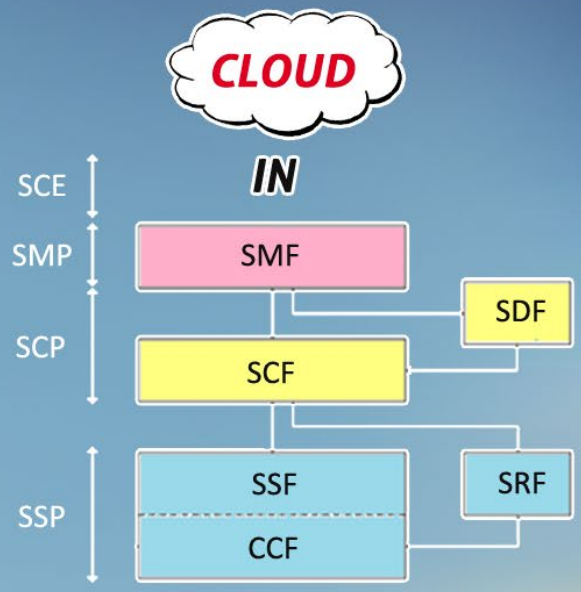
REŢELE INTELIGENTE DE TELECOMUNICAŢII



Dan Nicolae ROBU
Cosmin COSTACHE
- editori -

EDITURA UNIVERSITĂŢII TRANSILVANIA DIN BRAŞOV

D.N.Robu **C.Costache**

- editori -

REŢELE INTELIGENTE DE TELECOMUNICAŢII

Editura TRANSILVANIA

ISBN: 978-606-19-0820-2

Braşov - 2016

Cuprins

Pag.

1 Cuvânt înainte - *D.Robu, C.Costache*

4 Contextul Reţelelor Inteligente de Telecomunicaţii - *V.Cazacu, D.Robu, T.Bălan*

59 Modelul de Procesare a Apelurilor - *S.Bâlbâe, F.Sandu, T.Bălan*

85 Modelul de Controler al Reţelei Inteligente - *I.Rîpea, F.Sandu*

99 Servicii Tipice în Reţelele Inteligente - Implementare - *V.Cazacu, D.Robu*

127 Software pentru Crearea de Servicii - *C.Costache, V.Cazacu, G.Stelea*

135 JAIN - Java for Inteligent Networks - *S.Dovăncescu*

182 Mediul de Creare a Serviciilor pentru Retele Inteligente de Telecomunicatii - *R.David, C.Costache, F.Sandu*

249 Implementarea apelurilor cu extensia IBM Rational pentru SIP - *F.Sandu*

266 Comunicații Cloud - Abordarea orientată pe servicii - *C.Costache, G.Stelea, D.Robu*

314 WebRTC - Comunicații prin web în timp real - *G.Stelea, Ş.Şurariu, T.Bălan*

364 Managementul Business al Comunicațiilor - *A.Balica, F.Sandu*

435 Bibliografie

447 Lista abrevierilor

WebRTC - Comunicații prin web în timp real

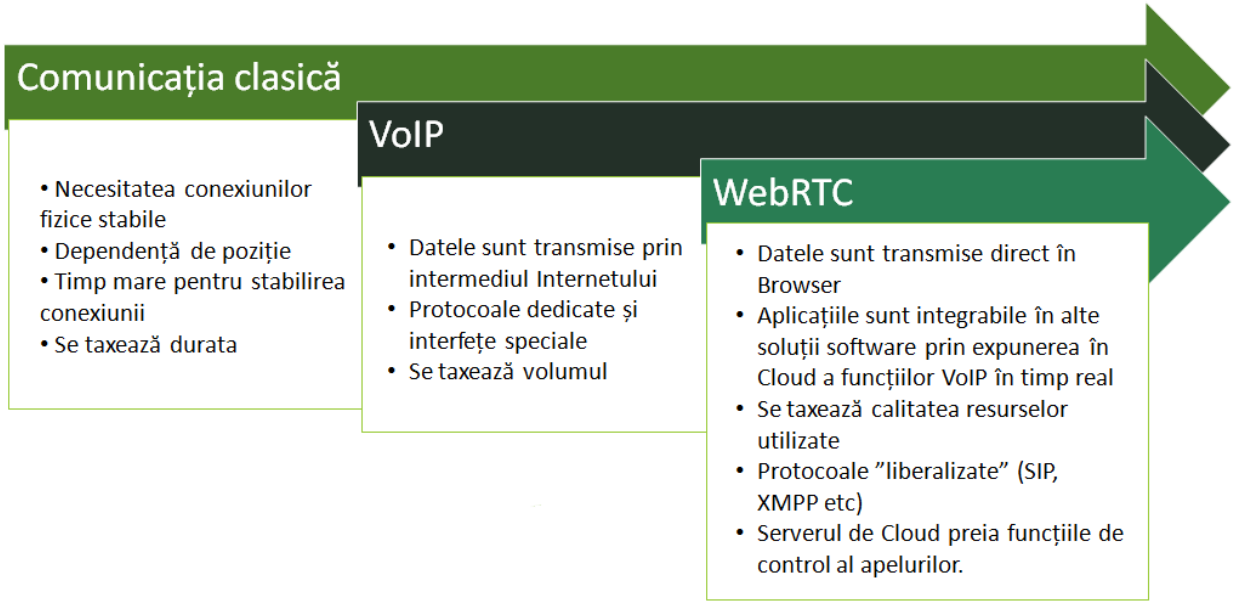
G.Stelea, Ş.Şurariu, T.Bălan

WebRTC prescurtarea pentru Web Real-Time Communications reprezintă un nou standard definit de W3C (World Wide Web Consortium) și IETF (Internet Engineering Task Force) pentru a permite browserelor (navigatoarelor) web să efectueze comunicații peer-to-peer în timp real.

Cu WebRTC (tehnologie care nu necesita instalarea de aplicatii sau module), HTML5 și JavaScript, pot fi folosite pentru a obține acces la camera web sau microfonul unui calculator și a trimite conținut data/audio/videode la un browser la altul.

WebRTC permite realizarea de comunicații peer-to-peer (punct la punct) sau peer-to-multipoint (punct-la-multipunct) fără plugin-uri, deoarece tehnologia este integrata nativ în browser, un serverweb fiind necesar doar pentru a face schimb de date și informații de control și între dispozitive.

Spre deosebire de comunicatiile clasice si VoIP unde datele sunt transmise prin intermediul Internetului sau prin conexiuni fizice stabile, in WebRTC datale sunt transmise direct in browser (navigator web) nemaifiind necesara instalarea de programe software dedicate, permitandu-se folosirea de protocoale "liberalizate" si integrarea aplicatiilor in alte solutii software prin expunerea in Cloud a functiilor VoIP in timp real, taxarea serviciilor facandu-se in functie de calitatea resurselor utilizate si nu a duratei conexiunii sau volumului de date transmise.

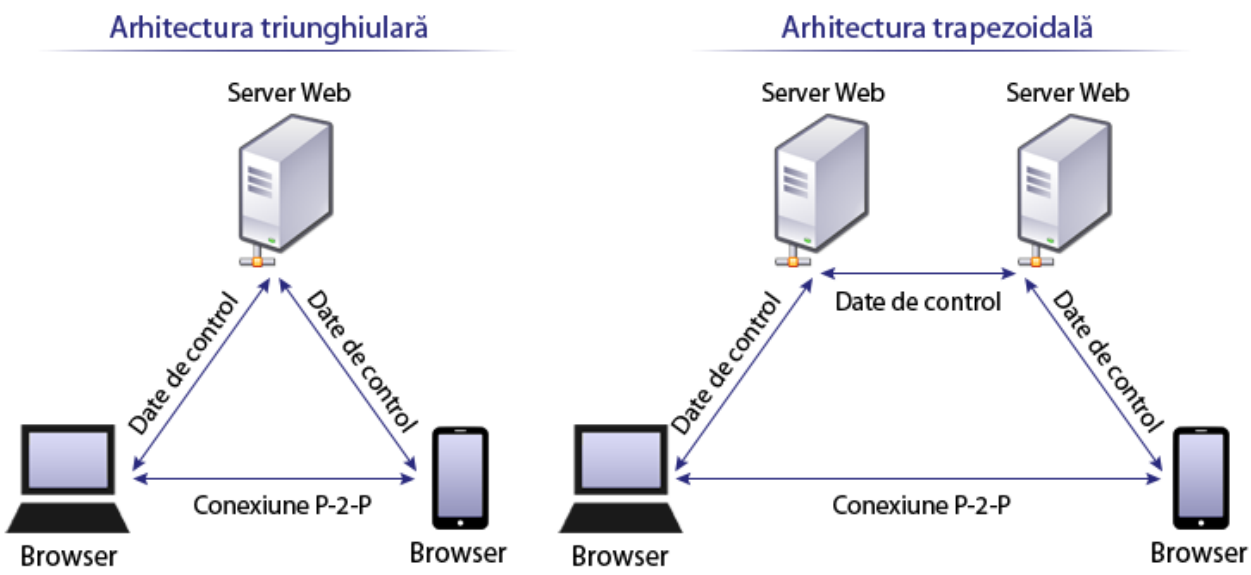


Comparatie *Comunicatia clasica*, *VoIP* si *WebRTC*

1. Arhitectura WebRTC

Arhitecturile web tradiționale sunt construite pe modelul client-server, caz în care comunicarea se face unidirecțional, de la server la client prin protocolul HTTP. Arhitectura WebRTC îmbunătățește modelul client-server prin adăugarea conceptului de comunicare peer-to-peer între browsere web.

În WebRTC cel mai comun model este arhitectura triunghiulară (sau in unele cazuri arhitectura trapezoidală) in care browsereacceseazao aplicatie pe un server prin care mesajele de semnalizare sunt transmise via HTTP sau WebSocket . Serverul este utilizat doar pentru a realiza controlul de date, apoi fluxul de date este efectuat peer-to-peer între browsere.



(a) (b)

Arhitectura WebRTC

2. Semnalizarea in WebRTC

În telecomunicații,semnalizarea are următoarele semnificații:

- utilizarea semnalelor pentru controlul comunicațiilor;
- schimbul de informații privind stabilirea și controlul unui circuit de telecomunicații și de gestionare a rețelei;
- trimiterea unui semnal de la capătul de emisie al unui circuit de telecomunicații pentru a informa un utilizator la capătul de recepțiecă un mesaj trebuie trimis.

Ideea generală din spatele design-ul WebRTC a fost aceea de a specifica pe deplin modul de controlmedia, lăsând în același timp partea de semnalizare cât mai mult posibil la nivelul aplicatiei. Raționamentul este faptul că diferite aplicații pot prefera să utilizeze diferite protocoale de semnalizarestandardizate (de exemplu, SIP sau eXtensible Messaging și Protocolul de prezență [XMPP])sau chiar unul personalizat.
Descrierea sesiunii reprezinta cea mai importanta informatie care trebuie schimbata intre browsere. Aceasta specifica informatiile de transport, tipul si formatul elementelor cat si configuratia parametrilor necesari penru a stabili calea media.
Initial s-a dorit ca schimbul de descriere a sesiunii sa se realizeze printr-un "blob" SDP (Session Description Portocol), insa aceasta metoda a prezentat o serie de deficiente.

dintre care unele s-au dovedit a fi foarte greu de abordat, IETF standardizeaza in momentul de fata protocolul JSEP (JavaScript Session Establishment Protocol) , care oferta interfata necesara pentru a gestiona descrierea sesiunii. JSEP deleaga in totalitate aplicatiei responsabilitatea semnalizarii prin API-uri si conversii ale descrierii sesiunii, in loc de a transmite direct mesajele emise de browser.

3. API-ul WebRTC

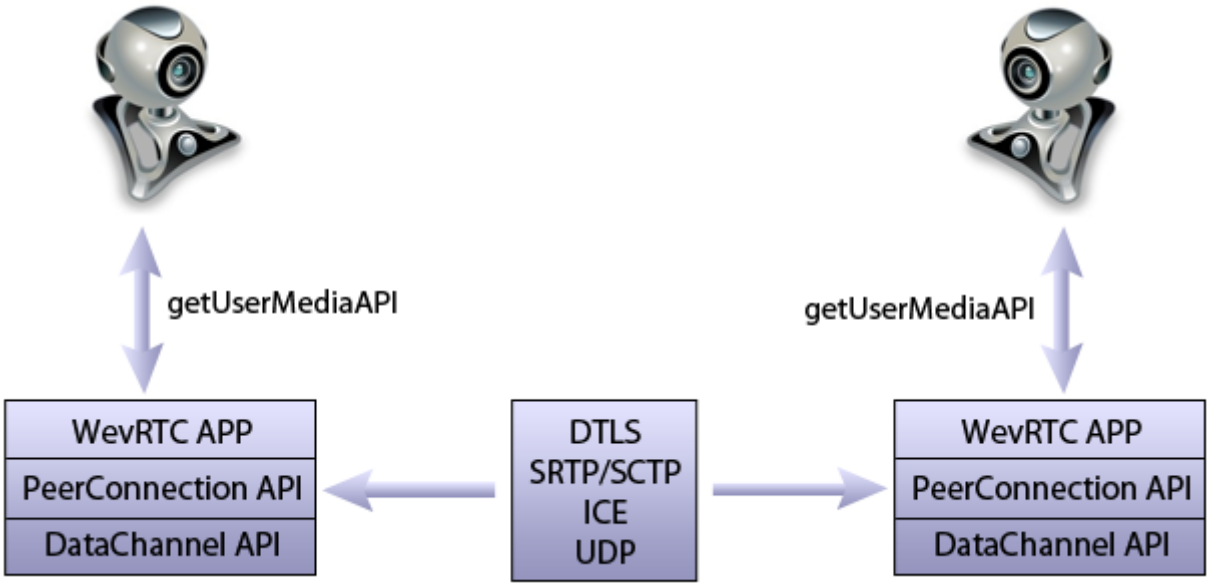
O aplicațieWebRTC (de obicei scrisa ca un mix de HTML și JavaScript) interacționează cu browsere web prin intermediul API-ul WebRTC standardizat, care permite exploatarea si controlul brwoserului corespunzator functiilor de comunicare in timp real.

O aplicatie WebRTC interacționează cu browser-ul, folosind atât API-ul WebRTC dar si alte API-uri standardizate, atât în mod proactiv (de exemplu, pentru a interoga capabilitățile browser-ului) cat și reactiv (de exemplu, pentru a primi notoficarile generate de browser). Prin urmare, API-ul WebRTC trebuie să furnizeze un set larg de funcții, cum ar fi mangementul conexiunii, codare / decodare, controlul fluxului media, firewall, NAT (Network address translation), stabilirea canalelor audio, video si de date. Toate fluxurile media și datele sunt criptate folosind protocolul DTLS.

Protocolul DTLS (Datagram Transport Layer Security) oferă securitatea comunicației pentru protocoalele de tip datagramă. DTLS permite aplicațiilor bazate pe datagrame să comunice într-un mod care este proiectatpentru a preveni interceptarea, alterarea sau falsificarea mesajelor.

Protocolul DTLS se bazează pe protocolul Transport Layer Security (TLS) și este destinat să ofere garanții de securitate similare.

API-ul WebRTC este construit în jurul a trei concepte principale: *MediaStream*, *PeerConnection* și *DataChannel*.



Stiva API-ului WebRTC

3.1 MediaStream

Un*MediaStream* este o reprezentare abstractă a unui flux real de date audioși / sau video. Acesta se foloseste pentru a manipula gestionarea acțiunilor pe fluxul media, cum ar fi afișarea conținutului fluxului, înregistrareasau trimiterea acestuia la un peer la distanță. O reprezentare MediaStream poate fi extinsa pentru a reprezenta fie un flux care provine de la un nod de comunicatie aflat la distanta sau un flux care este trimis catre un nod aflat la distanta (remote node) .

O reprezentare*LocalMediaStream* reprezintă un flux media de pe un dispozitiv de captare media locală (de exemplu, webcam, microfon, etc). Pentru a crea și folosi un flux local, aplicația web trebuie să solicite acces de la utilizator prin intermediul funcției *getUserMedia()*. Aplicația specifică tipul de suport audio sau video la care are nevoie de acces. Dispozitiviul de selectie din interfața browser-ul servește ca mecanism de acordare sau refuzarea accesului. Cand nu mai este nevoie de fluxul media, aplicatiaisi poate revoca propriulacces prin apelarea funcției stop() din*LocalMediaStream*.

3.2 PeerConnection

PeerConnection permite ca doi utilizatori să comunice direct, de la un browser la alt browser. Apoi, reprezintă o asociere cu un nod aflat la distanta (remote peer), care este,

de obicei, o altă instanță a aceleiași aplicații JavaScript care rulează pe nodul repectiv (remote). Comunicațiile suntcoordonate printr-un canal de semnalizare furnizatde codul de scripting (JavaScript) în pagina web intermediul serverului, de exemplu, folosind XMLHttpRequest sau WebSocket. Odată ce o conexiunea*peer-to-peer* este stabilita, fluxurile media pot fi trimise direct la browser-ul remote (aflat la distanta).

Mecanismul *PeerConnection* utilizează protocolul ICE (Interactive Connectivity Establishment), împreună cu servere STUN (Session Traversal Utilities for NAT) si TURN (Traversal Using Relays around NAT) pentru a permite fluxurilor media bazate pe UDP (User Datagram Protocol - utilizează un model simplu de transmisie fără conexiune)sa traversezedispozitive NAT (Network address translation) și firewall-uri.

3.2.1 Protocolalele TURNsi STUN

TURN (Traversal Using Relays around NAT) este un protocol folosit la translatareaadreselor de reteasi a firewall-urilor pentru aplicații multimedia. Acesta poate fi utilizat cu protocolul de control al transmisiei (TCP - Transmission Control Protocol) și User Datagram Protocol (UDP). Este folosit in mod special pentru utilizatorii de pe rețelele cu dispozitive NAT simetrice si oferta conexiunea unui utilizator in spatele unui NAT catre un singur nod(peer).

STUN (Session Traversal Utilities for NAT) este un protocol de rețea cu un set standardizat de metode folosit pentru a permite unei gazdă intr-o reteasa descopere adresa IP publică în cazul în care se află în spatele unui NAT. Este folosit pentru a permite traversarea NAT pentru aplicațiile de voce în timp real, video, mesagerie, precum și alte comunicații IP interactive.STUN este construit să fie un instrument pentru a fi utilizat de către alte protocoale, cum ar fi ICE.

Tehnologia ICEpermite browserelor să descopere suficiente informații despre topologia rețelei în care acestea sunt utilizate pentru a găsi cea mai bună cale de comunicare. ICE oferă, de asemenea, o măsură de securitate, deoarece împiedică paginile web și aplicațiile nesigure sa trimita date către gazde, care nu se așteaptă să le primească.

3.3 DataChannel

API-ul DataChannel este proiectat pentru a oferi un serviciu generic de transportcare permite browserelor web sa a faca schimb de date generice într-un mod bidirecțional de tip peer-to-peer.

DataChannelsuporta un set flexibil de tipuri de date. API-ul este proiectat pentru a imita exact conceptul de WebSocket, suportă siruri de caractere precum și tipuri binare de JavaScript, cum ar fi Blob, ArrayBuffer și ArrayBufferView. Aceste tipuri pot fi de ajutor atunci când se lucrează cu un transfer de fișiere sau aplicatii multiuser.

DataChannel poate lucra atat in mod UDP (comunicare nefiabila(*unreliable*) care nu este bazata pe conexiune) cat si TCP (comunicare fiabila (*reliable*) bazata pe conexiune) Cele două moduri se deosebesc prin:

- Modul fiabil garantează transmiterea mesajelor, precum și ordinea în care acestea sunt livrate. Acest mod necesita informatii adiacente incapuslate in pachetele transmise, ceea ce ar putea face acest mod mai lent.
- Modul nefiabi nu garantează ca fiecare mesaj va ajunge în cealaltă parte, nici in ce ordine vor mesajele ajunge. Astfel, se elimină informatiile suplimentare in pachetele transmise, facand acest mod de a lucra mult mai repaid.

Performanță pentru ambele moduri este aproximativ aceeașiatunci când nu există pierderi de pachete,cu toate acestea, în modul fiabil, un pachetpierdut va provoca blocarea altor pachete Se pot utiliza mai multe canale de date în cadrul aceleiași aplicații, fiecare cu propria lui semantica de încredere (*reliable/unreliable*).

Twilio - platformă de ”comunicații ca servicii” în Cloud

Twilio este o familie de soluții Cloud (API oferite pentru terți) cu utilizare in comunicatii.

Twilio permite dezvoltatorilor de software să programeze primirea si trimiterea de apeluri telefonice și mesaje de tip text folosind API-uri de servicii web. Serviciile Twilio sunt accesate prin HTTP și sunt facturate în funcție de utilizare.

Twilio utilizează serviciul Web Amazon (Amazon Web Services) pentru a găzdui infrastructura de telefonie și de a oferi conectivitate între HTTP și rețeaua telefonică

comutată publică (PSTN - Public Switched Telephone Network) prin intermediul API-urilor sale dedicate, folosind un set de principii de design arhitectural pentru protejarea intreruperilor neasteptate.

Servicii de mesagerie

Serviciile de mesagerie (SMS si MMS) Twilio oferă urmatoarele servicii:

- **Trimiterea de mesaje SMS și MMS prin intermediul API-urilor REST:**

Pentru a trimite un mesaj de tip SMS sau MMS, trebuie să se execute un POST de tip HTTP catre Twilio cu numarul de telefon catre care se doreste trimiterea mesajului si continutul mesajului care se doreste a fi trimis. Mesajele SMS trebuie să fie trimise de la numere de telefon ale platformei Twilio care au serviciul SMS preactivat, datorită arhitecturii rețelei SMS-uri la nivel mondial.

- **Primirea și răspunderea la mesajele SMS și MMS**

Atunci când se achizitioneaza un număr de telefon Twilio SMS preactivat, acesta se poate asocia cu o adresă URL. Atunci când cineva trimite un mesaj text la acel număr de telefon, Twilio face o cerere HTTP la URL-ul respectiv, cu continutul mesajului și numărul de telefon al expeditorului. Se poate apoi raspunde la mesajul primit prin returnarea unui mesaj de răspuns în headerul HTTP "response" (raspuns) catre Twilio.

- **SMS tracking (Urmarirea conversatiilor SMS)**

Folosind cookie-uri HTTP standard, se poate partaja starea în mai multe mesaje, permițând tratarea mesajelor separate ca o conversație și stochează date despre conversația în modulul cookie pentru referințe ulterioare.

- **Trimiterea de mesaje SMS în timpul apelurilor telefonice**

Se poate folosi serviciul TwiML (Twilio Markup Language) <Sms> in timpul unei convorbiri active pentru a permite trimiterea de mesaje SMS in timpul unei conversatii telefonice.

Controlul apelurile telefonice utilizând TwiML

Twilio Markup Language (TwiML) definește un simplu set de instrucțiuni pentru a permite aplicatiilor web scrise in limbaje precum PHP / Java / Ruby / C # sau Python sa

controleze apelurile telefonice în timp real. Instructiunile TwiML sunt preluate de Twilio de la serverul unde este gazduita aplicatia de la începutul fiecărui apel telefonic. La fel ca un browser web, de fiecare dată când telefonul conectat la Twilio efectuează o acțiune, cum ar fi apăsarea unei taste, Twilio face o cerere către server cu noile date, apeland un nou set de instrucțiuni TwiML.

De exemplu, atunci când cineva apelează numărul de telefon specificat, Twilio caută URL-ul de tip serviciu de voce al aplicației web, configurat anterior pentru acel număr de telefon și face o cerere la server, furnizând informații cu privire la apelant. Aplicația web poate returna apoi un set de instrucțiuni TwiML prin care se începe un dialog interactiv între Twilio și serverul aplicatiei web.

TwiML este un set de instrucțiuni care poate fi utilizat pentru a transmite platformei Twilio ce să facă atunci când se receptioneaza un apel sau SMS-uri.

Atunci când cineva face un apel sau trimite un SMS la unul dintre numerele Twilio, acesta va verifica adresa URL asociată cu acel număr de telefon și va face o cerere la această adresă. Twilio va citi instrucțiunile TwiML de la adresa URL pentru a determina actiunea dorită: înregistrarea apelului, redarea unui mesaj pentru apelant, solicitarea apelantului să apese cifre pe tastatura etc.

De exemplu, următorul text va returna un scurt mesaj, apoi va înregistra vocea apelantului:

```
<?xml version="1.0" encoding="UTF-8"?>
<Response>
<Say voice="woman">Please leave a message after the tone.</Say>
<Record maxLength="20" />
</Response>
```

Limbajul TwiML este similar cu HTML. Doar un document TwiML este redat apelantului o singura dată, dar mai multe documente pot fi legate împreună pentru a construi aplicatii complexe interactive de voce.

Solicitările catre un număr Twilio de la un număr exterior sunt controlate folosind TwiML în același mod ca și apelurile primite. URL-ul inițial pentru apel este furnizat ca un parametru catre API-ului REST Twilio pentru a iniția apelul.

Se pot utiliza diferite combinații ale comenzilor de mai jos, pentru a crea multiple aplicații interactive de voce și mesagerie:

- **Say** - apelantului ii este citit un text;
- **Play** - apelantului ii este redat un fișier audio;
- **Dial** - se adaugă un nou participant la apel;
- **Record** - se inregistrează vocea apelantului;
- **Gather** - se colectează cifrele tastate de catre apelant;
- **Sms** - se trimite un mesaj SMS în timpul unui apel telefonic;
- **Hangup** - se inchide apelul;
- **Queue** - se adaugă apelantul la o coadă(șir) de apelanți.
- **Redirect** - se redirecționează fluxul de apel la un document TwiML diferit;
- **Pause** - se așteptă înainte de a executarea mai multor instrucțiuni;
- **Reject** - se refuză un apel primit, fără a fi facturat;
- **Message** - se trimite un mesaj de tip răspuns (MMS sau SMS).

Exista posibilitatea de configurare a Twilio pentru a face cererile sale inițiale TwiML prin HTTP GET sau POST. Modificarea fluxurilor ulterioare de apel, cum ar fi colectarea de intrari de la tastatura (*input from the keypad*) folosind `<Gather>`sau înregistrarea mesajelor audio folosind `<Record>`,necesită solicitări suplimentare, care sunt, de asemenea configurabile ca metode GET sau POST prin parametrul "`method`" al fiecarei comenzi:

- **POST** - atunci când Twilio efectuează o cerere HTTP POST catre un server web, metadatele asociate cu apelul (numărul de telefon etc.) sunt trecut in corpul (body) POST ca și cand parametrii au fost introdusi ca parte a unui formular de pe o pagină web. O precizare importantă legata de metoda POST este fptul ca Twilio nu poate face "cache". Dacă se doreste ca Twilio să faca "cache" paginilor statice TwiML, trebuie folosita metoda GET;
- **GET** - Atunci când Twilio efectuează o cerere HTTP GET catre un server web, metadatele asociate cu apelul (numărul de telefon etc.), sunt trecute ca parametri de interogare in URL-ul aplicatiei ca și cand parametrii au fost introdusi ca parte a

unui formular de pe o pagină web. Twilio va procesa directivele standard de cache a protocolului HTTP pentru cererile HTTP GET. Metoda GET este recomandata atunci cand se folosesc fișiere media mari, precum MP3-uri.

API-ul de Mesagerie IP

API-ul REST de Mesagerie IP (IP Messaging REST API) permite controlarea aplicațiilor de mesagerie IP de la server în același mod în care se pot controla de la client. Exista posibilitatea de creare canale, trimitere mesaje și interogare a starii aplicațiilor de mesagerie utilizând resursele API REST.

Toate adresele URL din documentația de referință utilizează următoarea adresă URL de bază:

```
https://ip-messaging.twilio.com/v1
```

Toate solicitările către API-ul REST de Mesagerie IP sunt trimise prin HTTPS (HyperText Transfer Protocol/Secure). Protocolul HTTP necriptat nu este acceptat.

Cererile HTTP către API-ul REST sunt protejate cu autentificare de bază HTTP la fel ca si celelalte API-uri REST oferite de Twilio. Pentru a afla mai multe detalii despre modul în care Twilio se ocupă de autentificare, se poate consulta documentația de securitate de pe link-ul oficial:

```
https://www.twilio.com/docs/api/security
```

Se va utiliza contul SID (Security Identifier) Twilio cu numele de utilizator și token-ul de autentificare ca parola pentru autentificarea HTTP.

```
curl -G https://ip-messaging.twilio.com/v1/Services \
  -u '[YOUR ACCOUNT SID]:[YOUR AUTH TOKEN]'
```

Contul SID si token-ul de autentificare se regasesc în consola de administrare.

În mod alternativ, se pot utiliza chei secrete API pentru a autentifica solicitările REST (perechile de chei API se pot administra, de asemenea si din consolă).

Pentru a interactiona mai facil cu nodurile finale (endpoints) ale API-ului REST de Mesagerie IP se pot folosi bibliotecile Twilio Helper. Schimbul de mesaje IP este disponibil în versiunea stabilă a "Twilio Helper Libraries" (`https://www.twilio.com/docs/libraries`)

In tabelul de mai jos sunt prezentare resursele care interacționează cu API-ul REST:

Resurse	Descriere
`Channels`	O "cameră de chat" (chat room) - un mediu de aplicare în care mesajele pot fi trimise sau primite
`Credentials`	Datele de conectare folosite de Twilio pentru integrarea cu *Google Cloud Messaging* sau *Apple Push Notifications Service*
`Members`	Membrii unui "canal" (Channel)
`Messages`	Mesajele trimise la un "canal" (Channel)
`Roles`	Defineste actiuni pe care utilizatorii au voie sa le folosească într-un domeniu "canal" (Channel) sau "serviciu" (Service)
`Services`	Un serviciu de hărti (maps) al unei aplicații de mesagerie IP (canale, utilizatori, autorizări etc.). Pot defini adrese URL de tip "webhook" (cârlig web) pentru evenimente într-un serviciu
`Users`	Un utilizator cu o identitate unică în sfera serviciului definit

Serviciul NAT Twillio

Serviciul Twilio de translatare a rețelei (Network Traversal) este un serviciu de tip STUN / TURN distribuit la nivel global, care serveste la implementarea de aplicații fiabile de comunicații peer-to-peer. Acest serviciu este dedicat in special pentru aplicațiile WebRTC(Web Real-Time Communication) și VoIP (Voice over Internet Protocol) pentru translatarea si parcurgerea NAT / firewall-uri, astfel încât utilizatorii sa reusească sa se conecteze de fiecare dată indiferent in ce retele se află.

Securitate

Twilio suportă protocolul de criptare TLS pentru a proteja comunicațiile între Twilio și aplicația web. Trebuie doar să se specifice URL-ul HTTPS. (Notă: Twilio nu acceptă momentan certificate auto-semnate(Self-signed certificate).)

Protejarea prin parola a URL-urilor TwiML de pe serverul de web se face prin autentificare de bază HTTP (HTTP Basic and Digest Authentication) și autentificare de

tip "Digest" (Digest Authentication). Acest lucru vă permite protejarea prin parolă a URL-urilor, astfel încât numai proprietarul aplicatiei și Twilio le pot accesa.

Un nume de utilizator și o parola se pot prevedea prin următorul format de adresă URL:

```
https://username:password@www.numeserver.com/documentul_meu
```

Twilio se va autentifica pe serverul web folosind numele de utilizator și parola furnizate și va rămâne autentificat pe toata durata apelului. Se recomandă utilizarea autentificarii HTTP în conjuncție cu criptarea de tip "Digest Authentication".

Dacă se specifică o adresă URL protejată prin parolă, Twilio va trimite mai întâi o cerere fără antet de autorizare. Dupa ce răspunde serverul cu un răspuns neautorizat 401, Twilio va face aceeași cerere cu un antet de autorizare.

4. Implementarea, integrarea si testarea aplicatiei WebRTC pe platforma ”Circuit”

4.1. Modulul de control apeluri

4.1.1. Managementul participanților

1. În primul rând utilizatorii au nevoie de un cont pe site-ul https://www.circuit.com/web/developers pentru a putea utiliza platforma Circuit. În spatele acestei plaforme se află o bază de date care stochează acești utilizatori și este apelată prin funcția de mai jos:

```
app.use(express.logger('dev')); //verificare prin logare
```

În NodeJS, toate modulele exportă un obiect care poate fi apelat cu ușurință oriunde în cod, iar pentru logare am folosit o instanță a modulului *express*, numită app.
Express reprezintă un modul de dezvoltare web rapid care furnizează diferite funcții pentruvizualizare, testare, rutare, negociere a conținutului într-o pagină web, dar și un executabil pentru generare de aplicații.

```
var express = require('express') //generare proiect NodeJS Express
      , user = require('./routes/user') //user.js (users list)
```

Astfel pentru a putea folosi funcțiile de tip middleware, trebuie creată variabila *express* pentru a îngloba toate funcțiile necesare.

2. După apelarea funcției logger am folosit funcția app.get pentru a prelua utilizatorii din baza de date, prin subprogramul *user.js*.

Funcția **app.get** fiind structurată sub forma (rută, răspuns), se folosește pentru rutarea către puncte finale (URIs) și răspunsul la cererea clientului prin metode GET HTTP. În cazul acesta, funcția trimite către directorul *users*, cerând ca răspuns lista utilizatorilor.

```
app.get('/users', user.list); //din subprogramul user.js de tip
”cerere = req” si ”raspuns = res”
```

Subprogramul *user.js* cuprinde doar două linii de cod care exportă lista de pe platforma Circuit și o trimite ca răspuns-resursă pentru app.get.

```
exports.list = function(req, res){
res.send("respond with a resource");};
```

Subprogramul care reprezintă motorul soluției software face legătura între subprogramele ce compun aplicația web și creează o serie de variabile Java Script pe care le leagă în diferite pachete, dependințe, funcționalități Node sau rute, în momentul în care se începe un proiect nou Express Node JS.

3. Funcția de acces în aplicație prin logare și respectiv prin delogare se găsește în programul principal numit *client.js*.

Funcția de logare compară mai întâi emailul și parola cu baza de date de pe platforma Circuit, după care creează o instanță *Client*și prin metoda evenimentelor acordă acces utilizatorilor în aplicație.

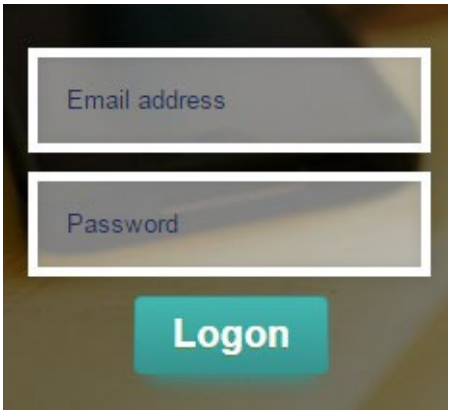
```
function logon() { //functie logare
        if (_client || !$email.value || !$password.value) {//verifica email si parola
        return;
         }
        _client = new Circuit.Client({domain: $domain.value}); //creare instanta
        client folosind modulul client
        addClientEventListeners();
```

Dacă datele de logare sunt corecte, se intră în aplicație, prima operație fiind cea de a afișa starea conexiunii utilizatorului în colțul din dreapta sus și a numelui utilizatorului.

```
_client.logon($email.value, $password.value).then(function (user){
_localUser = user;
$email.disabled = true; //camp completat implicit
$password.disabled = true; //camp completat implicit
```

```
$login_wall.style.display = 'none';
$content.style.display = '';
$logonButton.style.display = 'none';
document.getElementById('name').innerHTML = user.displayName; //preia nume
document.getElementById('right_side').style.display = ''; //afiseaza nume
})
```

Stare conexiune utilizator



Câmpurile pentru email și parolă completate implicit

Cea de a doua operație, după conectare este preluarea conversațiilor deja avute de utilizator:

```
.then(_client.getConversations) //preia conversatia in fct de utilizator
.then(populateDropdown) //populeaza lista cu conversatii
.then(_client.getCalls) //acces in conversatie
```

Pentru excepții am utilizat funcția *catch* care aici se referă la cazul în care logarea nu are loc cu succes, parola sau utilizatorul nu sunt corecte.

```
.catch(function (e) {   //cazul in care logarea nu a avut loc cu success
console.error(e);
alert('Unable to logon. Error: ' + e);//alerta de eroare
_client.removeAllListeners();
_client = null;
});
```

Funcția de ieșire din aplicație (delogare)dezactivează modulul *Client*, ieșind din aplicație și revenind la pagina inițială. De asemenea se afișează în colțul din dreapta sus starea conexiunii care va fi: ”Deconectat”.

```
function logout() { //iesire din aplicatie
        if (!_client) {
        return;
         }
        _client.logout().then(_client.removeAllListeners);
        _client = null; //modul client dezactivat
        _localUser = null;
        _conversations = {};
        _calls = {};
        $convList.options.length = 0;
        $email.disabled = false;//campurile implicite dezactivate
        $password.disabled = false;
        $logonButton.style.display = '';
        document.getElementById('right_side').style.display = 'none';
        $logonState.textContent = 'Deconectat'; //status conexiune
        location.reload();
}
```

Funcția care schimbă starea conexiunii în aplicație se face prin metoda addEventListeners() care atașează un eveniment către elementul *Client*, fără a suprascrie alte evenimente deja create.

```
function addClientEventListeners() { //adaugarea unui client in aplicatie
         if (_client) {
        _client.addEventListener('connectionStateChanged',
             function onConnectionStateChanged(evt) { //schimbare status conexiune
        prin crearea unui eveniment
        console.log('Received connectionStateChanged event - state = ',
         evt.state)
             $logonState.textContent = evt.state; //adaugarea textului "conectat" sau
        "deconectat"
        if (evt.state === Circuit.Enums.ConnectionState.Disconnected) { //in caz ca
        este deconectat se revine la interfata initiala
        resetCallUI(); //reseteaza interfata
        }
         });
```

4.1.2. Managementul conexiunilor

1. Server virtual. Port.

Pentru realizarea conexiunii între cele două entități, în primul rând am setat datele de nivel de legătură: cale, respectiv port în subprogramul *app.js*.
Se creează mai întâi variabilele prin modulul *express*:

```
var express = require('express') //generare proiect NodeJS Express
, routes = require('./routes') //cale director routes
, path = require('path'); //calea catre pagina web
```

Apoi se apelează prin *app.set*.
Funcția **app.set** (nume, valoare) setează numele și valoarea portului local pe care se rulează aplicația (`'port'`), un director sau mai multe, care în acest exemplu reprezintă directorul cu elementele de interfață și calea către acesta (`'views', __dirname + '/views'`) care aduce toate elemntele componente ale proiectului la nivel de prezentare pentru client.

```
app.set('port', process.env.PORT || 8080); //port local
```

Funcția **app.use** folosește funcția middleware care trimite către o destinație (`express.static(path.join(__dirname, 'public'))`). Rutarea se va face către orice cale care conține ”/”, iar funcția middleware se execută imediat ce are loc această portivire, dar secvențial, fapt pentru care ordinea este importantă.

```
app.use(app.router); //apelarea funcției de rutare
app.get('/', routes.index); //preia rutele din index.js
```

Subprogramul **index.js** conține funcția de tip cerere-răspuns care care exportă pagina de web care include aplicația.

```
exports.index = function(req, res){
res.render('index', { title: 'DEComunicator' });
};
```

Crearea serverului virtual care este gazda aplicației, în cazul de față https://localhost:8080 se face prin funcția de mai jos:

```
var httpsServer = https.createServer(credentials, app); //creare server virtual
httpsServer.listen(app.get('port'));
```

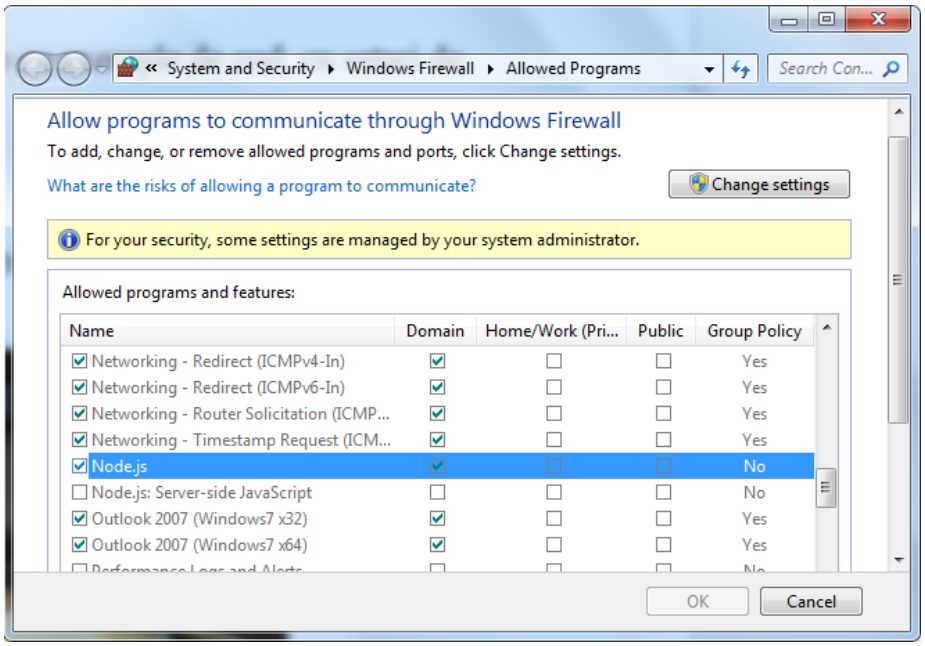
Un alt aspect al conexiunii este adăugarea etichetei <script> în subprogramul care conține elementele de interfață *index.ejs*. Etichetele trimit fie spre un link care conține un document de tip *script*, fie spre un document local, respectiv *client.js*.

```
<script src='https://circuitsandbox.net/circuit.js'></script>
<script src='js/client.js'></script>
```

Acestea conectează cu interfața *index.ejs* toate subprogramele și funcționalitățile din programul principal numit *client.js.*

2. Aspecte de securitate

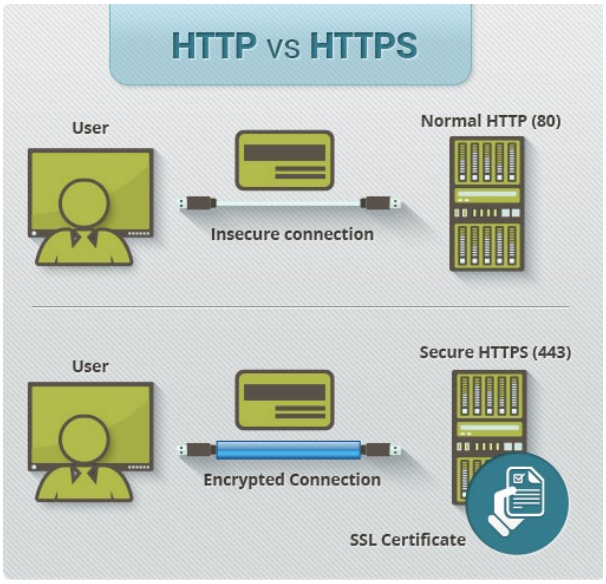
Pentru a putea permite mediului NodeJS să pătrundă în sistemul meu, ca dezvoltator al aplicației, a fost nevoie să adăug în lista programelor care pot trece prin Firewall și această platformă.



Accesul lui NodeJS în sistem

Odată setată această opțiune se pune problema securității de care se ocupă programul OpenSSL și cheile generate cu ajutorul acestuia.

Adresa de *localhost* va fi HTTPS și nu HTTP, datorită faptului că am folosit programul de securitate OpenSSL prin care am generat chei și certificate de securitate (vezi 3.2.6 OpenSSL). HTTPS reprezintă versiunea securizată pentru Hyper Text Transfer, protocol prin care datele se transmit de la browser-ul local la browser-ul celui cu care se comunică prin aplicație. Această ”linie” de comunicație este astfel criptată și ferită de interceptori nedoriți.



HTTP versus HTTPS
(https://www.instantssl.com/ssl-certificate-products/https.html)

Cheia și certificatul generate sunt adăugate în subprogramul *app.js*.

```
var privateKey  = fs.readFileSync('C:/OpenSSL-Win32/bin/unitBV-key.pem', 'utf8');
var certificate = fs.readFileSync('C:/OpenSSL-Win32/bin/unitBV-cert.pem', 'utf8');
```

4.1.3. Managementul fluxurilor

1. Elemente de control

Funcția de începere a unei conferințe verifică mai întâi dacă utilizatorul este logat și apoi creează variabile pentru timpurile de media:

```
function start() {
        if (!_client || !_localUser) { return alert('Caller is not logged in');}
         //cazul in care nu este logat
        var mediaType = {audio: true, video: enableVideo.checked}; //variabila
        audio, video doar ca optiune
```

Pasul următor pe care îl face utilizatorul este de a selecta o conversație:

```
         var selectedConvId = $convList.options[$convList.selectedIndex].value;
        //selecteaza numele conversatiei
         document.getElementById('conference_options').style.display = ''; //preia
        conversatie din lista
```

Pornește streaming media:

```
         _client.startConference(selectedConvId, mediaType) //porneste media
           .then(setCallUI) //reseteaza interfata pentru apel
```

```
    .catch(function (err) { //exceptia in caz de eroare
     alert('Error starting conference. ' + err);
     console.error(err);
    });
};
```

Funcția de participare la o conferință deja începută verifică de asemenea dacă utilizatorul este logat, crează variabilele pentru tipurile media și îi permite acestuia să aleagă o conversație. Pentru a fi participant la conversație, aceasta trebuie să aibă *starea* "**started**". Astfel se găsește în plus funcția join() cu variabila call.

```
function join() { //participa la conferinta
    if (!_client || !_localUser) { ...
    ...
    var call = _calls[selectedConvId]; //selecteaza conversatia
     document.getElementById('conference_options').style.display = '';
     _client.joinConference(call.callId, mediaType)
```

Funcția de ieșire dintr-o conferință are același elemente ca funcția de participare, dar folosește modulul leave() și în funcție de ID-ul conversației în care se află utilizatorul, acesta este scos din conferință.

```
 function leave () { //functia de iesire din conferinta
    if (!_client || !_localUser) { ...
    ...
     _client.leaveConference(call.callId); //scoaterea clientului din conferinta
    };
```

Am creat de asemenea un eveniment (`'callEnded'`) care să marcheze faptul că unul din utilizatori iese din conferință.

```
_client.addEventListener('callEnded', function (evt) { //terminare apel
var call = _calls[evt.call.convId]; //variabila call care preia ID-ul
conversatiei
if (call && call.callId === evt.call.callId) {
  console.log('Received callEnded event. state: ' + evt.call.state,evt.call);
//schimbare status conexiune
delete _calls[evt.call.convId]; //sterge client din conversatie
updateList(evt.call); //actualizeaza lista de conversatii
  resetCallUI();} });//actualizeaza interfata
```

Selectarea unei conferințe din listă se face cu ajutorul unui buton, fiecare conferință primind un ID. După selectarea conferinței se resetează interfața cu utilizatorul.

```
function onConversationSelected() { //selecteaza conversatie
        var convId = document.getElementById('convList').value; //fiecare
        conversatie are un ID
        var call = _calls[convId]; //call este asociat cu conversatia
         if(convId == "--Select conversation--") //selecteaza conversatia
        {
                 document.getElementById('buttons_area').style.display = 'none';
               }
        else{
        document.getElementById('buttons_area').style.display = '';
        //preia cu ajutorul butonului
             }
        if (call) {
        setCallUI(); //seteaza interfata pentru apel
        } else {
                 resetCallUI(); //reseteaza interfata
        }}
```

Încărcarea ID-urilor conversațiilor în listă folosește funcția populateDropdown(conversations) care preia cu ajutorul modulului Circuit.Enums numele conversațiilor printr-un ID (convId), dar care aici este atribuit unei variabile (c).

```
function populateDropdown(conversations) { //populeaza lista de convorbiri
         _conversations = {};
        conversations.filter(function (c) {
        if (c && c.type === Circuit.Enums.ConversationType.GROUP) {
        //enumera conversatiile din lista celor deja create pe platforma Circuit
               _conversations[c.convId] = c;
               returntrue; //returneaza true daca se gaseste in lista
               }
        returnfalse;
        }).forEach(function (c) {
        if (c.type !== 'DIRECT') {
        $convList.options[$convList.options.length] = new Option(c.title,
         c.convId); //titlul si ID-ul conversatiei
         }});
```

Starea unei conferințe, respectiv ***"Active"*** în cazul în care conversația a început deja, dar utilizatorul nu este angajat în acea conferință sau ***"Started"***, în cazul în care clientul face parte din conferință.
Această funcționalitate este realizată tot prin adăugarea unui`EventListener` (`callStatus`)care este apelat în momentul în care un utilizator se loghează în aplicație.

```
_client.addEventListener('callStatus', function (evt) { //este apelata in
momentul in care se adauga un nou client
var call = _calls[evt.call.convId]; //variabila call care preia ID-ul conv.
if (!call || call.callId === evt.call.callId) { //asculta eveniment
        console.log('Received callStatus event. state: ' + evt.call.state + ',
         reason: ' + evt.reason, evt.call); //receptioneaza starea
        _calls[evt.call.convId] = evt.call; //modifica in functie de eveniment
        updateList(evt.call); //reincarca lista dupa stabilire status
        setCallUI();}//seteaza interfata cu utilizatorul
```

Schimbarea stării unei conferințe în cazul în care se încheie conferința are loc prin funcția updateList(call) după cum urmează:

```
 function updateList(call) { //actualizeaza lista de conversatii
        if (call.state === Circuit.Enums.CallStateName.Terminated) { //daca se
        incheie conferinta
        getListOption(call.convId).textContent =_conversations[call.convId].title;
         } else {
                  getListOption(call.convId).textContent                    =
        _conversations[call.convId].title + '(' + call.state + ')'; } //in cazul in
        care nu s-a terminat afiseaza statusul in continuare
```

2. Fluxul audio și video

În momentul în care se intră într-o conferință se activează automat fluxul audio (vezi **function** start()).

```
var mediaType = {audio: true, video: enableVideo.checked};
```

Fluxului video se activează doar prin bifarea unei căsuțe de opțiuni după cum urmează:

```
 function onVideoChange() { //flux video
        var selectedConvId = $convList.options[$convList.selectedIndex].value;
        //preia conversatia din lista
         var call = _calls[selectedConvId]; //selecteaza conversatia
         if (call && call.localMediaType.video !== enableVideo.checked) {
        //activeaza video daca se bifeaza optiunea
        _client.toggleVideo(call.callId).then(function () { //se deschide fluxul
         Video
        console.log('Local video was successfully toggled');
         }).catch(function (err) { //exceptie in caz de eroare
        console.error('Failed to toggle video. ', err); //eroare video
        })}
```

4.2. Interfața grafică cu utilizatorul (GUI)

4.2.1. Nivelul de prezentare al serviciilor

Am creat o ***funcție numită CallUI()*** pe care am apelat-o de fiecare data când a fost vorba de setarea sau resetarea interfeței cu utilizatorul. În funcția care urmează se modifică butonul prin care se accesează conferința. Butonul are două opțiuni: ***"Începe"*** – care se referă la pornirea unei conferințe, sau ***"Intră"*** – care se referă la intrarea într-o conferință deja începută.

```
function setCallUI() {
    updateButtons(); //text buton in functie de conversatie
     var convId = document.getElementById('convList').value;
    if (!_conversations[convId]) {
         return;
    }
    var call = _calls[convId]; //returenaza IDul conversaiei
    $convId.textContent = call.convId;
     $callState.textContent = call.state; //status conversatie
    $callId.textContent = call.callId;
```

În momentul în care se accesează o conversație se activează automat fluxul audio, iar pentru video trimite la funcția `onVideoChange()`:

```
    call.activeMediaType && ($media.textContent = (call.activeMediaType.audio ?
    'audio' : '') + (call.activeMediaType.video ? '+video' : ''));
    $localVideo.src = call.localVideoUrl || ''; //trimitere catre fct video
    if (call.isEstablished) {    //daca se stabileste o conversatie
    $remoteAudio.src = call.remoteAudioUrl;  //se activeaza audio
    document.getElementById('users').innerHTML =
    call.participants.length+1; //flux video pentru restul participantilor
    for(var i = 0 ; i< call.participants.length ;i++)
        { remoteVideo[i].src = call.participants[i].videoUrl; }
    }
    else {
    $remoteAudio.src = ''; //flux audio
    $remoteVideo1.src = ''; //flux video pentru fiecare utilizator
```

```
$remoteVideo2.src = '';
    ...
    } }
```

Funcția de resetareresetCallUI() a interfeței am apelat-o pentru fiecare modificare făcută de celelalte funcții în timpul accesării aplicației sau conversației.

```
function resetCallUI() {
 updateButtons(); //schimba buton dupa ce intra in conversatie
$callState.textContent = ''; //status buton dupa intrare in conversatie
$media.textContent = '';
$callId.textContent = '';
$convId.textContent = '';
$localVideo.src = ''; //video local
$remoteVideo1.src = ''; //video de la ceilalti utilizatori
$remoteVideo2.src = '';//... pentru 6 utilizatori
$remoteAudio.src = '';//audio de la ceilalti utilizatori
$enableVideo.checked = false; //nu permite flux video
$enableVideo.disabled = true; }//video activ, checkbox inactiv
```

Funcția `updateButtons()` reprezintă butonul care își schimbă textul în funcție de starea conversației. Dacă a început deja conferința va scrie: ”Intră”, iar dacă ultilizatorul începe conferința, va afișa ”Începe” și va ascunde butonul cu descrierea ”Intră”. Dacă nu este începută conferința, atunci butonul cu descrierea ”Ieșire din conferință”.

```
function updateButtons() {
 $startButton.classList.add('hide'); //ascunde butonul ”Incepe”
$joinButton.classList.add('hide'); //ascunde butonul ”Intra”
$leaveButton.classList.add('hide'); //ascunde butonul ”Iesire”
 if (!call) { //daca utilizatorul nu a intrat intr-o conferinta

$startButton.classList.remove('hide'); //ascunde butonul ”Incepe”
} elseif (call.state === Circuit.Enums.CallStateName.Started) { //daca
oconversatie este deja ”Started”
$joinButton.classList.remove('hide'); //ascunde butonul ”Intra” (Join)
} elseif (call.state === Circuit.Enums.CallStateName.ActiveRemote) {
$leaveButton.classList.remove('hide'); //ascunde butonul ”Iesire” daca
utilizatorul nu este in conferinta}}
```

Toate funcțiile menționate mai sus sunt cuprinse în subprogramul ***client.js***. Elementele care alcătuiesc pagina HTTP, respectiv interfața sunt cuprinse în subprogramul ***index.ejs***, ***main.css*** sau ***style.css*** și fac trimitere la cele principale.

Funcția de intrare în aplicațieeste reprezentată în limbaj HTML în felul următor:

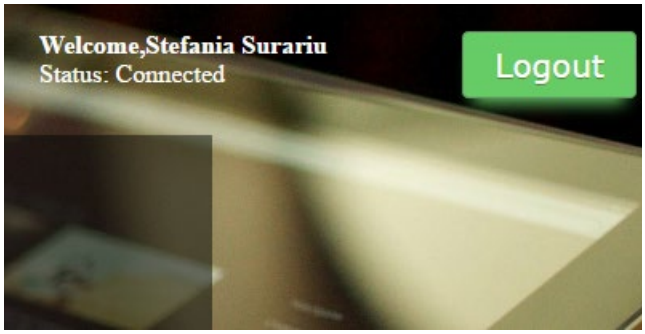
```
<input type="email" id="email" placeholder="Email address" />

<input type="password" id="password" placeholder="Password"/>
<button id="logon" class="button" onclick="logon()" style="">Logon</button>
```

Accesul în aplicație

Ieșirea din aplicație se face cu ajutorul butonului ”Logout” din colțul din dreapta sus a paginii. Această funcție se transpune în interfață prin:

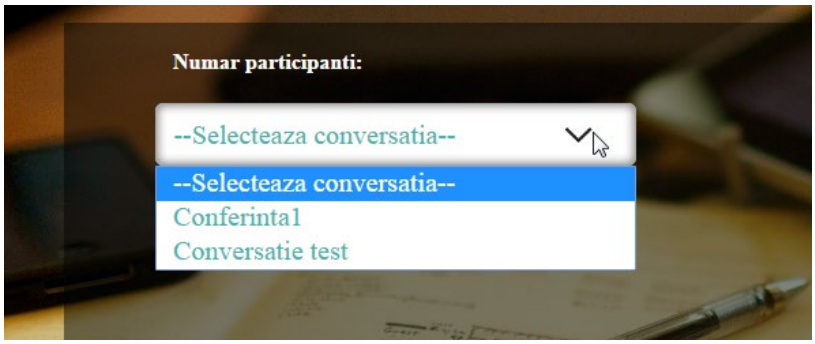
```
<div class="welcome_logout">
        <button id="logout" class="button" onclick="logout()">Logout</button>
</div>
<div class="welcome_logout" style="display:inline-block;">
```



Ieşirea din aplicație

Selectarea conversației din listă:

```
<div id="select_conv ">
<select   id="convList"   onchange="onConversationSelected()"   style="display:inline-
block">
<option selected="true">--Selecteaza conversatia--</option></select>
```



Selectarea conversației din listă

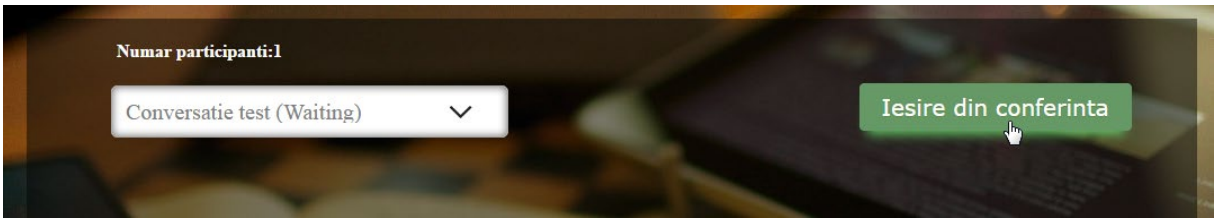
Secțiunea care se ocupă de resetarea stării conversației dacă aceasta a început sau nu:

```
<section class="button" style="display:none" id="buttons_area">
    <div onclick="start()" id="startButton"class="hide">Incepeconferinta</div>
    <div onclick="join()" id="joinButton" class="hide">Intra in  conferinta</div>
    <div onclick="leave()" id="leaveButton" class="hide">Iesire
    dinconferinta</div></section>
```

Incepe conferința

Stare conferință deja începută

Starea de așteptare după intrarea în conferință.



Starea conexiunii în așteptare

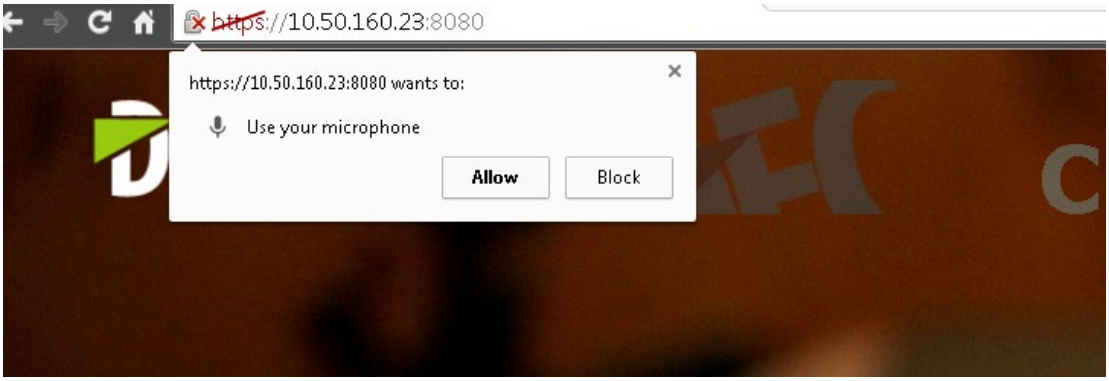
Primul lucru care semnalează conectarea la conferință este afișarea stării ”Active”.

Starea conexiunii active

Activare streaming media:

```
<div>Media: <span id="media"></span></div>
```

Înaintea activării fluxului media, browser-ul va cere permisiunea de lansare a microfonului, respectiv a camerei video:



Permisiunea folosirii uneltelor media

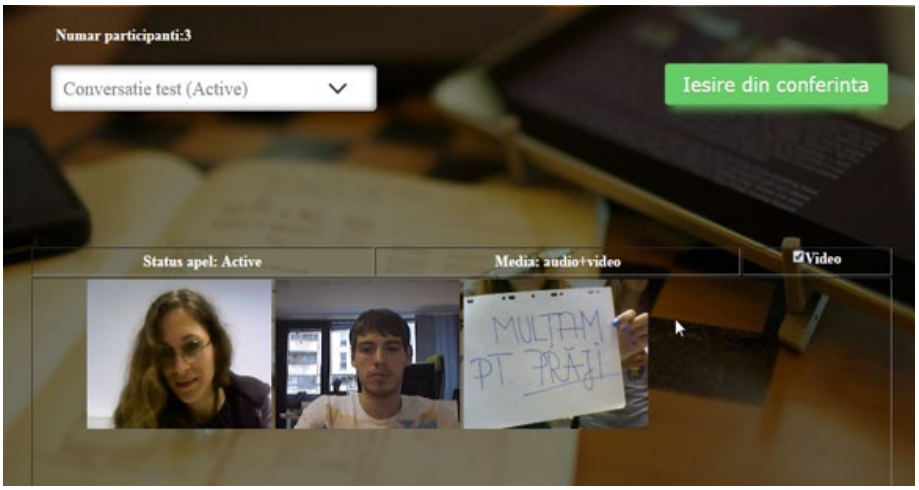
Activarea fluxului video se face prin bifarea căsuței:

```
<input id="enableVideo" type="checkbox" onclick="onVideoChange(this)"/>Video
```

Activarea fluxului video

Pornirea video-ului pentru utilizatorul local, urmat de video-ul pentru utilizatorul *remote*.

```
<video id="localVideo" width="200px" autoplay="true"></video>
<video id="remoteVideo1" width="200px" autoplay="true"></video>
```



Flux video și audio pentru un utilizator local și doi remote

5. Avantaje si motivații pentru a folosi tehnologia WebRTC

5.1. Nevoia de dezvoltare rapida - într-o lume aflata în continua mișcare posibilitatea de a construi rapide și siguraplicații poate să devină un atu important încrestereasectorul business. Cu ajutorul WebRTC si a tehnologiilor încorporate în direct browser-ul web se permite implementarea rapid de solutii și dezvoltarea de noi aplicații scalabile, care pot fi accesate de instantaneu de către utilizatori;

5.2. Comunicatii incorporate (Embedded communication)- deoarece browserele web folosesc protocoalele HTTP (HyperText Transfer Protocol) și HTTPS (HTTP Secure)impreuna cu libajul de markup HTML (HyperText Markup Language) care reprezinta formatul informațiilor afisate, tehnologia WebRTCpoate fi integrata oriunde, nu numai pe browserele web.Deorarece tehnologia web este reprezentata de abilitatea de a prelua informatii de la servereleweb pentru a le interpreta si afișa, aceasta poate fi integrata in aproape orice (inclusiv in aplicații native de pe smartphone-uri sau alte dispozitive moderne);

5.3. Virusi, Malware, Ransomware si spionaj industrial - Utilizatorii sunt si devin din ce in ce mai reticenti in descărcarea de plugin-uri și instalarea de noi programe, in companii se practica blocarea sistemelor, care nu permite angajaților să instaleze programe fără acceptul organizației, deoarece descărcarea si rularea de programe pot deschide riscuri de securitate, ceea ce poate duce la o supraaglomerarea a departamentul

IT. Cu tehnologia WebRTC costurile cu departamentul IT poate fi reduse, fara mai a avea griji cu privire la descărcarea de programe (software) noi, deoarece tehnologiile necesare pentru comunicare sunt construite nativ în browser-ul web si in protocolul pe care acesta il folosește. În comunicarea WebRTC criptarea semnalizarilor se poaterealizat cu DTLS (Datagram Transport Layer Security) iarfluxurile media pot fi criptate folosind SRTP (Secure Protocol de Transport în timp real) pentru evitarea atacurilor de tip "man-in-the-middle";

5.4. Internetul lucrurilor (IoT - Internet of Things) este conceptul care presupune controlul prin intermediul Internetului a dispozitivelor active (si conectate la Internet) care ne inconjoara precum telefoane mobile (smartwatches), aparatura electrocasnica, sisteme de iluminat și chiar automobile. Tehnologia WebRTC poate oferi conectivitate rapida, acces facil la date si interconectarea ecosistemelor generand costuri mai mici și optimiza eficientă autilizarii resurselor.

6. Biblioteci si seturi de instrumente WebRTC

- **PeerJS** esteo bibliteca care adapteaza si sporeste implementarea WebRTC in browser pentru a oferi un API complet configurabil si usor de utilizat pentru conexiunile peer-to-peer;
- **EasyRTC** este un set de instrumente WebRTC adecvat pentru construirea de aplicații fiabile si de inalta securitate. Este un pachet de aplicatii web, fragmente de cod, biblioteci și componente de server, scrise și documentate pentru acces si dezvoltare facila.
- **adapter.js** este o biblioteca care, pentru fiecare versiune a fiecărui browser care acceptă WebRTC, implementează blocurile de cod necesare, stabilește denumirile non-prefixat de API-uri, șiaplică modificările necesare pentru a face browser-ul sa ruleze corespunzator codul sursa conform specificatiilor WebRTC.
- **simpleWebRTC**esteo biblioteca compusa din mai multe module independente precum:
    - **Signalmaster** - server de semnalizare alimentat de **node.js/socket.io***;*
    - **RTCPeerConnection** - normalizeaza conexiunile peer.
    - **getUserMedia** - normalizeaza obtinerea de acces la camera/microfonul utilizatorului
    - **attachMediaStream** - normalizeaza adăugarea de fluxuri la reprezentarile video și gestioneaza elementele *muting/mirroring* într-un singur API simplu.
    - **webrtcsupport** - verifica capacitățile browser si suportul pentru canalele de date precum si managementul constructorilorpentru RTCPeerConnection și RTCSessionDescription;.

  - **getScreenMedia** - extinde metoda *getUserMedia* pentru accesul la ecranul dispozitivului (screensharing).
  - **mediastream-gain** - oferă un controler de volumpentru intrarile audio (audio input).
  - **WildEmitter** - este un emitator de evenimente cu suport pentru metacaractere.
- **Socket.IO** este o biblioteca JavaScript care comunicarea bidirectionalain timp real bazată pe evenimente. Este o bibliotecamulti-platformă, multi-browser simulti-dispozitiv, concentrându-se în mod egal asupra fiabilității și vitezei.
- **rtc.io** este o colecție de module **node.js** care simplifică dezvoltarea WebRTC oferind diferite caracteristici precum:
  - acces la camera web si microfonul local;
  - creare de apeluri audio si video intre browsere;
  - creare de canale de date intre browsere;
  - asigurare management de comunicare;
  - creare server de semnalizare.
- **ShareFest**este un instrument construit pe canalele de date ale API-ului WebRTC folosit pentru partajarea peer-to-peer a fisierelor;
- **Hoodie**este un set de instrumente cu o arhitectura "offline-first" si fara back-end scrisa in JavaScript si NodeJS si bazata pe CouchDB. **Hoodie** ofera o serie de unelte precum stocare de date, inregistrare si adminsitrare utilizatori, e-mail-uri, plati online sau sincronizari.

Tehnologia WebRTC reprezintă un element de succes în viitorul comunicațiilor.
Aspectele pozitive sunt nenumărate, însă nu trebuie neglijate cele negative.
Astfel, din punctul de vedere al clientului, accesarea unei aplicații WebRTC este cât se poate de simplă. Acesta nu are nevoie de niciun program instalat local pe dispozitiv (telefon, tabletă, laptop etc.), iar accesul se face printr-un singur click într-un browser obișnuit.
Pe de altă parte, cu cât devine mai ușor pentru client, cu atât devine mai complex pentru dezvoltatorul de serviciu care trebuie să creeze un mediu propice pentru dezvoltarea și susținerea (*debugging, mentenanță, customising*) unei soluții WebRTC.

Dincolo de mediul de dezvoltare care necesită o serie de programe software și timp îndelungat pentru configurare, există și câțiva factori de risc.

- În acest sens se pune **problema durabilității** unei soluții bazate pe un terț furnizor de servicii (în cazul de față *Circuit*) care ar putea da faliment în orice clipă.
- O altă **problemă** importantă este cea **a securității**, în condițiile în care platforma gazdă ar putea intercepta informații private (*snnifing*) ale companiei sau utilizatorului individual, indiferent de nivelul de criptare folosit.

În ciuda celor două aspecte menționate mai sus, s-a decis folosirea platformei Circuit pentru a crea o soluție de comunicație în timp real, rapidă și ușor de accesat utilizând serviciile oferite de aceasta (PaaS – Platform as a Service) – suport în Cloud, metode WebRTC și bază de date cu utilizatori. Soluția cuprinde funcționalități audio și video de înaltă definiție prin opțiunea unei conferințe multi-utilizator. Tehnologiile web de timp real sunt în prezent cele mai căutate atât de utilizatori cât și de dezvoltatori, dat fiind domeniul vast în care poate fi integrată orice soluție WebRTC.

Dezvoltările ulterioare vor duce la creearea inteligenței artificiale performante, capabilă să citească și să analizeze reacțiile umane, la înlocuirea resurselor umane în diferite posturi (spre exemplu un agent de call-center înlocuit cu un avatar), mutarea convorbirilor din centralele fizice către controlarea funcționalităților acestora în Cloud, ceea ce duce la o scădere considerabilă a cheltuielor aferente unei convorbiri etc. Toate aceste schimbări au scopul de a ușura realizarea de comunicații și servicii web atât pentru dezvoltator, cât și pentru client.
Aceleași mecanisme de sinteză emoțională pot fi folosite și în analiza emoțională a participanților dintr-o video-conferință.

PROGRAMUL SURSĂ

1. App.js

```
var express = require('express') //generare proiect NodeJS Express
  , routes = require('./routes')
  , user = require('./routes/user') //user.js (users list)
  , path = require('path'); //calea catre web page

var fs = require('fs'); //referinta la modulul fs(file system)
var https = require('https'); //server securizat
var privateKey  = fs.readFileSync('C:/OpenSSL-Win32/bin/unitBV-key.pem', 'utf8');
//certificate securitate
var certificate = fs.readFileSync('C:/OpenSSL-Win32/bin/unitBV-cert.pem', 'utf8');
//generate de OpenSSL
var credentials = {key: privateKey, cert: certificate};
var app = express();

//mediile de dezvoltare apelate
app.set('port', process.env.PORT || 8080); //portul local pe care se ruleaza
aplicatia
app.set('views', __dirname + '/views'); //calea pentru fisier ejs - interfata
app.set('view engine', 'ejs'); //trimitere catre fisierul care leaga elemetele de
interfata: index.ejs
app.use(express.favicon()); //indicatie vizuala pentru browser pentru a identifica
site-ul
```

```
app.use(express.logger('dev')); //verificare prin logare
app.use(express.bodyParser()); //analizator de obiecte pentru rute middleware
app.use(express.methodOverride());
app.use(app.router);
app.use(express.static(path.join(__dirname, 'public'))); //serveste obiecte statice
din directorul ”public”

// pentru dezvoltare aplicatie web
if ('development' == app.get('env')) {
  app.use(express.errorHandler());   } //functie erori

app.get('/', routes.index); //preia rutele pentru transmiterea informatiilor
app.get('/users', user.list); //din subprogramul user.js de tip ”cerere” si ”raspuns”
var httpsServer = https.createServer(credentials, app); //creare server virtual
httpsServer.listen(app.get('port'));
```

2. **Client.js**

```
var _client;
var _localUser;
var _conversations = {};  // trimitere catre conversatii cu convId ca si cheie
var _calls = {};          // trimitere catre apeluri cu convId ca si cheie

var $domain = document.getElementById('domain');
var $email = document.getElementById('email');
var $password = document.getElementById('password');
var $logonButton = document.getElementById('logon');
var $logoutButton = document.getElementById('logout');
var $logonState = document.getElementById('logonState');
var $mediaType = document.getElementById('mediaType');
var $localVideo = document.getElementById('localVideo');
var $remoteVideo1 = document.getElementById('remoteVideo1');
var $remoteVideo2 = document.getElementById('remoteVideo2');
var $remoteVideo3 = document.getElementById('remoteVideo3');
var $remoteVideo4 = document.getElementById('remoteVideo4');
var $remoteVideo5 = document.getElementById('remoteVideo5');
var $remoteVideo6 = document.getElementById('remoteVideo6');
var $remoteAudio = document.getElementById('remoteAudio');
var $enableVideo = document.getElementById('enableVideo');
var $startButton = document.getElementById('startButton');
var $joinButton = document.getElementById('joinButton');
var $leaveButton = document.getElementById('leaveButton');
var $pullButton = document.getElementById('pullButton');
var $convList = document.getElementById('convList');
```

```
var $callState = document.getElementById('callState');
var $media = document.getElementById('media');
var $convId = document.getElementById('convId');
var $callId = document.getElementById('callId');
var $errorMessage = document.getElementById('errorMessage');
var $login_wall = document.getElementById('login_wall');
var $content = document.getElementById('content');
var remoteVideo =
[$remoteVideo1,$remoteVideo2,$remoteVideo3,$remoteVideo4,$remoteVideo5,$remoteVideo6]
;

//pentru cazul in care sunt probleme cu platforma Circuit
if (typeof Circuit === 'undefined') {
  $errorMessage.textContent = "Could not load SDK (circuit.js). Try again later.";
  $errorMessage.style.display = 'block';
} elseif (!Circuit.isCompatible()) {
  $errorMessage.textContent = "Sorry, your browser is not supported. Chrome works
:)"; // in cazuri exceptionale
  $errorMessage.style.display = 'block';
} else {
//document.getElementById('mainWrapper').style.display = 'block';
}

Circuit.logger.setLevel(Circuit.Enums.LogLevel.Debug); //enumerare set de nivele de
logare

//inserarea listei si preluarea IDurilor
function getListOption(convId) {
for (var i = 0; i < $convList.options.length; i++) {
if ($convList.options[i].value === convId) {
return $convList.options[i];
    }
  }
}

// Reset interfata pentru sectiunea de call
function resetCallUI() {
  updateButtons(); //schimba buton dupa ce intra in conversatie
  $callState.textContent = ''; //status buton dupa intrare in conversatie
  $media.textContent = '';
  $callId.textContent = '';
  $convId.textContent = '';
  $localVideo.src = ''; //video local
  $remoteVideo1.src = ''; //video de la ceilalti utilizatori
  $remoteVideo2.src = '';
```

```
  $remoteVideo3.src = '';
  $remoteVideo4.src = '';
  $remoteVideo5.src = '';
  $remoteVideo6.src = '';
  $remoteAudio.src = ''; //audio de la ceilalti utilizatori
  $enableVideo.checked = false; //nu permite flux video
  $enableVideo.disabled = true; //video activ, checkbox inactiv
}

// Update interfata utilizator sectiune call pentru optiunea selectata
function setCallUI() {
  updateButtons(); //text buton in functie de conversatie
var convId = document.getElementById('convList').value; //preia conversatie din lista
if (!_conversations[convId]) {
return;
  }
var call = _calls[convId]; //returenaza IDul conversaiei
  $convId.textContent = call.convId;
  $callState.textContent = call.state; //status conversatie
  $callId.textContent = call.callId;
  call.activeMediaType && ($media.textContent = (call.activeMediaType.audio ? 'audio'
: '') + (call.activeMediaType.video ? '+video' : '')); //activare audio si video
  $callId.textContent = call.callId;
  $convList.disabled = !call.isRemote && call.isEstablished; //dupa ce se intra in
conversatie se dezactiveaza lista
  $enableVideo.disabled = !(!call.isRemote && call.isEstablished); //dupa ce se
activeaza video

// video
  $localVideo.src = call.localVideoUrl || ''; //trimitere catre functia video
// Enable video checkbox
if (call.isEstablished) {    //daca se stabileste o conversatie

    $remoteAudio.src = call.remoteAudioUrl;  //se activeaza audio
    document.getElementById('users').innerHTML = call.participants.length+1; //flux
video pentru restul participantilor

for(var i = 0 ; i< call.participants.length ;i++)
      {
       remoteVideo[i].src = call.participants[i].videoUrl;
      }
  }
else {
    $remoteAudio.src = ''; //flux audio
    $remoteVideo1.src = ''; //flux video pentru fiecare utilizator
```

```
    $remoteVideo2.src = '';
    $remoteVideo3.src = '';
    $remoteVideo4.src = '';
    $remoteVideo5.src = '';
    $remoteVideo6.src = '';
  }
}

// Arata buton in funtie de status conversatie
function updateButtons() {
  $startButton.classList.add('hide'); //ascunde butonul ”Incepe”
  $joinButton.classList.add('hide'); //ascunde butonul ”Intra”
  $leaveButton.classList.add('hide'); //ascunde butonul ”Iesire”
// $pullButton.classList.add('hide');

var convId = document.getElementById('convList').value;
if (!_conversations[convId]) {
return;
  }
var call = _calls[convId];

if (!call) { //daca utilizatorul nu a intrat intr-o conferinta
    $startButton.classList.remove('hide'); //ascunde butonul ”Incepe” (Start)
  } elseif (call.state === Circuit.Enums.CallStateName.Started) { //daca o
conversatie este deja ”Started”
    $joinButton.classList.remove('hide'); //ascunde butonul ”Intra” (Join)
  } elseif (call.state === Circuit.Enums.CallStateName.ActiveRemote) { //daca
utilizatorul este in conferinta
    $pullButton.classList.remove('hide');
  } else {
    $leaveButton.classList.remove('hide'); //ascunde butonul ”Iesire” daca
utilizatorul nu este in conferinta
  }
}

// Schimbare status conexiune pentru logare
function addClientEventListeners() { //adaugarea unui client in conversatie
if (_client) {
    _client.addEventListener('connectionStateChanged', function
onConnectionStateChanged(evt) { //schimbare status conexiune prin crearea unui
eveniment
      console.log('Received connectionStateChanged event - state = ', evt.state)
      $logonState.textContent = evt.state; //adaugarea textului "conectat" sau
"deconectat"
if (evt.state === Circuit.Enums.ConnectionState.Disconnected) { //in caz ca este
deconectat se revine la interfata initiala
```

```
        resetCallUI(); //reseteaza interfata
      }
    });

// Status conexiune - "Started"
    _client.addEventListener('callStatus', function (evt) { //este apelata in
momentul in care se adauga un nou client
var call = _calls[evt.call.convId]; //variabila call care preia ID-ul conversatiei
if (!call || call.callId === evt.call.callId) { //asculta eveniment
        console.log('Received callStatus event. state: ' + evt.call.state + ',
reason: ' + evt.reason, evt.call); //receptioneaza starea
        _calls[evt.call.convId] = evt.call; //modifica in functie de eveniment
        updateList(evt.call); //reincarca lista dupa stabilire status
        setCallUI(); //seteaza interfata cu utilizatorul
      }
    });

// Terminare apel
    _client.addEventListener('callEnded', function (evt) { //terminare apel
var call = _calls[evt.call.convId]; //variabila call care preia ID-ul conversatiei
if (call && call.callId === evt.call.callId) {
        console.log('Received callEnded event. state: ' + evt.call.state, evt.call);
//schimbare status conexiune
delete _calls[evt.call.convId]; //sterge client din conversatie
        updateList(evt.call); //actualizeaza lista de conversatii
        resetCallUI(); //actualizeaza interfata
      }
    });
  }
}

// Update status conexiune in lista de conversatii
function updateList(call) { //actualizeaza lista cu conversatii
if (call.state === Circuit.Enums.CallStateName.Terminated) { //daca se incheie
conferinta
    getListOption(call.convId).textContent = _conversations[call.convId].title;
//preia titlul si IDul
  } else {
    getListOption(call.convId).textContent = _conversations[call.convId].title + ' ('
+ call.state + ')'; //in cazul in care nu s-a terminat afiseaza status in continuare
  }
}
```

```
// Incepe conferinta
function start() {
if (!_client || !_localUser) { return alert('Caller is not logged in'); }  //cazul in care nu este logat
var mediaType = {audio: true, video: enableVideo.checked}; //variabila audio, video doar ca optiune
var selectedConvId = $convList.options[$convList.selectedIndex].value; //selecteaza numele conversatiei
  document.getElementById('conference_options').style.display = ''; //preia conversatie din lista
  _client.startConference(selectedConvId, mediaType) //porneste media tipurile de mai sus
    .then(setCallUI) //reseteaza interfata pentru apel
    .catch(function (err) { //exceptia in caz de eroare
    alert('Error starting conference. ' + err);
    console.error(err);
  });
}

// Participa la o conferinta
function join() {   //participa la conferinta
if (!_client || !_localUser) { return alert('Caller is not logged in'); }
//verificare logare
var mediaType = {audio: true, video: enableVideo.checked};  //variabila tipuri media
var selectedConvId = $convList.options[$convList.selectedIndex].value; //selecteaza din lista
var call = _calls[selectedConvId]; //selecteaza conversatia
  document.getElementById('conference_options').style.display = ''; //preia conversatie din lista

  _client.joinConference(call.callId, mediaType)
    .catch(function (err) {
return alert('Failed to join the call: ' + err);
    console.error(err);
  });
}

// Iesire din conferinta
function leave() { //iesire din conferinta
if (!_client || !_localUser) { return alert('Caller is not logged in'); }
//verificare logare
var mediaType = {audio: true, video: enableVideo.checked}; //tipurile media
var selectedConvId = $convList.options[$convList.selectedIndex].value;
var call = _calls[selectedConvId]; //conferinta in care se afla
  document.getElementById('conference_options').style.display = 'none'; //preia conferinta dupa ID
```

```
  _client.leaveConference(call.callId); //iesire din conferinta in functie de IDul
conferintei
}

//Pull a remote call
function pull() {
if (!_client || !_localUser) { return alert('Caller is not logged in'); }

var selectedConvId = $convList.options[$convList.selectedIndex].value;
var call = _calls[selectedConvId];

  _client.pullRemoteCall(call.callId);
}

function cacheCalls(calls) {
  _calls = {};
  calls.forEach(function(call) {
    _calls[call.convId] = call;
  });
}

// Functie logare cu email si parola
function logon() {
if (_client || !$email.value || !$password.value) {
return;
  }
  _client = new Circuit.Client({domain: $domain.value}); //creare instanta client
  addClientEventListeners();  //folosind modulul client

  _client.logon($email.value, $password.value).then(function (user) { //verifica
email si parola
    _localUser = user;
    $domain.disabled = true;
    $email.disabled = true; //camp completat implicit
    $password.disabled = true; //camp completat implicit
    $login_wall.style.display = 'none';
    $content.style.display = '';
    $logonButton.style.display = 'none';
    document.getElementById('right_side').style.display = ''; //afiseaza in dreapta
numele  //////INVERSEAZA
    document.getElementById('name').innerHTML = user.displayName; //preia numele
utilizatorului
  })
    .then(_client.getConversations) //preia conversatia in functie de utilizator
    .then(populateDropdown) //populeaza lista cu conversatii
```

```
    .then(_client.getCalls) //acces in conversatie - poate fi sunat
    .then(cacheCalls)   //TAIEEEEEEEEE
    .catch(function (e) {   //cazul in care logarea nu a avut loc cu succes
    console.error(e);
    alert('Unable to logon. Error: ' + e); //alerta de eroare
    _client.removeAllListeners();
    _client = null;
  });
}

// Functie delogare
function logout() { //iesire din aplicatie
if (!_client) {
return;
  }
  _client.logout().then(_client.removeAllListeners);
  _client = null; //modul client dezactivat
  _localUser = null;
  _conversations = {};
  _calls = {};

  $convList.options.length = 0;
  $domain.disabled = false; //campurile implicite dezactivate
  $email.disabled = false;
  $password.disabled = false;
  $logonButton.style.display = '';
  document.getElementById('right_side').style.display = 'none';
  $logonState.textContent = 'Disconnected'; //status conexiune
  location.reload();
}

//Functie video
function onVideoChange() {
var selectedConvId = $convList.options[$convList.selectedIndex].value; //preia
conversatia din lista
var call = _calls[selectedConvId]; //selecteaza conversatia
if (call && call.localMediaType.video !== enableVideo.checked) { //activeaza video
daca se bifeaza casuta
    _client.toggleVideo(call.callId).then(function () { //se deschide fluxul video
      console.log('Local video was successfully toggled');
    }).catch(function (err) { //exceptie in caz de eroare
      console.error('Failed to toggle video. ', err); //eroare video
    })
  }
}
```

```
// Select conversatie
function onConversationSelected() { //selecteaza conversatie
var convId = document.getElementById('convList').value; //fiecare conversatie are un
ID
var call = _calls[convId]; //call este asociat cu conversatia
if(convId == "--Select conversation--") //selecteaza conversatia
        {
             document.getElementById('buttons_area').style.display = 'none';
        }
else{
        document.getElementById('buttons_area').style.display = ''; //preia cu
ajutorul butonului
  }
if (call) {
    setCallUI(); //seteaza interfata pentru apel
  } else {
    resetCallUI(); //reseteaza interfata
  }
}
// Completeaza lista de convorbiri - conexiunea cu platforma Circuit
function populateDropdown(conversations) { //populeaza lista de convorbiri
  _conversations = {};
  conversations.filter(function (c) {
if (c && c.type === Circuit.Enums.ConversationType.GROUP) { //enumera conversatiile
din lista celor deja create pe platforma Circuit
      _conversations[c.convId] = c;
returntrue; //returneaza true daca se gaseste in lista
    }
returnfalse;
  }).forEach(function (c) {
if (c.type !== 'DIRECT') {
      $convList.options[$convList.options.length] = new Option(c.title, c.convId);
//titlul si ID-ul conversatiei
    }
  });
  $convList.size = Math.min($convList.options.length, 1); //dimensiune lista
}
  });
}
resetCallUI();
```

3. **Index.ejs**

```
<html>
	<head>
		<title>DEComunicator</title>
		<!-- css -->
	<linkrel="stylesheet"type="text/css"href="stylesheets/main.css">
	</head>
	<body>
<divid="header">
	<divid="left_side"><divid="logo">
<h>comunicator</h>
</div></div>
<divid="right_side"style="display: none">
			<divclass="welcome_logout">
		<buttonid="logout"class="button"onclick="logout()">Logout</button>
			</div>
		<divclass="welcome_logout"style="display:inline-block;">
<bstyle="color:white; float:left;">Welcome,  </b><bid="name"style="color:white"></b>
<divstyle="color:white;">Status: <spanid="logonState"><i>Disconnected</i>
</span></div>
			</div></div></div>
		<divid="container">
			<divid="login_wall">
			<h2>Bine ati venit!</h2>
				<divid="login">

					<selectid="domain"style="display:none">
<optionvalue="circuitsandbox.net">circuitsandbox.net</option>
				</select>
<inputtype="email"id="email"placeholder="Email address"/>
<br><inputtype="password"id="password"placeholder="Password"/>
<br><buttonid="logon"class="button"onclick="logon()"style="">Logon</button>
				</div>
			</div>
			<divid="content"style="display:none">
			<table>
			<th>
			<div>
				<divid="select_conv">
<selectid="convList"onchange="onConversationSelected()"style="display:inline-block">
	<optionselected="true">--Selecteaza conversatia--</option>
					</select>
				</th>
				<th>
```

```
<sectionclass="button"style="display:none" id="buttons_area">
<divonclick="start()"id="startButton"class="hide">Incepe conferinta</div>
<divonclick="join()"id="joinButton"class="hide">Intra in conferinta</div>
<divonclick="leave()"id="leaveButton"class="hide">Iesire din conferinta</div>
<!--  <div onclick="pull()" id="pullButton" class="hide">Apeleaza</div> -->
                                </section>
                                </th>
                                <tr>
            <bstyle="color:white; padding-left:80px">Numar participanti:</b>
      <bid="users"style="color:white"></b>
                                </tr>
</div></div>
</table>
                          <!-- TABELUL -->
                          <divid="conference_options"style="display:none;" >
                                <tableborder="1"style="padding-top:30px">
                                <tr>
            <th><div>Status apel: <spanid="callState"></span></div></th>
            <th><div>Media: <spanid="media"></span></div></th>
            <thstyle="vertical-align:middle">
<inputid="enableVideo"type="checkbox"onclick="onVideoChange(this)"/>Video
                                      </th>
                                </tr>
                                <tr>
                                <tdcolspan="3">
      <divstyle="display:none">Conversation ID: <spanid="convId"></span></div>
            <divstyle="display:none">Call ID: <spanid="callId"></span></div>
                                      <center>
                                      <div>
            <videoid="localVideo"width="200px"autoplay="true"></video>
            <videoid="remoteVideo1"width="200px"autoplay="true"></video>
            <videoid="remoteVideo2"width="200px"autoplay="true"></video>
            <videoid="remoteVideo3"width="200px"autoplay="true"></video>
            <videoid="remoteVideo4"width="200px"autoplay="true"></video>
            <videoid="remoteVideo5"width="200px"autoplay="true"></video>
            <videoid="remoteVideo6"width="200px"autoplay="true"></video>
                                      </div>
                                      </center>
                                </td>
                                </tr>
                                </table>
                                </div>
                          <!-- SFARSIT TABEL -->
                          <spanid="errorMessage"style="display: none"></span>
```

```
                        <audioid="remoteAudio"autoplay="autoplay"></audio>
                        </div>
                    </div>
                </div>
        </body>
        <!-- Surse scripturi -->
        <scriptsrc='https://circuitsandbox.net/circuit.js'></script>
<scriptsrc='js/client.js'></script>
        </html>
```

4. **Main.css**

```
html{
background:url(../Images/background.jpg)no-repeatcentercenterfixed;
-webkit-background-size:cover;
-moz-background-size:cover;
-o-background-size:cover;
background-size:cover;
width:100%;
height:100%;
}

htmlbody{
        overflow:hidden;
        width:100%;
        margin:0auto;
}


/*scris "comunicator"*/
h{
font-family:"Verdana",serif;
color:#6c807b;
font-size:540%;
font-weight:bold;
font-variant:small-caps;
opacity:0.6;
margin:58%;
background-repeat:no-repeat;
        background-size:contain;

}
```

```
/*"Bineaivenit"*/

h2{
font-family:"Times New Roman",serif;
color:white;
font-size:190%;
font-weight:bold;
opacity:0.7;
margin-left:28%;
padding-top:20%;


}

htmlbody#header#left_side{
	width:70%;
	height:110px;
	display:inline-block;
	padding-top:20px;

}



htmlbody#header#right_side{
	width:50%;
	height:100px;
	float:right;
	display:inline-block;
	margin:0auto;
}

htmlbody#header#left_side#logo{
	width:80%;
	height:80%;
	background-image:url("../Images/logo.png");
	background-repeat:no-repeat;
	background-size:contain;
	margin-left:5%;
	margin-top:0px;
}
```

```
/*container cu login*/

htmlbody#container{
width:100%;
	height:100%;
	padding-top:90px;

}

htmlbody#container#login_wall{
	width:30%;
	height:450px;
	background:rgba(0,4,5,0.5);
margin:0auto;
}

/*chenarnegru user si pass*/
htmlbody#container#login_wall#login{
	width:35%;
	height:30%;
	padding-top:10%;
margin:0auto;
margin-left:28%;
}

/*container conferinta*/
htmlbody#container#content{
	width:65%;
	height:500px;
	background:rgba(0,4,5,0.5);
	margin:0auto;
	padding-top:20px;
}

#conf_state{
	color:white;
	display:inline-block;
	float:left;
	margin-left:40px;
}
#select_conv{
	display:inline-block;
	float:left;
	margin-left:20px;
```

```
}
#video_zone{
	float:left;
	margin-top:60px;
width:100%;
}

/*casute login siparola*/
input{
	color:black;
border:5pxsolidwhite;
-webkit-box-shadow:
inset008pxrgba(0,0,0,0.1),
0016pxrgba(0,0,0,0.1);
-moz-box-shadow:
inset008pxrgba(0,0,0,0.1),
0016pxrgba(0,0,0,0.1);
box-shadow:
inset008pxrgba(0,0,0,0.1),
0016pxrgba(0,0,0,0.1);
padding:5%;
background:rgba(255,255,255,0.5);
margin:0010px0;
}

.button{
	margin-left:32%;
	-moz-box-shadow:0px10px14px-7px#66CC66;
	-webkit-box-shadow:0px10px14px-7px#66CC66;
	box-shadow:0px10px14px-7px#66CC66;
	background:-webkit-gradient(linear,lefttop,leftbottom,color-
stop(0.05,#66CC66),color-stop(1,#66CC66));
	background:-moz-linear-gradient(top,#66CC665%,#66CC66100%);
	background:-webkit-linear-gradient(top,#66CC665%,#66CC66100%);
	background:-o-linear-gradient(top,#66CC665%,#66CC66100%);
	background:-ms-linear-gradient(top,#66CC665%,#66CC66100%);
	background:linear-gradient(tobottom,#66CC665%,#66CC66100%);
	filter:progid:DXImageTransform.Microsoft.gradient(startColorstr='#43b5af',endC
olorstr='#37948f',GradientType=0);
	background-color:#66CC66;
	-moz-border-radius:4px;
	-webkit-border-radius:4px;
	border-radius:4px;
	border:1pxsolid#669966;
	display:inline-block;
```

```
	cursor:pointer;
	color:#ffffff;
	font-family:Verdana;
	font-size:20px;
	font-weight:normal;
	padding:8px19px;
	text-decoration:none;
	text-shadow:0px1px0px#5b8a3c;

}

.button:hover{
	background:-webkit-gradient(linear,lefttop,leftbottom,color-
stop(0.05,#669966),color-stop(1,#669966));
	background:-moz-linear-gradient(top,#3366995%,#669966100%);
	background:-webkit-linear-gradient(top,#3366995%,#669966100%);
	background:-o-linear-gradient(top,#3366995%,#669966100%);
	background:-ms-linear-gradient(top,#3366995%,#669966100%);
	background:linear-gradient(tobottom,#6699665%,#669966100%);
	filter:progid:DXImageTransform.Microsoft.gradient(startColorstr='#37948f',endC
olorstr='#43b5af',GradientType=0);
	background-color:#336699;
}

.button:active{
	position:relative;
	top:1px;
}
::-webkit-input-placeholder{
color:#1F3059;
}
:-moz-placeholder{
color:#1F3059;
}
::-moz-placeholder{
color:#1F3059;
}
:-ms-input-placeholder{
color:#1F3059;
}
.hide{display:none;}

pre{padding:5px; margin:5px; }
.string{color:green; }
.number{color:darkorange; }
```

```
.boolean{color:blue; }
.null{color:magenta; }
.key{color:red; }
.error{color:red; }
#outputspan{color:blue; }

#errorMessage{
margin-top:10px;
color:red;
}

.welcome_logout{
	height:50px;
	float:right;
margin-right:50px;
margin-top:25px;
}
#video_zone{
	color:white;
}


select{
	background-image:url(../Images/arrow.png);
	background-repeat:no-repeat;
	background-position:300px;
	background-size:20px;
	width:353px;
	padding:12px;
	margin-top:8px;
	font-family:TimesNewRoman;
	line-height:1;
	border-radius:5px;
	background-color:white;
	color:grey;
	font-size:20px;
	-webkit-appearance:none;
	box-shadow:inset0010px0rgba(0,0,0,0.6);
	outline:none
}
select:hover{
	color:#43b5af
}
table{
```

```
        color:white;
        width:90%;
        margin-top:10px;
        margin-left:5%;
        border-style:none;
}
#conference_options{
        margin-top:100px;
}
```

5. **Style.css**

```
body{
font-family:sans-serif;
font-size:12px;
}
section{padding-top:20px; }
input{margin-right:10px; }
button{
margin-right:10px;
}
pre{padding:5px; margin:5px; }
.string{color:green; }
.number{color:darkorange; }
.boolean{color:blue; }
.null{color:magenta; }
.key{color:red; }
.error{color:red; }
#outputspan{color:blue; }
#errorMessage{
margin-top:10px;
color:red;
}
#right{
        background-color:black;
}
        .hide{display:none;}
```

6. **Package.json**

Sursa cod a fișierului de mai jos cuprinde date despre numele pachetului, versiunea, dicționarul de comenzi (*scripts*) și lista de dependințe instalate în node_modules.

```
{
"name": "circuit",
```

```
"version": "0.0.1",
"private": true,
"scripts": {
"start": "node app.js"
  },
"dependencies": {
"express": "3.2.6",
"ejs": "*"
  }
}
```

7. **Index.js**

```
/*
 * (GET) preia ”home page”
 */
exports.index = function(req, res){
  res.render('index', { title: 'Express' });
   };
```

8. **User.js**

```
/*
 * (GET) preialistautilizatorilor
 */

exports.list = function(req, res){
  res.send("respond with a resource");
};
```