

UNIVERSITATEA „ALEXANDRU IOAN CUZA” IAȘI

Facultatea de Informatică



Lucrare de licență

Instant – Web Messenger

Propusă de

ORZU IONUȚ

Sesiunea: Iulie, 2017

Coordonator Științific:

Lector, dr. Cosmin Vârlan

UNIVERSITATEA „ALEXANDRU IOAN CUZA” IAȘI

Facultatea de Informatică

Instant – Web Messenger

Orzu Ionuț

Sesiunea: Iulie, 2017

Coordonator Științific:

Lector, dr. Cosmin Vârlan

DECLARAȚIE PRIVIND ORIGINALITATE ȘI RESPECTAREA DREPTURILOR DE AUTOR

Prin prezenta declar că Lucrarea de licență cu titlul “Instant – Web Messenger” este scrisă de mine și nu a mai fost prezentată niciodată la o altă facultate sau instituție de învățământ superior din țară sau străinătate. De asemenea, declar că toate sursele utilizate, inclusiv cele preluate de pe Internet, sunt indicate în lucrare, cu respectarea regulilor de evitare a plagiatului:

- toate fragmentele de text reproduse exact, chiar și în traducere proprie din altă limbă, sunt scrise între ghilimele și dețin referința precisă a sursei;
- reformularea în cuvinte proprii a textelor scrise de către alți autori deține referința precisă;
- codul sursă, imagini etc. preluate din proiecte open source sau alte surse sunt utilizate cu respectarea drepturilor de autor și dețin referințe precise;
- rezumarea ideilor altor autori precizează referința precisă la textul original.

Iași,

Absolvent Orzu Ionuț

(semnătura în original)

DECLARAȚIE DE CONSIMȚĂMÂNT

Prin prezenta declar că sunt de acord ca Lucrarea de licență cu titlul „Instant – Web Messenger”, codul sursă al programelor și celelalte conținuturi (grafice, multimedia, date de test etc.) care însoțesc această lucrare să fie utilizate în cadrul Facultății de Informatică. De asemenea, sunt de acord ca Facultatea de Informatică de la Universitatea „Alexandru Ioan Cuza” Iași să utilizeze, modifice, reproducă și să distribuie în scopuri necomerciale programele-calculator, format executabil și sursă, realizate de mine în cadrul prezentei lucrări de licență.

Iași,

Absolvent Orzu Ionuț

(semnătura în original)

Cuprins

Introducere	6
1. Motivație	6
2. Context	6
3. Cerințe funcționale	7
4. Abordare tehnică	8
1. Contribuții.....	10
2. Dezvoltarea aplicației web	11
2.1 AdonisJS	11
2.2 MySQL.....	13
2.3 Socket.io	15
2.4 WebRTC	18
2.5 Kurento Media Server	25
2.6 WebTorrent	28
2.7 EcmaScript2015	31
2.8 Sass	33
3. Descrierea aplicației	35
3.1 Pagina de întâmpinare (Landing page).....	35
3.2 Pagina de autentificare.....	35
3.3 Pagina de înregistrare.....	36
3.4 Pagina de acasă	36
3.5 Pagina de apel video.....	41
3.6 Pagina de apel voce	42
3.7 Pagina de cinema online.....	43
3.8 Pagina cu lista de prieteni	45
3.9 Pagina cu lista de utilizatori ai aplicației	46
Bibliografie.....	47

Introducere

1. Motivație

Comunicarea reprezintă o nevoie esențială și foarte importantă pentru oameni. Fără posibilitatea de a comunica și vorbi cu alți oameni nu ar putea exista și prospera niciun individ, grup sau comunitate. Aceasta permite oamenilor să se exprime și să își arate punctele de vedere.

Odată cu evoluția tehnologiei oamenii au găsit tot mai multe și diverse modalități de comunicare, fie că e vorba de mesaje text, poșta electronică, apel voce prin telefon fix, telefon mobil, apel video pe internet sau altele. Nevoia de comunicare, rapidă, în orice moment al zilei, este reflectată în această multitudine de modalități.

Aceste noi tipuri de comunicare au deschis orizontul la posibilități pe care în trecut nici nu ni le imaginam, acum putem să luăm legătura cu o persoană dragă aflată la mii de kilometri distanță în doar câteva secunde, tot ce avem nevoie este de o conexiune de internet (lucru la care din ce în ce mai mulți oameni au acces) și un dispozitiv care se poate conecta la internet. Putem să ne sunăm prietenii, să ne apelăm video părinții și să îi vedem în față noastră pe ecran ca și cum ar fi în același loc cu noi, să împărtășim amintiri cu persoane dragi cu ajutorul fotografiilor și clipurilor video. Posibilitățile de comunicare sunt nelimitate.

Am ales această aplicație deoarece am dorit să ofer o soluție pentru modalitățile de comunicare de mai sus, rapidă, gratis, direct în browser.

2. Context

Conceptul de mesagerie instantă a apărut în prim-plan în anii 1990, permițând prietenilor, cunoștințelor, colegilor din toată lumea să se conecteze în timp real.

De atunci, mesageria instantă a revoluționat felul cum comunicăm, iar astăzi peste 2 miliarde și 500 de milioane de oameni sunt înscrși în cel puțin una din aplicațiile de acest fel.

Până în anul 2009, serviciul de mesagerie prin intermediul furnizorilor de servicii telefonice avea monopol pe piață, însă în acel an a fost lansată aplicația WhatsApp. De atunci tot mai mulți oameni au început să folosească opțiunea de a trimite mesaje online, deoarece era mai ieftină, Wi-Fi-ul acoperind tot mai mult teren.

Mesageria instantă este un tip de conversație online care oferă transmisiuni în timp real cu ajutorul internetului. Mesajele sunt trimise între două sau mai multe părți, de obicei cunoscute, dintr-o lista de contacte (de asemenea cunoscută și ca listă de prieteni). Aplicațiile au devenit din ce în ce mai avansate cu trecerea timpului, acum oferindu-se și posibilitatea

transferului de fișiere, apelurilor voce sau apelurilor video, de tip unu-la-unu sau de tip conferință.

În ceea ce privește aplicațiile deja existente ce abordează un subiect similar, enumerăm:

Facebook Messenger¹ – este o aplicație de mesagerie instantă, dezvoltată inițial ca Facebook Chat în 2008. Aceasta permite utilizatorilor Facebook² să schimbe mesaje între ei, iar pentru a oferi o experiență cât mai completă, Messenger oferă utilizatorilor posibilitatea inițierii apelurilor video de tip unu-la-unu sau de grup. Este cea mai populară aplicație de acest tip având 1.2 miliarde de utilizatori în aprilie 2017.

WhatsApp³ – este un serviciu gratis, pe mai multe platforme, de tip mesagerie instantă. Acesta folosește internetul pentru a face apeluri voce, apeluri video unu-la-unu, pentru a trimite mesaje text, imagini, GIF-uri, etc. Se diferențiază de restul aplicațiilor deoarece pentru crearea contului necesită doar numărul de telefon al utilizatorului, spre deosebire de alte aplicații care necesită o adresă de email și parolă. Datele trimise sunt criptate de la capăt-la-capăt.

Majoritatea aplicațiilor de acest tip vin cu limitări:

- nu suportă toate platformele
- este necesar un program de instalat
- mărimea fișierelor de transferat este mică (25 MB Facebook, 300 MB Skype)
- este necesară o conexiune de internet rapidă pentru apelurile video deoarece datele trec prin server (deseori aflat la distanță foarte mare de utilizatori)

3. Cerințe funcționale

Crearea unui cont - direct din aplicație, necesar pentru a avea o listă de prieteni și un istoric al conversațiilor

Accesul la lista de prieteni - aceștia fiind adăugați (sau șterși) în urma căutării în lista de utilizatori ai aplicației, fie vizual, fie după nume

Accesul la conversațiile cu persoane sau grupuri de persoane – aplicația oferă stocarea persistentă a listei de conversații pentru fiecare utilizator în parte

Trimiterea de mesaje instant prietenilor – funcționalitatea de bază a aplicației, funcționează în timp real

¹ <https://www.messenger.com/>

² <https://facebook.com>

³ <https://www.whatsapp.com/>

Posibilitatea de a crea grupuri de prieteni – pentru conversațiile cu grupul de prieteni, colegii de la școală, colegii de la muncă, familia

Apelul voce unu-la-unu sau de tip conferință – aplicația permite apeluri voce gratis, cu ajutorul internetului

Apelul video unu-la-unu sau de tip conferință – această funcționalitate permite utilizatorilor să comunice cu o persoană sau un grup ca și cum ar fi față în față, chiar dacă aceștia sunt în altă țară sau la mare distanță

Transfer de fișiere de-la-egal-la-egal între persoane sau grupuri de persoane – aplicația permite utilizatorilor să trimită fișiere de orice tip (fotografii, videoclipuri, documente, etc.) și de orice dimensiune în mod confidențial (fișierele nu ajung la server în niciun moment).

Cinema online – în care se încarcă un fișier video pentru a fi vizualizat de două sau mai multe persoane în același timp, aflate într-un apel video, pentru a simula experiența cinematografică

4. Abordare tehnică

Pentru partea de server (back-end) am folosit **Node.js**, cu ajutorul framework-ului **AdonisJS**, a librăriilor **Socket.io**, **WebTorrent**, a tehnologiei **WebRTC** și a unui server media WebRTC **Kurento Media Server**. Baza de date este de tip MySQL.

Pentru partea de client (front-end) am folosit **HTML5**, **SASS** (compilat în CSS cu ajutorul librăriei **Gulp**), **EcmaScript2015** (compilat în JavaScript obișnuit pe care toate browserele îl suportă, cu ajutorul librăriilor **Babel** și **Webpack**).

Ca sistem de administrare a bazelor de date relaționale am folosit **MySQL**, fiind un sistem cu sursă deschisă. A fost integrat în aplicația „Instant – Web Messenger” în scopul stocării datelor referitoare la identitatea utilizatorului, conversațiile acestuia, lista de prieteni și altele.

Comunicarea între client și server se va realiza prin intermediul websocket-urilor (librăria **Socket.io**) în timpul conversațiilor, apelurilor voce și video, transferului de fișiere și cinematografului online, iar pentru restul funcționalităților se utilizează serviciile web REST.

Apelul voce și cel video se realizează cu ajutorul colecției de protocoale de comunicare **WebRTC** care oferă comunicare nativă între browsere de tip peer-to-peer (de la egal la egal), fără a fi necesar software adițional. Apelul voce sau video între mai mult de 2 utilizatori, pentru a fi eficient, necesită un punct central care să primească fluxul de date de la fiecare transmițător și să îl trimită la fiecare participant în parte (tehnică

cunoscută sub numele de rutare). Pentru a rezolva această problemă am folosit **Kurento Media Server**.

Transferul de fișiere și cinematograful online folosesc librăria **WebTorrent**. Această librărie conectează utilizatorii împreună pentru a forma o rețea distribuită, decentralizată, browser la browser pentru transfer de date eficient.

JavaScript este un limbaj de programare dinamic, multi-paradigmă, orientat pe obiecte, bazat pe conceputul prototipurilor, care a apărut în anul 1995, dezvoltatorul inițial fiind Brendan Eich. Creat inițial pentru a controla paginile web pe partea de client a browserelor, a fost dezvoltat mai târziu și pentru partea de server și chiar dezvoltarea aplicațiilor mobile.

Node.js este un mediu de programare cu sursă deschisă, multi-platformă care folosește JavaScript pentru a executa cod pe partea de server. Node.js aduce serverelor web un mediu de programare asincron, bazat pe evenimente, oferind posibilitatea realizării de aplicații web scalabile, care necesită operații multiple de intrare/ieșire, fiind ideal pentru aplicații de tip chat.

1. Contribuții

Prezenta lucrare este împărțită în două capitole, unde voi face o descriere a procesului de dezvoltare și voi menționa, unde este cazul, resursele necesare:

- Dezvoltarea aplicației web
- Descrierea aplicației

Primul capitol prezintă tehnologiile folosite în dezvoltarea aplicației web, atât pe partea de client (front-end), cât și pe partea de server (back-end).

Cel de-al doilea capitol prezintă succint fiecare cadru din aplicația client împreună cu o scurtă descriere a funcționalităților acestora. De asemenea se vor atașa capturi de ecran realizate pe un dispozitiv desktop sau mobil, după caz.

Contribuția mea în dezvoltarea acestui proiect este:

- implementarea aplicației pe partea de client, unde s-a construit o interfață intuitivă și plăcută din punct de vedere vizual prin respectarea standardelor *Material Design* și care este compatibilă cu orice dimensiune a ecranului
- implementarea aplicației pe partea de server, unde a fost nevoie de o aprofundare a tehnologiei WebRTC care oferă capacități pentru transferul fluxului de date media
- configurarea serverului media Kurento Media Server

2. Dezvoltarea aplicației web

2.1 AdonisJS

AdonisJS este un framework de tip MVC (Model-View-Controller) scris în EcmaScript2015, care capătă popularitate din ce în ce mai multă, din cauza asemănării evidente cu alte framework-uri din alte limbaje de programare (*Laravel* din PHP, *Ruby on Rails* din Ruby).

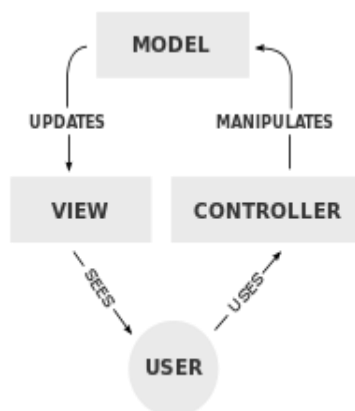


Figura 1: Structură de tip MVC⁴

Din structura de baza a unui nou proiect se merită a fi menționate următoarele dosare și fișiere importante:

- Dosarul */app* conține toată logica aplicației server și este încărcat automat (autoloaded) sub spațiul de nume *App*. Ca subdirectoare găsim următoarele:
 - Dosarul */Commands* dedicat pentru a stoca comenzi
 - Dosarul */Http* care conține entități legate de un server Http: **Controller**, **Middleware** și **Routes**
 - Dosarul */Listeners* utilizat pentru a organiza ascultătoare de evenimente (event listeners)
 - Dosarul */Model* care conține clasele ce ajută la reprezentarea datelor din baza de date
- Dosarul */bootstrap* conține fișiere care unesc piesele necesare aplicației. Aici adăugăm furnizori de servicii sau înregistrăm evenimente
- Dosarul */config* aici definim configurația aplicației (de exemplu datele de conectare la baza de date)

⁴ <https://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93controller>

- Dosarul */database* care conține toate fișierele legate de baza de date. Putem adăuga clase pentru migrații (migrations), seeds sau factories.
- Dosarul */public* în care, după cum spune și numele, ținem fișiere publice servite static peste HTTP
- Dosarul */resource* dedicat pentru paginile web (views), fișierele Sass care vor fi compilate în CSS, fișierele EcmaScript2015 care vor fi compilate în JavaScript standard
- Dosarul */node_modules* conține toate librăriile externe ce au fost instalate cu ajutorul managerului de pachete **npm** prin rularea comenzii *npm install nume_pachet*
- Fișierul *package.json* ce conține toate datele instalării dependențelor aplicației pe o mașină ce rulează Node.js

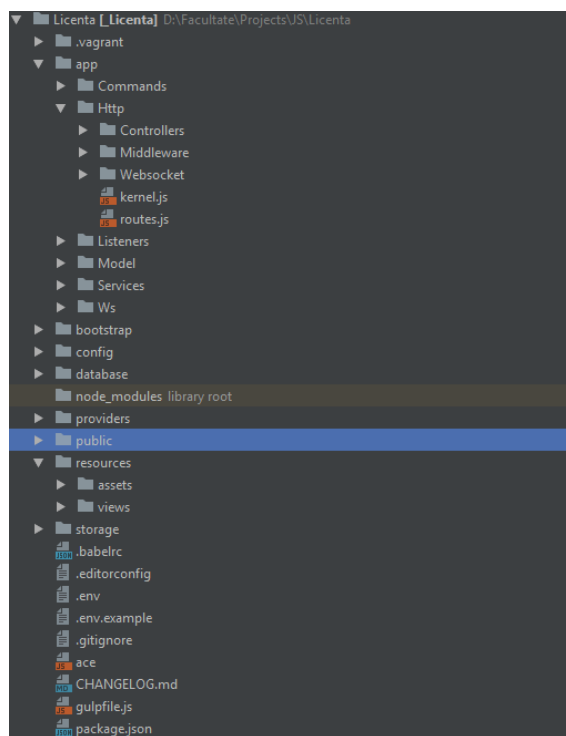


Figura 2: Structura unui proiect in AdonisJS

O interacțiune cu serverul este exemplificată în pașii de mai jos:

- Utilizatorul trimite o cerere către server printr-o metodă (ex: GET, POST) la o anumită adresă (endpoint) existentă în fișierul *routes.js*
- Cererea trece printr-o serie de politici (middleware) definite pentru acea rută, una câte una, iar apoi dacă este în regulă ajunge la controller-ul asociat în fișierul *routes.js*
- Controller-ul întoarce un răspuns (response) utilizatorului și în funcție de necesitate face niște acțiuni înainte (interoghează baza de date, validarea cererii, etc.)

- Dacă este necesară interogarea bazei de date, aceasta se face cu ajutorul unui model definit pentru acea resursă (ex: Modelul *User* pentru tabela *users* din baza de date)
- Răspunsul întors de controller este de obicei o pagină web cu anumite date

2.2 MySQL

Pentru stocarea datelor utilizatorilor avem nevoie de o bază de date. Unul din avantajele framework-ului AdonisJS este faptul că suportă o multitudine de tipuri de baze de date, cum ar fi: PostgreSQL, Oracle, MongoDB, MySQL, etc.

MySQL este un sistem cu sursă deschisă de administrare a bazelor de date relaționale.

Este o componentă centrală în stiva LAMP⁵, împreună cu Linux, Apache și PHP.

Am ales acest sistem din cauza popularității acestuia (este folosit în prezent de către Facebook, Twitter, Flickr, Youtube), el primind critici pozitive pentru performanță și stabilitate, și din cauză că are o documentație foarte bună.

Manipularea datelor se realizează în totalitate cu ajutorul ORM⁶-ului din AdonisJS numit *Lucid*, care este o implementare Active Record⁷ (un model arhitectural care stochează și manipulează datele SQL ca și obiecte, de exemplu pentru tabela *Users* avem clasa *User*).

Pentru modelarea datelor din server am folosit 5 tabele în baza de date, după cum se poate observa în Figura 3:

1. **Users** – stochează informații despre toți utilizatorii aplicației. Are următoarele câmpuri:
 - a. *id*: câmp de identificare unic, nu poate fi NULL și se auto-incrementează odată cu adăugarea unui utilizator nou
 - b. *first_name*: prenumele utilizatorului
 - c. *last_name*: numele utilizatorului
 - d. *email*: adresa de email a utilizatorului, nu trebuie să fie NULL întrucât este necesară autentificării
 - e. *password*: parola utilizatorului, nu trebuie să fie NULL deoarece, de asemenea, este necesară autentificării

⁵ [https://en.wikipedia.org/wiki/LAMP_\(software_bundle\)](https://en.wikipedia.org/wiki/LAMP_(software_bundle))

⁶ https://en.wikipedia.org/wiki/Object-relational_mapping

⁷ https://en.wikipedia.org/wiki/Active_record_pattern

2. ***Friends*** – stochează informații despre relațiile de prietenie a utilizatorilor. Este alcătuită din câmpurile:

a. *id*: câmp de identificare unic, nu poate fi NULL și se auto-incrementează odată cu adăugarea unei noi prietenii

b. *user_id_1*: identificatorul unic al primului utilizator ce face parte din relația de prietenie

c. *user_id_2*: identificatorul unic al celui de-al doilea utilizator ce face parte din relația de prietenie

Cheia primară este identificatorul unic *id* și există și index pe acest câmp deoarece este cheie primară. Avem două chei străine, *user_id_1* și *user_id_2* ce fac legătura între tabelul *Users* și acest tabel. Identificatorii unici ai utilizatorilor sunt sortați înainte de a se insera în baza de date, pentru a elimina duplicitatea (se înserează în *user_id_1* valoarea mai mare dintre cei doi identificatori, iar în *user_id_2* valoarea mai mica).

3. ***Conversations*** – Stochează conversații realizate între utilizatori și are următoarele câmpuri:

a. *id*: identificator unic pentru fiecare conversație în parte, nu poate fi NULL și se auto-incrementează odată cu adăugarea unei noi conversații

b. *name*: câmp ce conține numele unei conversații, poate fi NULL, dacă conversația este între doi utilizatori

4. ***Conversation_user*** – Tabela folosită ca tampon, stochează relațiile dintre utilizatori și conversații. Este necesară pentru conversațiile de grup. Are următoarele câmpuri:

a. *user_id*: identificatorul unic al unui utilizator participant într-o conversație, nu poate fi NULL. Este cheie străină la tabela *Users*

b. *conversation_id*: identificatorul unic al unei conversații, nu poate fi NULL. Este cheia străină la tabela *Conversations*.

Cele două câmpuri formează o cheie primară compusă, asigurând unicitatea, faptul că un utilizator nu poate participa de două sau mai multe ori într-o singură conversație.

5. ***Messages*** – Stochează mesajele dintr-o conversație. Are următoarele câmpuri:

a. *id*: identificatorul unic pentru fiecare mesaj în parte. Acesta nu poate fi NULL și se auto-incrementează odată cu adăugarea unui nou mesaj

b. *conversation_id*: identificatorul unic al conversației din care face parte mesajul, nu poate fi NULL, este cheie străină la tabela *Conversations*

c. *user_id*: identificatorul unic al utilizatorului care a trimis mesajul, nu poate fi NULL și este cheie străină la tabela *Users*

d. *message*: câmpul în care se află conținutul mesajului, trebuie să conțină măcar un caracter, deci nu poate fi NULL

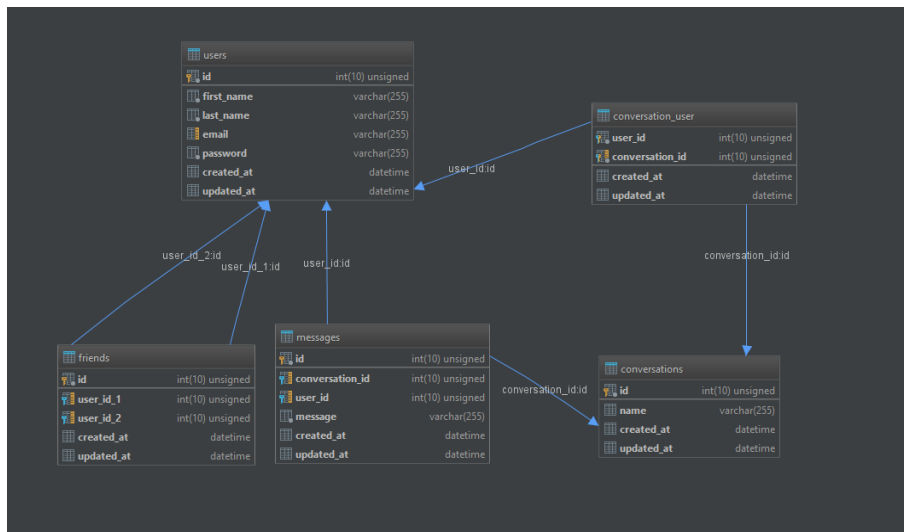


Figura 3: Modelarea datelor în server

2.3 Socket.io

Socket.io este o bibliotecă care facilitează lucrul cu protocolul WebSocket, permițând comunicarea bidirecțională, în timp real, între server și client (spre deosebire de protocolul HTTP care suportă comunicarea unidirecțională). Comunicarea este bazată pe evenimente (modelul Pub/Sub).

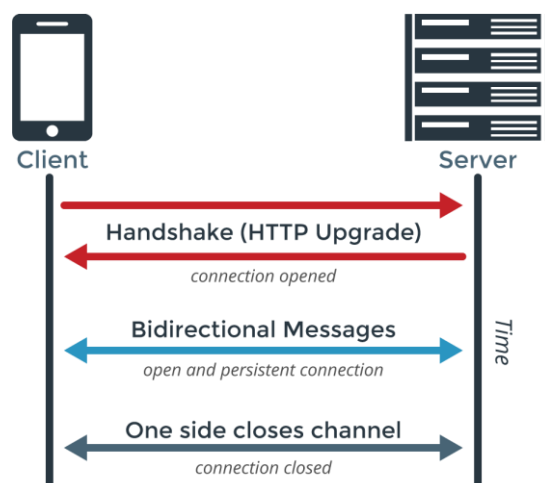


Figura 4: Diagrama WebSocket

⁸ Sursa: <https://www.pubnub.com/blog/2015-01-05-websockets-vs-rest-api-understanding-the-difference/>

Această librărie este ideală pentru o aplicație de mesagerie instantă deoarece oferă conexiune în timp real între client și server.

Librăria este alcătuită din două module:

- un modul (librărie) pentru partea de client
- un modul pentru partea de server

2.3.1 Conversația

Un exemplu de scenariu în care un utilizator participă într-o conversație este definit în felul următor:

1. Clientul apelează funcția *connect()*, parametrii fiind un URL și un port
2. Serverul acceptă cererea de conectare de la client (Figura 7)
3. Serverul ascultă pentru anumite evenimente, de exemplu:
 - i. *Join* – un utilizator se alătură unei conversații
 - ii. *Init* – un participant dorește să primească mesajele existente din conversație
 - iii. *Input* – un participant a trimis un mesaj
4. Clientul emite prin *socket.emit(event, data)* un eveniment în care anunță că dorește să se conecteze la o anumită conversație
5. Serverul îl alătură conversației prin *socket.join(conversation_id)*
6. Clientul emite evenimentul de inițiere *init*, serverul interoghează baza de date pentru mesajele conversației curente și le trimite (prin *socket.emit(„init”, data)*) doar participantului care a emis evenimentul de inițiere
7. Clientul, care ascultă pentru mesajul *init*, execută o funcție care afișează mesajele
8. Atunci când un participant trimite un mesaj nou se emite un eveniment *input* (Figura 5), serverul, care ascultă pentru eveniment, îl preia, îl inserează în baza de date și îl trimite tuturor participanților din conversație mai puțin celui care l-a trimis, prin *socket.broadcast.to(conversation_id).emit(„input”, data)*.
9. Clientul se deconectează de la server, fie apelând funcția *disconnect*, fie obiectul care ține conexiunea este distrus


```

self.DOM.footer.$submit_button.on('click', function ()
{
    let message = self.DOM.footer.getMessage();
    self.messageSubmitted(message);

    data = {
        user_id: self.user_id,
        message: message,
        user_name: self.user_name,
        conversation_id: self.id
    };
    self.socketIO.sendMessage('input', data);
});

```

Figura 5: Exemplu de cod când clientul emite evenimentul *input*

```

// New message from a user
socket.on('input', function (data)
{
    saveMessage(data);

    sendMessageToParticipants(socket, data.room, 'output', data);
});

```

Figura 6: Exemplu de cod din server în care se ascultă pentru evenimentul *input*

```

module.exports = function (io) {

  /**
   * Connection request received from a client
   */
  io.on('connection', function (socket)
  {
    // User joins a conversation
    socket.on('join', function (data)
    {
      socket.join(data.room);
    });

    // User changes conversation
    socket.on('roomChanged', function(data)
    {
      socket.leave(data.leaveRoom);
    });

    // Client loads all messages for a conversation
  });
}

```

Figura 7: Secțiune de cod din aplicație din partea de server ce acceptă conexiune de tip WebSocket

2.4 WebRTC

Web Real-Time Communications⁹ (WebRTC) este un nou standard care permite browserelor să comunice în timp real folosind o arhitectură de-la-egal-la-egal¹⁰ (peer-to-peer). Această tehnologie este folosită în mod preponderent ca și comunicare securizată, bazată pe consimțământ pentru schimbul de date de tip audio, video sau de alt fel între browsere HTML5.

Această tehnologie a schimbat total evoluția aplicațiilor web, deoarece oferă posibilitatea, pentru prima dată, dezvoltatorilor să construiască aplicații multimedia în timp real fără nevoia de plugin-uri proprietare.

WebRTC extinde semantica client-server prin introducerea paradigmei comunicării de-la-egal-la-egal între browsere. Cel mai comun scenariu WebRTC este cel în care ambele browsere rulează aceeași aplicație, descărcată de pe aceeași pagină web. Arhitectura este prezentată în Figura 8.

⁹ <https://webrtc.org/>

¹⁰ <https://en.wikipedia.org/wiki/Peer-to-peer>

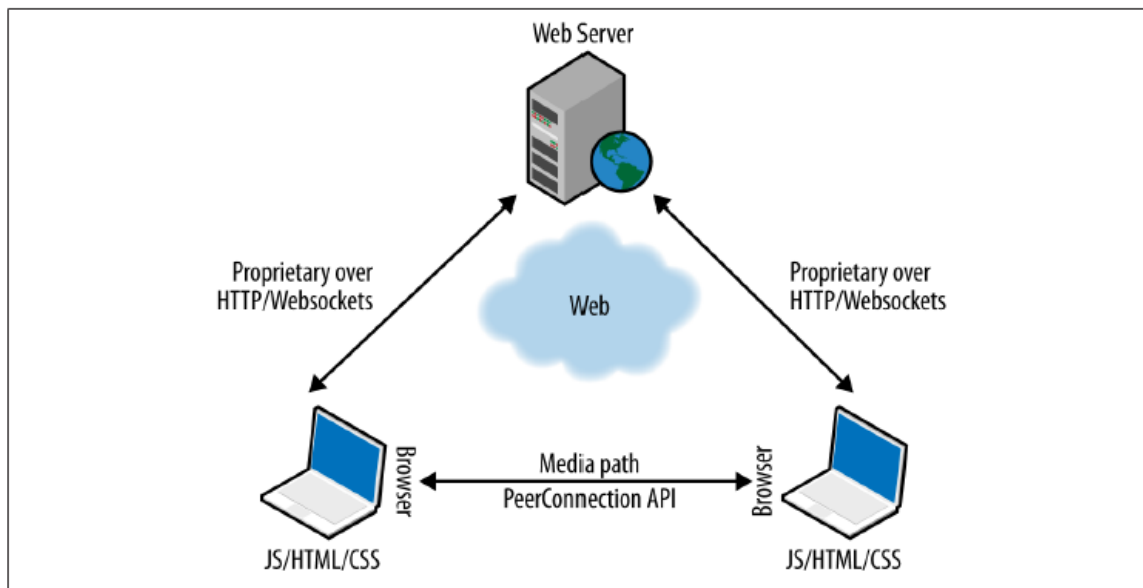


Figura 8: Arhitectura WebRTC¹¹

2.4.1 Cum funcționează

În general, o aplicație WebRTC necesită următoarele:

- Să obțină un flux audio, video sau de alt tip
- Să strângă informații de rețea (adrese IP și porturi) și să le schimbe cu alți clienți WebRTC
- O comunicare de semnalizare (signaling) este folosită pentru a iniția și termina sesiuni și pentru a raporta erori
- Clienți trebuie să schimbe între ei informații despre datele media (cum ar fi rezoluția și codecurile), de obicei se folosesc WebSocketi
- Să afișeze fluxul de date audio, video sau de alt tip

WebRTC implementează trei API¹²-uri (Application Programming Interface):

- *MediaStream* – oferă posibilitatea clienților (cum ar fi browserele) să aibă acces la un flux media, cum ar fi cel obținut de la o cameră web
- *RTCPeerConnection* – oferă transferul de date audio sau video cu suport pentru criptare și administrare a benzii de rețea

¹¹ <http://shop.oreilly.com/product/0636920030911.do>

¹² https://en.wikipedia.org/wiki/Application_programming_interface

- *RTCDataChannel* – oferă transfer de date de orice tip în mod de-la-egal-la-egal

MediaStream

Acest API reprezintă flux-uri media sincronizate. De exemplu, un flux obținut de la camera și microfonul unui dispozitiv are datele video și audio sincronizate.

Fiecare *MediaStream* are o admisie (input), care poate fi generată, de exemplu, de *navigator.getUserMedia()* și o producție (output) care poate fi atașată ca sursă la unu element video sau pasată unei conexiuni *RTCPeerConnection*.

Funcția *getUserMedia()* admite trei parametri:

- Un obiect cu constrângeri
- O funcție callback¹³ de succes, căreia i se transmite *MediaStream*
- O funcție callback de eșec, căreia i se transmite un obiect de eroare

Semnalizarea (signaling)

WebRTC folosește *RTCPeerConnection* pentru a comunica fluxul de date între browsere (sau peers), dar necesită de asemenea un mecanism pentru a coordona această comunicare și pentru a trimite mesaje de control, un proces denumit semnalizare. Acest proces nu este însă specificat de WebRTC, în schimb dezvoltatorii pot alege orice protocol de mesaje doresc, cum ar fi SIP¹⁴, XMPP¹⁵ sau WebSocket.

Acest schimb de informații trebuie să fie complet pentru ca fluxul de date de-la-unu-la-unu să poată începe.

Procesul este folosit pentru următoarele schimburi de informații:

- Mesaje de control a sesiunii: inițiere și închidere a comunicării, erori
- Configurarea rețelei: aflarea adresei IP și a portului pentru lumea exterioară
- Capacități media: ce codec-uri și rezoluții suportă browserele care doresc să comunice

De exemplu, doi utilizatori Alice și Bob doresc să comunice.

Întâi cei doi fac schimb de informații de rețea (folosind framework-ul ICE¹⁶ pentru găsirea interfețelor de rețea și porturilor).

- Alice creează un obiect *RTCPeerConnection* cu o metodă care manevrează *onicecandidate*

¹³ https://developer.mozilla.org/en-US/docs/Mozilla/js-ctypes/Using_js-ctypes/Declaring_and_Using_Callbacks

¹⁴ https://en.wikipedia.org/wiki/Session_Initiation_Protocol

¹⁵ <https://xmpp.org/about/>

¹⁶ <https://www.html5rocks.com/en/tutorials/webrtc/basics/#ice>

- Metoda este rulată când candidații de rețea sunt disponibili
- Alice trimite datele cu candidații către Bob (prin mecanismul de semnalizare, de exemplu WebSocket)
- Bob primește mesajul cu candidatul de la Alice, apelează metoda *addIceCandidate* (pentru a adăuga candidatul la *descrierea partenerului la distanță*)

Clienții WebRTC (cum ar fi Alice și Bob) trebuie de asemenea, după cum a fost menționat anterior, să schimbe și informații de capacități media (cum ar fi codecuri și rezoluții).

Acest schimb se face prin schimbarea unei *oferte* (offer) și a unui *răspuns* (answer) folosind Protocolul de Descriere a Sesiunii (Session Description Protocol¹⁷ – SDP).

- Alice execută metoda din `RTCPeerConnection` *createOffer()*. Metodei *callback*¹⁸ dată ca argument îi este pasată o *descriere a sesiunii locale* ale lui Alice (de tip `RTCSessionDescription`)
- În acea metodă callback Alice își va seta *descrierea locală* folosind *setLocalDescription()*, iar după îi va trimite lui Bob această descriere (prin mecanismul de semnalizare)
- Bob va seta descrierea trimisă de Alice ca și *descrierea îndepărtată* folosind *setRemoteDescription()*
- Bob execută metoda din `RTCPeerConnection` *createAnswer()*, dându-i ca argument descrierea primită de la Alice pentru a i se genera o sesiune locală compatibilă cu a ei. Similar cu primul pas, în callback se setează *descrierea sesiunii locale* a lui Bob și o va trimite lui Alice
- Alice primește descrierea sesiunii lui Bob, o va seta ca și *descrierea îndepărtată* folosind *setRemoteDescription()*

După schimbul de descrieri de sesiune (SDP) cei doi utilizatori încep să schimbe candidați ICE. Fiecare candidat ICE descrie o metodă prin care o persoană poate comunica. Aceasta trimite candidații în ordinea descoperirii lor și trimite până când rămâne fără sugestii, chiar dacă fluxul media a pornit. În momentul în care cei doi egali sugerează un candidat compatibil, fluxul începe să curgă. Dacă cei doi se înțeleg mai târziu în privința unei împerecheri mai potrivite (oferă performanță mai bună) fluxul poate schimba formatul după nevoie.

“candidate”: “candidate:2795255774 1 **udp** 2122260223
192.168.1.7 53196 **typ host** generation 0”

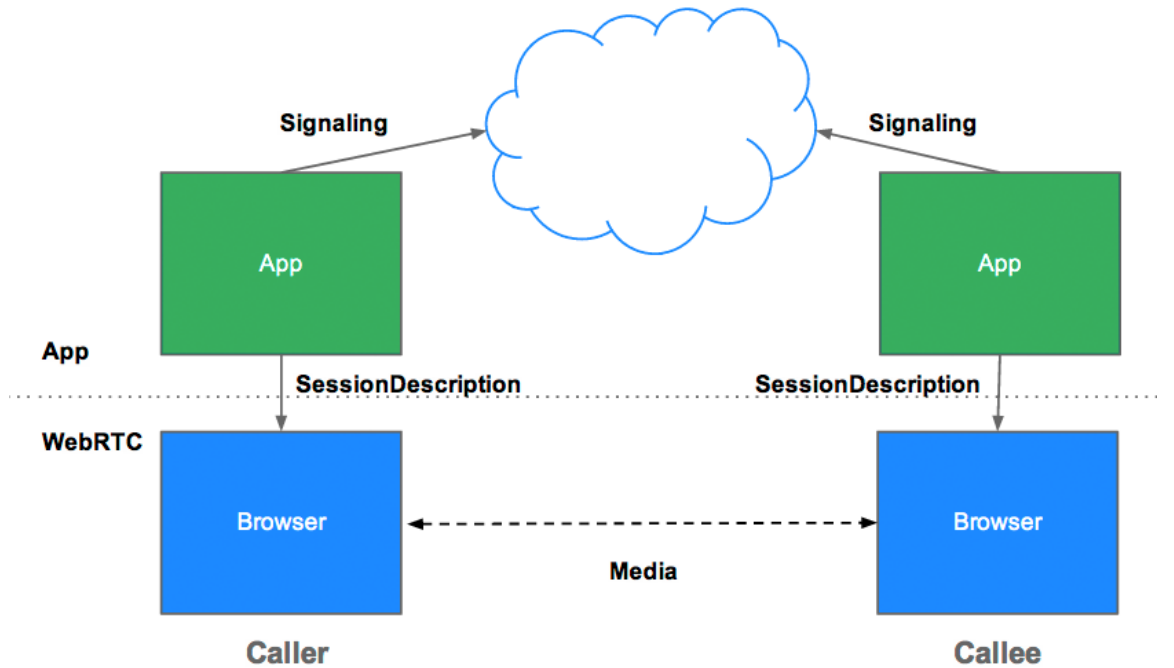
“medium”: “video”

¹⁷ <https://tools.ietf.org/html/rfc4566>

¹⁸ [https://en.wikipedia.org/wiki/Callback_\(computer_programming\)](https://en.wikipedia.org/wiki/Callback_(computer_programming))

Figura 9: Exemplu de candidat ICE

Când acest procedeu de semnalizare este terminat cu succes, fluxul de date este trimis direct de-la-egal-la-egal (Peer-to-Peer) între apelant și apelat.



19

Figura 10: Arhitectura ofertă/răspuns descrisă se numeste JSEP²⁰

RTCPeerConnection

Acest API este cel care manevrează fluxul de date între participanți.

Pentru ca procedeul de-la-egal-la-egal să aibă loc este necesară parcurgerea NAT (Network Address Translation²¹) implementată cu ajutorul framework-ului ICE, care folosește protocolul STUN²² și extensia acestuia, numita TURN²³.

Serverul NAT oferă suportul pentru traducerea adresei IP interne (private) în adresă publică și invers.

¹⁹ <https://www.html5rocks.com>

²⁰ <http://tools.ietf.org/html/draft-ietf-rtcweb-jsep-00>

²¹ https://en.wikipedia.org/wiki/NAT_traversal

²² <https://en.wikipedia.org/wiki/STUN>

²³ https://en.wikipedia.org/wiki/Traversal_Using_Relays_around_NAT

Acest framework (ICE) încearcă inițial să conecteze participanții direct, prin UDP. Se folosesc serverele STUN pentru a se descoperi adresa publică și portul unui participant din spatele unui NAT.

Dacă procedeul eșuează prin UDP, se încearcă TCP. De asemenea dacă procedeul eșuează prin serverele STUN se folosesc serverele TURN (Relay).

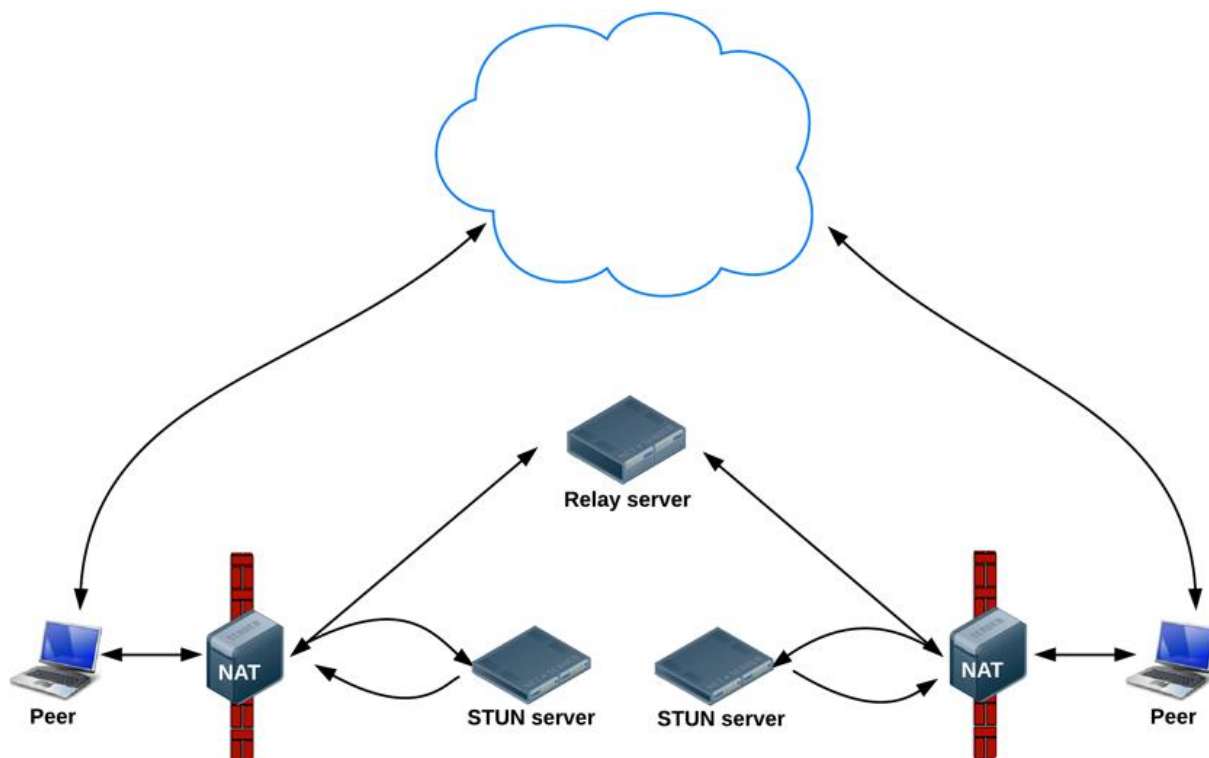


Figura 11: Găsirea candidaților²⁴

RTCDataChannel

Cel de-al treilea API al WebRTC oferă suport pe lângă date audio și video și pentru alte tipuri de date cu latență scăzută, printr-un canal de date, comunicarea fiind direct între browsere, deci ideală pentru transfer de fișiere sau mesaje text (de exemplu pentru anonimitate).

Acest API este similar cu cel al WebSocket-ului, datele fiind trimise prin metoda *send* și primite prin evenimentul *onmessage*.

²⁴ Sursa: <https://www.html5rocks.com/en/tutorials/webrtc/basics/stun.png>

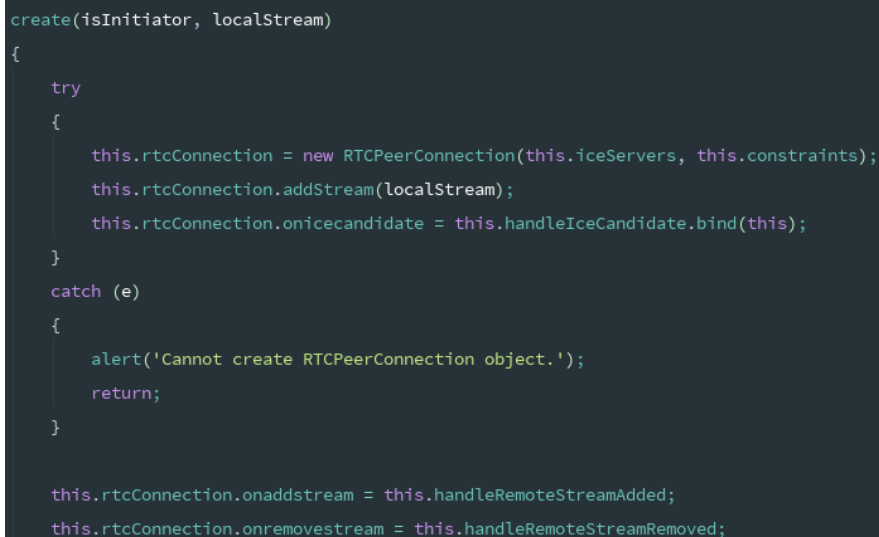
2.4.2 Apel video unu-la-unu

Un participant într-o conversație unu-la-unu inițiază un apel video. Acesta este redirecționat din pagina conversațiilor către pagina cu apelul. Dacă acesta nu a mai fost întrebat în trecut, este întrebat dacă dorește să acorde dreptul de acces aplicației la camera web și microfon (WebRTC necesită consimțământ). În momentul în care utilizatorul aprobă, începe capturarea datelor video și audio, se creează un element video, i se setează sursa un obiect BLOB creat din captura de mai sus. Se emite un eveniment la server că un utilizator a inițiat apelul.

Se așteaptă să se conecteze celălalt participant.

Când s-a conectat și celălalt participant, se repetă procesul de mai sus, iar când acesta își dă acordul și el, se creează un obiect `RTCPeerConnection` și procesul de semnalizare între cei doi poate începe.

```
create(isInitiator, localStream)
{
    try
    {
        this.rtcConnection = new RTCPeerConnection(this.iceServers, this.constraints);
        this.rtcConnection.addStream(localStream);
        this.rtcConnection.onicecandidate = this.handleIceCandidate.bind(this);
    }
    catch (e)
    {
        alert('Cannot create RTCPeerConnection object.');
```



```
        return;
    }

    this.rtcConnection.onaddstream = this.handleRemoteStreamAdded;
    this.rtcConnection.onremovestream = this.handleRemoteStreamRemoved;
}
```

Figura 12: Crearea obiectului `RTCPeerConnection`

Utilizatorul inițiator creează o ofertă SDP, o setează ca descriere locală și o trimite, prin server, la celălalt participant, care și-o setează ca fiind descrierea la distanță (remote description), creează un răspuns SDP, își setează descrierea locală (local description) și o trimite la utilizatorul inițiator. Acesta, după ce a primit răspunsul SDP își va seta descrierea la distanță.


```

doCall()
{
    this.rtcConnection.createOffer(this.setLocalAndSendMessage.bind(this), this.onSignalingError, this.sdpConstraints);
}

doAnswer()
{
    this.rtcConnection.createAnswer(this.setLocalAndSendMessage.bind(this), this.onSignalingError, this.sdpConstraints);
}

setLocalAndSendMessage(sessionDescription)
{
    this.rtcConnection.setLocalDescription(sessionDescription);

    this.sendMessage({
        sd: sessionDescription,
        channel: this.room
    });
}

```

Figura 13: Utilizatorul inițiator creează oferta SDP, o setează ca descriere locală și o trimite la celălalt participant

Urmează faza de descoperire a candidaților ICE, iar după ce se găsesc doi compatibili fluxul media poate începe să fie trimis între clienți.

2.5 Kurento Media Server

WebRTC și capacitățile sale de-la-egal-la-egal sunt ideale pentru comunicații unu-la-unu. Însă dacă ne dorim să trecem mai departe, la servicii mai-mulți-la-mai-mulți cum ar fi conferințele video, se ridică o întrebare: „Ce arhitectură ar trebui să folosesc pentru asta?”.

Există mai multe tipuri diferite de arhitecturi care ar putea rezolva problema de față:

- **Soluția de tip „plasă” (mesh):** este cea mai simplă deoarece nu necesită infrastructură în plus, însă este cea mai ineficientă deoarece creează mai multe fluxuri media unu-la-unu de la fiecare transmițător la fiecare alt participant
- **Soluția mixer:** această arhitectură este bazată pe un punct central care păstrează un flux unu-la-unu cu fiecare participant. Acest punct central primește și amestecă (mixează) fiecare flux audio și video într-unul singur și îl trimite la fiecare participant în parte. Este ineficientă și ea deoarece este costisitoare când codează și decodează datele media
- **Soluția ruter:** este cea mai folosită din noile platforme WebRTC. Arhitectura se bazează pe un punct central care primește fluxul de la fiecare transmițător și

îl trimite la fiecare participant în parte, deci nu necesită codări și decodări, ceea ce o face cea mai eficientă soluție

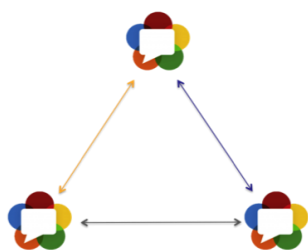


Figura 14: Soluția mesh²⁵

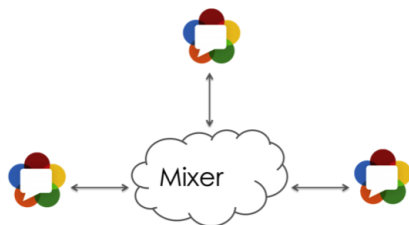


Figura 15: Soluția mixer²⁶

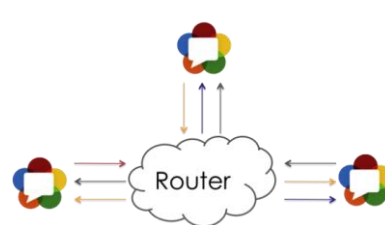


Figura 16: Soluția ruter²⁷

Kurento Media Server²⁸ este un server media de tip WebRTC responsabil pentru transmiterea, procesarea și încărcarea fluxurilor media.

API-ul Kurento este bazat pe conceptul de **MediaElement**. Un MediaElement deține o capacitate media specifică. De exemplu, un element media numit *WebRtcEndpoint* deține capacitatea de a trimite și primi fluxuri media WebRTC, elementul media numit *RecorderEndpoint* are capacitatea de a înregistra fluxul media primit și a-l salva în sistemul de fișiere.

2.5.1 Crearea aplicațiilor cu Kurento

„Din perspectiva dezvoltatorilor, elementele media (MediaElement) sunt ca piesele Lego. Trebuie doar să iei elementele de care ai nevoie pentru aplicația ta și să le conectezi urmând topologia dorită.,”²⁹.

Un graf de elemente media conectate este numit **Media Pipeline** (conductă media).

Conectivitatea este controlată prin primitiva *connect*, expusă de API, care este întotdeauna invocată în elementul sursă și primește ca argument elementul chiuvetă (sink), de exemplu: *sourceMediaElement.connect(sinkMediaElement)*.

²⁵ Sursa: <http://webrtcchacks.com>

²⁶ Sursa: <http://webrtcchacks.com>

²⁷ Sursa: <http://webrtcchacks.com>

²⁸ http://doc-kurento.readthedocs.io/en/stable/introducing_kurento.html

²⁹ http://doc-kurento.readthedocs.io/en/stable/introducing_kurento.html

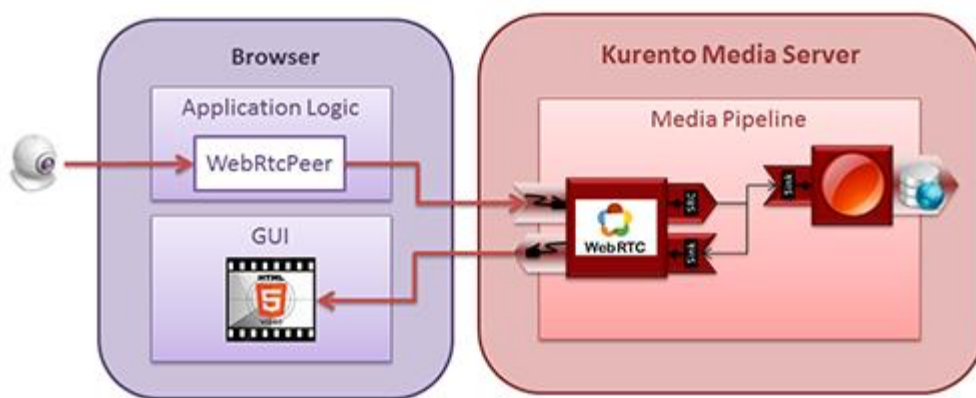


Figura 17: Exemplu simplu de Media Pipeline³⁰

2.5.2 Conferința video

Fiecare conversație în parte are asociată propria conductă media (Media Pipeline), cu scopul de a fi izolată de celelalte.

Fiecare participant într-o conversație își va trimite propriul flux media și în schimb va primi de la fiecare alt participant câte un flux media. Acest lucru va rezulta într-un total de $n*n$ de **WebRtcEndpoint** în fiecare conversație, unde n este numărul de participanți.

În continuare vom prezenta un scenariu în care participanții într-o conversație de grup inițiază o conferință video.

O participantă (numită în continuare Alice) se alătură unei conferințe:

1. Clientul emite un eveniment *joinRoom* prin care anunță că dorește să se alăture conferinței
2. Evenimentul este interceptat în server, se verifică dacă conferința există deja, dacă nu atunci se creează și i se asociază un obiect **MediaPipeline** în care sunt inserate conexiunile **WebRtc** de la participanți, apoi serverul o alătură pe Alice la conferință
3. Se creează un obiect **WebRtcEndpoint** pentru aceasta și se anunță ceilalți participanți, dacă există, că un nou participant s-a alăturat
4. Dacă există, fiecare din ceilalți participanți sunt notificați că un nou utilizator s-a alăturat (serverul emite un eveniment *newParticipantArrived*). În partea de client se creează un obiect **WebRtcPeer** de tip **Recvonly** pentru primirea fluxului de date și se generează o ofertă pentru noul venit.

În același timp noul venit este informat de server prin evenimentul *existingParticipants* ca există alți participanți în conferință. Acestuia i se creează un obiect

³⁰ Sursa: http://doc-kurento.readthedocs.io/en/stable/introducing_kurento.html

WebRtcPeer de tip **Sendonly** și un element video și se trimite la server pentru a fi trimis la ceilalți participanți.

Fiecare participant care își trimite fluxul video creează o oferta SDP, iar fiecare participant care primește fluxul este informat de oferta primită, creează un răspuns, astfel se face schimb de descrieri de sesiuni (SDP), de candidați ICE, după cum este descris la Apelul video unu-la-unu, și la final fluxul de date este permis între cei doi.

2.6 WebTorrent

Majoritatea aplicațiilor de tip mesagerie instantă moderne vin la pachet cu soluții de transfer de fișiere pentru a oferi o experiență cât mai completă utilizatorilor. Însă de multe ori aceste soluții sunt incomplete sau au neajunsuri.

Un exemplu ar fi limitarea dimensiunilor fișierelor (25MB Facebook sau 16MB WhatsApp, de exemplu), utilizatorii fiind obligați să caute alte platforme sau aplicații.

O altă problemă pentru transferul de fișiere între utilizatori este că dacă, spre exemplu, Alice dorește să trimită un fișier lui Bob, aceasta încarcă fișierul în aplicație, este trimis la server întâi, după abia ce s-a terminat de încărcat, Bob poate începe descărcarea. Deseori fiind limitată viteza de încărcare/descărcare acest lucru face ca transferul să dureze o perioadă îndelungată.

O soluție ideală pentru transferul de fișiere este lipsită de inepedimentele prezentate mai sus.

Librăria WebTorrent³¹, inspirată din protocolul BitTorrent³², care oferă distribuire de-la-egal-la-egal de date, este perfectă pentru acest caz.

WebTorrent este un client pentru browsere și desktop scris în JavaScript care folosește WebRTC pentru transport de-la-egal-la-egal, care nu necesită pluginuri, extensii sau instalare.

2.6.1 Transfer unu-la-unu

Un exemplu de scenariu este prezentat mai jos:

- Alice încarcă fișierul dorit, cu dimensiuni nelimitate
- Clientul WebTorrent creează un torrent din fișier și începe să îi facă seed³³ (în browser), rezultatul fiind un obiect ce conține, printre altele, un magnet URI (Figura 18)
- Bob dorește să descarce fișierul încărcat de Alice

³¹ <https://webtorrent.io/>

³² <https://en.wikipedia.org/wiki/BitTorrent>

³³ https://en.wikipedia.org/wiki/Glossary_of_BitTorrent_terms

- Clientul WebTorrent al lui Bob primește, printr-un mecanism de semnalizare (WebSocket, de exemplu), obiectul ce conține magnet URI-ul și îl adăugă (Figura 11)
- Torrentul începe să se descarce, iar când este finalizat Bob îl va putea salva pe disc

După cum se poate observa în scenariul de mai sus, dimensiunea fișierului nu este limitată, iar problema a doua expusă mai sus nu este prezentă deoarece fișierul nu este încărcat întâi în server, ci trimis direct către Bob, imediat, bucată cu bucată.

```
var dragDrop = require('drag-drop')
var WebTorrent = require('webtorrent')

var client = new WebTorrent()

// When user drops files on the browser, create a new torrent and start seeding it!
dragDrop('body', function (files) {
  client.seed(files, function (torrent) {
    console.log('Client is seeding ' + torrent.magnetURI)
  })
})
```

Figura 18: Exemplu de cod în care un utilizator încarcă un fișier prin WebTorrent

```
var WebTorrent = require('webtorrent')

var client = new WebTorrent()

var magnetURI = 'magnet: ...'

client.add(magnetURI, { path: '/path/to/folder' }, function (torrent) {
  torrent.on('done', function () {
    console.log('torrent download finished')
  })
})
```

Figura 19: Exemplu de cod în care un utilizator descarcă un fișier prin WebTorrent

2.6.2 Transfer unu-la-mai-mulți

Această soluție este și mai performantă atunci când un utilizator dorește să transfere un fișier către mai multe persoane, ca într-o conversație de grup, de exemplu.

Într-un scenariu standard (Figura 20):

- Alice încarcă un fișier în aplicație, acesta este trimis la server, ceilalți participanți așteaptă
- În momentul în care fișierul s-a încărcat complet, ceilalți pot începe descărcarea acestuia
- Serverul trebuie să suporte lățimea de bandă, memoria ocupată, puterea de procesare

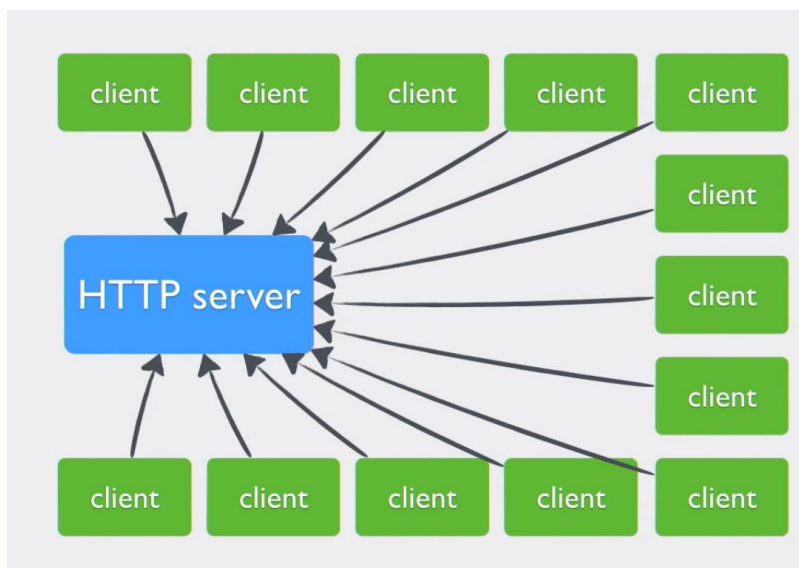


Figura 20: Scenariu standard de transfer de fișier cu un server HTTP³⁴

În scenariul nostru, în care folosim librăria WebTorrent (Figura 21):

- Alice încarcă un fișier în aplicație și începe să îi aplice procesul de seed
- Ceilalți utilizatori încep imediat să îl descarce și să facă seed bucaților descărcate
- În acest fel se creează o rețea de egali (peers) care își trimit unu altuia bucați din fișier, făcând procesul mai eficient și mai sigur, deoarece nu mai există un singur punct central de eșec (decentralizare)
- Când un participant are toate bucățile din fișier îl va salva local, pe disc

³⁴ Sursa: <https://www.youtube.com/watch?v=kxHRATfwnlw>

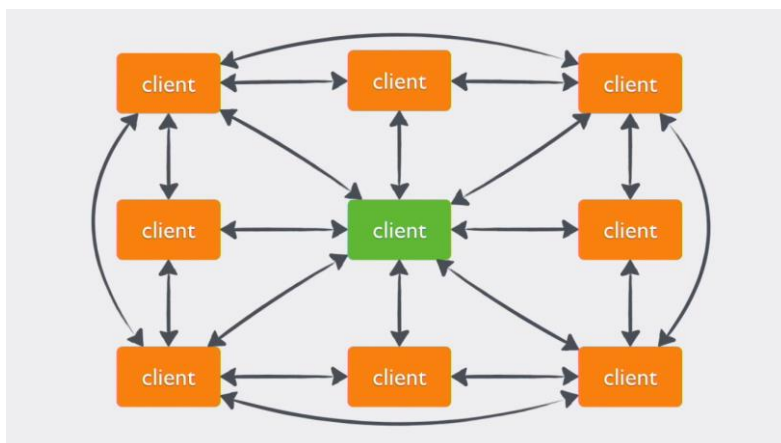


Figura 21: Scenariu de transfer de fișier folosind WebTorrent³⁵

2.7 EcmaScript2015

EcmaScript (sau ES) este o specificație creată pentru a standardiza limbajele de scripting, iar JavaScript este o implementare a sa.

Există mai multe versiuni în momentul actual:

- Versiunea 5: suportată de toate browserele, standarul pe care se bazează versiunea actuală de JavaScript
- Versiunea 6: sau 2015 este implementată parțial în browserele moderne, aduce o mulțime de noi caracteristici: **const** și **let** pentru variabile, **iteratori**, o noua sintaxă pentru a construi obiecte (**class**), **import-uri**, **export-uri** de obiecte, **etc.**
- Versiunea 7: sau 2016 a fost finalizată în iunie 2016

Am ales versiunea 2015 deoarece ajută la modularizarea codului, care simplifică administrarea dependențelor, organizarea codului (împărțim funcționalitatea aplicației, furnizează encapsulare), oferă reutilizarea codului, decuplarea, etc.

³⁵ Sursa: <https://www.youtube.com/watch?v=kxHRATfvnlw>

```

import {Config} from '../_config';
import {SocketIO} from '../modules/socket';

class FileTransfer
{
  constructor(id, user_id, user_name)
  {
    this.id = id;
    this.user_id = user_id;
    this.user_name = user_name;

    this.socketIO = new SocketIO(io, 'http://localhost:8181/chat');
    this.socketIO.setRoom(this.id);

    this.files = {};

    this.bindListeners();
  }

  bindListeners()
  {
    PubSub.subscribe(Config.getFileDroppedMessage(), this.handleFileDropped.bind(this));

    PubSub.subscribe(Config.getConversationSwitchMessage(), this.handleConversationSwitched.bind(this));

    this.socketIO.socket.on('new_file', this.handleNewFile.bind(this));

    PubSub.subscribe(Config.getDownloadFileMessage(), this.handleFileDownload.bind(this));
  }
}

```

Figura 22: Exemplu de cod din partea de client scrisă în ECMAScript2015

Din cauză că aceasta versiune nu este încă suportată în totalitate de browsere, codul trebuie să fie compilat în versiunea 5, care este prezentă în toate browserele.

Compilarea necesită 2 librării: **Webpack** și **Babel**.

Librăria Webpack împachetează fișierele care trebuie să fie împreună (belong together) într-un singur fișier cu scopul de a fi servit într-un răspuns la o singură cerere. Un pachet de acest fel poate conține mai multe tipuri de fișiere: JavaScript, CSS, HTML.

Librăria Babel oferă un plugin pentru Webpack, care este utilizat la compilarea codului ECMAScript2015 în cod ECMAScript 5.


```

module.exports = {
  context: `${__dirname}/resources/assets/webpack`,
  entry: {
    app: './app.js',
    voice_two_peers: './voiceCall/two_peers',
    voice_conference: './voiceCall/conference',
    conference: './conference/index.js',
    conversation: './conversation/index.js',
    videocall: './videocall/index.js',
    cinema: './cinema/index.js',
    fileTransfer: './file_transfer/index.js',
    friends: './friends/index.js',
  },
  output: {
    path: `${__dirname}/public/js/app`,
    filename: '[name].js'
  },
  module: {
    loaders: [
      {
        loader: 'babel-loader',
        query: {
          presets: ['es2015']
        }
      }
    ]
  }
};

```

Figura 23: Configurare Webpack și Babel

2.8 Sass

Sass este o extensie pentru CSS, care adaugă reguli înlănțuite, variabile, moștenirea selectorilor și multe altele.

Este compilat în CSS standard, bine formatat, folosind o librărie sau o unealtă de tip linie de comandă.

Pentru compilarea codului am utilizat **Gulp** care este o unealtă pentru automatizarea sarcinilor. Suportă acțiuni ca compilarea fișierelor Sass, compresia și minificarea fișierelor JavaScript.

```

#cinema-container-vertical
  display: flex

  .card-panel
    padding: 0

  // Less than or equal to 3 people in conversation
  #film-container
    flex-basis: 75%
    flex-grow: 0
    flex-shrink: 0

    display: flex
    align-items: center
    justify-content: center

    #film-placeholder
      background-color: #d0d0d0
      color: #BF1E2E
      font-size: 30px
      text-align: center
      border: $primary 2px dashed

```

Figura 24: Exemplu de cod Sass, în care se prezintă înlănțuirea regulilor și utilizarea variabilelor

3. Descrierea aplicației

3.1 Pagina de întâmpinare (Landing page)

Primul ecran cu care utilizatorul este întâmpinat.

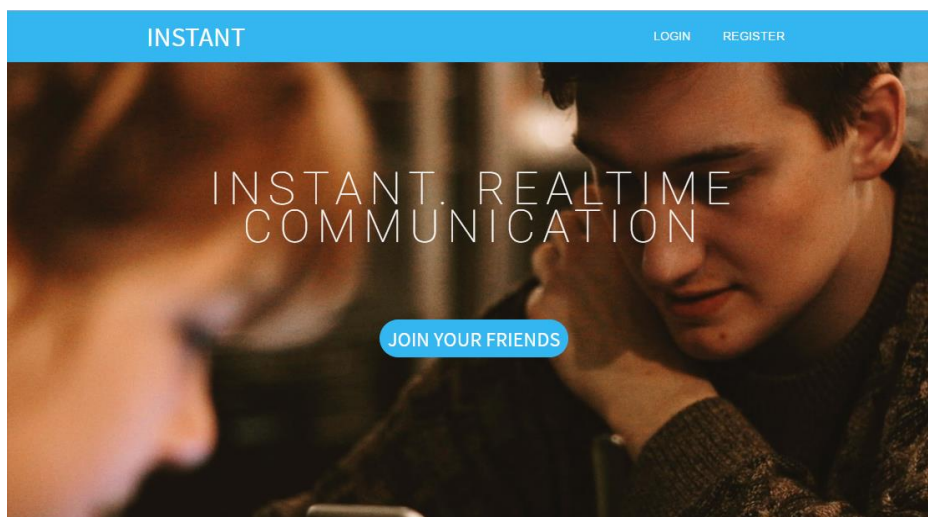


Figura 25: Ecranul de întâmpinare

3.2 Pagina de autentificare

Accesul în aplicație se face numai după ce utilizatorul trece de acest ecran și primește de la server o identitate (session token), pentru a putea fi recunoscut mai târziu în celelalte ecrane. Autentificarea se face cu ajutorul unui cont creat în prealabil în ecranul de înregistrare, fiind necesare două câmpuri: adresa de email și parola.

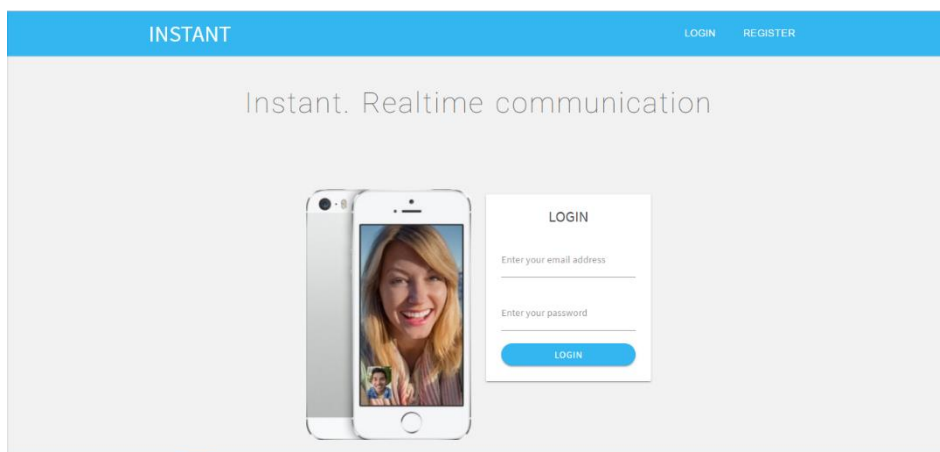


Figura 26: Ecranul de autentificare

3.3 Pagina de înregistrare

Acest ecran este cel ce furnizează utilizatorului posibilitatea de a-și crea un cont pentru a avea acces la toate funcționalitățile aplicației.

Pentru înregistrare sunt necesare patru câmpuri: prenume, nume, email și parola. Ultimele două câmpuri sunt necesare autentificării, iar primele două facilitează interacțiunea cu ceilalți utilizatori (poți căuta utilizatori după nume și prenume).

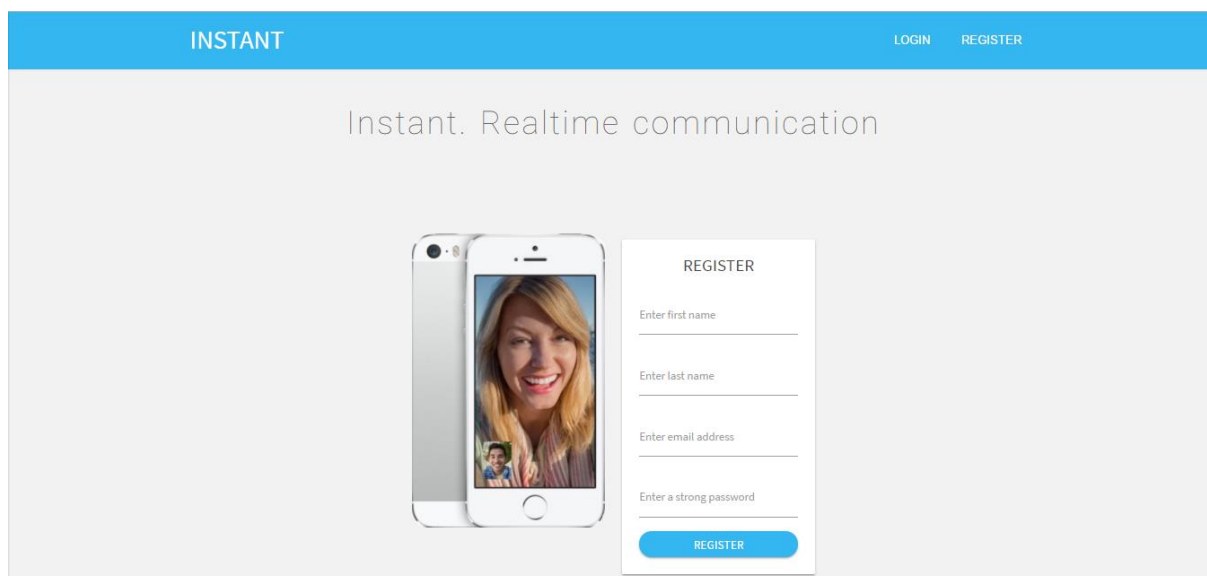


Figura 27: Ecranul de înregistrare

3.4 Pagina de acasă

Ecranul Acasă reprezintă ecranul cu care este întâmpinat un utilizator înregistrat și autentificat al site-ului.

Acest ecran este împărțit în două părți principale:

- Lista de conversații
- Conversația deschisă

Aceste două părți sunt secționate la rândul lor în alte componente mai mici.

1. Lista de conversații

- Secțiunea de sus
 - o bară de filtrare a conversațiilor după nume
 - un buton de creare a unui nou grup de conversație
- Secțiunea de jos
 - o stivă de conversații ale utilizatorului

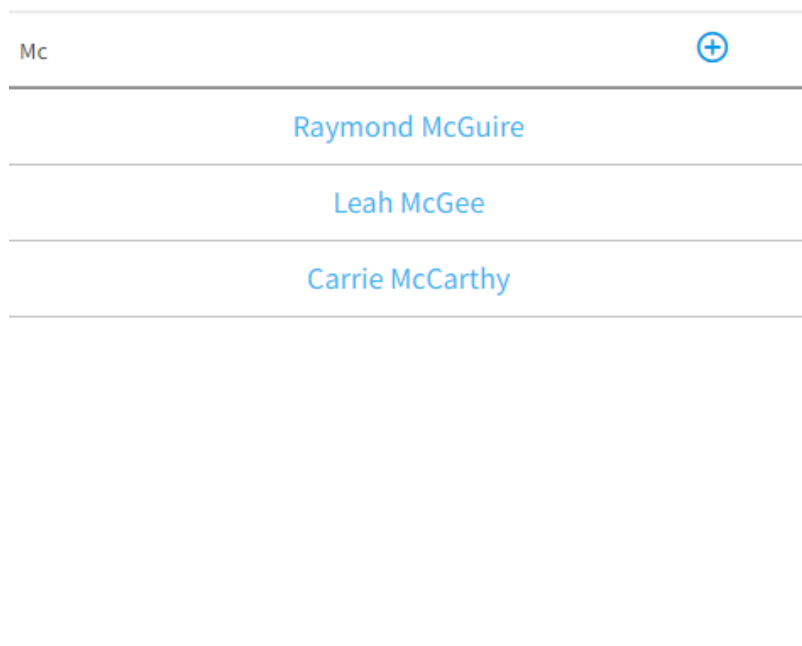


Figura 28: Ecranul de Acasă. Lista de conversații, a fost aplicat filtrul după nume

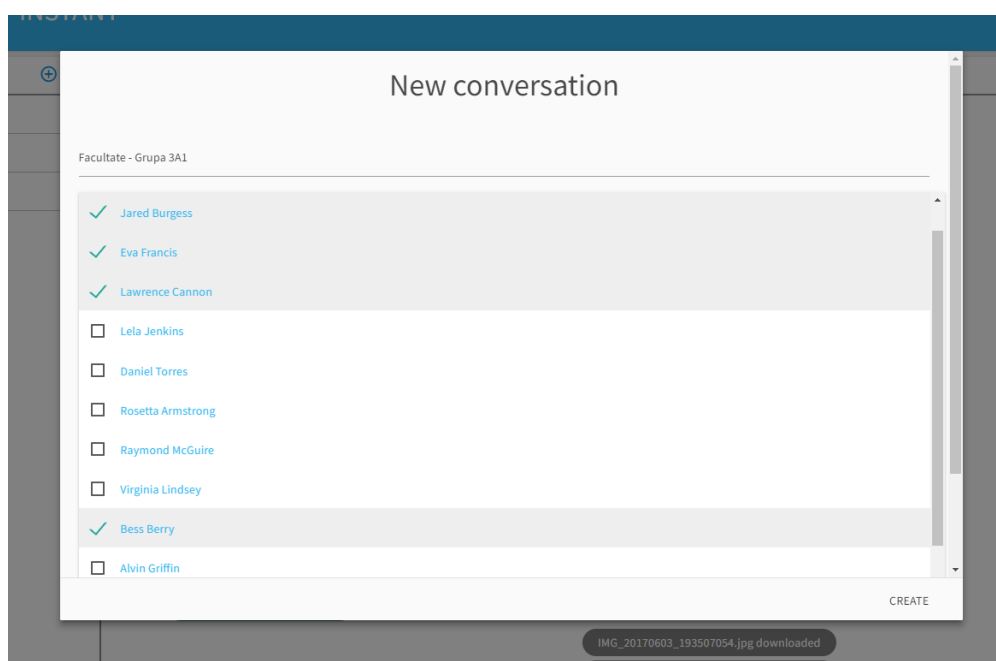


Figura 29: Ecranul de Acasă. Crearea unui nou grup de conversație

2. Conversația deschisă

- Secțiunea I: un grup de patru butoane
 - Buton pornire apel voce
 - Buton pornire apel video

- Buton deschidere cinema
- Buton setări (posibilitatea ștergerii conversației)
- Secțiunea II:
 - Mesajele aparținând conversației (mesajele utilizatorului curent sunt colorate cu alb pe un fundal albastru, iar cele ale celorlalți utilizatori au culoarea neagră pe un fundal gri, însoțite de numele autorului deasupra lor)
 - Această secțiune este de asemenea și o zonă Drag-and-Drop pentru transferul de fișiere, după ce un utilizator încarcă un fișier acesta va apărea ca un mesaj text, însă colorat cu alb pe fundal verde deschis pentru utilizatorul autor și colorat cu negru pe fundal verde închis pentru ceilalți participanți
 - Când un utilizator apasă click pe un mesaj conținând un fișier, fișierul se va descărca, când se termină utilizatorul trebuie să dea click pe un nou mesaj pentru a-l salva local
- Secțiunea III:
 - Câmpul de intrare pentru compunerea unui nou mesaj
 - Butonul de trimitere a mesajului compus

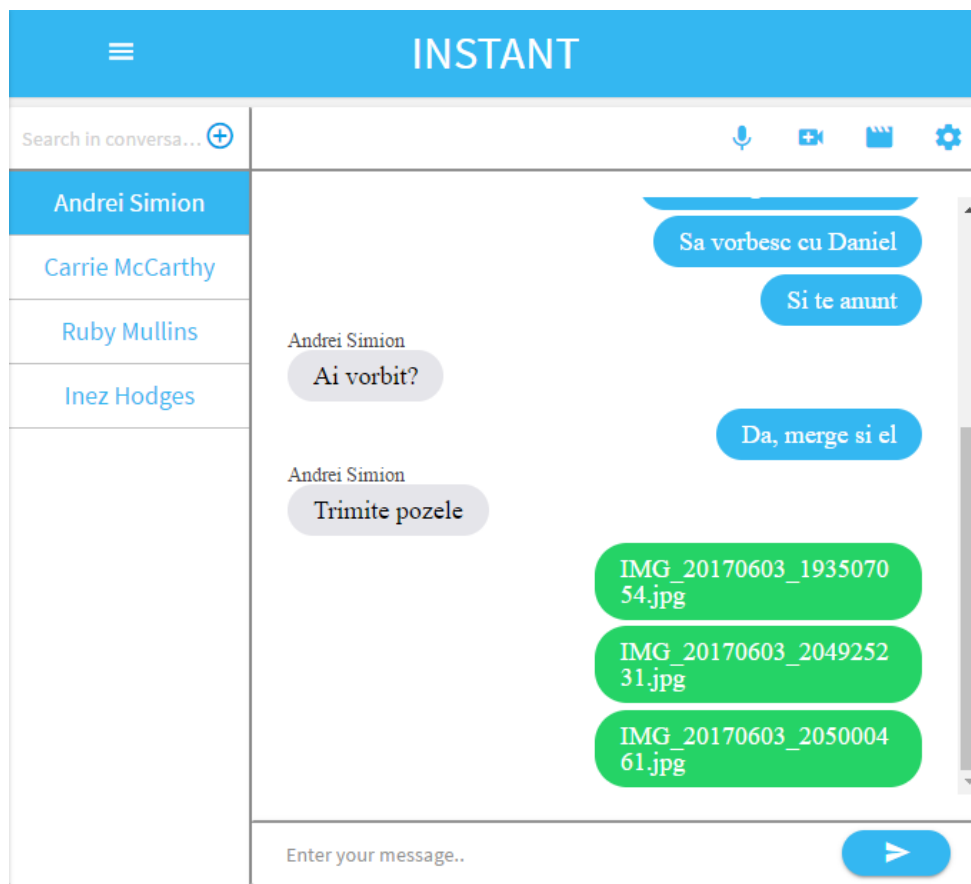


Figura 30: Ecranul de Acasă (Perspectiva 1)

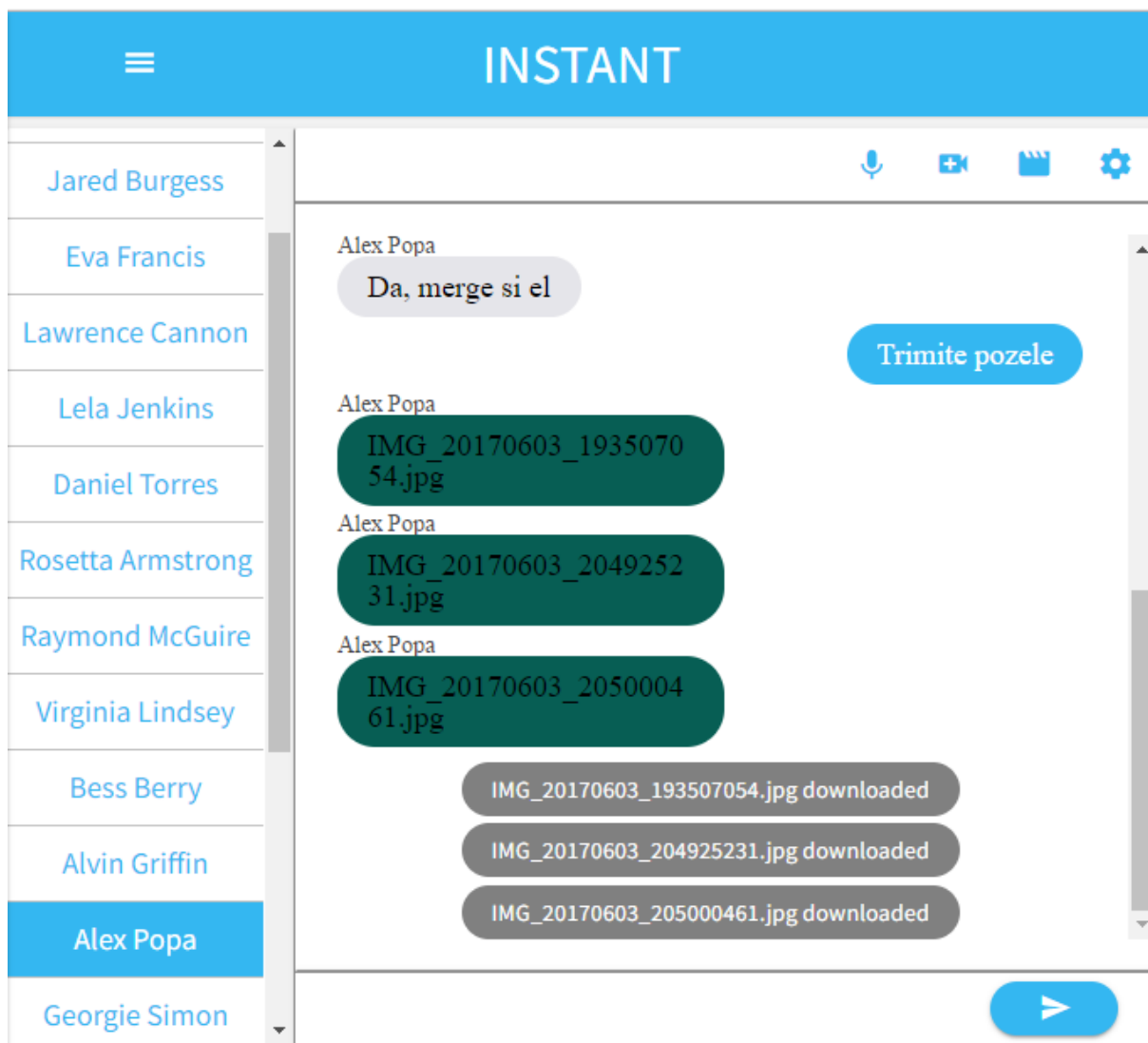


Figura 31: Ecranul de Acasă (Perspectiva 2)

În Figura 31 se observă că după ce celălalt utilizator participant în conversație a încărcat (prin Drag-and-Drop) mai multe imagini, utilizatorul curent a selectat (apăsând click pe fiecare mesaj asociat fișierului) fișierele dorite spre a fi descărcate.

După finalizarea procesului de descărcare au apărut 3 noi mesaje informative, afișând faptul că fișierele au fost descărcate (în memorie). Pentru ca imaginile descărcate să fie salvate local (pe disc) utilizatorul trebuie să dea click pe mesajul informativ asociat imaginii dorite.

Dacă există un apel (fie voce sau video) ceilalți participanți în conversație sunt informați (Figura 32).

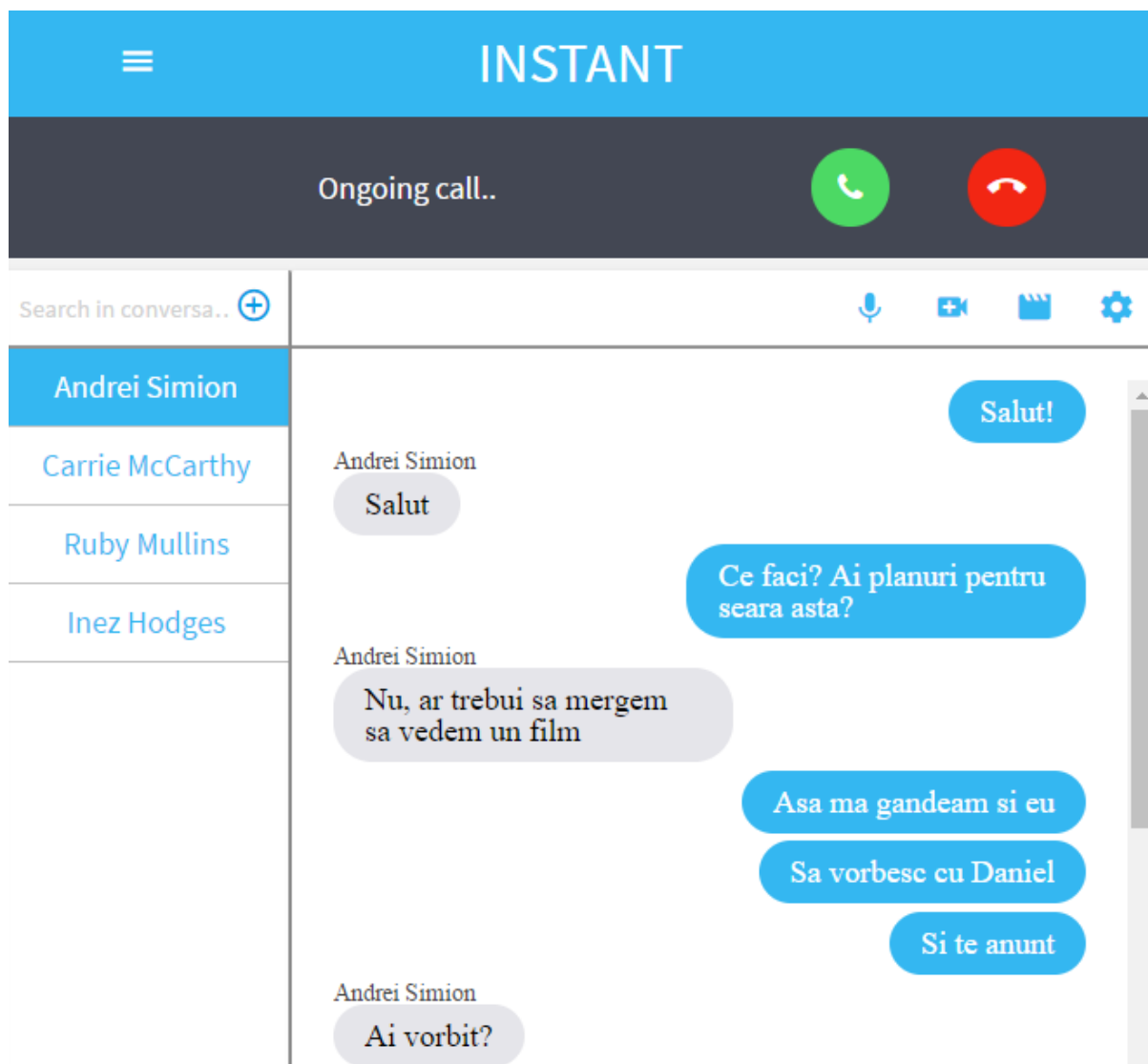


Figura 32: Un alt participant în conversație a pornit un apel voce, apel video sau cinema.

În funcție de opțiunea aleasă, utilizatorul va fi trimis pe pagina apelului dacă a apăsă butonul verde sau ascunde mesajul informativ dacă apasă butonul roșu.

3.5 Pagina de apel video

Această pagină oferă utilizatorilor posibilitatea comunicării în cadrul unui apel video de tip unu-la-unu sau de tip conferință, în funcție de numărul de participanți în conversație.

Ecranul este împărțit în două părți principale:

- **Secțiunea de videoclipuri** ce conțin capturi în timp real ale camerelor web ale utilizatorilor
- **Secțiunea de conversație** în care utilizatorii pot schimba mesaje sau le pot vizualiza pe cele existente

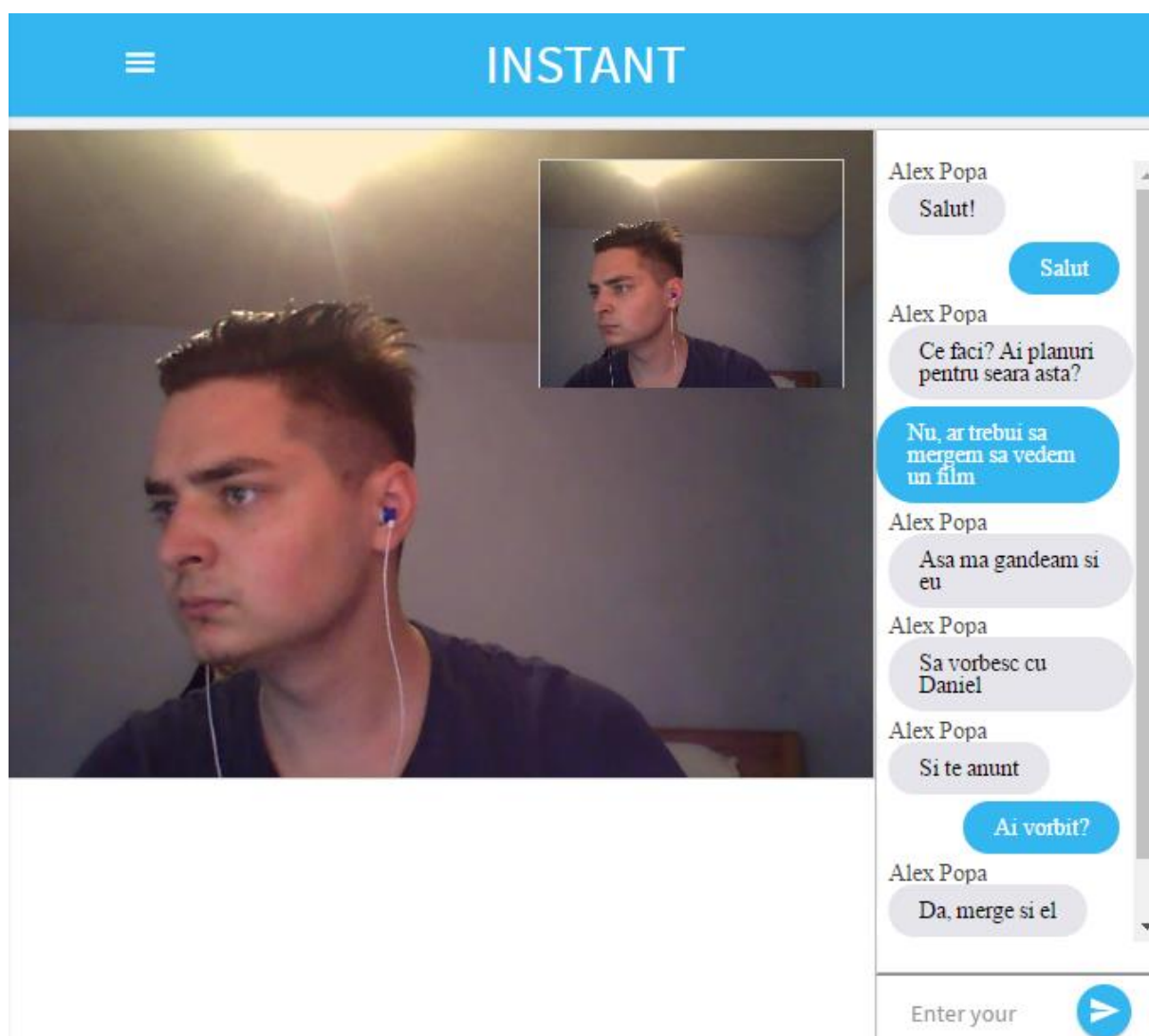


Figura 33: Ecranul de apel video de tip unu-la-unu, doi utilizatori diferiți folosesc aceeași cameră web (scop demonstrativ)

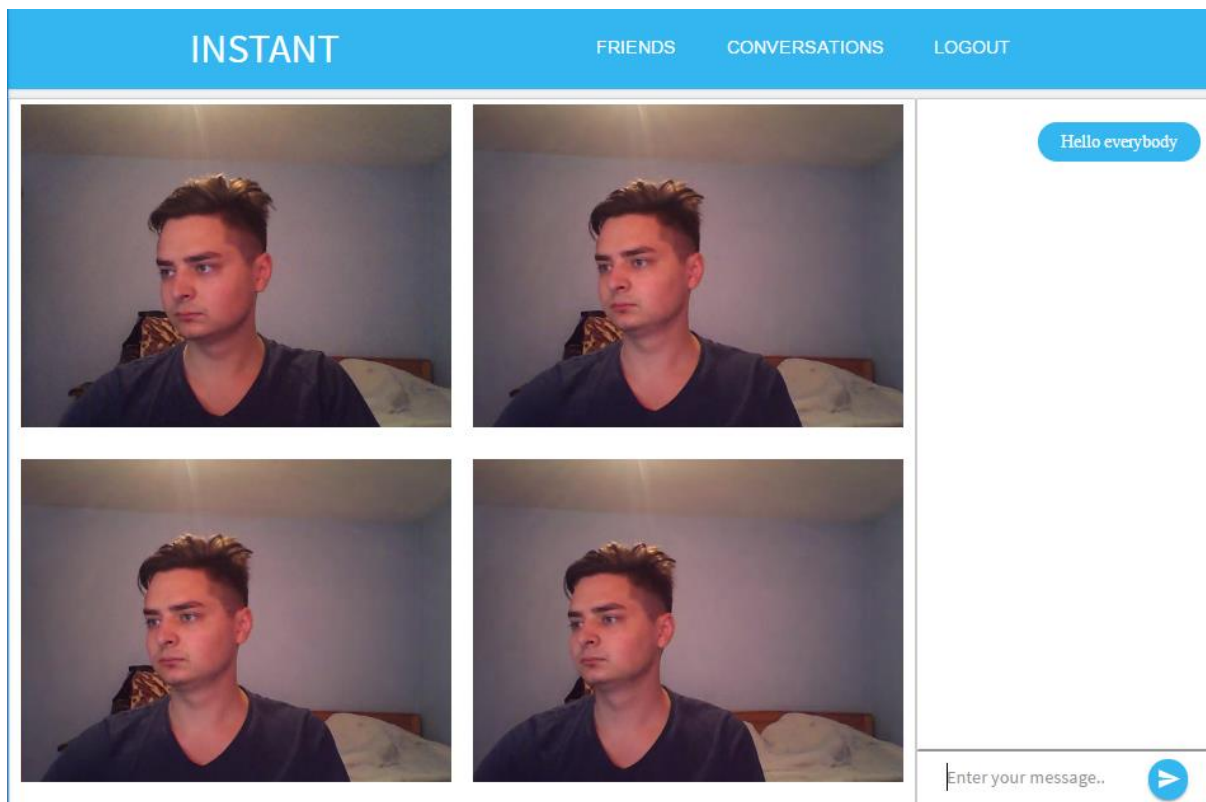


Figura 34: Ecranul de apel video de tip conferință, patru utilizatori diferiți folosesc aceeași cameră web (scop demonstrativ)

3.6 Pagina de apel voce

Această pagină este similară cu cea de apel video, singura diferență fiind faptul că nu se mai distribuie capturi video între utilizatori, ci doar de sunet.

Această modalitate de comunicare este mai eficientă din punct de vedere al vitezei și al consumului de bandă de rețea.

Ecranul diferă de cel al apelului voce prin înlocuirea capturii video a camerei web cu o imagine de profil a utilizatorului (inițial una implicită).

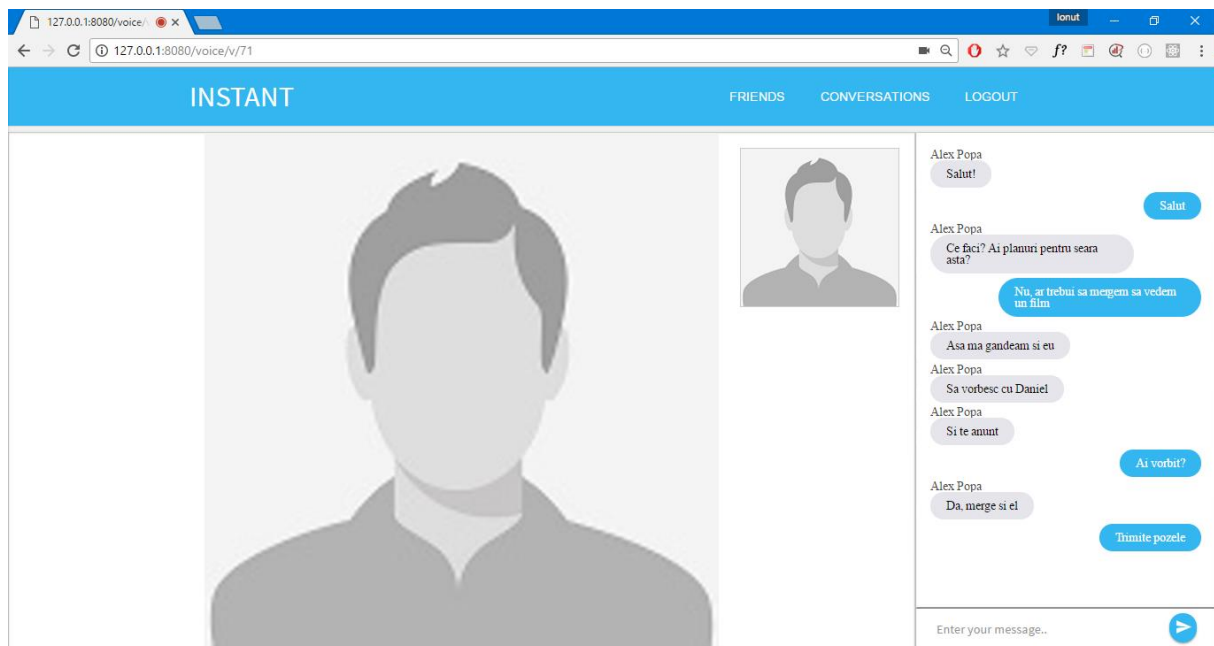


Figura 35: Ecranul de apel voce de tip unu-la-unu

3.7 Pagina de cinema online

În această pagină utilizatorii participanți într-o conversație pot încărca un fișier video pentru a fi vizualizat în mod comun și în același timp având posibilitatea de a comunica între ei fie prin apel video, fie prin schimb de mesaje, fiind simulată experiența de cinema.

Acest ecran este împărțit în două secțiuni mari:

- **Secțiunea principală**, împărțită la rândul ei în două subsecțiuni (doar una din ele fiind afișată la un moment dat)
 - O subsecțiune de Drag-and-Drop în care unul dintre participanți va putea încărca un fișier video
 - O subsecțiune în care se va afișa fișierul video încărcat în subsecțiunea precedentă
- **Secțiunea secundară**, de asemenea împărțită în două subsecțiuni (la fel ca mai sus, doar una din ele fiind afișată la un moment dat)
 - O subsecțiune în care se afișează capturile video ale camerei web ale participanților
 - O subsecțiune care conține conversația cu mesajele schimbate de participanți

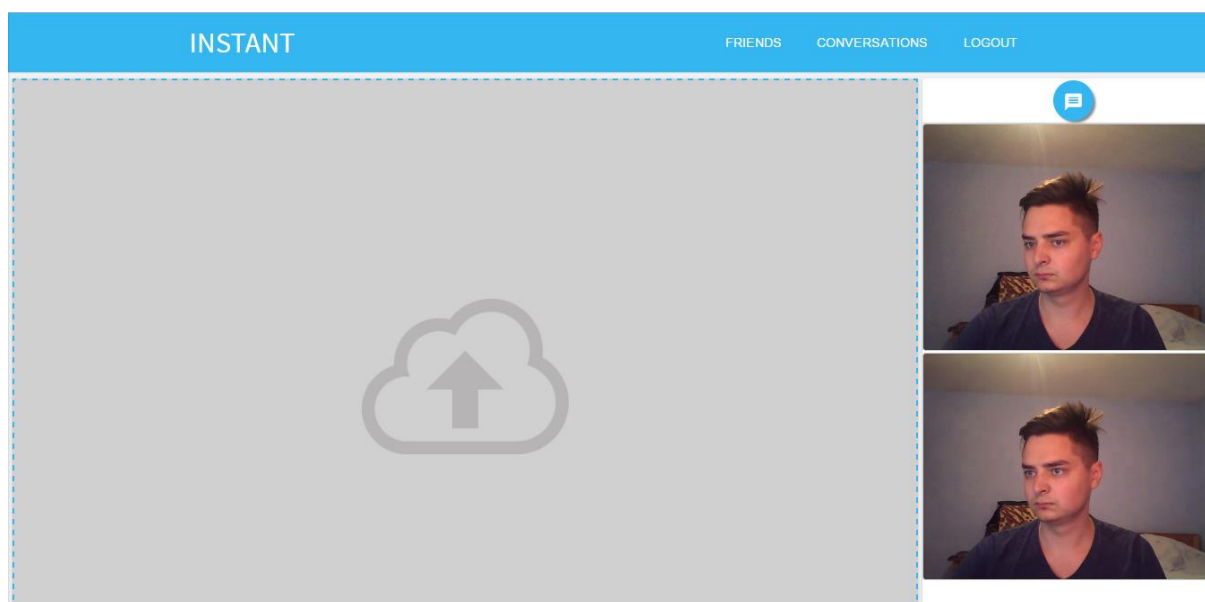


Figura 36: Ecranul de cinema online înainte de a fi încărcat un fișier video

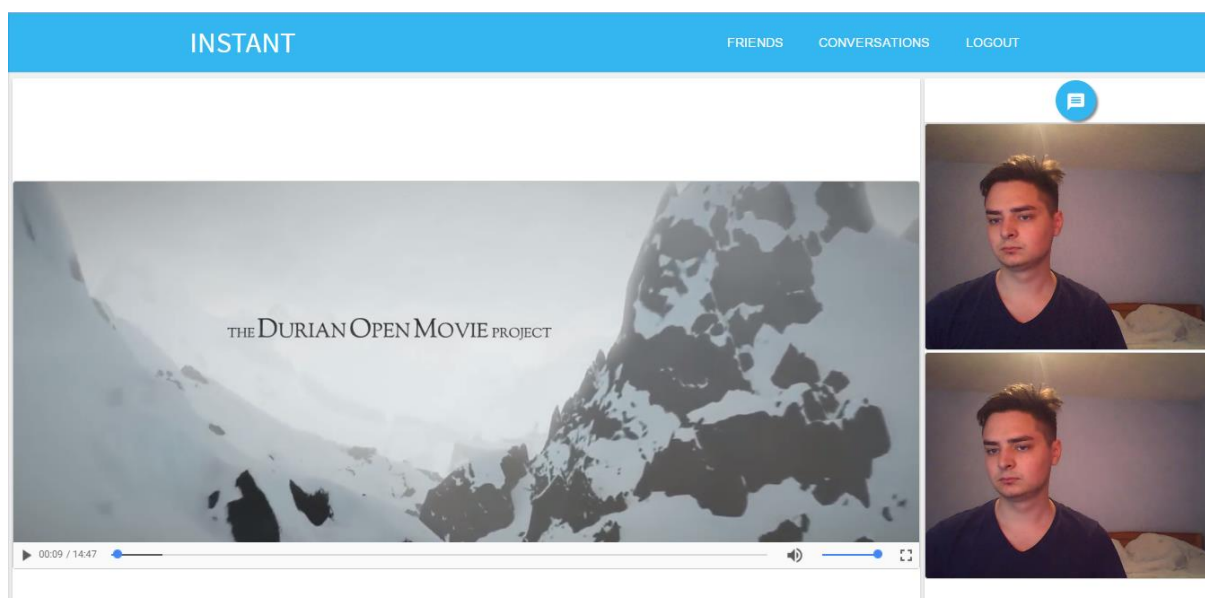


Figura 37: Ecranul de cinema online după ce un participant a încărcat un fișier video

3.8 Pagina cu lista de prieteni

În acest ecran, după cum se poate observa în Figura 38 avem:

- un tab în care putem schimba între pagina cu lista de prieteni sau pagina cu lista de utilizatori din aplicație
- un câmp de intrare de tip căutare în care putem filtra lista de prieteni după nume
- o listă de prieteni ai utilizatorului conectat, afișând imaginea de profil a acestora (una implicită inițial) și numele și prenumele lor

Din această pagină se pot șterge utilizatori din lista de prieteni, dând click pe numele lor și apăsând *unfriend* din modalul apărut.

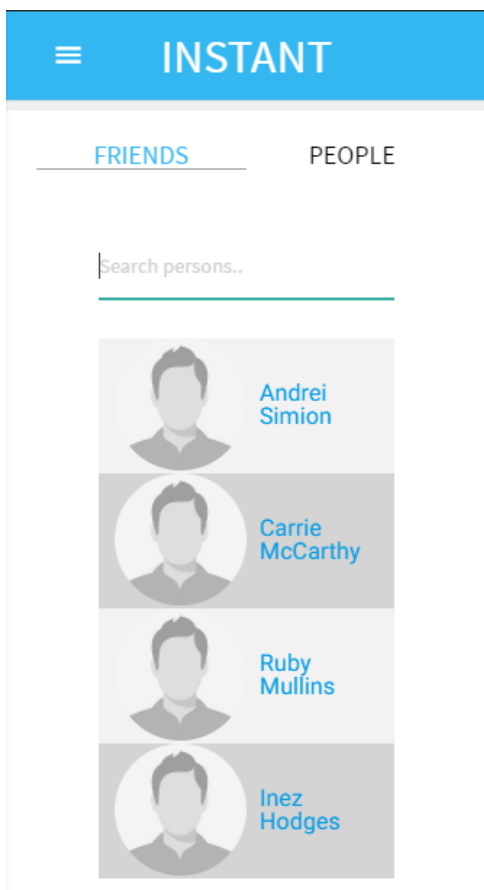


Figura 38: Ecranul cu lista de prieteni

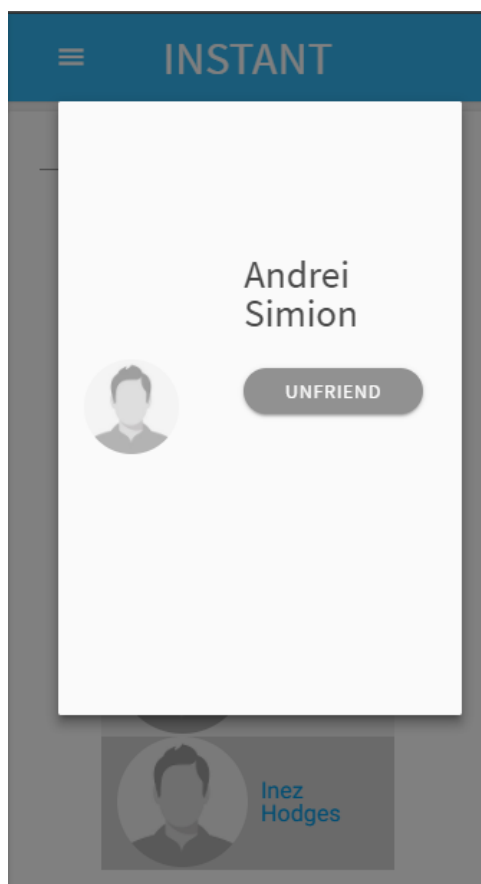


Figura 39: Ecranul ștergere prieten

3.9 Pagina cu lista de utilizatori ai aplicației

Această pagină este similară cu pagina cu lista de prieteni, după cum se poate observa comparând Figurile 38 și 39 cu Figurile 40 și 41.

Se regăsește tab-ul în care putem schimba între cele două pagini, câmpul de intrare de tip căutare și lista, însă de această dată aceasta este compusă din utilizatori ai aplicației, fie prieteni ai utilizatorului conectat, fie alții.

Din această pagină se pot adăuga noi prieteni din lista de utilizatori, apăsând click pe numele lor și dând click pe butonul *Add friend/Add* din modalul afișat.

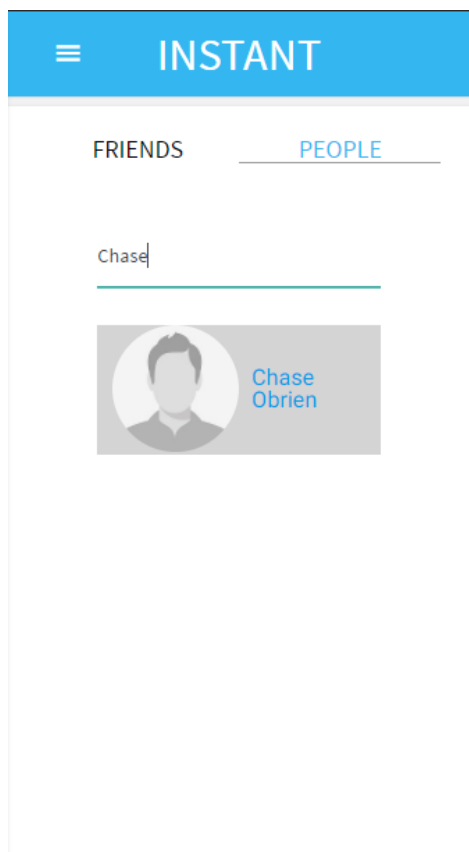


Figura 40: Ecranul cu lista de utilizatori filtrată

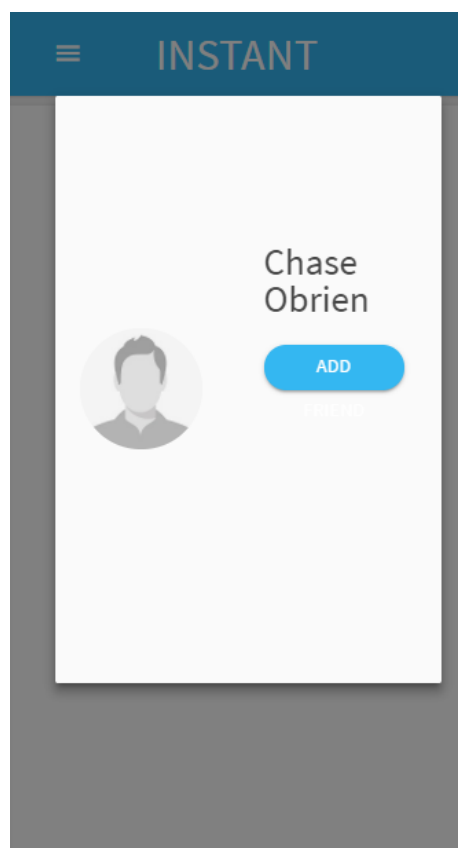


Figura 41: Ecranul adaugă prieten

Bibliografie

Socket.io

1. <https://www.pubnub.com/blog/2015-01-05-websockets-vs-rest-api-understanding-the-difference/>

WebRTC

2. <https://webrtc.org/>
3. <https://en.wikipedia.org/wiki/Peer-to-peer>
4. Real-Time Communication with WebRTC de Salvatore Loreto și Simon Pietro Romano găsită la adresa <http://shop.oreilly.com/product/0636920030911.do>
5. https://en.wikipedia.org/wiki/Session_Initiation_Protocol
6. <https://www.html5rocks.com/en/tutorials/webrtc/basics>
7. https://en.wikipedia.org/wiki/NAT_traversal
8. <https://en.wikipedia.org/wiki/STUN>
9. https://en.wikipedia.org/wiki/Traversal_Using_Relays_around_NAT

Kurento Media Server

10. <http://doc-kurento.readthedocs.io/en/stable/>
11. <http://webrtcchacks.com>

WebTorrent

12. <https://webtorrent.io/>
13. https://en.wikipedia.org/wiki/Glossary_of_BitTorrent_terms
14. O prezentare a fondatorului Feross Aboukhadijeh:
<https://www.youtube.com/watch?v=kxHRATfwnlw>