afișat la o dimensiune diferită de rezoluția sa nativă.

4. Configurare și utilizare

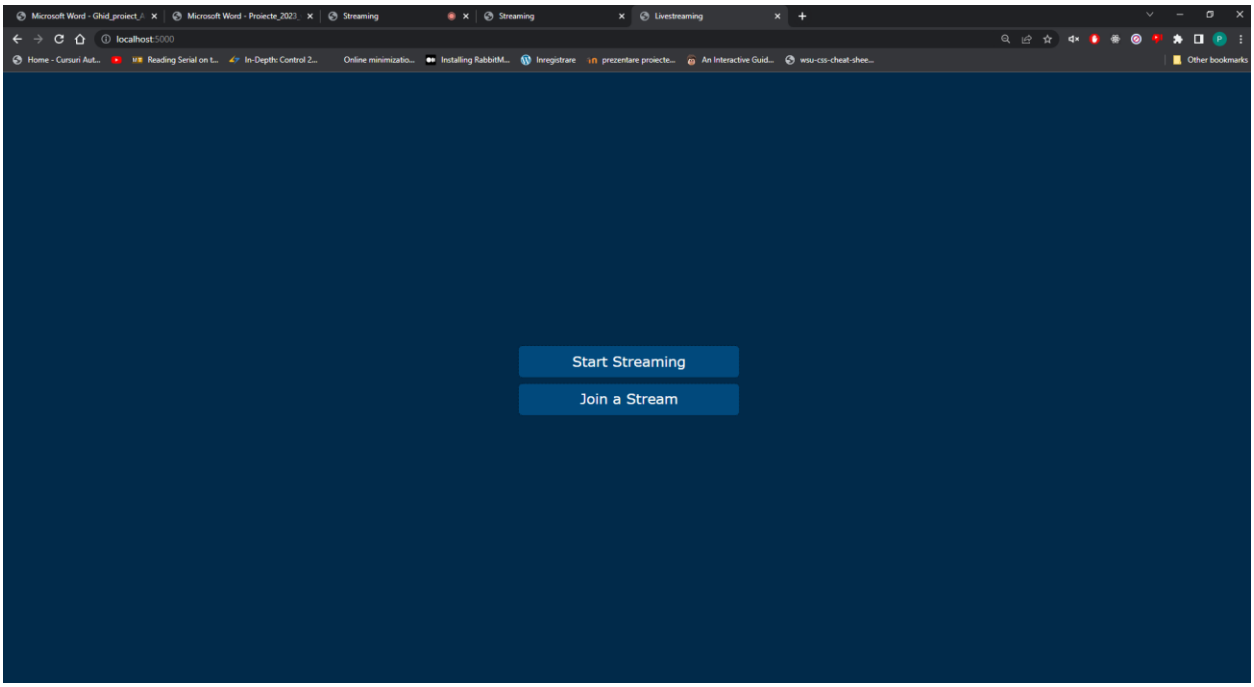
Configurare

Pentru configurarea aplicației este necesară instalarea Node.js și npm. Acestea pot fi descărcate de pe site-ul oficial pentru Node.js. De asemenea, este necesar folosirea unui browser modern precum Chrome și browsere bazate pe Chromium, Firefox sau opera. Aplicația poate fi descărcată de pe repo-ul de Git. Pentru instalarea aplicației trebuie rulată comanda npm install în directorul rădăcina al proiectului, astfel încât să se instaleze toate modulele necesare. Pentru a porni aplicația se rulează comanda npm run start.

Mod de utilizare

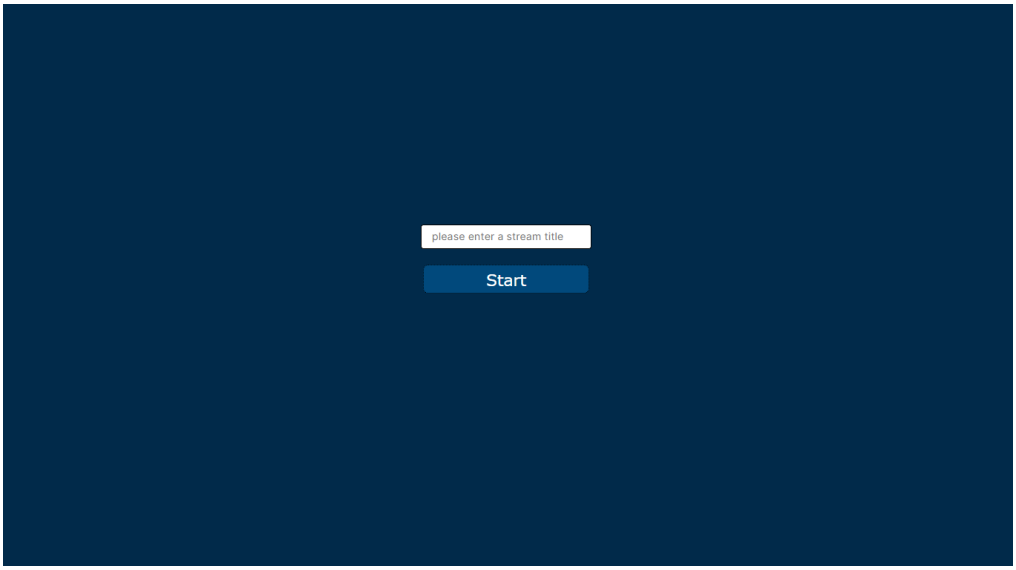
Aplicația pornește pe portul 5000, așadar este necesară accesarea url-ului localhost:5000, ca în figura 4-1.

Figură 4-1 Pagina de home



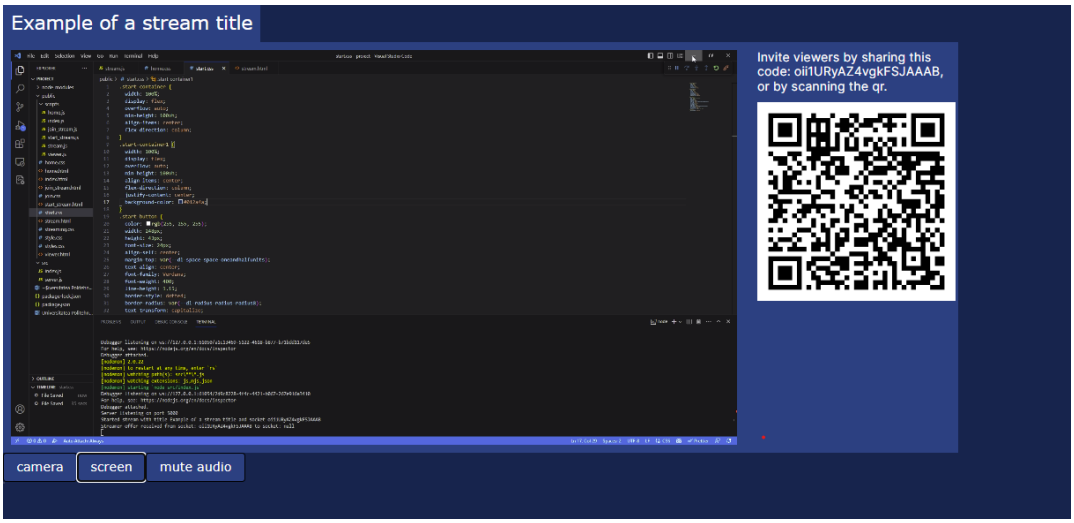
Utilizatorul poate selecta două opțiuni: “Start streaming” și “Join a stream”. În momentul în care utilizatorul selectează “Start streaming”, acesta este redirecționat către o pagină în care trebuie sa introducă titlul stream-ului.

Figură 4-2 Pornirea unui stream



În momentul apăsării butonului start după introducerea unuit titlu, se generează un id pentru cameră care e folosit pentru accesarea stream-ului.

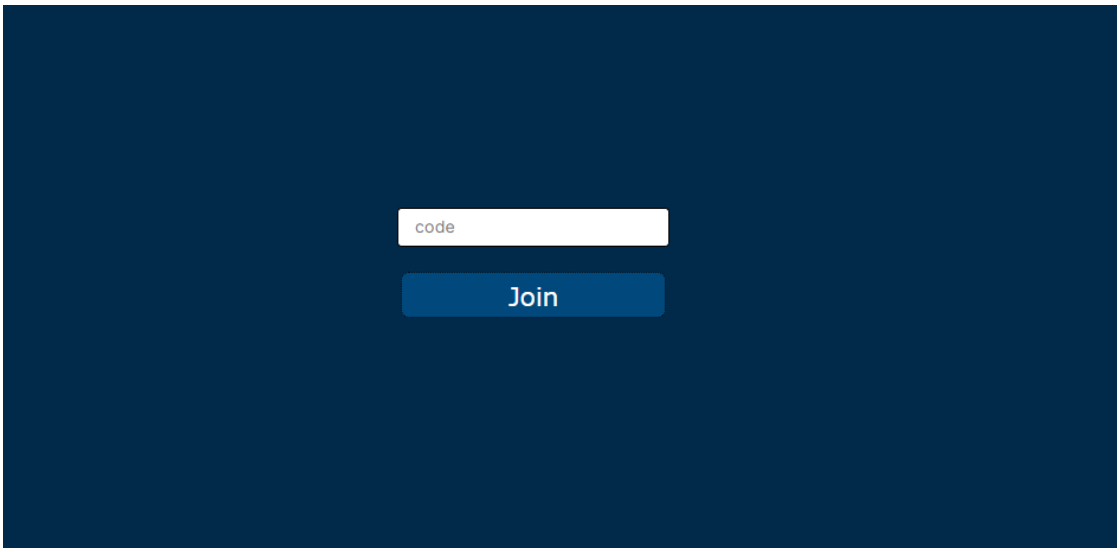
Figură 4-3 Interfața pentru streaming



Utilizatorul este redirecționat la o pagină în care are acces asupra configurării streamului. Astfel, acesta poate să își partajeze camera, un ecran, o aplicație sau un tab din browser prin apăsarea unuia din butoanea “camera” sau “screen”, moment în care apare o căsuță de dialog în browser pentru confirmarea permisiunilor aplicației de a accesa aceste resurse. De asemenea, utilizatorul poate să scoată sunetul de pe stream prin apăsarea butonului “mute audio”.

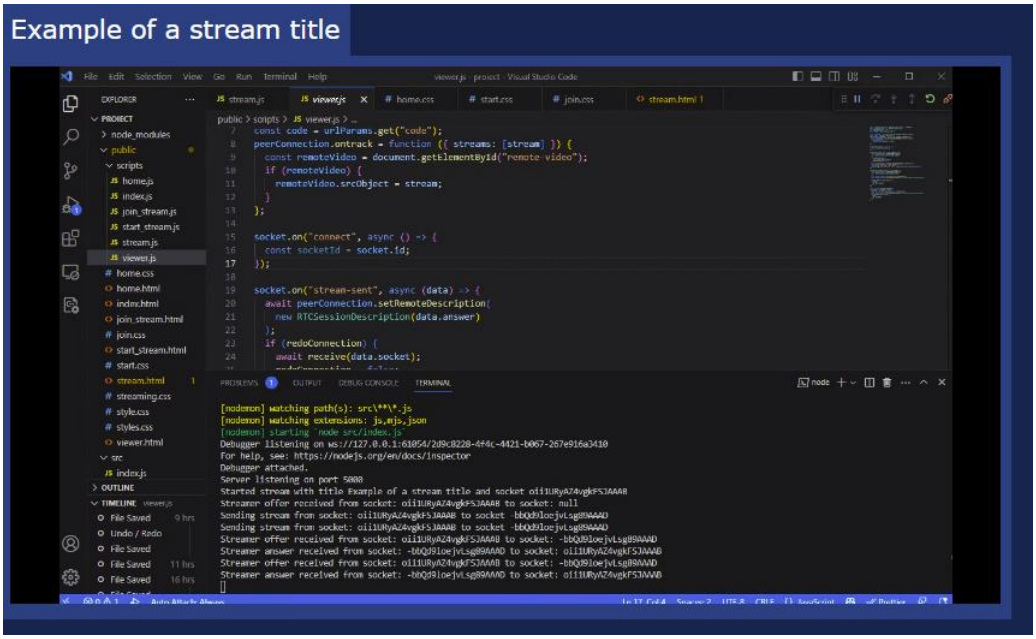
Vizualizarea unui stream se face prin alegerea opțiunii “Join stream” în meniul din figura 4-1 sau direct prin scanarea codului qr. Odată aleasă opțiunea “Join stream”, utilizatorul este redirecționat către o pagină în care poate să introducă codul stream-ului.

Figură 4-4 Meniu vizualizare stream



Dacă codul este valid, atunci se deschide o pagină cu livestream-ul asociat codului.

Figură 4-5Vizualizare stream



5. Concluzii

În concluzie, aplicația de livestreaming dezvoltată a îndeplinit cu succes obiectivele propuse. Am creat o soluție accesibilă și ușor de utilizat pentru transmiterea și vizionarea evenimentelor în direct, cu ajutorul unei conexiuni peer-to-peer. Aplicația nu necesită autentificare și oferă o pornire rapidă a transmisiei.

În ceea ce privește performanța, aplicația este rapidă și stabilă, oferind o calitate bună a stream-ului video. Deși conexiunea peer-to-peer prezintă avantajele detaliate, folosirea acesteia nu este recomandată pentru aplicații de acest gen, datorită faptului că aceasta nu este securizată împotriva atacurilor cibernetice. Astfel, pentru o implementare completă a acestei aplicații ar trebui trimis fluxul live mai întâi către un server, iar abia apoi către alți utlilizatori.

Acest proiect a avut ca scop recapitularea conceptelor de protocoale de comunicații și transmitere video/audio, dar nu reprezintă o implementare validă pentru un program de uz comercial.

6. Referințe bibliografice

1. Discord, https://discord.com/app
2. Generare qr code, https://medium.com/@facucarbonel_97514/how-to-create-a-qr-generator-using-javascript-4b5ce1b6ec27
3. Node.js, https://nodejs.org/en
4. Peer-to-peer, https://en.wikipedia.org/wiki/Peer-to-peer
5. Repo GitHub,
6. TCP, https://en.wikipedia.org/wiki/Transmission_Control_Protocol
7. tutorial Socket.io, https://tsh.io/blog/how-to-write-video-chat-app-using-webrtc-and-nodejs/
8. Twitch, https://www.twitch.tv/
9. WebRTC API, https://developer.mozilla.org/en-US/docs/Web/API/WebRTC_API