MINISTERUL EDUCAȚIEI ȘI CERCETĂRII ȘTIINȚIFICE

UNIVERSITATEA TEHNICĂ
DIN CLUJ-NAPOCA

FACULTATEA DE AUTOMATICĂ ŞI CALCULATOARE
DEPARTAMENTUL CALCULATOARE

Implementarea instrumentelor pentru video colaborare în domeniul educațional folosind tehnologii open source

LUCRARE DE LICENŢĂ

Absolvent:

Coordonator ştiinţific: **S.l.dr.ing. Bogdan Iancu**

2017

MINISTERUL EDUCAȚIEI ȘI CERCETĂRII ȘTIINȚIFICE

UNIVERSITATEA TEHNICĂ
DIN CLUJ-NAPOCA

FACULTATEA DE AUTOMATICĂ ŞI CALCULATOARE
DEPARTAMENTUL CALCULATOARE

DECAN,
Prof. dr. ing. Liviu MICLEA

DIRECTOR DEPARTAMENT,
Prof. dr. ing. Rodica POTOLEA

Absolvent:

Implementarea instrumentelor pentru video colaborare în domeniul educațional folosind tehnologii open source

1. **Enunţul temei:** *Aceasta lucrare urmărește dezvoltarea unui set de instrumente software de video colaborare ce permit cominicarea de la distanta intre profesor si stundeti pe baza elementelor transmise in timp real si a comunicarii realizate cu ajutorul mesajelor de tip text.*

2. **Conţinutul lucrării:** *Introducere, Obiectivele proiectului, Studiu bibliografic, Analiza si fundamentare teoretica, Proiectare in detaliu si implementare, Testare si validare, Concluzii și Bibliografie*

3. **Locul documentării**: Universitatea Tehnică din Cluj-Napoca, Departamentul Calculatoare

4. **Consultanţi**: S.l.dr.ing. Bogdan Iancu

5. **Data emiterii temei:** 1 septembrie 2016

6. **Data predării:** 14 iulie 2017

Absolvent: _____

Coordonator ştiinţific: _____

MINISTERUL EDUCAȚIEI ȘI CERCETĂRII ȘTIINȚIFICE

UNIVERSITATEA TEHNICĂ
DIN CLUJ-NAPOCA

FACULTATEA DE AUTOMATICĂ ŞI CALCULATOARE
DEPARTAMENTUL CALCULATOARE

Declaraţie pe proprie răspundere privind autenticitatea lucrării de licenţă

Subsemnatul(a)_____
_____,
legitimat(ă) cu _____ seria _____ nr. _____
CNP _____, autorul lucrării
_____
_____
_____elaborată în vederea susţinerii examenului de finalizare a studiilor de licență la Facultatea de Automatică și Calculatoare, Specializarea _____ din cadrul Universităţii Tehnice din Cluj-Napoca, sesiunea _____ a anului universitar _____, declar pe proprie răspundere, că această lucrare este rezultatul propriei activităţi intelectuale, pe baza cercetărilor mele şi pe baza informaţiilor obţinute din surse care au fost citate, în textul lucrării, şi în bibliografie.

Declar, că această lucrare nu conţine porţiuni plagiate, iar sursele bibliografice au fost folosite cu respectarea legislaţiei române şi a convenţiilor internaţionale privind drepturile de autor.

Declar, de asemenea, că această lucrare nu a mai fost prezentată în faţa unei alte comisii de examen de licenţă.

In cazul constatării ulterioare a unor declaraţii false, voi suporta sancţiunile administrative, respectiv, *anularea examenului de licenţă*.

Data _____

Nume, Prenume _____

Semnătura

De citit înainte (această pagină se va elimina din versiunea finală):

1. Cele trei pagini anterioare (foaie de capăt, foaie sumar, declaraţie) se vor lista pe foi separate (nu faţă-verso), fiind incluse în lucrarea listată. Foaia de sumar (a doua) necesită semnătura absolventului, respectiv a coordonatorului. Pe declaraţie se trece data când se predă lucrarea la secretarii de comisie.
2. Pe foaia de capăt, se va trece corect titulatura cadrului didactic îndrumător (consultaţi pagina de unde aţi descărcat acest document pentru lista cadrelor didactice cu titulaturile lor).
3. Documentul curent a fost creat în **MS Office 2007.** Dacă folosiţi alte versiuni e posibil sa fie mici diferenţe de formatare, care se corectează (textul conţine descrieri privind fonturi, dimensiuni etc.).
4. **Cuprinsul** începe pe pagina nouă, impară (dacă se face listare faţă-verso), prima pagina din capitolul **Introducere** tot aşa, fiind numerotată cu 1. Pentru actualizarea cuprinsului, click dreapta pe cuprins (zona cuprinsului va apare cu gri), Update field->Update entire table.
5. Vizualizaţi (recomandabil şi în timpul editării) acest document după ce activaţi vizualizarea simbolurilor ascunse de formatare (apăsaţi simbolul **π** din *Home/Paragraph*).
6. Fiecare capitol începe pe pagină nouă, datorită simbolului ascuns Section Break (Next Page) care este deja introdus la capitolul precedent. Dacă ştergeţi din greşeală simbolul, se reintroduce (*Page Layout -> Breaks*).
7. Folosiţi stilurile predefinite (Headings, Figura, Tabel, Normal, etc.)
8. Marginile la pagini nu se modifică (Office 2003 default).
9. Respectaţi restul instrucţiunilor din fiecare capitol.

Cuprins

Capitolul 1. Introducere – Contextul proiectului

În ultima perioadă, datorită dezvoltărilor rapide ale internetului și al tehnologiilor de învățare de la distanta, un model de învățare ce combina metoda clasica de învățare fata în fata cu cea de învățare de la distanta a evoluat. Totodată, aceasta maniera de învățare nu este adoptata în foarte multe instituții de învățământ superior sau în instituții de învățământ postliceale unde se studiază ingineria.

Unul dintre motivele principale care fac ca acest tip de învățare sa fie mai puțin popular este ca profesorii nu sunt obișnuiți sa își construiască cursurile după acest model de învățare ce combina cele doua tipuri de învățare. Un alt motiv ar fi ca o parte mare a profesorilor considera ca metodele de învățare de la distanta nu sunt destul de stabile si ca nu se pretează îndeajuns pentru a ajuta la studierea ingineriei, în special în a tine laboratoare și cursuri de la distanta.

Metoda de învățare ce combină învățarea de la distanță și metodele clasice, presupune ca profesorii dezvolta abilitățile studenților/elevilor și le asignează acestora anumite sarcini pe care aceștia trebuie sa le îndeplinească pentru a promova materia. Sarcina studenților este sa acceseze platforma educationala și întreține o colaborare sincrona cu profesorul. Profesorul răspund la eventualele întrebări ale elevului/studentului și ii oferă suport în situațiile dificile pe care le-a întâmpinat studentul. Totodată, aceștia pot folosi alte elemente multimedia cum ar fii partajarea ecranului sau a unei aplicații, prin care pot să ajute studentul.

Tendințele actuale ale pieței în domeniul educațional vizează soluții eficiente capabile să acopere nevoile de predare și învățare sincrone și asincrone. Piața cere ca aceste soluții să fie ușor de personalizat și livrat în funcție de diferitele cerințe și constrângeri ale beneficiarilor. Prin urmare s-a proiectat un cadru inovativ de învățare colaborativă și s-a implementat un sistem software de clasa virtuala care oferă capabilități de învățare combinate.

Deoarece ne aflăm în era tehnologiei dorim să comunicăm tot mai mult unul cu altul, mai ales dacă persoană cu care dorim să comunicam se află la distanță și nu putem comunica față în față cu această. Acest lucru este comun și domeniului educației, deoarece studenții au nevoie să comunice mai des între ei sau cu un cadru didactic iar un sistem de tip webinar ar fii perfect în aceste cazuri, profesorul sau unul dintre elevi planifică rapid un meeting iar la o anumită oră stabilita împreună de cei care doresc să participe, aceștia să folosească un instrument video comunicare.

În această lucrare s-a dezvoltat si implementau un sistem ce dispune de diferite instrumente de video colaborare cu ajutorul cărora această metodă de învățare ce combină atât elemente clasice de învățare cu cele moderne de învățare de la distanță să fie mai accesibilă și mai ușor de utilizat atât pentru profesori cât și pentru elevi/studenți. S-au implementat aceeste instrumente de video colaborare folosind doar tehnologii și librării oferite gratis de către dezvoltatorii lor.

În cadrul departamentului de dezvoltare a aplicațiilor real-time se pot dezvolta aplicații pentru conferințe online, jocuri în timp real cu mai mulți participanți, chat video, chat clasic (trimiterea de mesaje), transmiterea de fișiere, share screen, determinarea locației unui anumit punct în timp real etc. Cu alte cuvinte, interacțiunea în timp real a devenit un nou standard pentru aplicațiile care se dezvoltă în ultimul timp, deoarece sunt

mult mai interactive, promovează colaborarea atât pe plan profesional, cât și pe cel personal.

Tema “Implementarea instrumentelor pentru video colaborare în domeniul educațional folosind tehnologii open source” conturează modul în care se pot dezvolta instrumente de video colaborare în timp real, colaborare ce este dezvoltată pe toate dispozitivele existente, începând cu pc-uri în aplicații de sine stătătoare sau care rulează din un browser web, dispozitive mobile cu aplicații dedicate atât pe android și iOS cât și pe Windows phone.

Conținutul lucrării de față reprezintă componenta de colaborare sincronă prezentată în articolul [1]. Lucrarea este intitulată “Collaborative learning tools for formal and informal engineering education” și a fost prezentată în cadrul Conferinței Științifică a Studenților din data de 26 iulie 2017.

Capitolul 2. Obiectivele Proiectului

Lucrarea de fata urmărește îmbunatățirea procesului de învățare și adăugarea unor noi metode și instrumente moderne cu ajutorul cărora acest tip de învățare de la distanta sa fie mai accesibil și mai ușor de utilizat atât pentru pentru profesori cat și pentru elevi. Nu în ultimul rând studiul efectuat s-a focalizat și pe construirea acestor instrumente ale sistemului folosind doar tehnologii și librării oferite gratis de către dezvoltatori, având că scop atât minimizarea costurilor acestui sistem, flexibilitatea în ceea ce privește personalizările ce pot fii aduse acestor tehnologii/librării open-source.

Pentru a îndeplini toate obiectivele propuse mai sus este necesară crearea unui sistem compus din mai multe instrumente de video colaborare care să creeze un mediu prielnic pentru realizarea colaborării asincrone între profesori și elevi.

Instrumentele ce compun acest sistem trebuie să aducă benefici atât profesorului cât și elevului și să ușureze folosirea unui sistem de învățare de la distanță împreuna cu modelul de învățare clasic. Nu în ultimul rând lucrarea de față are că scop îmbunătățirea și metodelor de învățământ existente.

Pe lângă cele enunțate mai sus sistemul propus trebuie să fie flexibil și accesibil din toate sistemele de operare disponibile în momentul de față(Windows, Linux, macOS) prin aplicații native oferite pentru fiecare platforma menționată mai sus, dar flexibilitatea se referă și la posibilitatea utilizării aplicației din toate browser-ele compatibile cu tehnologiile utilizate, în momentul de față browserele ce oferă suport pentru WebRTC sunt Google Chrome, Mozilla Firefox, și Operă, urmând ca în perioada următoare și browserele proprietare Microsoft (Edge) și Apple(Safari) să introducă suportul pentru WebRTC.

În concluzie obiectivele principale pe care lucrarea de față le are sunt dezvoltarea unui framework ce conține diverse instrumente de video colaborare care să :

- permită colaborarea sincronă între profesor și studenți, prin utilizarea unui modul de tip video conferință.
- să faciliteze accesul studenților și al profesorilor la aceste instrumente, prin proiectarea unui framework ce poate fi integrat cu platformele educaționale deja existente pe piață cum ar fii Moodle, Blackboard, HyperEdu
- să fie disponibil și să poate să fie utilizat indiferent de sistemul de operare pe care studentul/profesor îl folosesc (Windows, Linux sau MacOs), iar totodată să fie accesibil din orice browser ce oferă suport tehnologia WebRTC.
- să ofere profesorului instrumente de video colaborare ca să ii permită să partajeze cu elevul conținut multimedia (partajare de ecran, partajare de aplicație)
- să permită unui profesor ce moderează o clasă virtuală să distribuie conținut multimedia extern (spre exemplu prezentarea unui video clip de pe YouTube) și să îi permită acestuia să dețină controlul asupra acestui conținut distribuit (show,hide, play, pause, mute, ș.a.ș.m.d)
- instrumentele de video colaborare în timp real să permită folosirea unor calități de înaltă definiție, iar consumul de resurse pentru clienți să fie unul mic
- framework-ul să fie unul generic iar instrumentele pe care le oferă să fie

ușor de personalizat pentru alte domenii, nu doar în domeniul educației ci și în domenii precum:
    - sanate
    - finanțe
    - transporturi
- aplicația trebuie să ofere utilizatorilor posibilitatea să comunice și în bază mesajelor de text, iar acestea să fie stocate într-o baza de date iar mai apoi să poată sa fie creat un istoric al conversațiilor
- unul dintre principalele obiective ale framework-ului face referire la securitate, din acest motiv printre obiective să număra și folosirea versiunii securizată a protocolului de comunicare HTTP, protocolul responsabil de transmiterea datelor de la client la server.
- framework-ul trebuie sa fie scalabil, în cazul în care se dorește utilizarea sa la scara larga cu un număr de utilizatori mare.
- instrumentele de video colaborare trebuie să permită crearea unor sesiuni paralele, iar conținutul multimedia și text partajate să fie disponibil doar utilizatorilor din cameră în carea aceștia sunt conectați

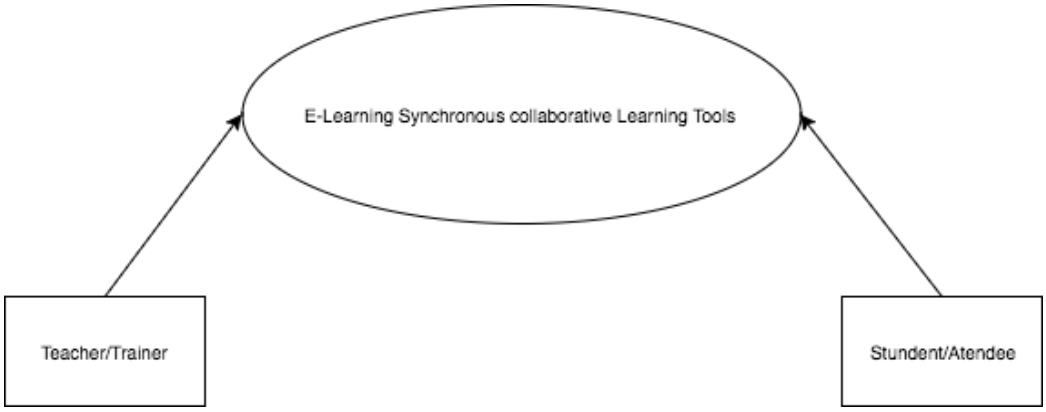

Fig 2.1 Diagrama de context a sistemului

Diagrama de context a aplicației este prezentată în figura 2.1. Din diagramă de context nu poate fii anticipată nici arhitectura generală a aplicației dar nici vreo funcționalitate a sistemului, aceasta prezentând doar domeniul vizat de vizat de aplicație, tipul acestei aplicații și principalii utilizatori pe care lucrarea de față îi are.

Capitolul 3. Studiu Bibliografic

O muncă laborioasă de cercetare a fost efectuată în domeniul educației și al metodelor prin care se pot aduce îmbunatățiri acestui domeniu. Ding și Cao în articolul [2] prezintă un model de învățare RECT (Remote Collaboration Tutor) și implementarea acestui model de învățare în cadrul unui program masteral din domeniul ingineriei software. Modelul de învățare RECT propus se bazează pe învățarea adaptivă și comunicarea față în față. Învățarea adaptivă o este metodă educațională ce presupune utilizarea calculatoarelor pentru comunicarea elev-profesor. Nu în ultimul rând au fost proiectate și instrumente dedicate colaborării asincrone, acestea adaptându-se la capabilitățile platformei de cloud cu scopul de a partaja resurse între între membri aflați la distanță ai unei echipe.

S. Ouya în articolul [3] propune un sistem de învățare colaborativa bazat pe tehnologia WebRTC, în contextul unui mediu de utilizare cu o conexiune Internet instabilă și limitări din punct de vedere al lățimii de bandă. Studenții participă la cursuri transmise în timp real, susținute cu ajutorul instrumentului de video colaborare; totodată aceștia participă la distanță, la activități practice desfășurate prin intermediul instrumentelor de tip laborator virtual. Autorii articolului subliniază, printre altele, importanța componentei de tip server de semnalizare cu rolul de a asigura comunicarea la nivel de grup respectând cerințele sistemelor de video colaborare - timp minim de răspuns, lățime de bandă suficientă pentru legătura audio-video, etc. Totodată, se asigură o lățime de bandă suficientă pentru partajarea aplicațiilor, respectiv ecranelor, un instrument necesar în procesul de predare ce permite cadrelor didactice și instructorilor prezentarea unor soluții la problemele practice cu care cursanții se întâlnesc pe durata actului de învățare. Mai mult, sistemul propus în articolul [13] oferă o interfață web selectivă ce asigură accesul la instrumentele de tip colaborative learning în funcție de rolul utilizatorului. Dintre acestea se disting: gestionarea cursurilor și conținutului aferent, acces la conținut și activități practice, agendă cursurilor, clasa virtuală sau transfer de fișiere în timp real s.a.m.d.

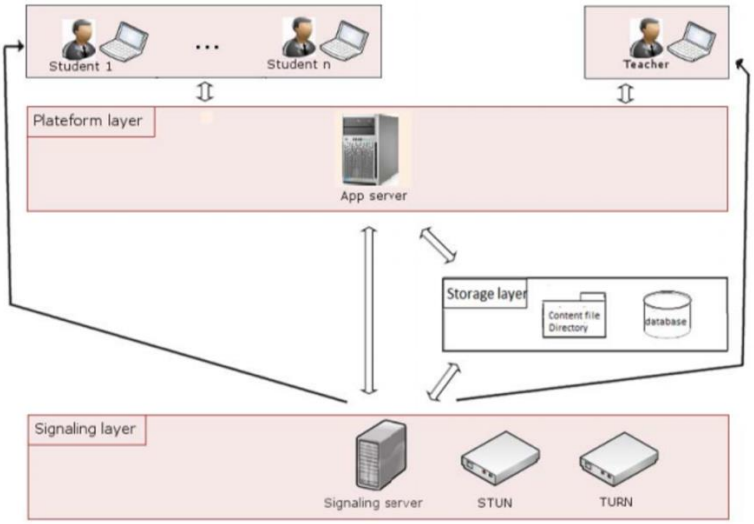

Fig 3.1 Arhitectura sistemului propus in lucrarea [3]

Figura 3.1 prezintă arhitectură sistemului propus în lucrarea [3]. Acesta este compus din mai multe componente dispuse pe mai multe nivele. Nivelul Platform Layer este o aplicație web hostata pe un server web, care este la rândul său un nivel intermediar între ușeri și serverul de semnalizare. Nivelul de stocare al datelor este de asemenea folosit de către nivelul aplicației client și de nivelul serverului de semnalizare în procesul de autentificarea, identificare și control al permisiunilor userilor.

Y. Liao prin lucrarea [4] prezintă conceptele de proiectare și implementare a unui sistem de predare video bazat pe tehnologia WebRTC dedicat dispozitivelor ce rulează platforma Android. Sistemul de învățare propus oferă un standard deschis și ușor de utilizat, ce permite predarea folosind componentele audio-video, permițând accesul participanților aflați în diferite locații, indiferent de rețeaua de acces pe care aceștia o folosesc. Soluția propusă nu promovează doar un mod eficient de învățământ la distanță, dar, ci reprezintă un mod de popularizare a unui model de educație și formare profesională continuă.

Figura 3.2 prezintă diagrama arhitecurala a sistemului de clasă virtuală prezentat în articolul [4]. Din figure pot fi observate cele patru tipuri de servere pe care modulul de clasă virtuală propus în lucrarea din articolul [4] le folosește:

- Classroom Server
- Signalling Server
- Stun Server
- Turn Server

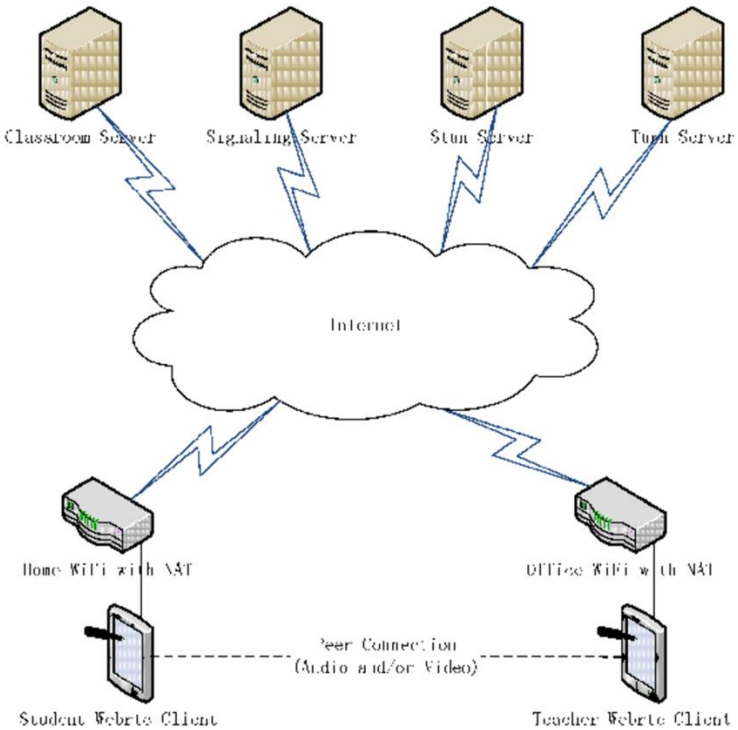

Fig 3.2 Diagrama arhitecturală a sistemului de clasă virtuală din lucrarea [4]

LL Fernandez în lucrarea [5] prezintă Kurento un server media care suportă comunicații audio-video folosind tehnologia WebRTC pentru aplicații web și mobile. Kurento are la o tehnologie open-source și asigură convergență WWW/mobile prin combinarea unui plan de semnalizare bazat pe SIP (Session Initiation Protocol) și HTTP (Hypertext Transfer Protocol), respectiv o infrastructură compresivă de servicii media construită pe fundamentele pachetului software GStreamer. Tehnologia prezentată în acest articol asigură funcționalități de tip real time media streaming folosind diferite protocoale și formate multimedia, la care se adaugă prelucrarea pachetelor de date complexe, transcodarea și filtrarea acestora. Autorii sublineaza ca, cu ajutorul celor mentionate mai sus Kurento ar putea impinge capabilitatiile WebRTC dincolo de o comunicare simpla peer-to-peer.

Figura 3.3 prezintă arhitectură serverului media propus care este împărțită în două secțiuni, prima fiind planul media iar cea de a două este planul de semnalizare. Primul are la bază stiva JBoss Java EE în schimb ce planul de semnalizare a fost dezvoltat peste framework-ul firului de execuție GStreamer.

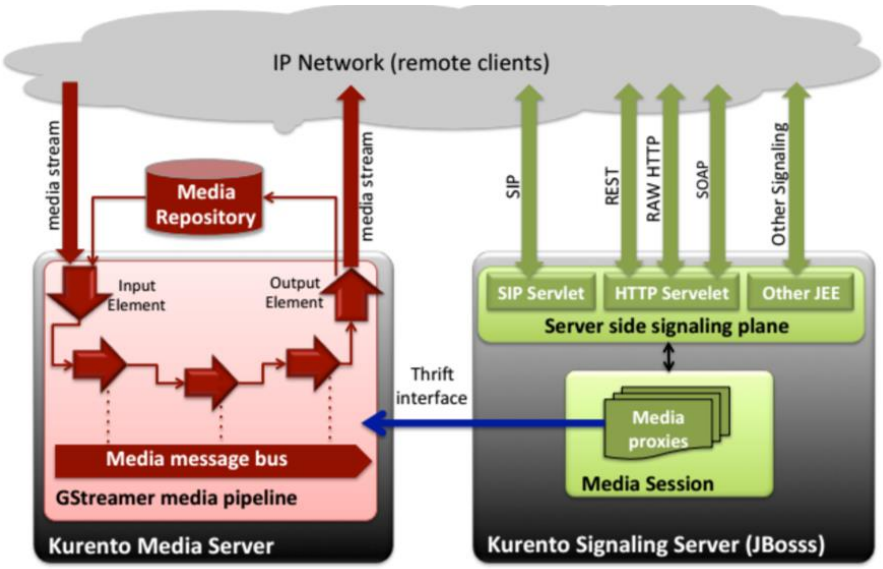

Fig 3.3 Arhitectura server media propus de LL Fernandez in articolul [5]

B. Garcia, LL. Fernandez, M Gallego și F. Gortázar prin lucrarea lor [6] intitulată "Analysis of video quality and end-to-end latency în WebRTC" fac o analiză calității video și a latenței de la un capăt la celălalt utilizând tehnologia WebRTC. În ciudă faptului că tehnologia este încă în dezvoltare ea câștigă rapid atenția tot mai multor dezvoltatori de aplicații de video colaborare. Autorii menționează cât de importantă este testarea acestei tehnologii și că mecanismele de asigurare a calității pentru WebRTC sunt cheia pentru pentru a avea aplicații sigure și performanțe. Articolul prezintă framework de testare pentru serverul media Kurento numit KFT(Kurento Testing Framework), ce este în fapt o bucată de software destinat simplificării activităților de testare a aplicațiilor și serviciilor ce au la bază tehnologia WebRTC. Framework-ul oferă funcții de evaluare avansate cu ajutorul cărora se poate realiză o evaluare aplicațiilor ce folosesc tehnologia amintită în frază anterioară.

Una dintre analizele realizate în lucrarea [6] poate fii observată în figura 3.4 care prezintă evoluția lantentei punct la punct. Putem observa că latență rămâne cât de cât stabilă pe durată exeperimtenutlui până când numărul de clienți ajunge la 180

.

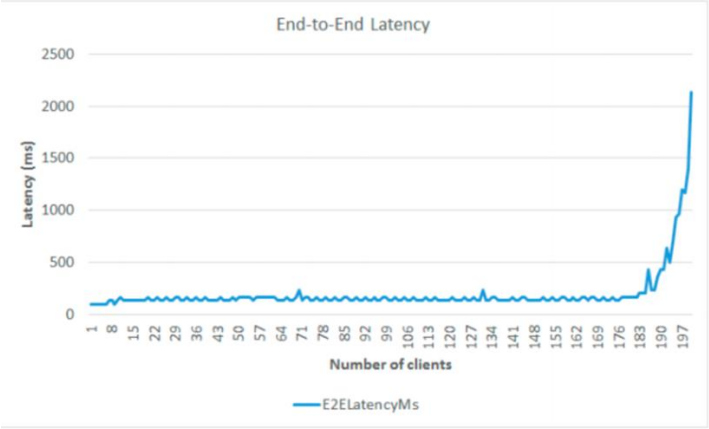

Fig 3.4 Analiza asupra evoluției latenței punct la punct efectuată în lucrarea [6]

C Spoială, A. Calinciuc, O Turcu și C Filote de la Facultatea de Inginerie Electrică și Calculatoare Ștefan Cel Mare din Suceava preconizează că o mulțime de servicii vor trece la folosirea tehnologiei WebRTC în detrimentul altor servicii [7]. Din acest motiv autorii consideră că ar fii utilă o comparație între *hostarea* unui server de WebRTC în containere Docker sau pe mașini virtuale. Pentru serverul de WebRTC aceștia au ales Kurento Media Server un server media open-source puternic cu multe caracteristici avansate. În această lucrare se prezintă care este soluția mai bună pentru o virtualizare mai potrivită pentru o aplicație multimedia bazată pe WebRTC, aceștia testând capabilitatiiile multimedia de care dispune acest server instalat în un container de Docker sau pe o mașină virtuală linux dedicată. Concluziile acestui articol sunt cu containerele Docker nu au aceleași costuri generale precum mașinile virtuale dedicate, iar datorită acestui fapt costurile de întreținere sunt mai mici și performațele unui sistem de video colaborare în timp real sunt mai bune dacă se rulează un server media din un container de Docker.

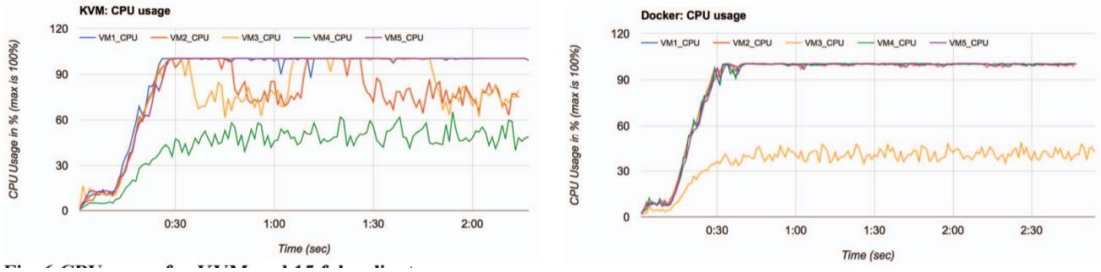

Fig 3.5 Comparație utilzare CPU în cazul utilizări KMS pe o mașină virtuală vs Docker prezentată în articolul [7]

În figura 3.5 se poate observa o analiză comparativă a utilizării procesorului în cazul unei gazde a serverului media prezentată în articolul [7]. Comparația din figură 3.5 a fost realizată folosind 15 clienți falși pentru ambele cazuri și aceeași configurație hardware.

Munca de cercetare depusă de către Chandran și prezentată în lucrarea din articolul [8] a fost concentrată pe definirea unui model arhitectural dedicat platformelor de învățare de la distantă, pornind de la problemele identificate în platformele de tip e-learning existența la momentul în care a fost efectuată cercetarea. Un alt punct de discuție ce poate fii găsit în această lucrare a fost prezentarea unui HMI (Hybrid Instrucțional Model) ce poate fii folosit atât în metodele de educație clasică(susținute fată în fată în o sală de curs) dar și în metodele ce presupun educarea studenților prin mediul online. Lucrarea prezentată de Chandran sublineaza cerințele funcționale și non-funcționale pe care un sistem de tip e-learning trebuie să le respecte. Cerințele menționate au fost scalabilitatea sistemului, costurile de dezvoltarea sau de personalizare a unui sistem de acest tip, dar totodată identifică și limitările și punctele slabe ale soluțiilor deja dezvoltate în acest domeniu, care sunt greu de scalat și de extins cu funcționalități noi. În completarea celor menționate deja autorul precizează faptul că integrarea cu sistemele existente de e-learning este scumpă.

M.A. Bochicchio în articolul [9] propune o abordare ierahica, ce constă în colaborarea mai multor entități folosind un sistem de conferința mulți video, ce are la bază WebRTC și care permite accesul la un microscop aflat în clădirea universității sub tipul unui echipament de laborator online. Aplicația rezultată în urmă cercetării a fost testată de elevii unui liceu în activitati biologice curriculare, cu scopul de a demonstra fezabilitatea sistemului propus și eficacitatea să pedagogică. În articol sunt descrise și informații de implementare pentru colaborarea sincronă folosind un server de semnalizare NodeJS și tehnologia WebRTC.

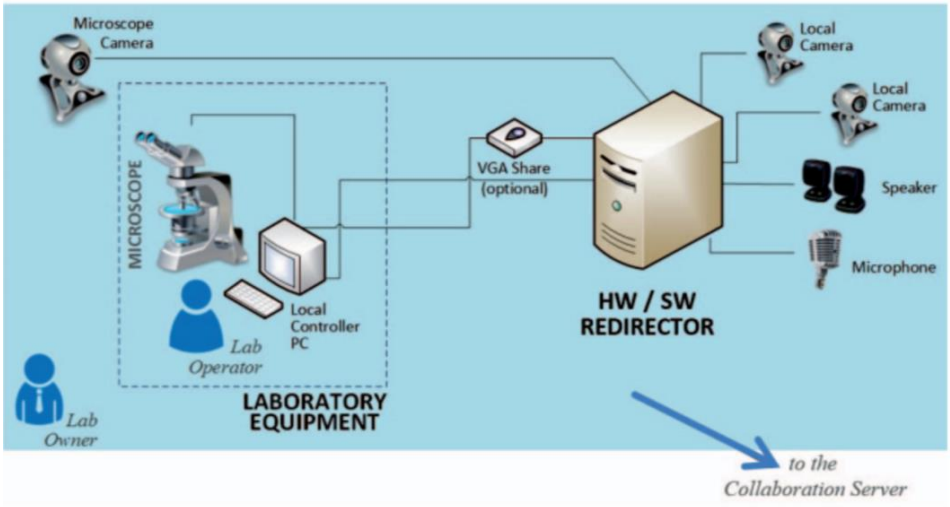

Fig 3.6 Arhitectura sistemului descis în lucrarea [9]

Figura 3.6 prezintă arhitectura fizca aleas de M.A. Bochicchio în implementarea sistemului de video colaborare pentru a avea acces de la distanță la un microscop al facultății de biologie unde a fost implementat acest sistem.

Moldovan în articolul [10] prezintă un framework-ul eRAF și inovațiile adnotării elementelor multimedia pe care acest framework le implementează. Acestea sunt realizate prin interacțiunea om-calculator și a conceptelor de colaborare interpersonala sincronă și asincrona. În lucrarea prezentată de către Moldovan se încearcă îmbunătățirea

experienţelor profesorilor şi a studenţiilor prin customizarea instrumentelor oferite de către framework-ul eRAF în sectorul educaţional şi a-l celui de pregătire.

Un model generalizat pentru implementarea sistememului propus de Moldovan în articolul din lucrarea [10].

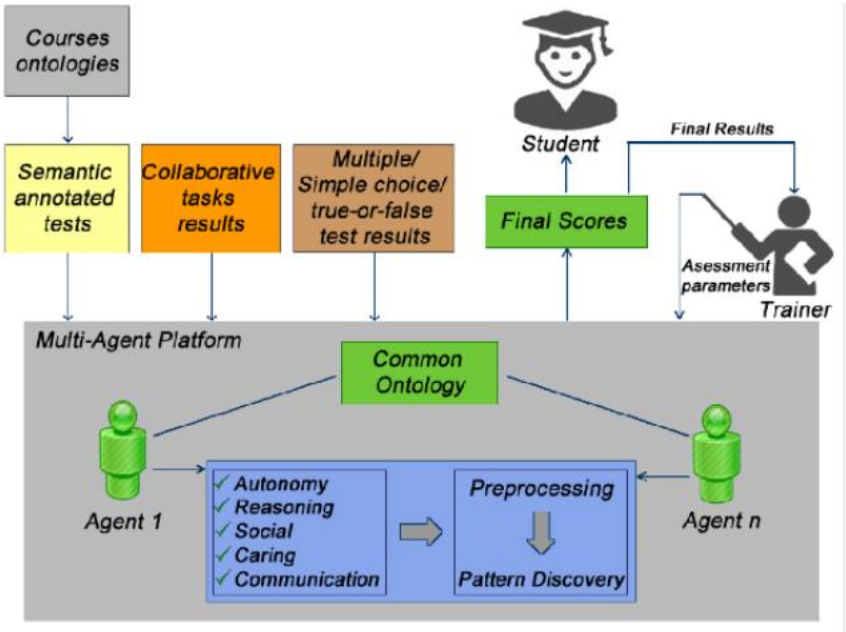

Fig 3.7 Arhitectura generalizată pentru un system de tip Blended learnig propus în lucrarea [10]

Nu în ultimul rând Qiu prin lucrarea de cercetare ce poate fii găsită în articolul [11] propune un sistem de învăţare ce presupune atât învăţarea după modelul clasic dar şi utilizarea instrumentelor de învăţare de la distanţă, adică un model de învăţarea şi formare profesională care necesită atât prezenţa fizică în anumite cazuri dar şi folosirea platformelor de tip e-learning. Din acest motiv sistemul propus implementează şi instrumente folositoare pentru învăţământul clasic cât şi instrumente e-learning folosite în ingineria software avansată. Sunt încurajate comunicarea în interiorul unui grup dar şi între mai mai multe grupuri de studenţi. Studiul efectuat a condus la un rezultat ce arată că studenţii acceptă destul de rapid şi adopta stilul acesta de învăţare, iar rezultate obţinute de către studenţi au fost mai bune decât s-a aşteptat. Metodologia lui Qui constă în împărţirea studenţiilor pe echipe/grupuri şi împărţirea semestrului pe proiecte care mai apoi sunt dezvoltate folosind iteraţii.

Alte instrumente colaborative comerciale oferite de WizIQ, Blackboard, Dokeos, Slack au fost luate în considerare. Conceptele de de ultimă generaţie propuse de aceste instrumente comerciale ce oferă soluţii pentru învăţarea colaborativa sincronă au fost luate în considerare atunci când am proiectat instrumentele de video colaborare.

Chiar dacă instrumentele de tip e-learning analizate în secţiunea curentă sunt specifice anumitor tipuri de metode de învăţare în un proces complex de dezvoltare a abilitaţilor şi a competenţelor de dezvoltarea a unui individ, aceste concepte au fost luate în considerare de către autori când au proiectat aceste framework-uri inovative, pornind de la concepte de e-learning şi algoritmi.

Capitolul 4. Analiză şi Fundamentare Teoretică

framework nu este altceva decât un instrument care ajută la organizare, modelarea structurii aplicației pentru a oferi un suport dezvoltatorilor. Ca un mod implicit framework-urile au la bază niște operații de bază.

Din punct de vedere arhitectural framework-ul propus nu are la bază arhitectură web clasică, chiar dacă din punct de vedere semantic, este bazată pe paradigma client-server.

Soluția tehnică propusă pentru implementarea sistemului de clasă virtuală a fost concepută pentru a se conforma cu arhitectura micro serviciilor, framework-ul de clasă virtuală fiind implementat că un set de componente independente responsabile fiecare de un tip de serviciu de învățare colaborativa sincronă

Implementarea folosind **arhitectura microserviciilor** [12] evidentiază un stil de programare a actorilor de reacție, în timp ce utilizatorii finali de astăzi se asteaptă la experiențe dinamice, dar consecvențe într-o gamă largă de dispozitive. Din acest motiv am ales să proiecteze și să implementeze o clasă virtuală scalabilă, adaptabilă, modulară și ușor de adaptat.

Din punct de vedere arhitectural se remarca patru blocuri distincte

1. Rich Client
2. Signalling server
3. Enterprise Information System
4. Media Server

4.1. Rich Client

Rich client este modulul care are rolul de a oferi utilizatorului o interfață web prin care poate să folosească sistemul. Diferența dintre un client simplu și rich client este aceea că pe o interfață web de tip “rich client” se procesează majoritatea datelor deoarece în cazul de față are rolul de a iniția comunicarea în timp real și anumitele configurări și personalizări în cea ce privește această comunicare pe care utilizatorii doresc să le facă. O altă responsabilitate a acestui bloc este să ofere capabilități audio-vizuale (sloturi video, tag-uri audio), cât și să permită controlul sesiunii și ofere gestionarea participanților.

Modulul a fost implementat folosind limbajele de programare JavaScript, Html5 și Css3.Acesta are la bază framework-ul AngularJS 1.0. Interfață web dedicată extinde capabiltatiile tehnologiei open-source WebRTC și invocă o librărie media oferită de către Kurento care facilitează interacțiunea cu mecanismul de video colaborare a-l tehnologiei WebRTC.

HTML5 este un limbaj de markup folosit pentru stricturarea și prezentarea conținutului WWW [13]. Versiunea aleasă este cinci, ultima apărută în anul 2014.Această versiune extinde și îmbunătățește versiunea anterioară prin adăugarea mark-upuri pentru API-uri(application programming interfaces).

CSS3 [14] este versiunea a treia a unui limbaj de stil folosit la descrierea și prezentarea unui document scris folosind limbaje de tip markup (HTML). CSS controlează stilul paginilor web, cum ar fi: poziția elementelor, culoarea elementelor, fontul elementelor, mărimea fonturilor. Rolul acestor stiluri de pagină este să ofere utilizatorului posibilitatea de a aranja elementele în pagină după cum dorește.

Saas [15] este o extensie a CSS care adaugă putere și eleganță limbajului de bază. Acesta permite utilizarea variabilelor, regulilor imbricate, importuri inline, sasmd. Sintaxa este una compatibilă cu limbajul CSS. Acesta ajută la organizarea și crearea mult mai rapid a design-ului unei aplicații folosind CSS.

JavaScript este un limbaj de programare de nivel înalt, bazat pe obiecte. Alături de HTML și CSS el stă la bază producției de conținut World Wide Web [16]. Este utilizat pentru a face paginile Web interactive. Deși inițial a fost implementat pentru a deservii partea de client și să fie folosit doar în browserele web, mai nou motoarele JavaScript sunt acum folosite și pe servere. Acest lucru face că JavaScript să fie diponibil pentru scrierea aplicațiilor mobile și desktop.

Framework-ul ales pentru implementarea clientului este **AngularJS1.0.** Acesta permite crearea unei aplicații de tip Single Page Aplication ușor, și având o structură bine organizată [17]. În cazul instrumentelor de colaborare sincrone de tip video-conference framework-ul simplifică aplicația datorită abstractizării, ceea ce conferă flexibilitate developer-ilor de a dezvoltă aplicații dinamice. Acest framework este perfect pentru aplicațiile CRUD, în schimb lasă de dorit în cazul în care au loc acțiuni intense de manipulare a DOM-ului aplicației, precum la jocuri. În acest context este de preferat Jquery, care are un nivel de abstractizare scăzut .

Arhitectura folosită în implementarea componenței Rich Client este **model-view-controller**. Această are la bază principiul de a izola conceptele, logica aplicației de interfață. Controller-ul primește toate cererile user-ului, care interacționează cu aplicația prin view, și trimite acele solicitări modelului. Modelul face legătura cu bază de date, pentru a pregăti toate datele cerute din view.

Controller-ul este partea unde se implementează logică aplicației. Mai exact, răspunde la cerințele view-ului și transmite aceste request-uri modelului. În controller se primesc datele din input, sunt validate, iar după poate să prelucreze acele date în funcție de cerințele user-ului.

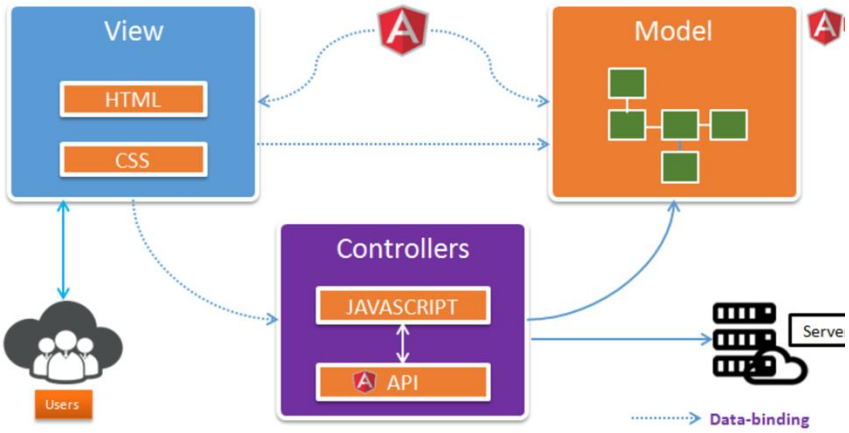

Fig 4.1 AngularJS Model View Controller. [18]

Aplicațiile care folosesc framework-ul AngularJs folosesc neapărat următoarele:

- ng-app: directiva ce leagă aplicația de HTML
- ng-controller: directiva care are scopul de a controla fluxul de date care circula intre model si view
- ng-model: stochează și modifică valorile din input-uri, acestea putând fi accesibile și în controller.

Așadar, între view și controller se află o variabilă care stochează datele din DOM și le trimite controller-ului pentru a le prelucra. Variabila în pricină este $scope. În figura 4.2 este reprezentarea grafică a elementelor care sunt implicate în transmiterea datelor de la view la controller, prin intermediul variabilei.

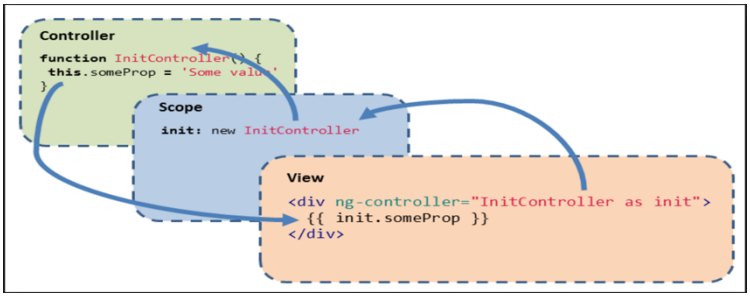

Fig 4.2 AngularJS View-Controller comunicare folosind $scope [19]

Deoare $scope este specific fiecărui controller pentru a putea controla și accesa elementele salvate în $scope din mai multe controllere, putem folosi $rootScope o variabilă globală ce conține toate variabilele de tip scope declarate în orice Controller al aplicației. Acest lucru poate fii observat și în figura 4.3.

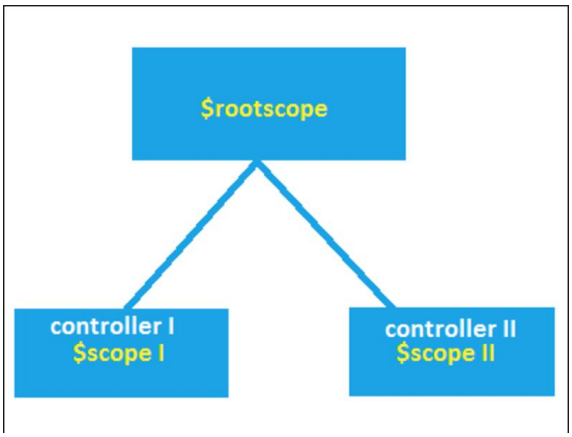

Fig 4.3 Relația $scope și $rootScope [20]

Pentru a abstractiza și mai mult conținutul unei aplicații realizate AngularJS oferă noțiunea de servicii. Acestea inglobează codul comun care este folosit de toate controllerele aplicației. Aceste tipuri de servicii pot fii de două feluri Service și Factory. Atunci când avem de-a face cu un sistem de mare anvergură se recomandă folosirea serviciilor deoarece o să fie ușor de manipulat în cazul în care se cer modificări

Partea de view a aplicație poate fii imbunatita și abstractizată folosind noțiunea de directivă pe care o oferă framework-ul și care premite programatorul să creeze o parte a aplicației ca o componentă indepenta cu codul HTML și controller separat. Folosirea directivelor este necesară când dorim să creem instrumente/componente decuplate, lucru ce permite eliminarea sau modificarea acestor componente foarte ușor.

Mai jos o să menționez câteva dintre avantajele Angular 1.0 care m-au făcut să aleg să implementez aplicația de video colaborare folosind acest framework:

- Oferă un grad mare de abstractizare a aplicației și putem să menținem codul mult mai ușor;
- are o arhitectura familiara și bine cunoscuta de multi dezvoltatori de software(Model-View-Controller)
- Controller-ul poate manipula datele din DOM cu ajutorul directivelor. Directivele sunt componente separate de aplicație, ele putând fi reutilizate în diverse proiecte, în funcție de nevoile programatorului.

Angular Material [21] este librăria folosita pentru a ajuta la crearea unei interfețe de utilizator consistente și atractive. Scopul angular material este de a furniza un set ușor și ușor de elemente UI native AngularJS care implementează specificațiile de proiectare a materialelor pentru a fi utilizate în aplicațiile single-page. Angular Material include un set bogat de componente UI reutilizabile, bine testate și accesibile. Totodată să respectăm principiile de dezvoltare a aplicațiilor web cum ar fii:

- portabilitate
- independenta fata de dispozitivul folosit

Doarece am dorit crearea unei interfețe web bine organizată și care să respecte standardele WWW am ales să folosesc un alt framework de data aceasta pentru partea de front-end. Am folosit **Bootstrap** [22] un framework web gratuit și open-source pentru

proiectarea de site-uri Web și aplicații web. Acesta conține șabloane de design HTML și CSS pentru formulare, butoane, navigație și alte componente de interfață, precum și extensii JavaScript opționale. Spre deosebire de multe alte framework-uri web, Bootstrap se referă numai la dezvoltarea front-end-ului.

Bootstrap oferă un set de elemente de design CSS care oferă definiții de design de bază pentru toate componentele HTML cheie. Acestea oferă un aspect uniform și modern pentru formatarea textului, a tabelelor și a elementelor unui formular.

Bootstrap vine în completarea elementelor HTML obișnuite prin adăugarea unora noi. Aceste componente sunt implementate că clase CSS care trebuiesc aplicate anumitor elemente HTML dintr-o pagină web.

Nu în ultimul rând bootstrap vine cu mai multe componente JavaScript sub formă unor plugin-uri jQuerry. Acestea oferă elemente noi de interfață cum ar fii dialog-box, carusel-uri ș.a.ș.m.d. Totodată aceste elemente noi extind unele elemente de interfață existente.

După cum spuneam interfața web dedicată extinde capabilitatiile tehnologiei open-source **WebRTC** [23]. Din punct de vedere cronologic WebRTC a fost dezvoltat inițial de către GIPS are oferă servicii de telefonie online și videoconferința. În anul 2010 Google a achiziționat nouă tehnologie, având că scop îmbunatățirea funcțiilor de comunicare a poștei electronice Gmail.

Acronimul WebRTC se trage de la denumirea “Web Real Time Communication”. După cum se poate observa pe site-ul oficial al tehnologie oferite de Google produsul este unul open-source. Cea mai importantă caracteristică a lui WebRTC constă în capacitatea de a putea accesă date de tip media(audio/video) direct în browser, eliminând instalarea vreunui plugin sau a unor extensii, prin JavaScript API. Astfel, putem menționa faptul că această nouă tehnologie adaugă noi funcționalități browserelor. Inițial tehnologia WebRTC a fost incorporată inițial doar în browser-ul de casă al celor de la Goolge, numit Google Chrome. Însă datorită faptului că Google a decis să ofere tehnologia open-source a permis tot odată ca aceasta să se dezvolte rapid iar developerii de la Mozilla au acceptat să introducă posibilitatea folosirii acestei tehnologii și în browser-ul Firefox. Mai nou și Opera a adoptat WebRTC. După ultimele informații oferite de către cei de la Apple și browser-ul lor Safari o să introdu-că tehnologia în versiunea 11 a acestuia, în schimb ce cei de la Microsoft au precizat că o personalizare a WebRTC va fii disponibilă și în browserul Edge în acest an.

Ținând cont că consumatorii de servicii web real-time preferă să utilizeze soft-uri care să fie ușor de folosit, fără să parcurgă un traseu anevoios de instalare a diferiteloor extensii, să își seteze camera sau microfonul și numai la finalizarea acestora să poată avea acces la serviciile în timp real, tehnologia WebRTC vine în față prezentabil cu o gamă de largă de componente, care pot ușor fi utilizate în aplicațiile care au nevoie de video/audio chat. Având în vedere faptul că poate fi la îndemâna oricărui programator web, fiind open-source în viitor ne așteptăm să fie dezvoltate în viitor aplicații din ce în ce mai performante.

La bazele tehnologiei WebRTC stau urmatoarele componente:

- **MediaStream**: este un API care oferă acces la cameră și microfon în browser, o componentă care ne ajută să afișăm conținutul unui stream video/audio, local sau prin conectarea cu un alt client un stream de tip

remote.

- ➔ **PeerConnection**: este una dintre funcționalitățile oferite de WebRTC. Conexiunea peer-to-peer este definită ca fiind legătura directă dintre două browsere, un "Peer" reprezintă o entitate care se află la finalul unei conexiuni bidirecționale.
- ➔ **DataChannel:** datele media – stream-urile sunt atașate acestei conexiuni pentru a putea fi trimise și atașate browser-ului aflat la celălalt capăt al conexiunii

RTCSessionDescription este un obiect care conține informații despre sesiune. Developer-ul poate să editeze acest obiect, având acces la API-ul componentei RTCPeerConnection.

Tehnologia WebRTC introducând o nouă paradigmă peer-to-peer. Protocolul SIP (Session Institution Protocol) este modelul după care a fost construită arhitectura WebRTC-ului. Protocolul de semnalizare în stiva OSI se situează la nivelul aplicație.

Acesta este utilizat pentru a realiza conexiunea asincronă între doi useri. Acest protocol se ocupă de crearea, modificarea, de asemenea și de încheierea sesiunii. SIP se folosește și pentru descrierea specificațiilor sesiunilor de semnalizare protocolul SDP (Session Description Protocol). Protocolul destinat descrierii sesiunii constituie o componentă importantă în procesul de semnalizare, realizat între userii prezenți într-o sesiune. Din punct de vedere arhitectural are un design asemănător cu HTTP. Protocolul SIP este folosit doar pentru inițializarea sesiunii.

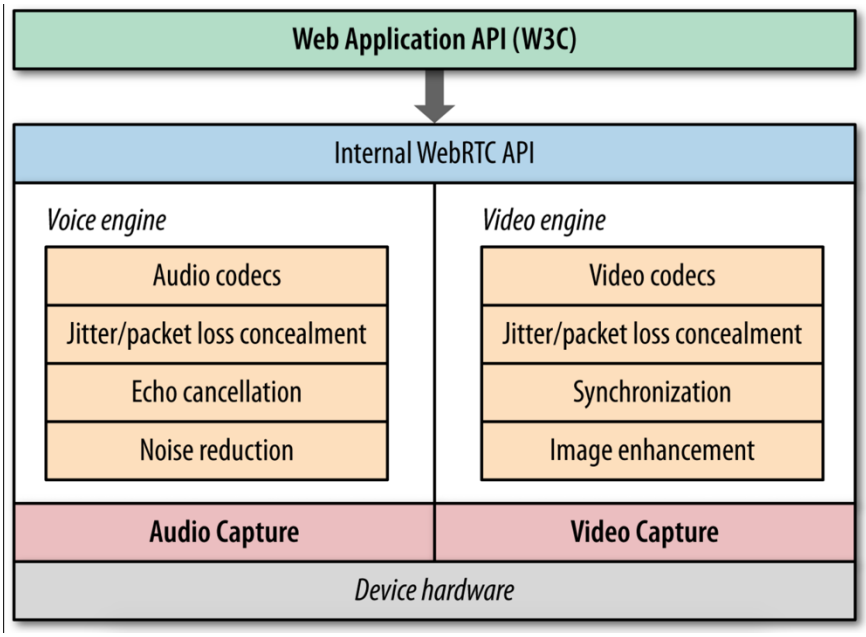

Fig 4.4 Arhitectura WebRTC [24].

In figura 4.4 este prezentata arhitectura ce sa la baza tehnologie WebRTC și se poate observa cat de complexa este aceasta.

O altă componentă care a trbuit luată în consideră în implementarea aplicației de video colaborare este ICE (Interactive Connectivity Establishment). ICE este implicat în direcționarea conexiunii dintre ușeri în cadrul rețelei de Internet locală și globală, folosind bine cunoscutele servere STUN și TURN. Pentru a stabili și întreține o conexiune peer-to-peer pe lângă UDP, sunt necesare și ICE, STUN și TURN. ICE este un framework care este utilizat într-o conexiune peer-to-peer. Acesta oferă suficiente înfomații despre topologia rețelei Internet pentru a realiza conexiunea care se dorește să se stabilească între participanți, oferind posibilitatea de a trece de bariera oferită de NAT și firewall-uri sau proxy-uri.

Primul pas în realizarea conexiuni folosind ICE este ca framework-ul să stabilească o conexiune utilizând adresă "host" a dispozitivului participantului, iar dacă aceasta se realizează cu succes atunci transferul de date între emițător și receptor poate avea loc. În caz contrar se apelează serverul STUN care încearcă să ofere o adresă publică pentru participant, iar dacă nici acesta nu reușește să ofere informații legate de adresa participantului se face un apel mai departe către serverul TURN care traversează din nou blocul NAT și firewall.

În figura 4.5 este exemplificat modul de funcționare a framework-ului ICE și se poate observă rolul și modul de funcționare a serverelor STUN respectiv TURN.

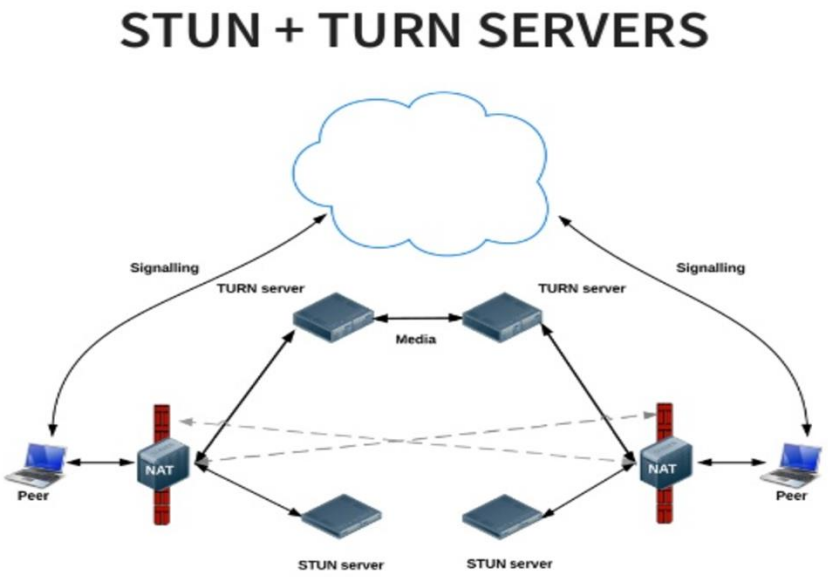

Fig 4.5 rolul serverelor STUN & TURN în un sistem ce folosește WebRTC [25]

Serverul STUN este oferit gratuit, de cele mai multe ori. Acesta permite clienților să afle adresa lor publică, tipul de NAT pe care îl au în spate și portul lateral pe Internet asociat NAT cu un anumit port local [26]. Aceste informații sunt utilizate pentru a configura comunicația UDP între client și furnizorul de servicii pentru a stabili un o legătură ce permite transmisia de date în timp real. Protocolul STUN este definit în RFC 3489. Serverul STUN este contactat implicit pe portul UDP 3478.

Fig 4.6 funcționarea serverului STUN [26]

Serverul TURN este de asemenea un server de traversare NAT și un gateway pentru traficul media [27]. Acesta poate fii utilizat folosind TCP sau UDP. TURN este specificat de RFC 5766. O actualizare a TURN pentru IPv6 este specificată în RFC 6156. Deși în general serverul TURN nu este gratuit am instalat versiunea open-source numită COTURN [27], pe o mașină virtuală la fel cum am procedat și cu serverul STUN.

Deoarece serverul TURN este unul intermediar datele care se transmit prin el consumă foarte multă lățime de bandă iar prioritara este folosirea serverului STUN.

In continuare o sa prezint câteva dintre avantajele folosirii WebRTC-ului:

- comparativ cu soluțiile existente deja în domeniul colaborării video WebRTC nu are nevoie de nici un plugin pentru a putea captura date de la un client (microfon/camera web).
- costul redus: nu necesită cumpărarea unei licențe, deoarece proiectul este open-source, fiind oferit de cei de la Google
- datorită implicării marilor companii - Google, Mozilla - a developer-ilor de la W3C, IETF, care au drept obiective să dezvolte și să standardizeze produsul WebRTC, bug-urile pot fi rapid rezolvate.
- WebRTC suportă ori ce mod de codificare a datelor: H.264, VP8 pentru video și G.711 sau OPUS pentru audio. Codificare datelor are loc, în cazul arhitecturii peer-to-peer pe care se bazează WebRTC, la nivel de browser.

Iar unul dintre marile dezavantaje ale tehnologiei este că nu este disponibila în browserele proprietar Apple și Microsoft. Figura 4.7 ilustrează browserele în care acesta tehnologie open-source poate fii folosită.

IE	Edge	Firefox	Chrome	Safari	Opera	iOS Safari
11	14	51	49	10	43	9.3
	15	52	55	10.1	44	10.2
		53	56		45	10.3
		54	57		46	
		55	58			
			59			
			60			

Fig 4.7 Browsere ce oferă suport pentru WebRTC [28]

Pentru a simplifica dezvoltarea instrumentelor de video colaborare bazate pe tehnologia WebRTC am decis să folosesc un API oferit de Kurento numit **Kurento-Utils**.

Această librărie este disponibila pentru aplicațiile ce folosesc limbajul de programare JavaScript. Librăria Kurento-Utils conține un set de componente reutilizabile. Codul sursă al acestui proiect poate fi clonat din depozitul GitHub.

Singura clasă a librăriei este WebRTCPeer. Instanța acestei clase este un obiect ce maschează complexitatea RTCPeerConnection. Scopul acestui obiect este simplificarea dezvoltării aplicațiilor bazate pe WebRTC. Cu ajutorul acestui obiect se pot apela metode precum addIceCandidates(), processOffer(), processAnswer(), dispose(), ș.a.ș.m.d. Rolul și utilizarea acestor funcții o să fie explicate în capitolul ce urmează. Pentru instalarea librărie s-a folosit package manager-ul npm.

Pentru instalarea tuturor librăriilor folosite s-a folosit package manager-ul npm. **NPM** [29] este un managerul de pachete pentru limbajul JavaScript și este tot odată package-managerul prestabilit pentru aplicații server side ce folosesc NodeJS. Acesta se poate folosi din linia de comandă, iar după instalare clientul este numit npm. Acesta folosește o bază de date online unde sunt stocate pachetele publice, acesta fiind numit registrul npm. Registrul este accesat prin folosirea clientul din linia de comandă iar pachetele disponibile pot fi accesate și căutate prin intermediul site-ului oficial npm. NPM poate gestiona pachetele care sunt de dependințe locale ale unui proiect însă și pachete JavaScript Globale.

Grunt [30] este un task runner a unor sarcini JavaScript. Acesta este utilizat pentru a efectua automat sarcini frecvent utilizate cum ar fi minimizarea dimensiunii unui fișier, compilarea, testarea, ș.a.ș.m.d. Acesta folosește linia de comandă pentur a executa sarcini personalizate definite într-un fișier definit cu numele “Gruntfile.js”. Grunt este scris în limbajul JavaScript cu ajutorul Node.JS. Librăria este disponibilă prin npm. Prezent există mai mult de cinci mii de plugin-uri disponibile în ecosistemul Grunt. Printre companiile care folosesc Grunt se numără AdobeSystems, jQuery, Twitter, Mozilla, Bootstrap, Cloud, Opera, WordPress, Walmart și Microsoft. Task-urile sunt sarcinile care efectuează o un anumit job. Acestea sunt definite în fișierul Gruntfile. Developerii pot descărca și utiliza task-uri predefinite de la plugin-urile Grunt existente și/sau își pot defini propriile sarcini în funcție de cerințele lor. Odată definită această activitate poate fii rulată din linia de comandă.

În dezvoltarea instrumentelor de video colaborare am folosit Grunt pentru compilarea codului scris cu ajutorul librăriei Sass în cod Css cu ajutorul librariei grunt-sass și pentru împachetarea interfeței web sub forma unei aplicații de sine stătătoare ce rulează ca o aplicație nativă pe diferite sisteme de operare. Pentru finalizarea celui de-al doilea task am folosit task-ul predefinit al plugin-ului grunt-nw-builder.

Interfața web poate fii accesata atât prin utilizarea:

- browser web care suporta WebRTC
- fie prin instalarea și rularea aplicație native

Pentru a face posibila accesarea aplicație folosind cele doua metode a trebuie sa asiguram următoarele lucruri.

Pentru a putea utiliza aplicația din un browser web aplicația trebuie să fie publicate pe un server web. Am ales să folosesc serverul web **nginx [31]** care pe lângă acest lucru poate fii utilizat și ca un reverse proxy, sau load balancer. Serverul este unul proxy HTTP generic TCP/UDP sau un server de poștă electronică care a fost inițial scris de către Igor Sysoev. Am ales această variantă deoarece este un software gratuit și open-source chiar dacă pe piață serverelor web cea mai cunoscută și utilizată variantă este cea oferită de către Apache. Pe lângă cele menționate serverul web nginx facilitează comunicarea cu server-ul, se ocupă de cereri și răspunsuri către acesta și dinspre acesta, asigurând în același timp și de securitatea acestora.

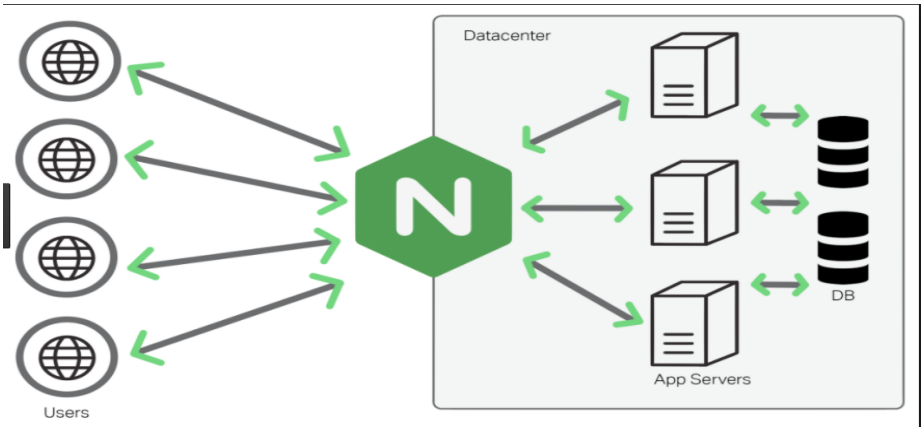

Fig 4.8 exemplu de utilizare Nginx [31]

Cea de a două metodă este de a accesă interfața web dedicată clienților este folosind nativă relizata cu ajutorul unor module npm și a unui plugin grunt. Pentru a putea împacheta o aplicație scrisă în limbaje de programare precum JavaScript, HTML, CSS runtime-ul pentru aplicații numit **nw.js** [32], care inițial a purtat numele de Node-Webkit. Acesta este bazat pe Chromium și Node.JS. Pe lângă faptul ca permite dezvoltarea aplicațiilor native folosind limbajele menționate mai sus acesta permite și apelarea modulelor de Node.Js direct din DOM-ul aplicației. Acesta a fost creat și dezvoltata de către Intel Open Source Technology Center și este un proiect oferit gratis de către aceștia.

Printre principalele funcționalitate ale aplicației sunt :

- permite scriere de aplicații standalone folosind tehnologii moderne cum ar fii HTML5, CSS3, JavaScript si WebGL
- oferă suport complet pentru API-urile Node.JS și toate modulele third-party
- oferă performanta buna deoarece Node.Js si WebKit rulează în același thread, apelurile de funcții sunt făcute simplu iar obiectele sunt stocate în aceeași zona de memorie *heap* și se pot apela reciproc
- aplicațiile sunt ușor împachetat și distribuit
- aplicația este disponibila atât în Linux, MacOs dar și pe Windows

Pentru a simplifica procesul de împachetarea aplicației web based în o aplicație stand alone sub forna nw.js am folosit utilitarul **nw-builder** [33]**.** După instalarea acesta

oferă folosirea unei comenzi în terminal urmată de anumite opțiuni pentru configurarea și împachetarea implicită a aplicației în formatul dorit.

Printre opțiunile folosite pot fii :

- platforma pentru care se face împachetarea aici pot fii trecute una dintre următoarele variante sau mai multe separate prin virgula(win32, win64, osx32, osx64, linux32, linux64)
- versiunea de nw.js pe care aplicația sa o folosească
- dacă platforma respectiva sa folosească Node.JS
- dezactivarea logurilor
-

Pentru a nu repeta introducerea manuală a comenzii în terminal am recurs la folosirea unui plugin de Grunt oferit de către nw.js pentru instrumentul de împachetare nw-builder. Acesta se numește **grunt-nw-builder** [34] și permite realizarea acestor împachetări sub forma unei aplicații Nw.Js. Acesta va descărca fișierele binare preconfigurate pentru versiunea specificată, îl va despacheta și crea fișierul app.nw.

Toate cele 3 instrumente au fost instalate folosit managerul de pachetele NPM fiind disponibile ca pachete distribuite gratuit.

Pentru a ușura distribuirea și folosirea instrumentelor de video colaborare pentru utilizatorii care dorească să utilizeze aplicația nativă am decis că aplicația are nevoie și de un installer. Pentru aceasta am folosit următoare două module de NodeJs din pachetul celor de la npm:

- **node-appdmg** [35] acesta permite generarea de imagine de tip DMG atractive pentru utilizator pentru aplicațiile native dedicate sistemului de operare MacOS. Pentru a crea o imagine de tip dmg e nevoie instalarea cu ajutorul package-managerului npm. Mai apoi se rulează aplicația din terminal și se dă input un fișier de tip JSON și locul în care fișierul să fie creat. Un exemplu minimal de fișier de configurare de tip JSON se poate observă în figura 4.9 și din figura 4.10 ne putem da face o idee cum afectează aceste opțiuni rezultatul final . Modulul permite o configurare amplă oferind foarte multe opțiune pentru configurarea acestui imagini de tip dmg

```
{
  "title": "Test Application",
  "icon": "test-app.icns",
  "background": "test-background.png",
  "contents": [
    { "x": 448, "y": 344, "type": "link", "path": "/Applications" },
    { "x": 192, "y": 344, "type": "file", "path": "TestApp.app" }
  ]
}
```

Fig 4.9 Exemplu de fișier de imput în format JSON

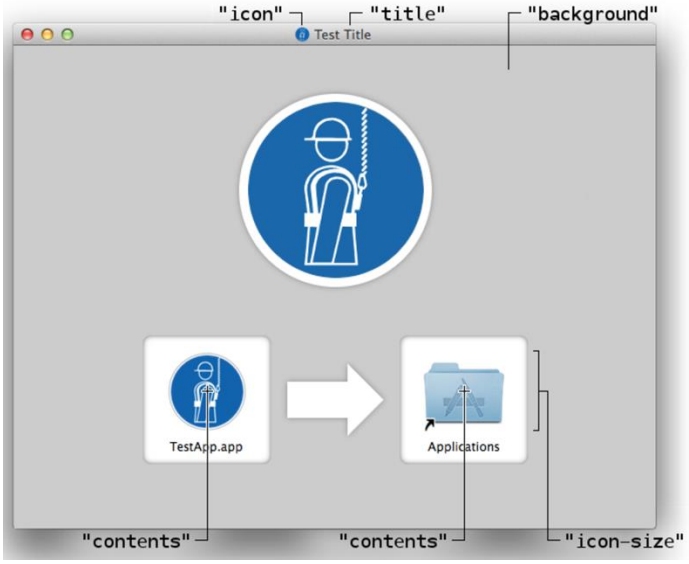

Fig 4.10 exemplu imagine DMG și componentele sale de baza

- **msi-packager** [36] este un modul Node.Js care permite pachete MSI (Windows Installer) pe Mac și Linux. Pentru a crea pachete MSI de pe Windows se poate folosi Visual Studio. După instalarea acestui modul de Node.JS se crează un fișier de tip javaScript unde este definit un task pentru crearea instalerului de windows iar mai apoi este executat cu comanda node și numele fișierului. Programul de instalare nu are un wizard , utilizatorii rulează doar programul de instalare și aplicația va fi instalată și vor fi create comenzi rapide. Implicit, aplicația va fi instalată pentru toți utilizatorii în folderul Program Files. Dacă se specifică localUser:true că opțiune, aplicația va fi instalată în folderul AppData al utilizatorului. Acest lucru permite utilizatorilor non-administratori să instaleze aplicația.

4.2. Signalling server

Serverul aplicației reprezinta al doilea nivelul logic al sistemului. Principalele responsabilități pe care acest bloc al sistemului trebuie sa le acopere sunt:

- gestionarea sesiunilor sincrone, având rolul de server de semnalizare intre clienți și serverul media. Prin aceasta se înțelege ca trebuie sa răspunda la comenzi precum crearea unei sesiuni de colaborare sincrona, negocierea cu intre server și client, stabilirea apelurilor.
- implementarea logicii de business a sistemului și asigurarea înregistrării utilizatorilor, distribuirea lor pe room-uri, prezenta utilizatorilor, asigurarea comunicării folosind metode bazate pe trimiterea de mesaje text, și gestionarea și salvarea acestor date în o baza de date relaționala.
- accesul și gestiunea resurselor de pe serverul media.
- acces la fișierele sistemului

Pentru a oferi serviciile de care are nevoie blocul Rich Client am decisă să ridic un server web Node.JS. **NodeJS** [37] este un framwork open-source pentru implementarea unui server cu ajutorul limbajului de programare JavaScript. Acesta extinde API-ul JavaScript pentru a oferi funcționalități obișnuite pentru servere.API de bază folosit de NodeJS poate fi extins utilizând sistemul de module CommonJS.

Scopul acestui tip de server este de a oferi o modalitate ușoară și sigura de a construi aplicații performante și scalabile în JavaScript.

Aceste obiective sunt realizate datorita arhitecturii sale:

- single thread :Nodul utilizează un fir unic pentru a rula spre deosebire de un server ca Apache HTTP care produce un fir pe cerere, această abordare conduce la evitarea comutării contextului procesorului și a stivelor de execuție masive în memorie. Aceasta este, de asemenea, metoda folosită de nginx și alte servere dezvoltate pentru a contracara aceste probleme.
- Event Loop : (Buclă de evenimente) este scris în C ++ folosind biblioteca libev a lui Marc Lehman, bucla evenimentului utilizează epoll sau kqueue pentru mecanismul de notificare a evenimentelor scalabile.
- Non blocking I/O : (I/O fără blocare) NodeJS evită pierderea de timp a procesorului, de obicei, prin așteptarea unui răspuns de intrare sau de ieșire (bază de date, sistem de fișiere, serviciu web, ...), datorită funcțiilor de intrare / ieșire asincrone, furnizate de biblioteca libeio a lui Marc Lehmann.

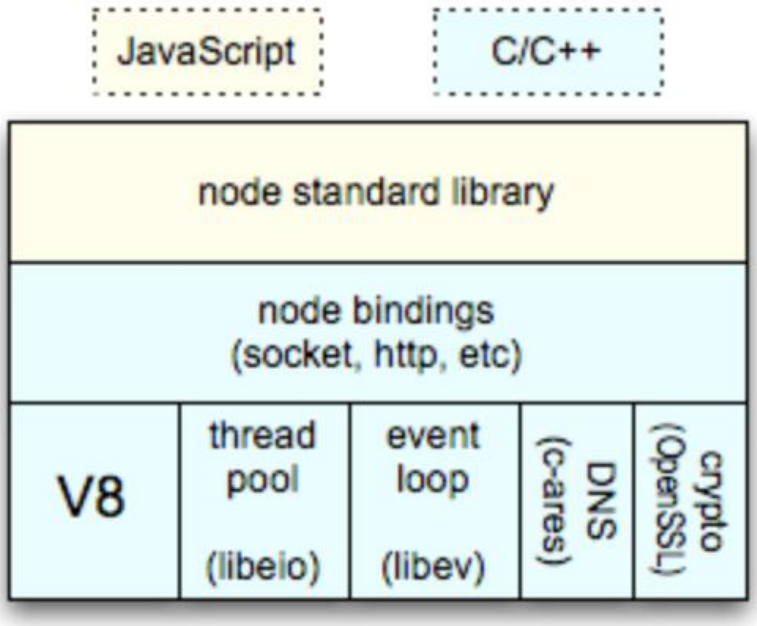

Fig 4.11 Arhitectura NodeJS [38]

NodeJS rulează pe diferite platforme (Windows, Linux, MacOS, Unix) și folosește motorul JavaScript V8. Motorul V8 este scris în C ++ și este cunoscut mai ales pentru utilizarea în Google Chrome.

Dezvoltarea acestui framework de server bazat pe limbajul de programare JavaScript a început în anul 2009.

NodeJS folosește o tehnică de programare asincrona bazată pe evenimente. Acesta folosește implicit un singur thread, fără blocări aceasta fiind o metodă foarte eficientă din punct de vedere al memoriei.

Câteva dintre lucrurile pe care le poate face un server NodeJS:

- NodeJS poate genera conținut dinamic pentru o pagina HTML
- poate crea, deschide, citi, scrie, șterge, și închide fișiere de pe un server
- NodeJS poate adaugă, șterge sau modifica date ale unei baze de date

NodeJS are un set de module incorporate care nu necesita instalare înaintea folosirii lor. Dintre acestea am ales sa folosesc următoarele modulele incorporante:

- HTTP(s) - modulul ce permite NodeJS să transfere date prin protocolul HTTP(Hyper Text Transfer Protocol). Modulul HTTP poate crea un server HTTP care ascultă porturile de server și oferă un răspuns înapoi clientului. În aplicația am folosit modul secure a acestui modul HTTPS ce permite că transferul să se facă prin protocolul HTTPS TLS/SSL care este protocolul HTTP securizat.
- fs - este modulul ce oferă o modalitate de lucru cu sistemul de fișiere al serverului
- path - modulul path permite NodeJS sa lucreze cu directoare și căile spre anumite fișiere.

Pe lângă cele menționate am importat alte module oferite de diferite companii importate folosit managerul de pachete NodeJS NPM. Acestea sunt:

- winston : este o librărie folosita pentru a salva log-urile serverului. A fost necesara deoarece serverul este rulat in consola iar log-urile nu sunt disponibile după închiderea consolei. Acesta a fost conceput pentru a fii o biblioteca universala ce permite logarea pe mai multe nivele (error, info, warning). Aceste nivele sunt utile în monitorizarea log-urilor. Librăria permite salvarea unui nivel, spre exemplu error in o baza de date nu doar într-un fișier pe server.
- redis : deoarece NodeJS este un server ce rulează pe un singur thread pentru a putea folosi intreaga putere de procesare a unui server cu procesor potent cu mai multe core-uri fizice am încercat scalarea acestui bloc. Pentru aceasta am folosit modul redis. Acesta este un broker de mesaje ce permite comunicarea intre doua instante de NodeJS. Deoarece am ridicat doua servere la porturi diferite pentru a putea gestiona sesiunea sincrona s-a folosit acest broker de mesaje ce permite comunicarea intre aceste instante de server la porturi diferite. Acest modul este gratuit și open-source. Totodată acesta implementează paradigma Public-Subscribe.
- kurento-client : este api-ul gratuit și open-surce oferit de Kurento. Cu ajutorul acestui API se realizează gestionarea sesiuni sincrone și comunicarea cu serverul media. Acesta permite crearea resurselor necesare ca stream-urile clienților sa ajungă la serverul media central(pipeline-uri media, endpoints etc.) ,procesarea ofertelor clienților s.a.s.m.d .

Pentru a include atât modulele incorporate in NodeJS dar și cele importate de utilizatori din alte surse se folosește funcția require().

Comunicarea între blocurile Rich Client și Signalling server se realizează prin folosea WebSocket. **WebSocket** [39] este un protocol de comunicare pentru calculatoare ce oferă canale de comunicare full-duplex printr-o singură conexiune TCP. Singură relație dintre WebSocket și HTTP este că hand shake-ul este interpretat de serverele HTTP că o solicitare de upgrade. Protocolul WebSocket este în prezent acceptat în majoritatea browserelor importante, inclusiv Google Chrome, Microsoft Edge, Internet Explorer, Firefox, Safari și Opera. WebSocket necesită, de asemenea, aplicații web pe server pentru al susține.

Pentru a eficientiza comunicarea între nivele precizate s-a folosit librăria **Socket.IO** [40]. Această este librărie JavaScript pentru aplicații web în timp real. Librăria este împărțită în două părți, una care rulează în browser pe client iar cealaltă rulează în un serverul NodeJS. Ambele componente au un API aproape identic. Această librărie la fel că și framework-ul NodeJS este de tipul event-driven.

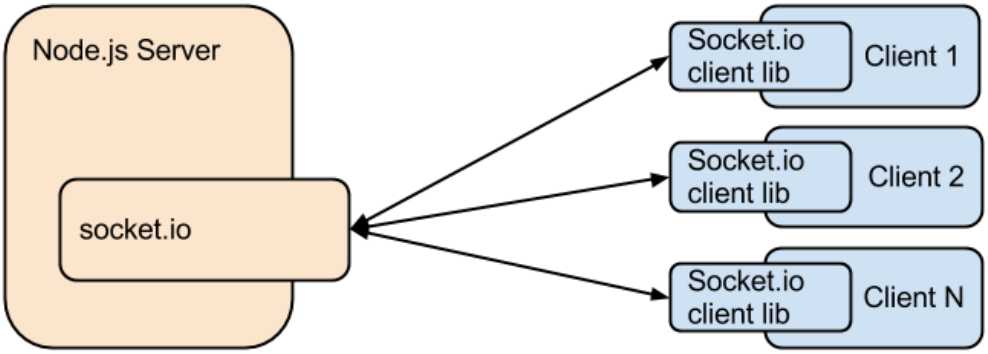

Fig 4.12 Comunicare client -> server folosind Socket.IO

Socket.IO utilizează în principal protocolul WebSocket și a fost instalat folosind managerul de pachete NPM. Socket.IO gestionează conexiunea în mod transparent și se va actualiza automat la WebSocket dacă este posibil. Părțile sale de negociere a protocolului determină că un client care suportă WebSocket standard să nu poată contacta un server Socket.IO. Și un client de implementare Socket.IO nu se poate cu un server WebSocket sau Long Polling Comet bazat ce nu folosește Socket.IO. Prin urmare, Socket.IO necesită utilizarea bibliotecilor Socket.IO pe ambele parți client și server.

Doua dintre opțiunile de configurarea ale socket.io pentru partea de server au fost:

- PingTimeout : căruia trebuie sa ii atribuim un număr sub forma unor milisecunde și reprezinta câte milisecunde fără un pachet de pong pentru a considera conexiunea închisă .
- PingInterval: la fel ca PingTimeout trebuie setata o valoare în milisecunde ce

reprezinta câte milisecunde trebuie sa aștepte serverul înainte de a trimite un nou pachet ping

Acești doi parametri vor avea impact asupra întârzierii înainte că un client să știe că serverul nu mai este disponibil. De exemplu, dacă conexiunea TCP nu este închisa corect din cauza unei probleme de rețea, este posibil că un client să trebuiască să aștepte pană la pingTimeout + pingInterval milisecunde înainte de a obține un eveniment de deconectare.

Pe partea de client se returnează o nouă instanță Socket pentru spațiul de nume specificat de numele căii din adresă URL. De exemplu, dacă URL-ul este http://localhost/users, va fi stabilită o conexiune de transport la adresa http: //localhost și o conexiune Socket.IO va fi stabilită pentru /users.

Trimiterea și primirea evenimentelor emiterea și recepționarea unor mesaje personalizate. Pe lângă mesajele implicite ale librărie cum ar fii connect, disconnect, reconnect sau connect_error se pot crea mesaje personalizate care să fie emise de pe client/server și recepționate în celălalt bloc.

Totodată socket.IO permite și trimiterea unor mesaje de tip broadcast care este foarte utili în cazul instrumentelor de colaborare sincronă. Acest tip de mesaj permite difuzarea unui anumit pache unui set de mai mulți participanți înregistrați într-un room, mai puțin socket-ului care a inițiat acest mesaj de broadcast.

În interiorul fiecărui namespace se pot defini canale arbitrare la care socket-urile se pot alătura sau pe care le pot părăsi. Acestea sunt numite room-uri iar un socket se conectează folosind comanda join și poate părăsi acest room folosind comanda leave. Room-urile sunt folositoare în crearea și gestiunea sesiunii sincrone. Acest concept este utilizat și în cazul mesajelor de tip broadcast un se poate specifică în câmpul to numele room-ului iar acesta să facă broadcast în interiorul acestui canal.

Există mai multe modalități de a rezolvă problemă emiteterii de evenimente în din afară procesului Socket.IO, cum ar fi implementarea propriului canal pentru a trimite mesaje în acest proces. Pentru a facilita acest caz de utilizare, se pot utiliza modulele socket.io-Rediș și socket.io-emitor prin implementarea Rediș Adapter, modul ce facilitează emiterea de mesaje de la orice alt proces către orice canal.

Pentru mesajele personalizate am folosit pentru comunicare un protocol standardizat numit **JSON** [41], iar toate datele care circulă de la server la client și invers sunt încapsulate în sub acest format. JSON are un format complet independent de limbajul de programare folosit, astfel JSON-ul este un format de schimb al datelor ideal.

Construcția unui obiect de tip JSON se poate face folosind una din deformatoarele doua structuri:

- JSON-ul ca un singur obiect ce conține o colecție de perechi de tip cheie-valoare.
- o lista de obiecte, fiecare având structura de tip pereche cheie-valoare;

4.3. Enterprise Information System

Acest bloc al sistemului este responsabil de manipularea datelor fie că vorbim despre accesul și stocarea lor în o baza de date fie că vorbim despre accesul și gestiunea fișierelor din sistemul de fișiere a-l serverului. Am decis să utilizez o bază de date de tip relațional și utilizarea MySQL, iar pentru gestiunea fișierelor am folosit modulul fs al NodeJS prezentat anterior.

MySQL [42] este un sistem de gestiunare a bazelor de date relaționale open-source. MySQL a fost detinută și sponsorizată de o singură firmă cu care avea că tel obinerea de profit de pe urmă acestui sistem adică compania suedeză MySQL AB, dar acum este detinută de Oracle Corporation.

MySQL este o componentă centrală a pachetului open-source LAMP pentru aplicații web. LAMP este un acronim pentru "Linux, Apache, MySQL, Perl / PHP / Python". MySQL este, de asemenea, utilizat în numeroase site-uri web de mari dimensiuni, inclusiv Google (deși nu pentru căutări), Facebook, Twitter, Flickr, Și YouTube.

MySQL este oferit în două ediții diferite server open-source MySQL Comunity Server și unul proprietar Entherprise Server. În implementare am folosit variantă open source.

Printre principalele funcționalități disponibile în versiune 5.6 a MySQL se regăsesc:

- suport pentru majoritatea sistemelor de operare
- posibilitatea scrierii unor proceduri stocate, folosind un limbaj procedural care respectă îndeaproape SQL
- posibilitatea utilizării triggerelor
- Un set de opțiuni de mod SQL pentru a controla comportamentul runtime, inclusiv un mod strict pentru a respecta mai bine standardele SQL
- posibilitatea utilizării tranzacțiilor cu puncte întoarcere ce permit executarea unui rollback în cazul în care o parte a tranzacției eșuează, atunci când se utilizează motorul implicit de stocare InnoDB.
- etc.

4.4. Media Server

Blocul serverului media al sistemului de colaborare sincronă este folosit pentru livrarea conținutului media clienților. Conceptual, un server media WebRTC este doar un fel de „middleware multimedia “în cazul în care traficul media trece prin acesta atunci când se deplasează de la sursă la destinație. Soluția propusă pentru serverul media este **Kurento Media Server** [43] un server media open source și gratuit. Kurento este un server media WebRTC ce oferă un set de API-uri ce abstractizează unele metode complexe ale tehnologiei WebRTC și simplifică dezvoltarea instrumentelor de video colaborare pentru platformele WWW și pentru dispozitivele mobile.

Kurento Media Server oferă funcționalități precum:

- ➔ comunicare în grup

- ➔ transcodare de fluxuri audio vizuale
- ➔ înregistrarea fluxurilor audio vizuale
- ➔ mixarea de fluxuri audio vizuale
- ➔ difuzarea fluxurilor audio vizuale
- ➔ rutarea de fluxuri audio vizuale

Ca o funcționalitate diferită, Kurento Media Server furnizează, de asemenea, capabilități avansate de procesare media care implică indexare video, realitate augmentată și analiză de vorbire. Arhitectură modulară Kurento simplifică integrarea algoritmilor de procesare media third-party (adică recunoașterea vorbirii, analiză sentimentului, recunoașterea feței etc.), care pot fi utilizate în mod transparent de către dezvoltatorii de aplicații ca și celelalte caracteristici incorporate ale Kurento.

În centrul arhitecturii Kurento există un server media numit Kurento Media Server (KMS). Protocolul Kurento permite controlul KMS și se bazează pe standarde clasice pentru a comunica ale internetului precum WebSocket și JSON-RPC.

Kurento poate fii utilizat diferite scenarii vizibile în figura 4.13,iar cele trei scenarii sunt următoarele:

- Utilizarea API-ului Kurento Client direct intr-un browser web compatibil cu tehnologia WebRTC
- Utilizarea librăriei Kurento Client pentru Java într-un server de tip Java EE Application
- Utilizarea API-ului pentru limbaju JavaScrip într-un server de NodeJS

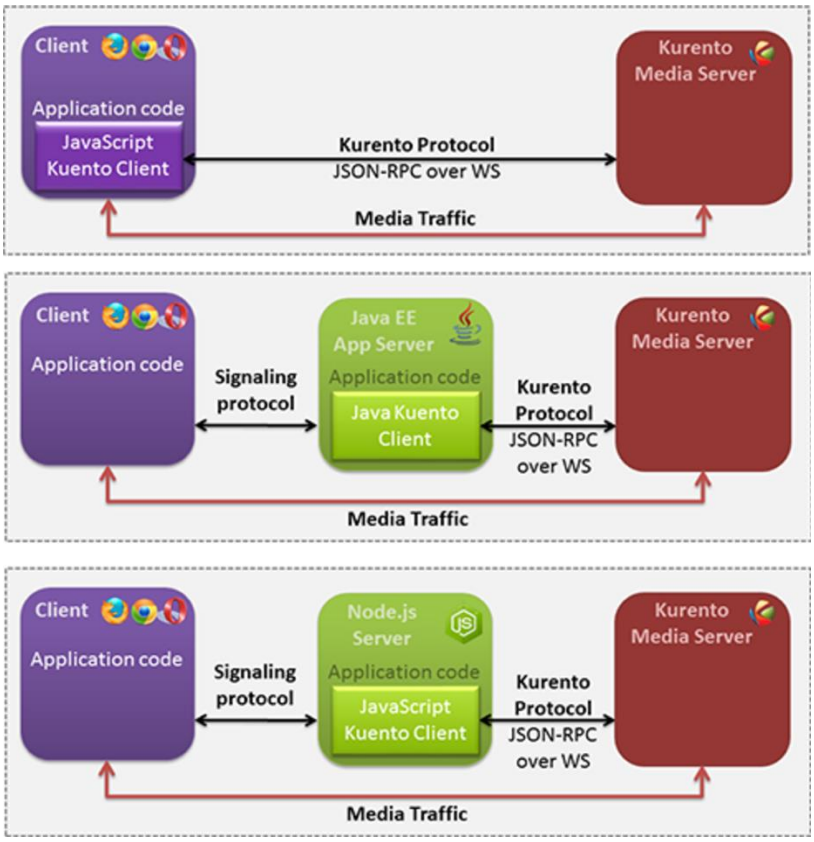

Fig 4.13 Moduri posibile pentru a stabili comunicarea Client - Kurento Media Server [44]

După cum am precizat anterior din motive de rapiditate, consum mai mic de resurse și crearea unor instrumente de video colaborare sigure și performanțe am ales ultima varianta ce presupune utilizarea librărie JavaScript integrată într-un server NodeJS.

API-ul Kurento Client se bazează pe conceptul Media Element. Un element media are un set de capacități media specifice. Spre exemplu elementul media WebRTCEndpoint deține capacitatea de a emite și recepționă fluxuri media WebRTC în timp real. RecordEndpoint are capacitatea de a stoca în sistemul de fișiere a-l serverului etc.

În jargonul Kurento, un grafic al elementelor media conectate se numește Media Pipeline. Conectivitatea este controlată prin intermediul primitei connect expusă de API-ul Kurento Client. Primitivă este invocată întotdeauna pe elementul care joacă rolul de element sursă(emițător) și primește ca argument elementul destinație(receptor)

Pentru a simplifica manipularea fluxurilor media de tip WebRTC pe partea de client Kurento oferă un obiect ajutător oferit de către librăria Kurento-Utilis numit WebRTCPeer. Nu în ultimul rând standardele tehnologiei WebRTC precum getUserMedia, RTCPeerConection pot fii folosite fără a utiliza librăria Kurento-Utils pentru a conecta clientul la un obiect de tipul WebRTCEndpoint.

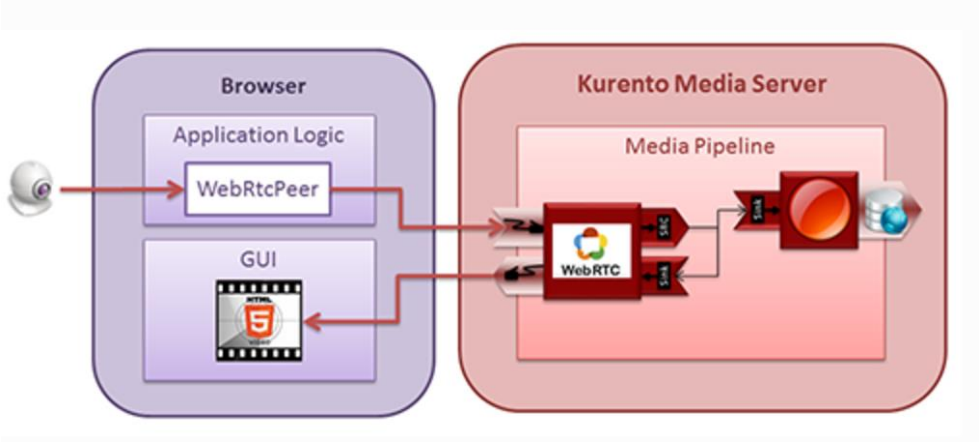

Fig 4.14 Exemplu simplu al unui Media Pipeline [45]

Blocul media serverului comunică cu blocul serverului de semnalizare folosind WebSocket ca protocol și pe se folosește JSON-RPC ca protocol de comunicare între aceste două niveluri ale aplicației

4.5. Codificarea si compresia datelor multimedia

Codificarea datelor multimedia constă în compresia acestora și înaintea trimiterii lor și decompresia lor când au ajuns la destinație. Acest proces de codificare și decodificarea poartă numele de codec. Compresia este necesară pentru ca timpul de trimitere să fie cât mai mic. În paragrafele ce urmează voi prezenta modul în care tehnologia WebRTC gestionează compresia și decompresia datelor. Următoarele tehnologii sunt cele implicite iar userul le poate modifică dacă dorește în momentul în care o conexiune ce conține date multimedia de tip audio-video este inițiată.

VP8 [46] este codec-ul folosit pentru codificarea datelor multimedia de tip video. Acest codec de codificare și decodificare este open-source și este deținut de către Google și vine cu îmbunătățiri substanțiale în cadrul compresiei datelor și oferă posibilitatea de a avea o compresie eficientă a datelor la decodificarea acestora. În același timp complexitatea de calcul a fost semnificativ redusă. VP8 face parte din familia VPx și a fost inițial dezvoltat de către cei de la ON2 Tehnologies.

Principalele funcționalități pe care VP8 dorește sa le ofere sunt următoarele :

- Lungimea de bandă să fie cât mai mică. Intervalul urmărit de dezvoltatori este între ~ 30dB și ~45dB
- sa acopere un număr cat mai mare de device-uri care sa adopte aceasta tehnologie de criptare/decriptare. Începând cu device-urile mobile, continuând cu tablete și nu în ultimul rând laptopurile, care au procesorul format din multe core-uri
- Key Frame: VP8 detectează scenele tăiate și intersecția key frame-urilor. Userul poate specifica numărul maxim de frame-uri/secundă. De exemplu, 30fps 120
- sa permită transformări hibride cu valori adaptive: VP8 utilizează pentru transformare DCT (discrete cosine transform) blocul 4x4. În cazul în care

avem coeficienți DC pentru blocul 16x16, aceștia vor fi transformați folosind transformarea 4x4 Walsh-Hadamard

VP8 nu este însă ultima versiune a tehnolgiilor VPx, deoarece există și VP9. WebRTC permite utilizarea acesteia însă deoarece este o tehnlogie este una recent apărută și care se află într-o versiune beta aceast codec nu este setat implicit de WebRTC.

Formatul implicit de codare și decodare audio se numește **Opus** [47]. Acesta a fost conceput pentru a eficeintiza vorbirea și sunetul general într-un singur format, ramamand în accelasi timp destul de eficient în realizarea unei comunicări interactive în timp real. Opus combină algoritmul SILK cu codificare predictivă liniară orientat pe vorbire și algoritmul CELT bazat pe MDCT cu latentă inferioară, comutând între ele sau combinându-le după cum este necesar pentru o eficientă maximă. Bitrate-ul, lărgimea de bandă audio, complexitatea și algoritmul pot fi ajustate fără probleme în fiecare cadru. Opus are întârzierea algoritmică mică (26,5 ms în mod implicit) cea ce permite să fie utilizat că parte a unor instrumente de comunicare în timp real, permițând conversații naturale. Întârzierea poate fi redusă la 5 ms. Întârzierea să este extrem de scăzută în comparație cu alte codecuri concurente, care necesită mult peste 100 ms, insă Opus este foarte competitiv aceste formate în ceea ce privește și calitatea pe bitrate.

4.6. Transmisia pachetelor multimedia audio-video

Pentru a realiza transmisia pachetelor multimedia in timp real WebRTC utilizează următoarele trei protocoale:

- SIP(Session Initiation Protocol)
- SDP (Session Descripting Protocol)
- RTP(Real Time Transfer Protocol)

SIP [48] Este protocolul destinat inițializării unui semnal. O altă funcție constă în controlul sesiunii. Cel mai des este folosit în cadrul tehnologiei VoIP (Voce peste Protocol de Internet), folosit la apelurile efectuate cu telefoanele mobile. Aceeași funcționalitate o are și în cazul setării unei sesiuni pentru conexiunii destinata ulterior transmiterii datelor multimedia.

SDP [49] este protocolul care standardizează descrierea unei sesiuni. Această descriere constă în "capacitățile multimedia (video/audio), adresa IP și portul disponibil pentru transmiterea datelor, protocolul de transmisie a datelor peer-to-peer (WebRTC transmite date securizate), lățimea de bandă utilizată pentru conexiune, de asemenea, și atributele sesiunii, precum numele acesteia, identificatori, durata activităților etc. SDP este folosit împreună cu SIP, acesta fiind atașat semnalului care se inițiază, având ca rezultat formarea sesiunii. Scopul transmiterii datelor despre sesiune este de a aduce la cunoștință fiecărui browser cine este user-ul care vrea să intre în legătură cu el. Astfel, fiecare browser are toate datele necesare, primite sub format text, pentru a putea lua legătura cu PC-ul utilizatorilor care i-au invitat în meeting.

RTP [50] este protocolul folosit pentru transportul de date în timp real. Datele care sunt transmise sunt informații multimedia (audio/video). Acestea sunt transportate în timp

real printr-o rețea de telecomunicații. Protocolul RTP facilitează transmiterea informațiilor audio și video de cele mai multe ori printr-o conexiune UDP. Protocolul RTP lucrează împreună cu protocolul de control al datelor RTCP (Real-time Control Protocol). RTP a fost dezvoltat pentru conexiuni unicast și multicast (de exemplu, MCU – Multipoint Control Unit). Conexiunile unicast sunt unu la unu, cum este conexiunea peer-to-peer, care include o singură conexiune între doi participanți. În schimb, conexiunea multicast implică mulți participanți

4.7. Diagrama cazuri de utilizare

Principalele cazuri de utilizare a aplicației pot fi deduse din figura 4.15 respectiv 4.16. Principalele tipuri de utilizatori ale sistemului sunt entități de tipul profesor sau elev, care în funcție de tipul cu care s-a înregistrat în sistem are diferite permisiuni și pot realiza activități specifice tipului de utilizator pe care îl au.

Fig 4.15 Cazuri de utilizare profesor

Figura 4.15 prezintă cazurile de utilizare ale profesorului. Dintre acestea se remarcă:

- Autentificare
- Emitere / Receptionare date multimedia in timp real

- Gestionare interbari
- Control asupra sesiunii
- Partajare de continut
- Comunicare prin mesaje text
- Control asupra surselor elementelor multimedia

Cazurile de utilizare ale studentului/elevului pot fii observate în figura 4.16. Dintrere acestea cele cu rolul cel mai semnificativ sunt:

- Autentificare
- Emitere / Receptionare date multimedia in timp real
- Adaugare/Vizualizare interbari
- Vizalizare continut partajat
- Comunicare prin mesaje text
- Control asupra surselor elementelor multimedia

Fig 4.16 Cazuri de utilizare student

Capitolul 5. Proiectare de Detaliu si Implementare

Așa cum am prezentat și în capitolele anterioare lucrarea de față are ca principal obiectiv dezvoltarea unui framework ce oferă utilizatorilor instrumente de video colaborare cu o aplicabilitate în domeniul educației și care trebuie să ofere un sistem intuitiv care să fie ușor de folosit atât de către profesori cât și de către studenți. Pentru a implementa sistemul am folosit mediul de dezvoltarea Sublime deoarece aceasta oferă o gamă mare de plug-uri pentru limbajele JavaScript și Html ce permit oferirea de sugestii și aranjarea a codului automată pentru a putea fii urmărit mai bine. Pentru versioanarea și management al codului sursă am folosit source-controlul GitLab ce permite gestionarea modificărilor aduse fișierelor ce conțin codul sursă al aplicației.

5.1. Arhitectura generala aplicație

Arhitectura generală a aplicației poate fii observată în figura 5.1 din care din punct de vedere arhitectural se remarcă cele patru blocuri (niveluri) logice ale sistemului. Aceste blocuri sunt nivelul de prezentare, unde este implementată interfața utilizatorului folosind tehnologii web precum Html, JavaScript, și Css. Următorul nivel este reprezentat de un server NodeJS care are rol de server de semnalizare pentru modulele de video comunicare, dar tot în acest nivel al aplicației este realizată și logică de bussines a aplicației. Cel de-al treilea nivel este reprezentat de nivelul de date care asigură stocarea datelor în baza de date sau pe sistemul de fișiere al serverului. Ultimul bloc este cel reprezentat de serverul media ce asigură livrarea conținutului media clienților.

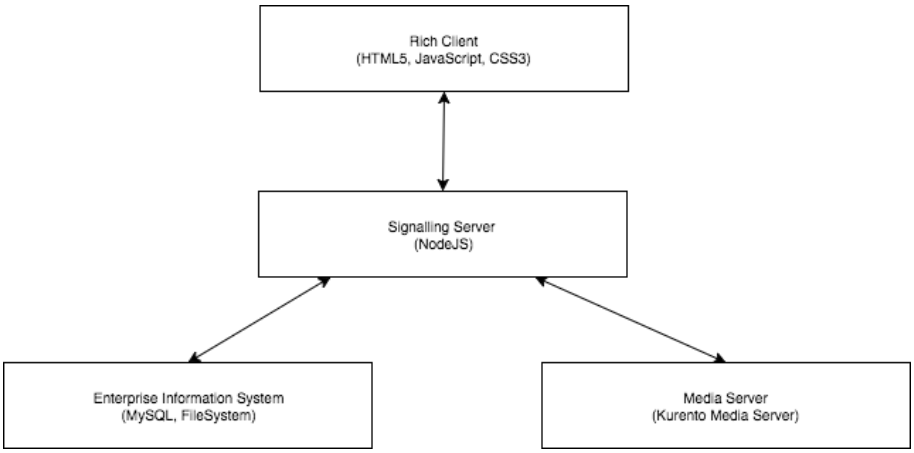

Fig 5.1 Arhitectura generala a sistemului

În figura 5.1 a fost prezentată doar arhitectura generală a sistemului, iar în figura 5.2 este prezentată arhitectura detaliată a sistemului colaborativ sincron din domeniul educației.

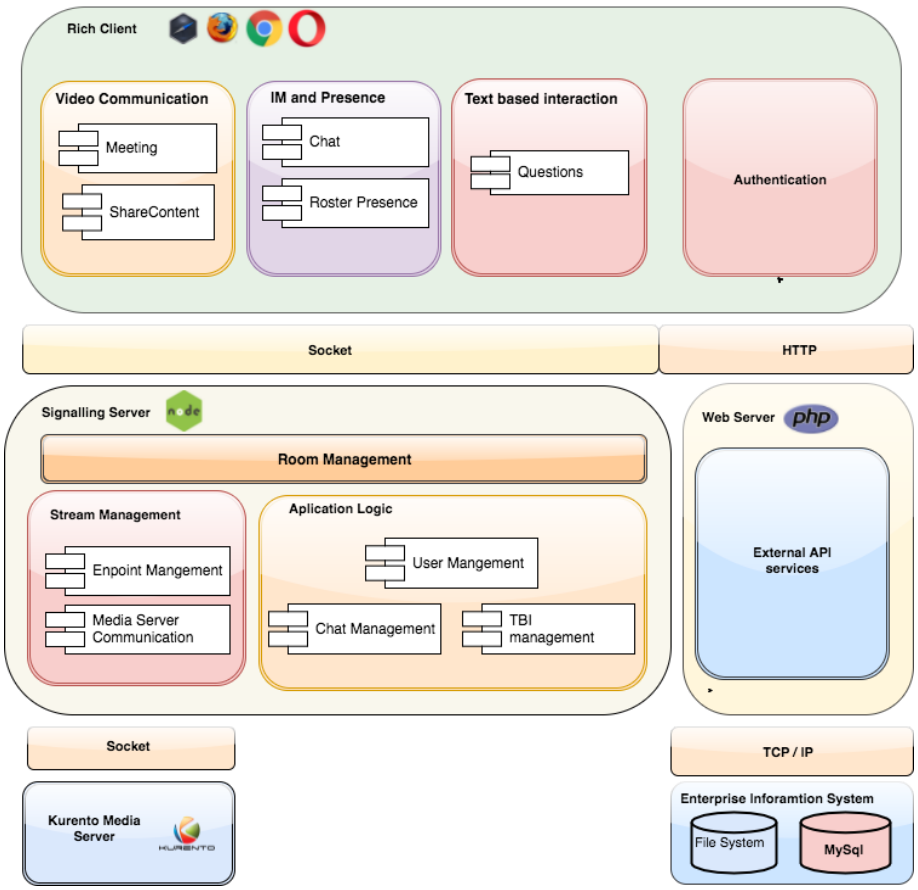

Fig 5.2 Arhitectura Instrumente de video colaborare

Această arhitectură prezintă modulule fiecărui nivel arhitectural. În Primul nivel putem observa modulele Video Communication, Im & Presence, Text Based Interraction și Authentication, respectiv componentele corespunzătoare fiecăruia dintre aceste module. Primul Modul (Video Communication) este responabil de inițierea și gestiunea pe partea de client a unei sesiuni de comunicare sincronă, în schimb ce Im & Presence se ocupă de comunicarea folosind mesajele de tip text sub formă chat și de afișarea listei cu utilizatori conectați în sistem. Ultimul modul asigură comunicarea pe baza mesajelor text fără a întrerupe sesiunea de comunicare sincronă.

În ceea ce privește serverul NodeJS aici putem identifica o componentă comună numită Room Management ce asigură gestiunea sesiuni și alte două module unul ce asigură managementul conținutului multimedia numit Stream Management iar cel de-al doilea numit Application Logic este responsabil cu asigurarea logicii de business a sistemului. Aceste ultime două componente au fost realizate cu ajutor unor componente ce asigură funcționalități specifice.

Nivelul serverului media este și el exemplificat in arhitectura prezentata, iar cu ajutorul figurii 5.2 se poate observa care este modulul cu care acest bloc al sistemului este în directă și strânsă legătură.

Componentele ultimului bloc sunt și ele exemplificate în figură, nivelul de date este format din o bază de date relațională MySQL și din sistemul de fișiere al serverului unde este găzduit (publicat) sistemul.

În figura 5.3 este descrisă distribuirea fizică a informațiilor ale modulelor software ce compun acest sistem de colaborare sincronă, și componentele cu care intereactioneaza utilizatorii când folosesc aplicația.

Instrumentele de video colaborare prezentate au nevoie de mai multe componente hardware pe care programele software să fie livrate. În diagrama de deployement din figura 5.3 sunt prezentate componentele harware care găzduiesc sistemul software implementat. În cazul lucrării de față avem nevoie de cinci servere independente sau mașini virtuale care să găzduiască cele patru blocuri ale sistemului și serverele de STUN și TURN pe ultima mașină.

După cum s-a prezentat și în capitolul 4 utilizatorii pot utiliza aplicația atât dintr-un browser web instalat ce suportă tehnologia WebRTC instalat local pe calculatorul personal al utilizatorului cât și dintr-o aplicație nativă descărcată și instalată peste sistemul de operare. Dacă interfața grafică este accesată dintr-un browser web acesta trimite o cerere către serverul web Nginx care încarcă în browserul utilizatorului componentele interfeței grafice. Dacă se utilizează aplicația nativă atunci componente sunt deja descărcate pe calculatorul utilizatorului și sunt încărcate din memoria lui. Din interfața grafică oferită utilizatorilor se realizează o conexiune socket cu serverul de semnalizare NodeJS, iar după realizarea acestei conexiuni se face un schimb de masaje bidirecțional între aceste două blocuri ale sistemului.

Serverul de semnalizare rulează în contextul unui server NodeJS la un port definit de utilizator. În cazul nostru 443 deoarece aplicația folosește protocolul secure HTTPs.

Media Serverul reprezintă o instanță livrată pe o mașină virtuală a serverului media WebRTC Kurento Media Server, ce rulează la portul 7000.

Blocul responsabil de gestiunea datelor rulează în contextul unui server de baze de date MySQL.

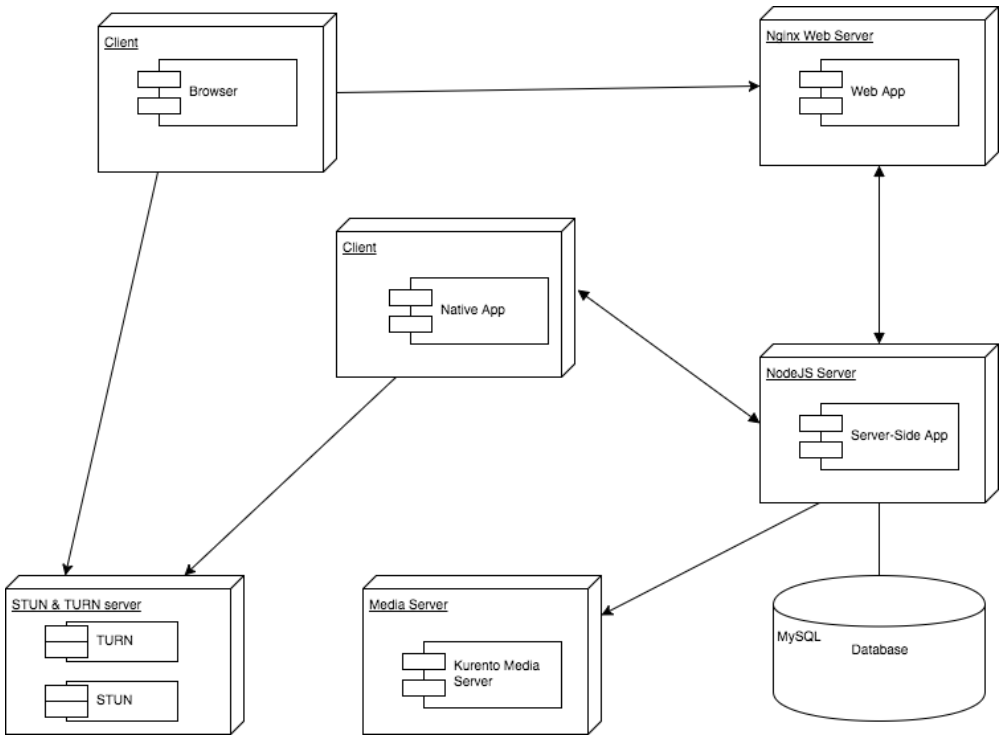

Fig 5.3 Diagrama de Deployement

5.2. Implementarea sistemului de colaborare sincrona

5.2.1. Implementare client

După cum a fost prezentat și în capitolul precedent nivelul cel mai înalt al aplicației cel numit anterior Rich Client este implementat folosind tehnologiile web amintite mai sus, și au fost folosite framework-urile AngularJS împreună cu șablonul arhitectural Model-View-Controller dar și framework-ul Bootstrap pentru a structura mai bine aplicația și pentru a putea permite ca aceasta să fie modificată cât mai ușor iar efortul depus în realizarea acestui demers să fie cel minim.

Pentru a avea datele într-o singură pagină am folosit conceptul de Single Page application, concept foarte bine expus de catre framework-ul ales.

Arhitectura aplicației de client poate fii identificată din figura 5.4 care prezintă diagrama de pachete al acestui bloc al sistemului. Pentru o oraginizarea mai bună am decis utilizarea mai multor directoare. Ramificarea acestor directoare poate fii obervata în figura 5.4.

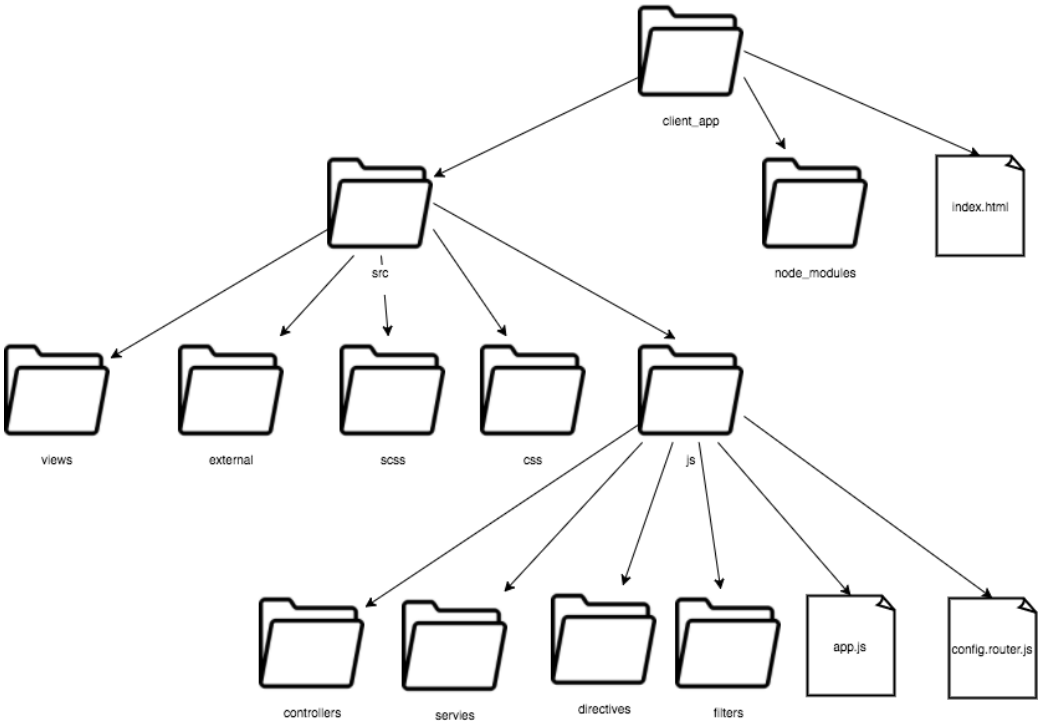

Fig 5.4 Diagrama de pachete a aplicație de client

În continuare o să prezint conținutul directoarelor din figură și rolul pe care acestea îl au:

- index.html: este indexul principal al proiectului. aici sunt incluse toate librăriile, fișierele JavaScript și css ale aplicației.
- node_modules: este directorul în care sunt instalate librăriile cu ajutorul managerul de pachete npm.
- src: este directorul unde se află întreg codul sursă al aplicației de client care

conține și el următoarele directoare

- views: director ce conține toate fișierele de tip HTML ale aplicației.
- scss: director ce conține fișierele de tip scss care mai apoi o sa fie compilate în fișiere de tip css pentru a putea fii interpretate de către browser și aplicațiile native
- css: conține fișierele compilate menționate mai sus
- external: director ce conține fișiere de configurare de tip JSON sau imagini folosite în aplicație.
- js: este directorul ce conține tot codul sursa de tip JavaScript, și acest director este împărțit în mai multe sub-directoare ce în funcție de rolul pe care aceste pe care le conțin îl au:
  - controllers: director ce conține toate controller-ele aplicației
  - sevices: conține fișiere ce reprezinta un fișier de service
  - directives: conține toate controller-ele directivelor
  - filters: fișiere care au rolul de a filtra anumite elemente de UI
  - acesta mai conține și două fișiere app.js și config.route.js elemente de bază ale unei aplicații ce folosește framework-ul AngularJS

Fișierul index.html conține atașat tag-ul ng-app=”app”, unde app reprezintă numele aplicație și acest obiect va fii folosit mai departe pentru a seta și defini celelalte componente.

Fișierul app.js conține definirea modului aplicației AngularJS iar în fișierul config.router.js este implementata rutarea paginilor folosind servicii precum $stateProvider, $urlRouterProvider, $locationProvider trimise ca parametru funcției .config conținută în acest fișier. Un exemplu de rutare poate fii observat în figura 5.5 unde este prezentată rutarea pagini de meeting unde este specificat ce view este folosit la ce url este găsită pagină și ce controller și fișier css să folosească. Pe lângă cele menționate sus în acest fișier este definită și funcția load care are rolul de a încărca pe rând fișierele și a evita cazuri în care fișierul css se încarcă mai repede decât cel html sau js.

```
.state('main.meeting', {
    url: '/meeting',
    templateUrl: 'src/views/meeting.html',
    resolve: load([
        "src/css/meeting.css",
        "src/js/controllers/MeetingCtrl.js"
    ]),

})
```

Fig 5.5 Rutarea pagini de login

În funcție de url-ul la cere ne aflăm în browserul web sau în aplicația nativă o să fie încărcate un view un controller, un fișier de design CSS și diferite fișiere de service sau

directive dacă acestea sunt necesare. Controller-ele sunt specifice unui singur view și fiecare view are specificat un controller folosind directiva ng-controller=”exemepluCtrl.js”.

După autentificare utilizatorul este redirectat automat în pagina main a aplicației, iar o directivă meeting se încarcă automat. Aplicația este divizată în trei taburi fiecare tab fiind definit independent. Aceste tab-uri sunt Meeting unde este implementată componenta de video comunicare, Content Sharing care are rolul de a asigura partajarea de conținut și Questions tab-ul cu ajutorul căruia a fost implementat modulul de Text Based Intercation. Utilizatorul poate comuta între aceste trei tab-uri iar dacă apare o modificare în ultimele două tabul primește o iconiță de notificare ce atenționează utilizatorul că s-a pus o întrebare de către un alt participant la sesiune sau a fost partajat un conținut de către profesor. Pe lângă aceste directive pagina de main conține partea de sus a pagini informații despre sesiune cum și anumite elemente ce permit controlul asupra elementelor multimedia proprii (cameră web, microfon), dar și alte două directive una pentru comunicarea text(Chat) și una pentru lista de participanți. Aceste două directive sunt și ele ascunse în spatele unui tab ce permite comutarea între directive după bunul plac. În funcție de rolul și primit la login elev/student/student activ elementele de UI sunt încărcate dinamic permițând anumite operații specifice sesiunii de comunicare sincronă doar utilizatorilor ce dețin acest drept.

În pagina Index.html a sistemului de colaborare sincronă este injectat controllerul AppCtrl.js ce conține funcții precum init sau anumite filtre de securitate pe care elemenetele multimedia video ce urmează a fii partajate tebuie să le treacă. Funcțiile deifinite în acest controller sunt accesibile din orice alt controller sau view ale aplicației. Asemantor cu acest controller este și controller-ul MainCtrl.js definit prin o directivă în view-ul Main.html, iar acesta este la rândul său accesibil în controller-ele directivelor mapate peste acest view central al aplicației.

MainCtrl are rolul de a asigura logica apelată de elementele comune ale aplicației și totodată asigură și funcții folositoare în restabilirea conexiuni cu serverul cazul unor reconectări datorate unor probleme și de a reoferi utilizatorului privilegiile pe care acesta le avea înaintea acestor probleme

Pentru a putea păstra datele sesiunii existențe în cazul unei reconectări, a reîncărcării pagini sau în cazul în care un utilizator părăsește aplicația și dorește să o refolosească și dacă sesiunea respectivă mai este valabilă s-a folosit serviciul implicit oferit de Angular numit $cookies care se folosește de abilitatea browserului web de a salva fișiere text în sistemul de fișiere al calculatorului. Acestea sunt păstrate un timp variabil în memoria calculatorului, acest timp este setat în funcție de timpul rămas al sesiuni în momentul în care dorim să salvăm aceste informații. Acest lucru poate fii schimbat ușor și să fie setat ca timp de expirare închiderea tab-ului sau a întregii sesiune a browserului web.

Controller-ul pagini de login are rolul de a face request către serviciul de autentificare cu email-ul sau parola introdu-se de către utilizator de a parsa datele primite și pregătirea lor pentru folosirea în inter-schimbul de mesaje cu serverul de semnalizare NodeJS.Nu în ultimul rând în acest controller sunt făcute anumite verificări precum:

- verificarea browserului web folosit și atenționarea utilizatorilor în cazul în care folosesc un browser ce nu oferă suport pentru tehnologiile utilizate (WebRTC)
- verificarea dacă sesiunea este încă valida iar în cazul în care această a expirat sau nu a început încă, utilizatorul să fie notificat aplicația să nu poată fii accesată.

După cum am precizat și în capitolul 4 pentru a avea acces la variabile locale și globale am utilizat variabilele implicite definite în framework, $scope ce este vizibil doar în interiorul controller-ului respectiv și $rootScope la care avem acces de orice controller sau directivă a aplicației.

Modulele aplicației de client sunt vizibile în figura 5.6 la fel și componente fiecărui modul. Acestea vor fii detaliate în următoarele paragrafe și sunt următoarele:

- VideoCommunication - compus din:
  - Meeting
  - Shared Content
- IM and Presence - cu urmatoarele componente:
  - Chat
  - Roster Presence
- Text Based Interraction, ce are o singura componenta:
  - Questions
- Autentication

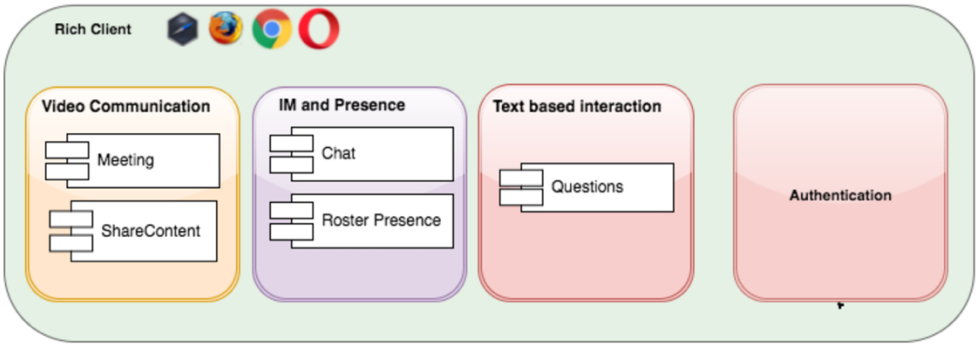

Fig 5.6 Module și componente aplicației client-side

Modulul ***Video Communication*** are la bază tehnologia WebRTC și are că scop asigurarea transmisiei video în timp real între clienți și partajară de conținut în timp real cum ar fii partajarea ecranului sau a unei aplicații deschide pe ecran dar și a unor elemente multimedia și controlul asupra acestor elemente

Prima componentă a modului, componenta ***Meeting*** are la bază controller-ul MeetingCtrl.js care asigură logica acestei componente dar și meeting.html și meeting.css ce asigură designul și modul de redarea al componentei.

În implementrea acestei componente au fost utilizate și un număr semnificativ de fișiere de tip service precum: *SocketFactory, CookieService, MediaDevicesFactory, PresenterVideoTagsFactory, VideoSlotSettingsFactory, MediaOptionsFactory, DisconnectingFactory*. Am ales crearea acestor service-uri deoarece multe funcții au fost preluate și în implementarea componenței ShareContent.

Pe lângă cele menționate mai sus am utilizat și directivele implicite oferite de către framework în încercarea de a dezvolta o aplicație care se bazează pe anumite standarde. Acestea sunt $scope, $rootScope, $http, $window, $cookies.

După ce conexiunea socket cu serverul de semnalizare a fost creată în modulul de autentificare, mai exact în controller-ul LoginCtrl.js, din acesta componentă putem comunica direct cu serverul NodeJS al aplicației. În cazul în care rolul primit la login este cel de profesor se primește un eveniment pe socket ce anunța acest lucru și începe procesul

de inițializare al comunicării în timp real, creându-se un obiect de tip WebRTCPeer pentru profesor și se urmează flow-ul de trimitere de date timp multimedia audio/video către serverul media Kurento Media Server. Dacă rolul primit este însă de student și a acesta a primit rol de user activ în prealabil se va inița flow-ul pentru ce asigură primirea datelor în timp real transmise de profesor și de studenții activați în prealabil de către profesor.

Pași care trebuie urmați pentru emiterea datelor în timp real și implementați în controller-ul MeetingCtrl.js sunt următorii:

1. stabilirea și setarea opțiunilor pe bază căruia se va crea un obiect de tip WebRTCPeer.Aceste constrângeri de comunicare sunt realizate cu ajutorul service-urilor și pot fii următoarele:
    a. tag-ul video unde se vor atașa elementele multimedia de tip audio/video locale
    b. constrângeri legate de dimensiunea imaginii video, frame rate-ul, și sursele camerei web sau a microfonului
    c. configurarea serverelor de STUN respectiv TURN
2. imediat după executarea primului pas browserul afișează o fereastră prin care userul este întrebat dacă dorește să partajeze date ce provin de la cameră web și microfon, iar dacă nu acceptă conexiunea se oprește aici în caz contrar se trece la punctul 3
3. crearea obiectului de tip WebRTCPeer care poate fi unul din următoarele 3 tipuri
    a. WebRtcPeerSendonly - doar emite date (utilizat în cazul de fata)
    b. WebRtcPeerRecvonly - doar recepționează date
    c. WebRtcPeerSendRecv - emite și recepționează date
4. dacă obiectul a fost creat cu succes se apelează funcția de generare a ofertei SDP pentru obiectul WebRTC proaspăt creat webRTCPeer.*generateOffer().*
5. trimiterea ofertei la serverul de semnalizare NodeJS
6. apelarea funcției de procesare a răspunsului primit de la server *webRTCPeer.processAnswer()*
7. generarea de candidați Ice locali și trimitere la server
8. adăugarea candidaților primiți de la serverul de semnaliza prin apelarea *webRTCPeer.addIceCandidate(candidate)*
9. generarea mesaj *ice.gathering.completed* ce reprezinta realizarea cu succes a conexiuni cu serverul media

Pașii pe care care trebuie urmați pentru recepționarea datelor multimedia de tip audio/video sunt în mare parte aceeași singurele modificări sunt la pasul 2 unde trebuie să setăm tag-ul video aferent stream-ului care urmează a fii primit de la media server, la pasul 3 unde obiectul creat este de tip WebRtcPeerRecvonly, iar la sfârșit apare un pas care adaugă stream-ul primit de la server în tag-ul video precizat la pasul 2, iar această flow este apelat recursiv pentru fiecare utilizator activ primit pe mesajul Joined de la server.

Utilizatorul fie ca el este student/profesor are control local asupra elememtelor multimedia randa-te în browser-ul sau aplicația lor nativa, modificând locul de randare, volumul, s.a.m.d.

Pasul 3 și pasul 4 pot fii observați și în figură 5.7, unde este prezentată porțiunea de cod din contoller-ul MeetingCtrl.js care asigură crearea obiectului WebRTCPeer și

apelarea funcției ce crează oferta sdp a acestui obiect în pentru crearea unui obiect ce recepționează stream-ul media în timp real.

```
$rootScope.webRtcPeers[presentersList[index].id] = kurentoUtils.WebRtcPeer.WebRtcPeerRecvonly(options, function(error) {
    if (error) return onError(error);
    this.generateOffer(function(error, offerSdp) {
        offerToReceiveVideoViewer(error, offerSdp, presentersList[index].id, messageId);
        index++;
        setTimeout(function() {
            createPeerForEachPresenter(presentersList, index, messageId, callback)
        }, 500);

    });
});
```

Fig 5.7 Creare obiect WebRTC și apelare funcție ce generează oferta sdp

Conceptul prezentat și utilizat în implementarea componenței Meeting a fost preluat și în componenta *SharedContent,* singura diferență majoră este că datele media nu sunt captate de la o cameră web ci ele sunt un ecran sau o aplicație ce rulează pe ecranul calculatorului în acel moment.

Fișierele de tip service utilizate în componenta Meeting au fost reutilizate și în implementarea componentei de față, alte diferențe ar fi numele mesajelor prin care se realizează comunicarea cu serverul.

În acest tab au fost reprodu-se componenta video pentru a avea și o imagine cu cel ce prezintă conținutul partajat pe lângă posibilitatea de a vedea acest conținut. Acest lucru a fost posibil prin adăugarea unui tag video unde este randat stream-ul video primit de la media server al profesorului și totodată au fost create tag-uri audio pentru a putea auzi și restul studenților activi în momentul respectiv.

Pentru a putea captura datele de la un ecran am fost nevoit să utilizez librăria open-sorce getScreenId ce reușește să captureze obiectul de stream media al ecranului, și returnează id-ul acestei surse care mai apoi este setat ca opțiune în crearea obiectului WebRTCPeer și trimis serverului. Pentru ca această librărie să funcționeze, utilizatorul este nevoit să instaleze în prealabil o extensie suplimentară în browser, însă aceasta instalare nu este necesară în cazul utilizării aplicației native .

Pentru a realiza partajarea unor elemente video care nu sunt în timp real nu este nevoie să folosim serverul media, profesorul partajează cu restul elevilor un url care conduce la un element multimedia, acesta este redat pe interfețele tuturor elevilor iar profesorul are control absolut al acestor elemente prin apelare funcțiilor play(), pause(), show(), hide() s.a.m.d.

Fișierele care compun acesta componenenta sunt controller-ul ShareScreen.js, view-ul shareScreen.html și shareScreen.css.

Modulul ***Im and Presence*** are la bază componentele *Chat și Roster Presence*. Acest modul are la bază Api-ul Socket.IO, iar contribuția personală în realizarea acestui modul a fost personalizarea framework-ului, cu scopul de a asigura afișarea studenților conectați în listă de participanți și comunicarea folosind mesaje de tip text timp real.

Aceste două componente sunt implementate cu ajutorul directivelor *chat și participants-list* ce sunt incluse în pagină Main.html a sistemului. Utilizatorul comuta între aceste două tab-uri pentru a afișa componenta pe care o dorește, iar dacă există un

eveniment pe componenta de chat utilizatorul va fii notificat cu ajutorul unui badge. Importul acestor doua directive este vizibil în bucata de cod din figura 5.8

```
<div class="tab-content">
    <div id="participants" class="tab-pane fade in active">
        <participants-list></participants-list>
    </div>
    <div id="chat" class="tab-pane fade in">
        <chat></chat>
    </div>
```

Fig 5.8 Integrarea directive chat și participants-list în pagina Main.html

Componenta *Chat* asigură comunicarea prim mesaje text între participații conectați la sesiune în momentul respectiv. Un mesaj este emis de client către server care mai apoi face un broadcast către restul utilizatorilor iar un eveniment de pe client interceptează mesajul și îl afișează. Momentan informațiile afișate în această componentă sunt mesajul text,numele utilizatorului, ora la care a fost recepționat mesajul și avatarul utilizatorului.

Roster Presence este componenta ce asigură afișarea utilizatorilor în lista de participanți și oferă profesorului control asupra listei cu studenții. Acesta poate activa un utilizator iar rolul acestuia se va modifica în participant activ la sesiunea de colaborare sincronă, îi poate modifica anumite setări asupra conexiuni, setări accesibile la nivel global cum ar fii să îi oprească microfonul sau camera web. Această componenta folosește de asemenea Socket.IO prin un set de evenimente emise către și recepționate partea de client din directiva participants-list.directive.js. Și studentul are control asupra userului propriu din lista de participanți și poate folosi butonul de *raise haind* care atenționează profesorul ca ar dori să participe la sesiunea sincronă.

Interacțiunea pe baza mesajelor text între studenți și profesor fără a întrerupe sesiunea în desfășurare se realizează cu ajutorul Modului Text Based Interraction și mai exact ajutorul componenței *Questions*.

Componenta este implementat în două view-uri diferite ce se încarcă dinamic pe baza tipului de utilizator obținut la autentificarea în aplicație. Prin urmare au fost create două stări în fișierul de configurare al rutelor.

```
.state('main.viewerQuestions', {
        url: '/viewerQuestions',
        templateUrl: 'src/views/viewer-questions.html',
        permission: "notpresenter",
        otherwise : 'main.presenterQuestions',
        resolve: load([
            "src/css/viewerQuestions.css",
            "src/js/controllers/ViewerQuestionsCtrl.js"
        ]),
    })

    .state('main.presenterQuestions', {
        url: '/presenterQuestions',
        templateUrl: 'src/views/presenter-questions.html',
        permission: "presenter",
        otherwise: 'main.viewerQuestions',
        resolve: load([
            "src/js/controllers/PresenterQuestionsCtrl.js",
            "src/css/presenterQuestions.css",
        ]),
    })
```

Fig 5.9 Încărcare dinamica a componentei Questions

Încărcarea dinamica a acestei componente poate fii observata în figura 5.9, unde sunt de asemenea prezentate permisiunile pe care un utilizator al sistemului trebuie sa le aibă pentru a a accesa pagina respectiva si varianta care va fii afisata in cazul in care permisiunile nu sunt îndeplinite.

Componenta de întrebări a profesorului are la bază următoarele fișiere presenter-questions.html, PresenterQuestionCtrl și presenterQuestion.css iar cea a studentului viewer-questions.html, ViewerQuestionCtrl și viewerQuestion.css.

La bazele componentelor stă Api-ul Socket.IO, iar componenenta este realizată prin inter-schimbul de mesaje între aplicația de client și cea de server de semnalizare, studenții emit evenimente pe socket precum postarea unei întrebării, like pentru o întrebare postată, dar și prinderea unor evenimente emise da către socket unde sunt expuse informații precum lista de întrebări acceptate, lista cu întrebările proprii etc. Profesorul primește întrebările postate de către studenți pe canalul socket pe care îl are deschis cu serverul și răspunde prin alte mesaje cum ar fii acceptarea sau refuzarea întrebări sau schimbarea stării pentru întrebarea respectivă.

Ultima componenta de autentificare în sistem apelează un api-extern implementat în php cu framework-ul php. Pentru a apela serviciul se trimit că parametri emailul și token-ul din formular pagini de login. Acesta trimite ca răspuns un obiect de tip JSON care este parsat iar mai apoi informațiile sunt trimise spre serverul de semnalizare pentru a accesa camera respectivă.

5.2.2. Implementare Server

La fel ca și în cazul clientului acest bloc a fost implementat folosind limbajul de programare JavaScript, diferența fiind că în acest context codul nu este interpretat de browser ci este rulat în contextul unui server NodeJS.

Modulele compun această componentă sunt *Stream Management, RoomManangement și AplicationLogic*. Acestea la rândul lor sun implementate cu ajutor unor componente independente pentru a se conformă arhitecturii alese, cea a microserviciilor

Figura 5.10 prezinta arhitectura acestui nivel al aplicatiei. Modulele mentionate pot fi observate in aceasta figura. Totodata din figura putem observa ca modulul RoomManagement este unul comun, iar modulele Stream Management si Application Logic folosesc acest modul in implementarea functionalitatiilor specifice pe care aceste doua module le acopera.

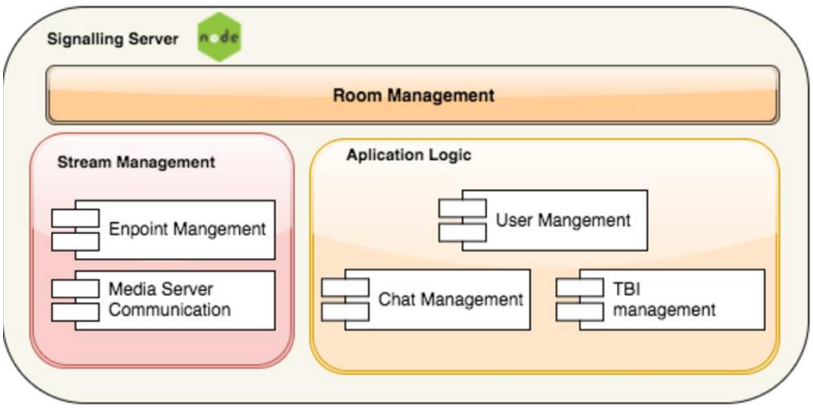

Fig 5.10 Arhitectura de detaliu server de semnalizare NodeJS

Pentru a putea utiliza aceste componente implementate precum și alte componente trebuie apelată funcția require("modulName"). Pentru a expune aceste componente am folosit noțiunea de module oferită de către nodeJS. Un modul încapsulează codul creat în fișierul respectiv între-un singur obiect și expune câmpuri sau metode ale sale clasei care crează o instanță a acestui modul. Din această cauza fiecare modul la sfârșitul sau în începutul fișierul trebuie să folosească funcția exports și să se specifice obiectul sau funcțiile pe care modulul creat le crează și dacă la instanțierea acestui modul este nevoie de anumiți parametri.

Ca și orice alt server și un server NodeJS rulează la un anumit port și folosește un anumit protocol de comunicare. În cazul nostru portul se introduce în linia de comandă în momentul lansării în execuție a serverului iar protocolul de comunicare folosit este https. Pentru a putea ridica un server https este necesar un obiect numit options care conține certificatele ssl. Serverul poate fii utilizat doar dacă în prealabil a fost instalat pe server Nodejs și este pornit utilizând comanda din terminal node server.js și portul în cazul de față 8443.

În figura 5.11 este prezentată metoda care permite ridicarea unui server https de NodeJS la un port specificat în linia de comandă și setarea că configurarea socket.io pentru un server de acest tip.

```
 */
var options = {
    key: fs.readFileSync('./key/server.key'),
    cert: fs.readFileSync('./key/server.crt')
};
/*
    node js server config server
 */
var app = https.createServer(options).listen(server_port, function(err) {
    console.log('Server is listening at port ' + server_port + '...');
    winstonLogger.info('Server is listening at port ' + server_port + '...');
});
/*
    socket io settings
 */
var io = require('socket.io', 'socket.io-emitter').listen(app, {
    'pingInterval': 2000,
    'pingTimeout': 6000,
    'origins': '*:*'
});
```

Fig 5.11 Configurarea serverului ce folosește ca protocol https și a socket.io

După cum spuneam componenta Stream Management are că principal scop asigurarea transmisiei conținutului media între participanti unei sesiuni sincrone, iar rolul său este să asigure consistența între obiectele de tip WerRTCPeer din blocul de client și obiectele WebRTCEnpoint de pe server. Nu în ultimul rând este responsabilă cu crearea unui media pipeline pentru fiecare cameră în care se desfășoară o sesiune de comunicare sincronă. Toate obiectele de tip WebRTCEndpoint din cameră respectivă vor fii mai apoi create în contextul acestui media pipeline.

Conexiunea cu serverul media este realizată cu ajutorul acestei componente, dezvoltatorul trebuie să specifice ip-ul și portul de websocekt la care ascultă serverul media. Pentru această trebuie importat modulul librărie Kurento-Client și creată o instanță a acestui modul trimițând că parametru path-ul către serverului media. Acest lucru este realizat cu ajutorul funcției getKurentoClient(), iar după executarea cu succes și crearea instanței se poate crea media pipelineul utilizând funcția kurentoClient.create('MediaPipeline', function(error, pipeline){}) iar mai apoi, se poate trece la crearea obiectelor de tip WebRTCEndpoint.

Pentru managementul obiectelor de tip WebRTCEndpoint este folosit modulul userSession, unde sunt stocate în patru liste obielele endpoint pentru fiecare ușer. cele patru liste sunt:

- WebRTCEndpoint_outgoingMedia[] stocate endpoint-urile care emit date multimedia de tip audio/video
- WebRTCEndpoint_incomingMedia[] ce stochează obiectele create petnru a recepționa datele multimedia în timp real de tip audio vide
- WebRTCEndpoint_outgoingMediaScreenShare[] asemănător cu WebRTCEndpoint_outgoingMedia dar sunt referințe către obiectele dedicate partajări de conținut
- WebRTCEndpoint_incomingMediaScreenShare[] sunt stocate obiecte care

recepționează conținut partajat în timp real.

Pe lângă aceste liste modulul implementează și funcții pentru managementul acestora(addEndpoint,deleteEntpoitn,getEndpoint,etc).

După crearea endpointurilor, trebuie facă conectarea lor unde la enpoin-nul sursă dorit din lista WebRTCEndpoint_outgoingMedia se face *connect* cu endpoint-ul destinație din lista WebRTCEndpoint_incomingMedia a utilizatorului care o să recepționeze datele. Dacă conectarea obiectelor a avut loc cu succes se trece la procesarea ofertei primite de la aplicația de client pentru fiecare user. Acest lucru este realizat cu ajutoul funcție connectViewerToPresenter().

Mai apoi cu ajutorul socket.io răspunsul obținut după procesarea ofertei este trimis clientului, și se trece la pasul de schimb de candidați Ice, lucru care este identic cu cel din aplicația de client prezentat în capitolul 5.2.1.

În cazul în care un participant activ își pierde drepturile de emitere resursele sunt eliberate cu ajutorul funcție deactivatePresenter() unde referințele obiectelor sunt șterse din listele prezentate iar obiectul în sine este distrus prin apelul userSessionData.outGoingMedia.release();

Modulul comun atât componentelor ce țin de componența de colaborare sincronă cât și cele care realizează comunicarea prin mesaje text sau modulul de questions este numit *Room Management*. Acesta folosește conceptul de room oferit de socket.io și se asigură că fiecare client să fie repartizat în camera în care trebuie și să creeze o un obiect de tip Ușer pentru utilizatorul care s-a conectat. Pentru a distribui utilizatorii pe mesajul inițial trimis de client *"create or join"* să implementat logică care verifică room-ul în care acesta trebui distribuit și se apeleaza funcția oferită de socket.IO socket.join(user.room). Funcția care crează un obiect de tip ușer este oferită de modulul UserRegistry, și se realizează cu ajutorul funcției registerUser(). Funcția registerUser este vizibilă în figură 5.12

```
function registerUser(socket, userEnteredInfo, callback) {
    var userSession = new UserSession(socket.id);
    userRegistry.register(socket.id);
    userRegistry.setName(socket.id, userEnteredInfo.name);
    userRegistry.setEmail(socket.id, userEnteredInfo.email);
    userRegistry.setType(socket.id, userEnteredInfo.type);
    userRegistry.setRole(socket.id, userEnteredInfo.role);
    userRegistry.setRoom(socket.id, userEnteredInfo.room);
    userRegistry.setRoomId(socket.id, userEnteredInfo.roomId);
    userRegistry.setSessionId(socket.id, userEnteredInfo.sessionId);
    userRegistry.setImage(socket.id, 'user1');
    userRegistry.setCurrentStatus(socket.id, 'online');
    userRegistry.setRaiseHand(socket.id, 'false');
    userRegistry.setSdpOffer(socket.id, null);
    userRegistry.setUserSession(socket.id, userSession);
    setUserIncomingMediaEndpoints(userRegistry, userSession, socket.id, function() {
        callback();
    });
}
```

Fig 5.12 Functie de creare user in serverul NodeJS

Figura 5.13 prezintă diagram de clase prezentă la nivelul nivelului al doilea al sistemului de comunicare sincronă. Din această putem observă că sistemul are o clasă centrală server.js care utilizează alte module pentru a asigură managementul elementelor multimedia transmise în timp real și a asigură logică de business a sistemului. Clasele instantiate direct în clasa centrală a sistemului sunt UserRegistry, QuestionServer, Room, VideoSharing. Primele două folosesc de asemenea câte o clasă ajutătoare, mai exact UserSession instantiata în clasa UserRegistry, respectiv QuestionRegistry utilizat de clasa QuestionServer.

Prin instantierea unor obiecte de tipul claselor ajutătoare se pot utiliza din modulele centrale metode definite în interiorul acestor clase. Prin împărțirea codului în mai multe clase sistemul este mai ușor de întreținut sau de urmărit.

Pentru ca serverul să fie cât mai bine definit, clasele prezentate în figura 5.13 implementează anumite funcționalități specific, excepție însă face clasa centrală care are rolul intermediar între aplicația de client și serverul media sau logica de business asigurată de clasele importate în acesta clasă centrală. După cum spuneam clasă UserRegistry asigură gestionarea utilizatorilor în schimb ce gestionearea elementelor multimedia ale fiecărui utilizator sunt asigurate de către clasa UserSession. Gestionarea resureselor fiecărei camere virtuale sunt asigurarate de către clasa Room în schimb clasa VideoSharing asigură gestionare și controlul asupra elementelor multimedia partajate. Gestionarea întrebărilor adresate de către studenți și partajarea lor întregii clase virtuale este realizată cu ajutorul clasei Question Server, iar QuestionRegistry se ocupă de stocare permanentă a tuturor informațiilor legate de acest tip de conținut.

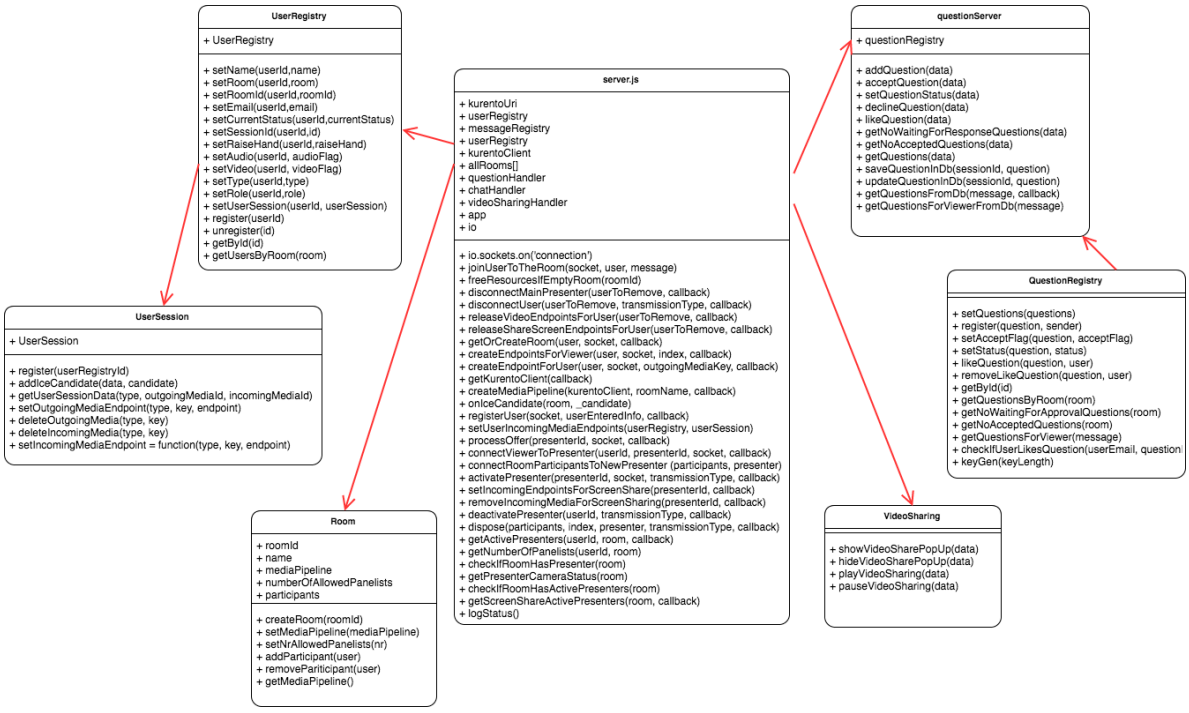

Fig 5.13 Diagrama de clase server de semnalizare NodeJS

Modulul *Application Logic* ce are la bază componentele *User Management, Chat Management, Text Based Interraction Management* trebuie să asigure consistență datelor afișate și oferite clienților. Această componentă are la bază Api-ul Socket.io cu ajutorul cărora se face o gestionarea a mesajelor text, și a întrebărilor adresate de studenți.

Componenta *User Management* verifică modificarea stărilor utilizatorilor (user nou, user deconectă, user reconectat) și în functie de starea respectivă apelează funcții care crearea endpoint-uri, eliberează resurse sau notifică restul clienților atunci când un apare un nou utilizator sau când se deconectează. unul existent.

Chat Management trebuie să asigure că un mesaj trimis pe socket de un client ajunge la toți utilizatorii camerei, și apelează un serviciu al api-ului extern de php ce oferă un serviciu rest de salvare a întrebării în bază de date sql. La crearea unui room componența interoghează bază de date prin un alt serviciu pentru vedea dacă sesiunea nu a existat în prealabil și au existat alte mesaje transmise înaintea redeschiderii sesiunii.

Ultima componentă a modului cea numită *Text Based Interraction Management* asigură distribuirea și gestionarea elementelor de colaborare bazate pe text precum cele ale întrebărilor postate de către studenți, care cu ajutorul socket.io sunt trimise profesorului. Dacă acesta accepta o întrebare serverul trebuie să facă întrebarea disponibilă și celorlalți studenți, pentru a putea fii vizualizata. Tot cu ajutorul socket.io profesorul poate să schimbe starea unei întrebări și să o marcheze că răspunsă, în așteptarea răspunsului sau că în momentul de față se răspunde la întrebarea respectivă. Asemănător cu componenenta Chat Management TBI Management consumă servicii de la același api pentru salvarea întrebărilor pe sesiuni și posibilă interogare în caz de necesitate.

5.3. Diagrame de secventa

În figura 5.14 este descrisă o diagrama de secvența prin care este prezentat interg proceul prin care un utilizator al aplicației în cazul de față profesorul inițiază comunicarea cu serverul media și trimite datele multimedia de tip audio/video în timp real. Pentru această este necesar că utilizatorul să acceseze aplicația nativă sau să încarce din browser pagină de login a aplicației. După acea urmează introducerea și trimiterea token-ului către pagină api-ul de autentificare care returnează anumite date legate de sesiune și le trimite serverului de nodeJS pentru a inițializa această comunicare sincronă inter user și server media. După înregistrarea user-ului pe serverul de nodeJS obiectul user este trimis clientului și sunt create resursele necesare comunicării în acest fel. După crearea lor sunt trimise serverului se crează și pe acesta resursele necesare și date precum răspunsul la oferta sdp sunt trimise înapoi clientului. După aceea începe procesul de interschimb de candidați ice între server și client. Iar când acest interschimb de mesaje e terminat fluxul media între clientul web și serverul media este realizat. Dacă clientul se deconectează acesta notifică severul care eliberează resursele create pentru a eficientiza consumul de resurse de pe serverul media.

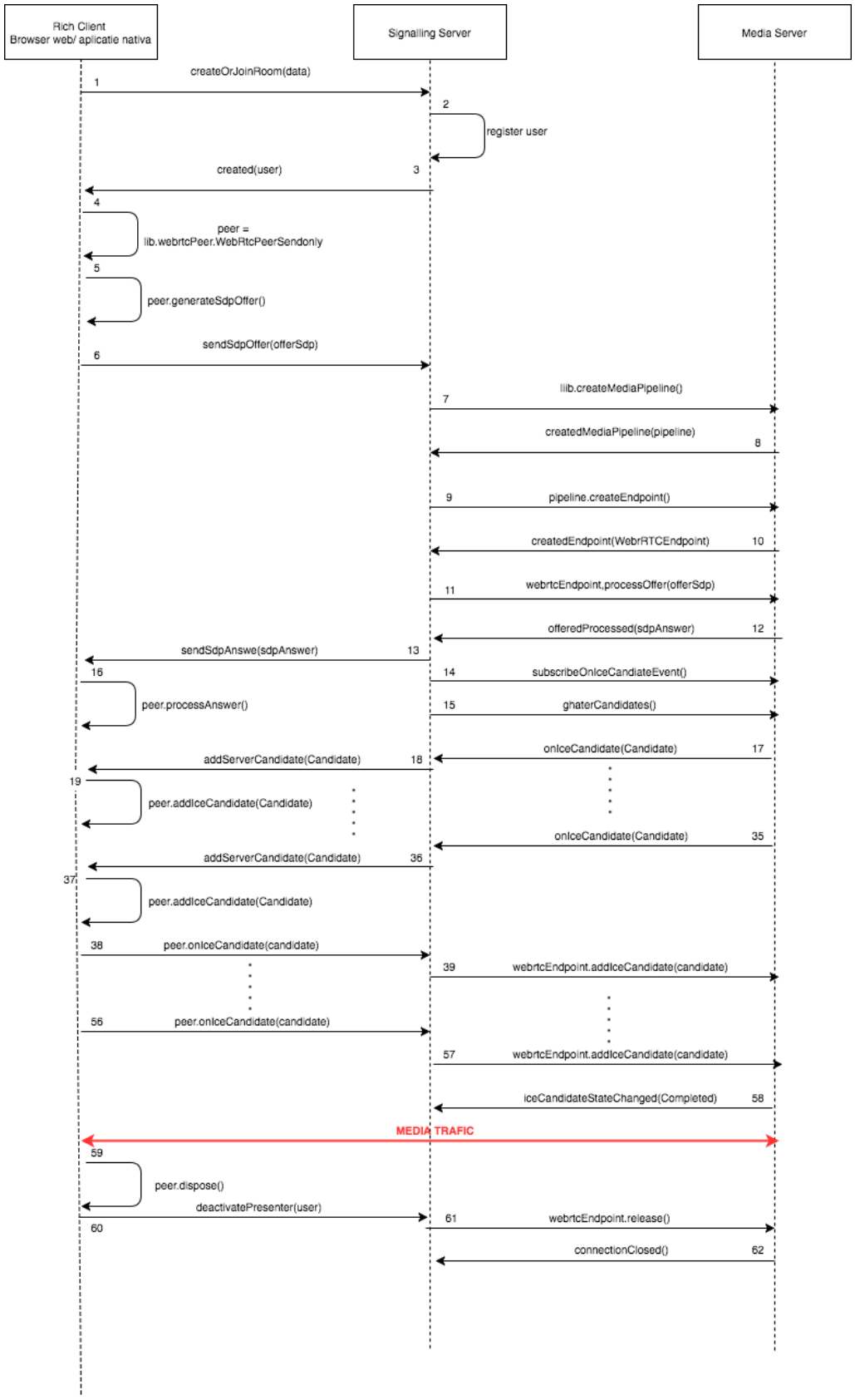

Fig 5.14 Diagrama secventa conexiune server media și transmiterea datelor multimedia

Capitolul 6. Testare şi Validare

Testarea este o parte esențială în dezvoltarea unui sistem software. Din acest motiv se consideră că timpul alocat testării trebuie să fie cel puțin la asemănător cu cel alocat dezvoltării sistemului. Testarea reprezintă analiza minuțioasă a componentelor și a funcționalilor sistemului.

A fost realizată atât o testare manuală a tuturor componentelor aplicațiilor, atât pentru cazurile de utilizare dedicate profesorului cât și pentru cele dedicate studenților. Pentru fiecare dintre aceste cazuri au fost folosite diferite seturi de date și configurări pentru a încerca acoperirea și evitarea a cât mai multe probleme ce pot apărea în utilizarea unor utilizatori nefamiliari cu funcționalitatile instrumentelor de video colaborare.

Procesul de testarea manuală al acestui sistem de colaborare sincronă s-a concentrat pe o testare funcitonala a componentelor, abordare ce permite testarea unui sistem de acest tip fără a dobândi în prealabil detalii legate de implementarea și structura componetelor testate.

Prin urmare au fost testate pas cu pas fiecare funcționalitate a sistemului. Prin urmare pentru componentele modulului de video comunicarea a fost verificat dacă stream-ul este afișat tuturor participanților, și anumite informații importante ale acestor elemente media însă testarea acestor componente multimedia o să le detaliez mai târziu. Pentru componentele bazate pe interacțiunea text s-a urmărit dacă mesajul trimis/întrebarea postată de un anumit utilizator este afișată și pentru restul participanților.

În timpul testării manuale a aplicației s-au urmărit și cât de solicitate sunt toate componentele fizice implicate în conferința la momentul respectiv. Din acest motiv am decis ca pentru exemplifiarea unui astfel de test, programamare unei sesiuni de comunicare sincronă formată din un profesor și patru studenți, iar încărcarea componentelor fizice pe care le vom urmării sunt încărcarea la nivel CPU, consum memorie RAM și consumul de bandă. Aceste componente se vor testa atât în cazul calculatoarelor ce rulează aplicația de client (se vor exemplifica doar la două), dar și al serverului de semnalizare NodeJS respectiv al serverului media Kurento Media Server.

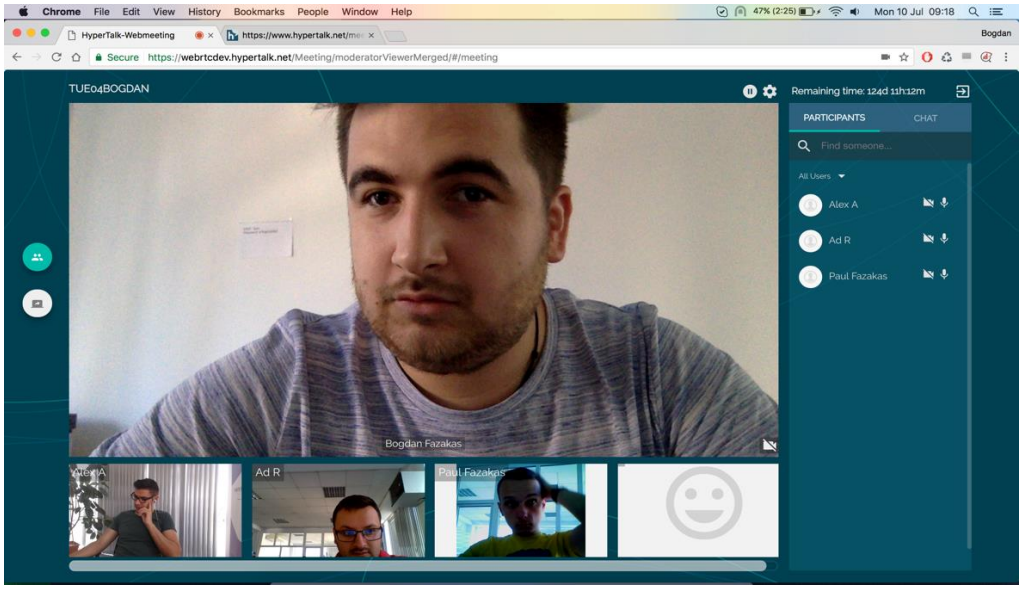

Fig 6.1 Aplicatie rulata pe un sistem de operare MacOS din browser-ul Chrome

Fig 6.2 Aplicație rulată pe un sistem de operare Windows din browser-ul Chrome

Figurile 6.1 respectiv 6.2 prezintă aplicația de client rulând pe o un sistem de operare MacOS respectiv Windows, în schimb ce figurile 6.3 și 6.4 prezintă comparative încărcarea procesorului pe ambele sisteme de operare pentru cazul de testare prezentat anterior.

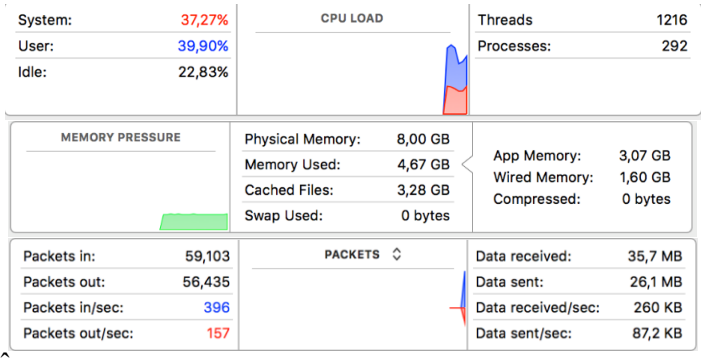

Fig 6.3 Încărcarea pe componentele hardware ale unui client Windows

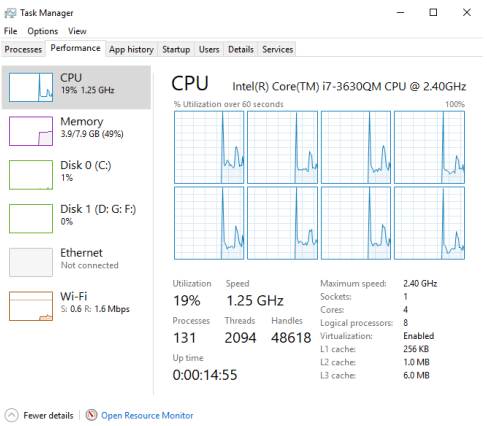

Fig 6.4 Încărcarea pe componentele hardware ale unui client Windows

După cum se poate observa din figurile 6.3 respectiv 6.4 încărcarea este aproximativ similară pe ambele sisteme de operare, iar pe randarea unor elemente ce conțin elemente multimedia de dimensiuni hd (1280 x 720) la 20 de cadre pe secundă consumă un număr de resurse limitate, fie că vorbim de încărcarea procesorului sau utilizarea memorie ram sau consumul de bandă/canal destul de mic 0,6 Mbps/s pe canalul de out iar la pe in pentru cele 4 canale audio/video este de 1,6 Mbps/s.

În figura 6.5 sunt prezentate informații ce oferă detalii de spre încărcarea componentelor mașinii virtuale pe care este instalat serverul de semnalizare. Acesta are un consum foarte mic de resurse având la dispoziție un un core al unui procesor și doar 2gb ram. Procesorul este foarte puțin utilizat în schimb ce consumul de memorie ram este destul de ridicat însă în contetul utilizării a doar 2 gb de memorie RAM.

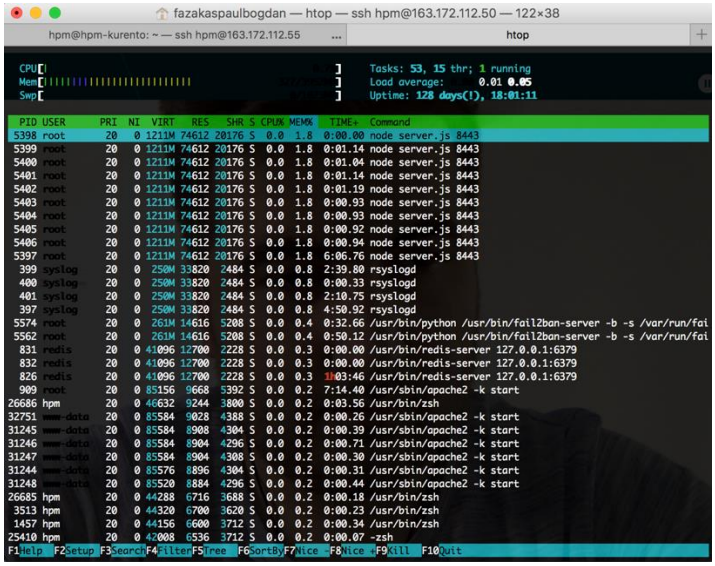

Fig 6.5 Încărcarea și procesele ce rulează pe serverul de semnalizare, NodeJS

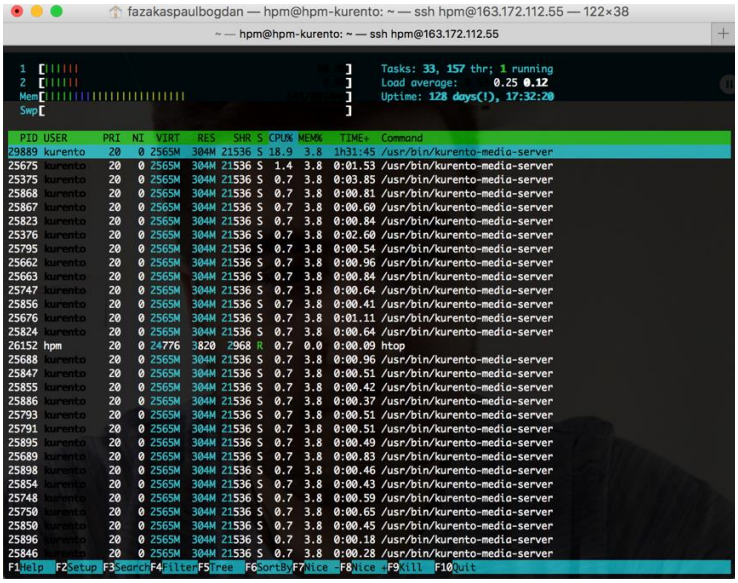

Fig 6.6 Încărcarea și procesele ce rulează pe serverul de media, Kurento Media Server

Încărcarea la nivel de procesor dar și consumul de momorie ram în cazul serverului media sunt prezentate în figura 6.6, în acest caz însă consumul procesorului este mai mare doarece el decodează și encodeaza toate streamurile media primate pentru a le livra mai departe tuturor participanților ai unei sesiuni sincrone. În cazul de față însă avem resursele pe care accesta le are sunt mult mai mari adică un procesor cu 4 core-uri respective 8 gb de Ram.

Pe lângă testarea funcționala a componentelor de UI pentru componentele video ale sistemului a fost folosit instrumentul WebRTC-Internals. Acest instrument este de mare ajutor în testarea aplicațiilor ce au la bază tehnologia WebRTC dar și în depănarea unor probleme existente deja.

Pentru a accesa acest instrument este necesară doar accesarea url-ului "chrome://webrtc-internals/" dintr-un browser web Goolgle Chrome în timp ce o sesiune de colaborare ce folosește WebRTC este deschisă. După această sunt afișate o serie de informații cu privire la intrarea sesiune în pagina menționată. Acestea sunt grupate cu ajutorul obiectelor de tip WebRTCPeer, iar informațiile sunt oferit pentru fiecare obiect în parte.

Pentru fiecare obiect de tip WebRTCPeer sunt disponibile următoarele informații:

- Sunt oferite opțiunile pe bază cărora obiectul a fost creat și ce servere de STUN și TURN sunt folosite
- un istoric al al apelul de metode ale acestui obiect (ex: generateOffer(),addStream()) respectiv răspunsul oferit de aceste apeluri dar și al evenimentelor de callback precum onIceCandidates
- statisticile colectate de API-ul getStats()
- și grafice generate de API-ul menționat mai sus.

În figurile 6.7 și 6.8 sunt prezentate exemple de grafuri surselor emise audio respectiv video, pentru cazul de utilizare în care profesorul emite aceste elemente video.

Graficul ce analizează elementele multimedia audio/video este prezentat în figura 6.7. Graficul este disponibil pentru fiecare canal audio deschis între un obiect WebRTCPeer și media server. Acest grafic permite analiza următoarelor informații:

- bitsSentPerSecond: număr de biți trimiși în fiecare secunda
- googRtt: (round trip) timpul de răspuns
- audioInputLevel: specifica volumul care este captat de sursa din care se emit datele audio
- googleJitterReceived: variatia intarzierii pachetelor receptionate
- s.a.m.d

În cazul surselor video graficul din figura 6.8 permite afișarea unor informații precum:

- bitsSentPerSecond: număr de biți trimiși în fiecare secunda
- packetsLost: numărul de pachete pierdute
- googRtt: (round trip) timpul de răspuns
- googFrameHeightSent: înălțimea stream-ului media transmis
- googFrameWidthSent: lățimea stream-ului media transmis

- s.a.m.d

Fig 6.7 Analiza canalului de comunicare audio folosind WebRTC-internals

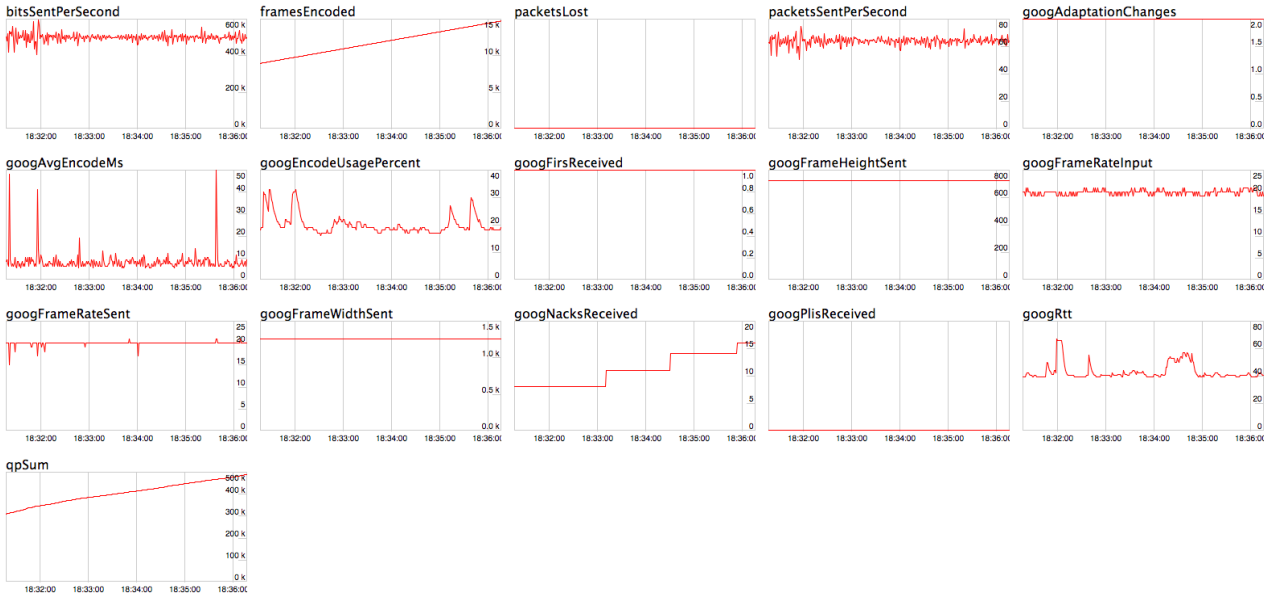

Fig 6.7 Analiza canalului de comunicare video folosind WebRTC-internals

Validarea sistemului a avut loc în cadrul evenimentului numit Cluj Innovation Days. Acest eveniment a fost transmis live din trei locații simultan, utilizatorii având posibilitatea să aleagă destinația pe care doresc să o vizualizeze. Numărul mediu de utilizatori conectați la datele media transmise în timp real a fost de 30 de utilizatori. Componentele validate cu ajutorul acestui eveniment au fost Video Communication, Content Sharing și Roster Presence.

Capitolul 7. Manual de Instalare si Utilizare

Rularea sistemului necesita pe lângă descărcarea codului sursă și instalarea unor dependințe și instrumente externe. Unele componente se pot utiliza din orice sistem de operare cum ar fii blocul aplicației de client,cel al serverului de semnalizare NodeJS și blocul ce se ocupă cu manipularea datelor(bază de date MySQL și sistemul de fișiere). Aceste două componente pot fii instalate atât pe sistemul de operare,windows, linux sau macOS. Instalarea serverului media însă este limitată la instalarea pe un sistem de operare linux de tip Ubuntu 14.04 LTS cu o arhitectură pe 64 de biți.

Deoarece am utilizat un set de mașini virtuale ce foloseau un sistem de operare linux în continuare voi prezenta o instalare a acestor dependințe în contextul unui sistem de acest tip.

Prima dependința care trebuie instalată este NodeJS. Acesta a fost instalat folosind doar instrucțiuni din linia de comandă a unui terminal linux. Comenzile necesare instalării pot fi găsite pe site-ul oficial al NodeJS la link-ul de la referință [51]. Cele două comenzi necesare instalării sunt următoarele:

- curl -sL https://deb.nodesource.com/setup_6.x | sudo -E bash -
- sudo apt-get install -y nodejs

Pentru a putea instala alte librării nodeJS se poate folosi managerul de pachete NPM care se instalează automat la instalarea nodeJS.

Următoarea dependință este instalarea serverului web nginx. Și această a fost instalată în același context cu cel al serverul de NodeJS, pe o mașină virtuală cu un sistem de operare ubuntu și instalarea dependinței utilizând doar terminalul acesteia. Pentru a instala nginx se recomandă un update la sistemul de operare înainte de a instală aplicația din biblioteca oficială a ubuntu. Comenzile necesare sunt:

- sudo apt-get update
- sudo apt-get install Nginx

A treia dependință necesară este instalarea bazei de date MySQL. Instalarea s-a făcut asemănător cu cea anterioară și aici se recomandă un update al sistemului înainte de a descărcă și instala dependința bazei de date folosind comanda apt-get install. Trebuie rulate următoarele comenzi pentru a instala dependință MySQL pe un sistem de operare Ubuntu:

- sudo apt-get update
- sudo apt-get install mysql-server

Nu în ultimul rând trebuie instalată dependință serverului media Kurento Media Server pe sistemul de operare Ubuntu 14.04 Lts. Comenzile necesarii instalării sunt diponibile pe site-ul oficial Kurento la secțiunea Kurento Media Server Installation , link ce poate găsit că că referința în [52]. Pentru a instala o instanță de server media kurento tebuie executate următoarele comenzi din linia de comandă:

- echo "deb http://ubuntu.kurento.org trusty kms6" | sudo tee
- /etc/apt/sources.list.d/kurento.list
- wget -O - http://ubuntu.kurento.org/kurento.gpg.key | sudo apt-key add -
- sudo apt-get update
- sudo apt-get install kurento-media-server-6.0

După cum spuneam toate comenzile mai puțin cele necesare instalării dependinței serverului media au echivalent și în sistemele de operare Windows și macOS.

Pentru a rula sau opri dependitentele instlate se ruleaaza următoarele comenzi în funcție de dependința dorită și acțiunea dorită:

- NodeJS signalling server
  - ridicare server web: node server.js port
- Nginx web server
  - pornire web server : sudo /etc/init.d/nginx start
  - oprire web server : sudo /etc/init.d/nginx stop
- MySQL database
  - pornire server baza de date: sudo service mysql start
  - oprire server baza de date: sudo service mysql stop
- Kurento Media Server
  - sudo service kurento-media-server-6.0 start
  - sudo service kurento-media-server-6.0 stop

Această aplicație poate fii folosită de către utilizatorii autentificați pe platformă educațională de care este integrată. Odată ce o sesiune este programată aceștia vor primi pe adresa de email specificată la înregistrare un token ce permite accesul la o sesiune programată de profesor.

În figura 7.1 este prezentată pagina de login a sistemului și câmpurile pe carea acesta trebuie să le completeze pentru a se putea alătura unei sesiuni de video colaborare în desfășurare.Aceste câmpuri sunt adresa de email și token-ul primit.

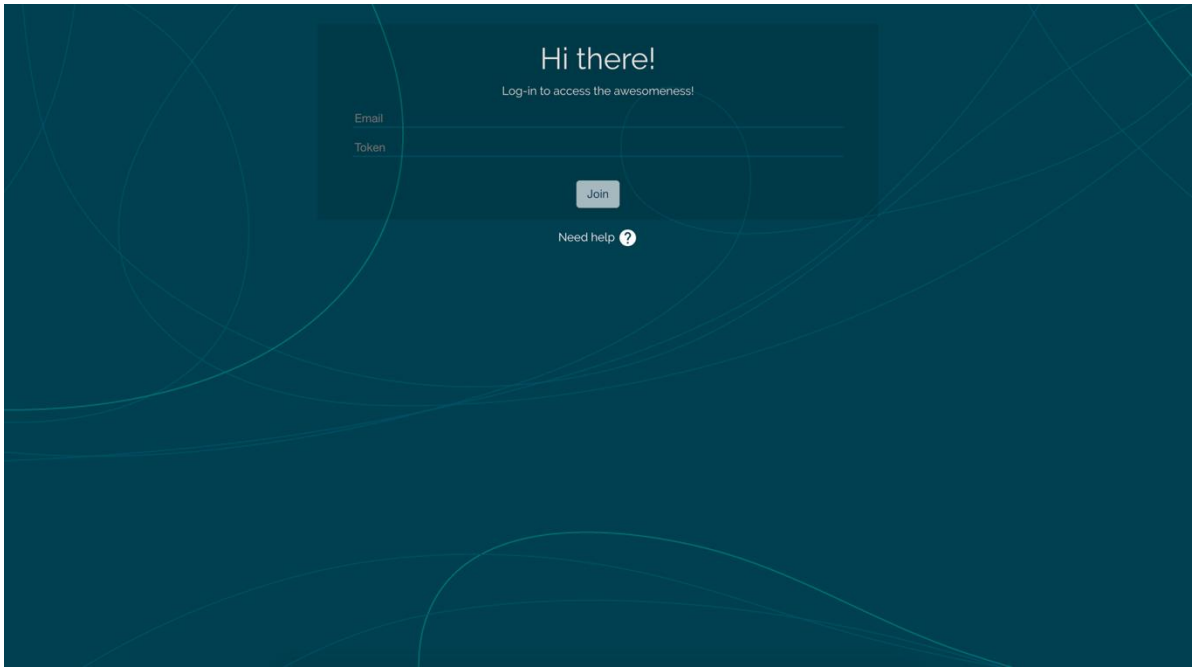

Fig 7.1 Pagina de login a sistemului

Dacă în momentul în care utilizatorul încearcă să se conecteze în sistem, sesiunea nu a început sau expirat acesta va fii notificat și îi este afișat orarul în care sesiunea a fost disponibilă. Un exemplu pentru acest caz poate fii observată în figura 7.2

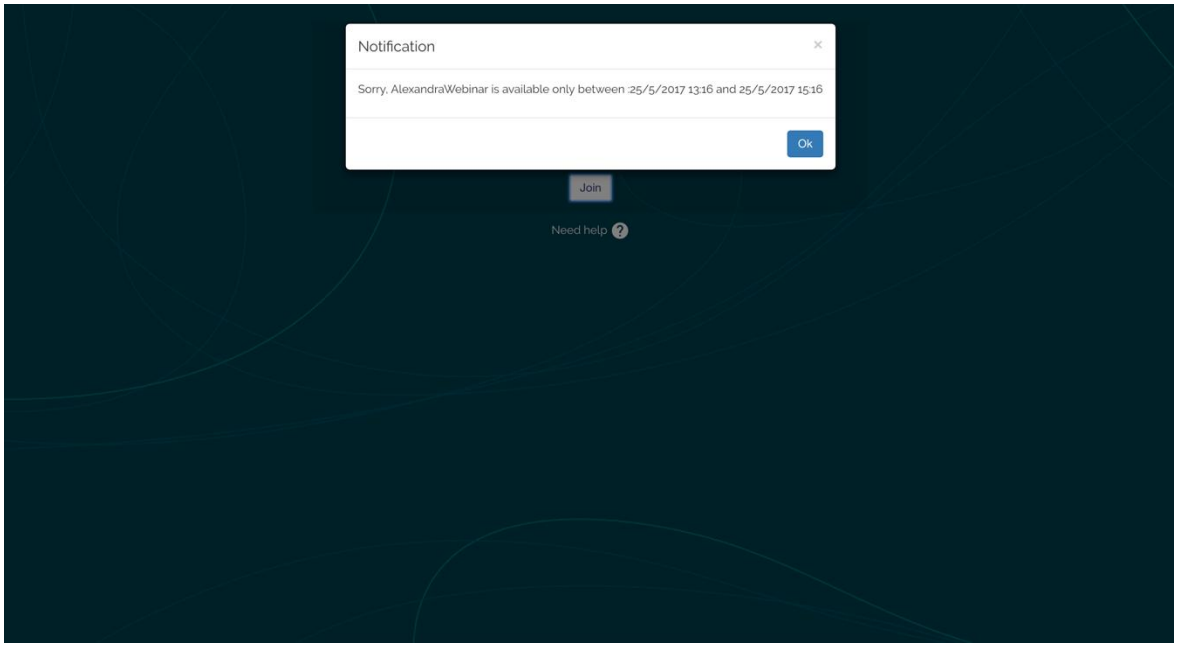

Fig 7.2 Notificare ca sesiunea nu e în desfășurare

Odată înregistrat în sistem se afișează pagina de main a sistemului. Aici avem disponibile informații generale ale sesiunii, butoane pentru schimbarea inter tab-urile aplicației, dar și instrumente de comandă asupra elementelor multimedia prezentate. În figură 7.3 sunt disponibile aceste informații, iar cu ajutorul figurii o să încer să explic rolul fiecărui element grafic prin cele ce urmează:

1. butoane selectare tab aplicație (Meeting,Content Sharing sau Questions)
2. buton schimbare tab intre lista participanților activi și chat
3. numele sesiunii
4. slot video profesor
5. sloturi video dedicate studenților
6. mesaj text afișat în componenta chat
7. buton deconectare
8. buton pauza sesiune
9. afișare meniu schimbare camera web sau microfon
10. input pentru adăugare mesaje text
11. buton oprire camera web și emitere doar stream audio
12. afișare timp rămas pana la închiderea sesiunii

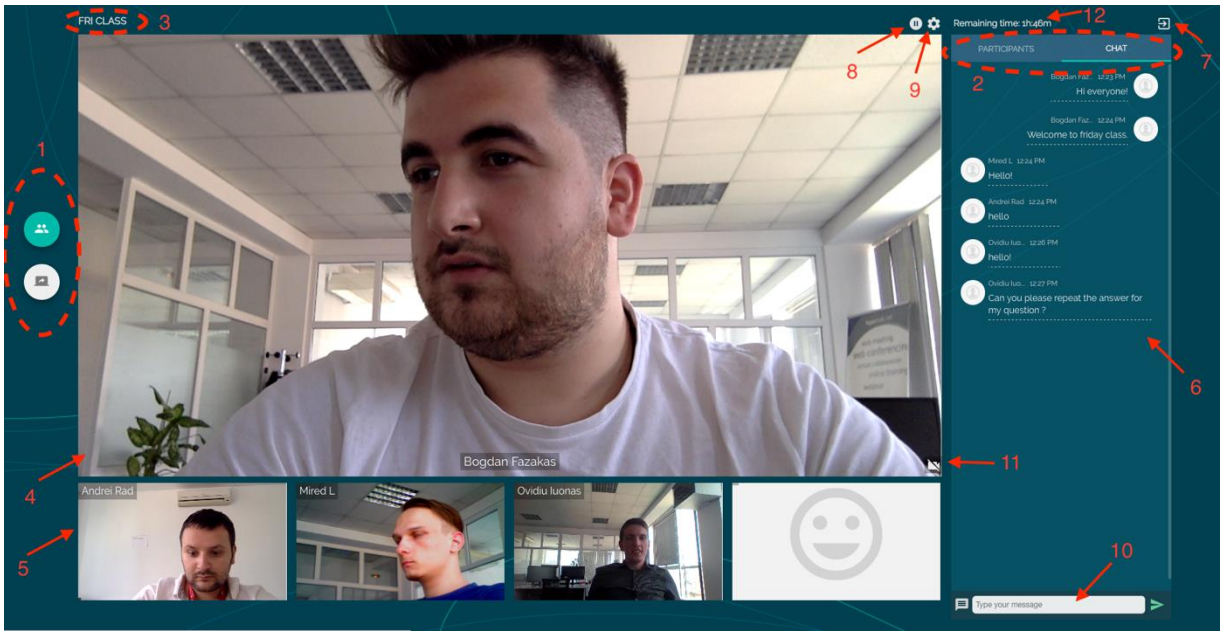

Fig 7.3 Pagina principala a sistemului și funcționalitatile oferite

În figura 7.4 este pagina dedică partajării conținutului, în cazul de față partajarea unui aplicații de editare text Sublime ce rulează pe ecranul profesorului. Din figură pot fii identificate următoarele componente:

1. slot video dedicat unde este redat conținutul multimedia partajat
2. slot video ce oferă imagini in timp real cu profesorul
3. lista de participanți
4. buton de control asupra listei cu participanți activare/dezactivare ca participanți activ
5. buton oprire stream audio student
6. buton partajare conținut multimedia preluat de la diferite url-uri
7. buton oprire partajare conținut

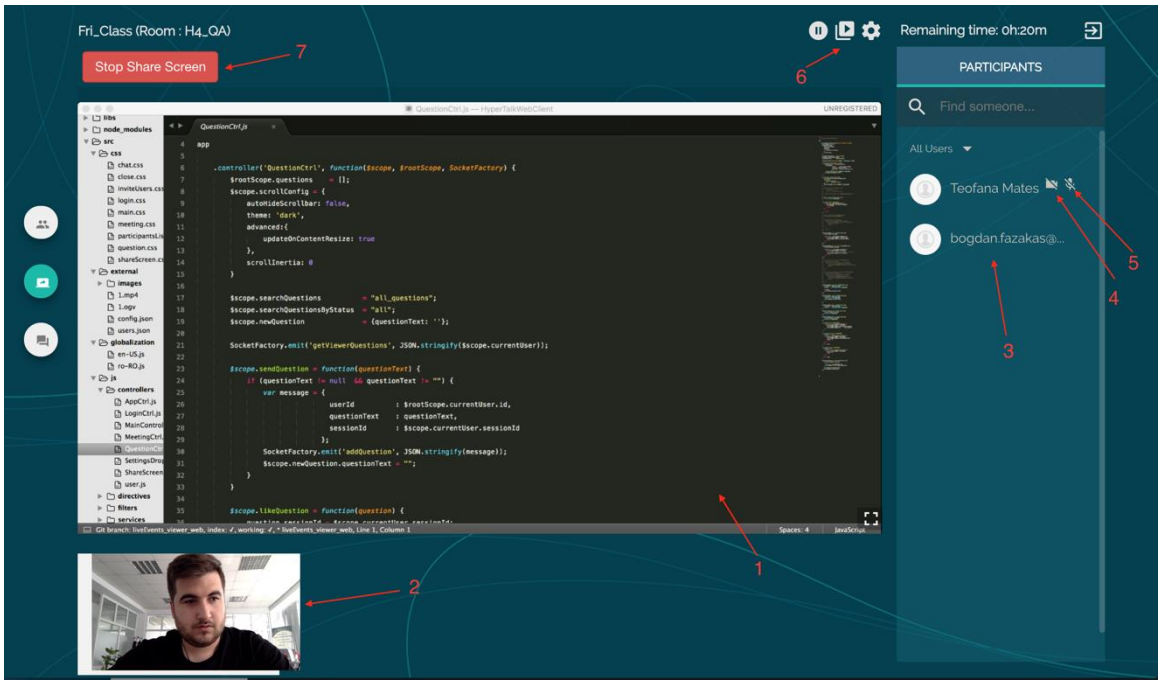

Fig 7.4 Pagina de partaje conținut

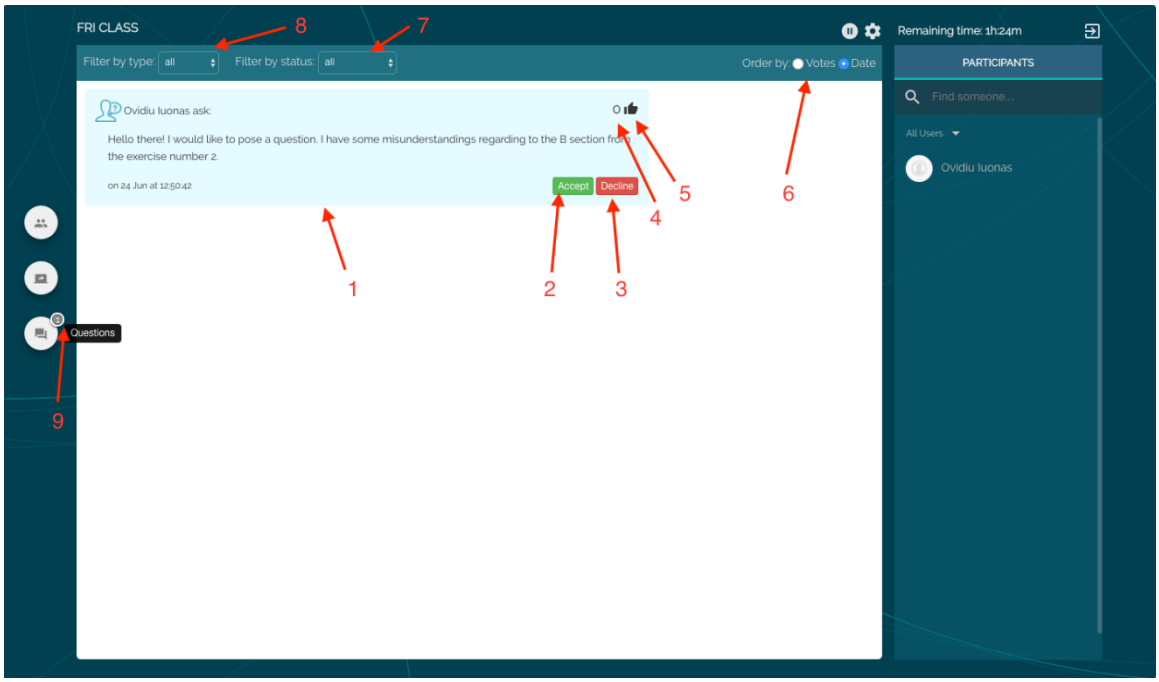

Fig 7.5 Componenta Questions a sistemului

Figura 7.5 prezintă componenta Question și funcționalitatile pe care un profesor le are pentru a gestiona această secțiune. Printre principale funcționalitati oferite de acesta pagină se număra:

1. zona afișare întrebare propriu-zisa și date despre elevul ce a adresat întrebarea respectiva și ora la care aceasta a fost adresata

2. buton de acceptare întrebare
3. buton de refuzare întrebare
4. numărul de like-uri ale întrebări
5. buton de like întrebare
6. buton selectare ordonare întrebări
7. filtru ce permite afișare întrebării cu un anumit status (ongoing,answered,waiting for answer)
8. filtru ce permite afișare întrebări cu un anumit tip (accepted/declined)
9. notificare întrebare noua

Capitolul 8. Concluzii

Lucrarea de față a avut că principal obiectiv crearea unui framework care se îmbunătățească comunicarea între profesori și studenți. Pentru ca acest obiectiv să fie îndeplinit au fost implementate un set instrumente de video colaborarea folosind doar tehnologii open-source. Realizarea acestor obiective fac ca o comunicare în timp real de la distanță între profesor și elevi să fie posibilă. Comunicarea poate fii realizată atât cu ajutorul instrumentelor de colaborare video, sau prin comunicarea în timp real bazată pe inter-schimbul de mesaje text sau a conținutului partajat.

Deoarece aplicația de client este una de tip rich client application acest fapt ne să rulăm aplicația atât din browere web cât și din contextul unei aplicații native independent de sistemul de operare deținut, și nu în ultimul rând aplicația este ușor de personalizat.

Deoarece soluția propusă este una de tip cloud-based acest lucru oferă avantaje precum scalabilitatea, acces facil indiferent de poziționarea geografică, și permite o integrarea facilă cu platforme educaționale existențe deja pe piață precum Moodle, Blackboard sau HyperEdu.

Implementarea sistemului de video colaborare utilizând conceptul micro-serviciilor oferă un control mai eficient asupra componentelor sistemului. Din acest motiv adăugarea unor funcționalități noi sau eliminarea unora existențe deja este rapid și ușor de efectuat. Nu în ultimul rând componentele sistemului fiind independente ne permit un management mai bun și mai ușor de efectuat, iar în cazul unor probleme componente se livrează independent pe server, o componente cu probleme nu afectează funcționarea celorlalte.

Componenta de video comunicare are la bază tehnologia WebRTC ce permite transmisia de conținut mulțimea de tip audio video în timp real între browsere, concept denumit peer to peer. Contribuția personală la soluția tehnică constă din introducerea unui modul de tip media server ce permite redirectarea fluxului media prin acest server și implicit eficientizarea consumului de resurse pe parte de client în cazul sesiunilor de tip *multi point video conferencing*.

Pentru realizarea celorlalte compoente ale sistemului contribuția proprie a fost personalizarea API-ului Socket.IO pentru a asigura interacțiunea text based și controlul sesiunii în cazul.

Putem consideră că obiectivele propuse la începutul lucrării au fost îndeplinite, chiar dacă pot fii aduse îmbunatățiri soluției tehnice propuse. În implementarea sistemului au fost folosite tehnologii și concepte noi precum WebRTC, Kurento, NodeJS, Angular. Tehnologiile menționate beneficiază de o comunitate extrem de vastă și ce se află într-o continuă expansiune, toate acestea din pricină faptului că aceste tehnologii sunt open-source. Acest lucru a contat destul de mult în alegerea tehnologiilor deoarece acestea vor avea suport pentru o perioadă mare de timp și se vor dezvoltă și îmbunătăți continuu.

8.1. Dezvoltări ulterioare

Printre îmbunătățirile ce pot fii aduse soluției tehnice propuse în lucrarea de față se numără:

- dezvoltarea unei componente care să ne permită control la distanță asemănătoare cu componenta de partajare a ecranului, ce însa o sa ofere profesorului control la ecranul partajat de student.

- dezvoltarea aplicațiilor native pe sisteme precum iOS și Android care sa ofere utilizatorilor aceleași funcționalități.
- implementarea sau integrarea unor componente ce permit adnotarea unor fișiere in timp real pe durata sesiunii
- crearea unui cluster de servere media pentru ca aplicatia sa suporte un numar mai mare de conexiuni punct la punct. O singura instanta a kurento media server suporta circa 800 de conexiuni simultane.

Opțiunile prezentate sunt doar câteva funcționalități, care pot fi implementate cu ajutorul noilor tehnologii.

Instrumentele ce compun acest system aduc benefici atât profesorului cât și elevului și usureza folosirea oferă utilizatorilor un sistem de tip blended learning ușor de utilizat, atât de către profesori cât și de catre elevi/student.

Pe langă cele enunțate mai sus sistemul propus este unul flexibil și poate fii utilizând orice sistem de operare disponibil în momentul de fată (Windows, Linux, macOS) prin aplicațiile native oferite pentru fiecare platformă mentionată mai sus, dar flexibilitatea se referă și la posibilitatea utilizării aplicației din toate browser-ele compatibile cu tehnologiile utilizate.

În concluzie, aplicațiile de video colaborare în timp real vor fi din ce în ce mai utilizate pe diferite dispozitive, oferind oportunitatea studenților și a profesorilor să comunice și după terminarea unui laborator sau a unei ore de curs.

Bibliografie

[1] Bogdan Fazakas, Ovidiu Iuonas, "Collaborative learning tools for formal and informal engineering education", Computer Science Students Conference, June 26 2017

[2] Ding Q., Cao S., RECT: A Cloud-Based Learning Tool for Graduate Software Engineering Practice Courses With Remote Tutor Support, IEEE Access, March 2017

[3] S Ouya, G Mendy, C Seyed, AB Mbacke,I Niang WebRTC platform proposition as a support to the educational system of universities in a limited Internet connection context, 14-16 Dec. 2015

[4] Y Liao, Z Wang, Y Luo, The Design and Implementation of a WebRTC based Online Video Teaching System, Oct 3-5,2016

[5] LL Fernández,, MP Diaz, RB Mejías, Kurento: a media server technology for convergent WWW/mobile real-time multimedia communications supporting WebRTC.June 4-7,2013

[6] B Garcia, LL Fernandez, M Gallego, F Gortázar, Analysis of video quality and end-to-end latency in WebRTC,Dec 4-8,2016

[7] CC Spoiala, A Calinciuc, CO Turcu, C Filote, Performance comparison of a WebRTC server on Docker versus Virtual Machine, 13th International Conference on DEVELOPMENT AND APPLICATION SYSTEMS, Suceava, Romania, May 19-21, 2016

[8] D. Chandran and S. Kempegowda, Hybrid e-learning platform based on cloud architecture model: a proposal, Proceedings of the International Conference on Signal and Image Processing,2010 534-537.

[9] M.A. Bochino, A. Longo, M Zappatore, D Tarantino, The role of online labs in the European e-Science Infrastrucure, 24-26 Feb 2016

[10] Moldovan, R., Orza, B., Vlaicu, A. and Porumb, C., Advanced human-computer interaction in external resource annotation, In Automation, Quality and Testing, Robotics, 2014 IEEE International Conference on (pp. 1-6) , IEEE

[11] M. Qiu, and L. Chen, A Problem-based Learning Approach to Teaching an Advanced Software Engineering Course, Proceedings of the 2nd International Workshop on Education Technology and Computer Science, pp 252-255,2011

[12] Microservices Arhitecture ,https://en.wikipedia.org/wiki/Microservices

[13] HTML5, https://www.w3schools.com/html/html5_intro.asp

[14] CSS3, https://developer.mozilla.org/en/docs/Web/CSS/CSS3

[15] Sass, http://sass-lang.com/

[16] JavaScript, https://www.w3schools.com/js/

[17] AngularJs tutorial, http://angularjstutorials.net/

[18] AngularJS 1.0 Model-View-Controller, http://angularjstutorials.net.html

[19] AngularJS View-Controller $scope, https://docs.angularjs.org/guide/concepts

[20] Anugular $scope vs $rootScope https://lazydevguy.wordpress.com/2015/02/04/broadcast-vs-emit-vs-on/
.

[21] Angular Material, https://material.angularjs.org/latest/

[22] Bootstrap, https://www.w3schools.com/bootstrap/

[23] WebRTC, https://webrtc.org/

[24] WebRTC Arhitecture, ,http://blog.csdn.net/fanbird2008/article/details/18623141
[25] Stun and Turn signalling, https://www.html5rocks.com/en/tutorials/webrtc/infrastructure/
[26] STUN server, https://www.3cx.com/pbx/what-is-a-stun-server/
[27] COTURN server, https://github.com/coturn/coturn
[28] Webrtc browser support, http://www.telepresenceoptions.com/webrtc/
[29] NPM, https://en.wikipedia.org/wiki/Npm_(software)
[30] Grunt, https://gruntjs.com
[31] Nginx, https://www.nginx.com/resources/wiki/
[32] NW.JS, https://nwjs.io/
[33] Nw-builder, https://github.com/nwjs-community/nw-builder
[34] Grunt nw-buidler plugin, https://github.com/nwjs/grunt-nw-builder
[35] Node-appdmg, https://github.com/LinusU/node-appdmg
[36] MSI-packager , https://www.npmjs.com/package/msi-packager
[37] NodeJS, https://nodejs.org/en/about/ Arhitectura NodeJS
[38] Arhitectura NodeJS, https://blog.zenika.com/2011/04/10/nodejs/
[39] WebSocket, https://en.wikipedia.org/wiki/WebSocket
[40] Socket.IO, https://socket.io/docs/
[41] JSON, http://www.json.org/
[42] MySQL, https://en.wikipedia.org/wiki/MySQL
[43] Kurento Media Server, http://doc-kurento.readthedocs.io/en/stable/
[44] Comunicarea Client - Kurento Media Server, http://doc-kurento.readthedocs.io/en/stable/introducing_kurento.html#kurento-api-clients-and-protocol
[45] Kurento Media Pipeline, http://doc-kurento.readthedocs.io/en/stable/introducing_kurento.html#creating-applications-with-kurento
[46] VP8 codec, https://en.wikipedia.org/wiki/VP8
[47] OPUS codec, https://en.wikipedia.org/wiki/Opus_(audio_format)
[48] SIP, https://en.wikipedia.org/wiki/Session_Initiation_Protocol
[49] SDP, https://en.wikipedia.org/wiki/Session_Description_Protocol
[50] RTP, https://en.wikipedia.org/wiki/Real-time_Transport_Protocol
[51] Link instalare NodeJS, https://en.wikipedia.org/wiki/Real-time_Transport_Protocol
[52] Link instalare Kurento Media Server , http://doc-kurento.readthedocs.io/en/stable/installation_guide.html