Building WebRTC Apps with JsSIP

José Luis Millán
jssip.net
joseluis.millan@frafos.com

JsSIP

- State of the art JavaScript SIP library
- Built in JavaScript from the ground up
- Uses WebSocket as SIP transport (RFC 7118)
- Uses WebRTC at media plane
- Widely used in open and private WebRTC apps

JsSIP

Born to extend the use of SIP to Web browsers

WebRTC interaction

- Makes use of the WebRTC API given by browsers
  - getUserMedia
    - To acquire the microphone and camera
  - RTCPeerConnection
    - SDP generation
    - STUN/ICE/DTLS/SRTP
- Integrates RTCNinja

WebRTC interaction

- RTCNinja (https://github.com/eface2face/rtcninja.js)
  - WebRTC API wrapper to deal with different browsers transparently
  - API
    - getUserMedia (rtcninja.getUserMedia())
    - getMediaDevices (rtcninja.getMediaDevices())
    - RTCPeerConnection (rtcninja.RTCPeerConnection())
    - RTCSessionDescription (rtcninja.RTCSessionDescription())
    - …

Where can we use it?

- Desktop
  - Natively: in Chrome, Firefox, Opera
  - Via external plugin: Safari, IE (https://www.temasys.com.sg/solution/webrtc-plugin)
- Mobile
  - Android
    - Chrome, Firefox, Opera
    - Android version >= 4.4, the WebView is based on the Chromium project
    - Android version < 4.4, CrossView provides a WebView with WebRTC capabilities
  - IOS
    - Cordova plugin IOSRTC
- Node.js

Where can we use it?

- Mobile
    - IOS
        - Using the plugin interface in rtcninja

```
// Just for Cordova apps.
document.addEventListener('deviceready', function () {
  // Just for iOS devices.
  if (window.device.platform === 'iOS') {
    // Load rtcninja with cordova-plugin-iosrtc.
    rtcninja({
      plugin: cordova.plugins.iosrtc.rtcninjaPlugin
    });
  }
});
```

JsSIP API

- Event driven
  - Successful / Failed WebSocket connection
  - Successful / Failed SIP registration
  - New Call
    - Call Answered
    - Call Failed
    - Call Terminated
    - …
  - New Message
    - Message Succeeded / Failed

JsSIP API

- Intuitive and easy to use
    - ua.start() / ua.stop()
    - ua.register() / ua.unregister()
    - ua.call()
        - session.terminate()
        - session.hold()
        - session.mute()
        - session.sendDTMF()
        - …
    - ua.sendMessage()

JsSIP API

- Configuration
  - Wide configurability
  - Only two mandatory parameters:
    - WebSocket server/s
    - SIP URI

JsSIP API

http://jssip.net/documentation/

JsSIP in examples

- Library download

```
<script src="http://jssip.net/download/releases/jssip-0.6.26.js"></script>
```

- JsSIP User Agent creation

```
var ua = new JsSIP.UA({
  'ws_servers': 'ws://tryit.areteasea.com:8080',
  'uri': 'sip:alice@areteasea.com'
});
```

JsSIP in examples

- SIP registration
  - Event callbacks definition (optional)

```
ua.on('registered', function(){
  console.log('Registered!');
});

ua.on('unregistered', function(){
  console.log('Unregistered!');
});

ua.on('registrationFailed', function(e){
  console.log('Registration failed! Cause: '+ e.cause);
});
```

  - User Agent registration

```
ua.register();
```

JsSIP in examples

- SIP Message
  - Event callbacks definition (optional)

```
ua.on('newMessage', function(e) {

  if (e.direction === 'local') {
    console.log('Sending Message!');
  }
  else if (e.direction === 'remote') {
    console.log('Received Message!’);
    e.message.accept();
  }
});
```

  - Outgoing Message

```
ua.sendMessage('sip:bob@areteasea.com', 'Hi Bob!', {
  eventHandlers: {
    'succeeded': function(){ console.log('Message succeeded!'); },
    'failed':    function(e){ console.log('Message failed! Cause: '+ e.cause); }
});
```

JsSIP in examples

- SIP Call
  - Event callbacks definition (optional)

```
ua.on('newRTCSession', function(e) {

  if (e.direction === 'local') {
    console.log('Outgoing call');
  }
  else if (e.direction === 'remote') {
    console.log('Incoming call');
    e.session.answer();
  }
});
```

  - Outgoing Call

```
ua.call('sip:bob@areteasea.com, {
  eventHandlers: {
    'accepted': function(){ console.log('Call accepted!'); },
    'failed':    function(e){ console.log('Call failed! Cause: '+ e.cause); }
});
```

Building an application

- How much of server logic do we need? It depends…
  - Every peer does WebRTC
    - WebSocket outbound SIP server
    - SIP Registrar
  - Not every peer does WebRTC
    - WebRTC gateway or B2BUA
  - Media management is needed (media recording, conferencing)
    - Media server supporting WebRTC

Building an application

- HTML button to make WebRTC calls
- WebRTC audio conference application

HTML button to make WebRTC calls

```
<button class="callme-btn callme-btn-lg” data-call-to="music@iptel.org" data-conf-ws_servers="wss://tryit.areteasea.com:8081" data-conf-uri =“anonymous@tryit.areteasea.com”>Call
</button>

<script>!function(d,s,id){var js,fjs=d.getElementsByTagName(s)[0],p=/
^http:/.test(d.location)?"http":"https";if(!d.getElementById(id))
{js=d.createElement(s);js.id=id;js.src=p+"://go.areteasea.com/assets/js/callme-widget.js";fjs.parentNode.insertBefore(js,fjs);}}(document, "script", “click2dial-widget");
</script>
```

HTML button to make WebRTC calls

- CSS classes defining different colours for the button
  - default (WS connected, call terminated)
  - dialling (call progress)
  - answered (call accepted)
  - error (call failed, WS disconnected, global error)
- On JsSIP events
  - button CSS class is changed
  - button click() method is bound

HTML button to make WebRTC calls

- WS connection callback

```
function() {
  jQuery('.callme-btn')
    .removeClass().addClass('callme-btn callme-btn-default')
    .attr('title','')
    .unbind('click').bind('click', function(){ self.call(this); });
});
```

- Call generation callback

```
function() {
  jQuery(this.dom).removeClass().addClass('callme-btn callme-btn-dialing');
  jQuery(this.dom).text(this.label_dialling);

  jQuery(this.dom).bind('click',function(){
    session.terminate();
  });
});
```

HTML button
to make WebRTC calls

http://go.areteasea.com

WebRTC conference application

- JsSIP
- Frafos WebRTC GW
- SEMS webconference module
- MeteorJS

WebRTC conference application

- MeteorJS -Client side-
  - template based HTML
  - reactive to JS data modifications
  - exchanges status data with the server
  - JSON messages sent over WebSocket

WebRTC conference application

- MeteorJS -Client side-

```
<template name="joinParticipant">
   <div class="panel panel-default {{cssJoined}}">
     <div class="panel-body">
       <div class="col-xs-4">
         {{name}}
         <br />
         <small>{{callState ../participant}}</small>
         {{#if isMutedByOrganiser}}
           <small>(muted by organiser)</small>
         {{/if}}
       </div>
       {{#unless declined}}
         <div class="col-xs-4">
           {{#if isJoined ../participant}}
             {{> joinButton}}
             {{> muteButton}}
           {{/if}}
         </div>
         <div class="col-xs-4">
           {{> volumeMeter}}
         </div>
       {{/unless}}
     </div>
   </div>
 </template>

 <template name="joinButton">
   <button class="btn {{btnClass}} join-button">
     <i class="fa fa-phone"></i> {{callAction}}
   </button>
 </template>
```

WebRTC conference application

- MeteorJS -Server side-
    - SEMS conference module (XML-RPC)
    - MeteorJS Clients (WSS)
    - receives status data from participants and broadcasts to others
    - pulls SEMS for participants volume info and broadcasts it

WebRTC conference application

http://go.areteasea.com

Thank You

```
(function() {

  var ua = new JsSIP.UA({
    'uri': 'joseluis.millan@frafos.com',
    'ws_servers': ‘wss://fraunhofer.fokus.de'
  });

  ua.on('connected', function() {
    this.sendMessage('audience@KamailioWorld2015.de', 'Thank You!');
  });

  ua.start();
}());
```