UTM

Universidad Tecnológica de la Mixteca

Labor et Sapientia Libertas

Chat con WebRCT y PeerJS

Castro Guerrero José Luis

Programación en web

José Figueroa Martínez

702-A

Acatlima Febrero 2017

WebRCT

Con la llegada de HTML5 se han ido agregando grandes capacidades a las páginas web de hot en día; etiquetas como header, nav, article, section, aside y footer se piensa en una mejor organización de las páginas web.Así como también mejoramiento de contenido multimedia, sean estos audio y video. Este últimas características son de las que se van a hacer uso en esta práctica.

La idea de WebRCT es permitir a las aplicaciones web realizar llamadas de voz, chat de video y uso compartido de archivos P2P sin plugins, todo esto en tiempo real. Y lo mejor, sin plugins.

PeerJS

PeerJS es una biblioteca de JavaScript que simplifica las llamadas de datos, video y audio de WebRTC peer-to-peer.

Lo que hace es actuar como una envoltura de la implementación WebRCT de los disntintos navegadores. Es ya bien sabido, que cada navegador implementa de distinta forma las características. PeerJS actua como contenedor de código para dististos navegadores.

PeerServer

WebRCT no es completaente P2P. En una aplicación real, siempre se encuentra un servidor que hace de intermediario para que se haga posible la conexión entre dos peers. Uno se puede encontrar con cortafuegos, routers y demás obstáculos que pueden impeir la comunicación P2P, es ahí en donde un servidor entra en acción.

PeerServer es un componente del servidor de PeerJS y permite que dos o más dispositivos se conecten entre sí.

Implementación

Lo primero que se procederá es la explicación de implementación del código del cliente, teniendo la siguiente extructura:

- public
  - css
    - style.css
  - index.html
  - js
    - script.js
  - package.json

Las dependencias se encuentran en el archivo package.json, las cuales se pueden instalar simplemente ejecutando el comando npm install.

Entre las dependencias tenemos:

- Picnic - un marco CSS ligero.
- jQuery - sirve para seleccionar elementos en la página de eventos y manejo.
- PeerJS - el componente del lado del cliente de PeerJS.
- Handlebars - una biblioteca de plantillas JavaScript.

Index.html

El código que se tiene en el archivo raiz de la página las partes importantes son que se crean dos elementos de video, uno para la cámara local y la otra del remoto. Y para el chat se creó una pequeña plantilla de handlebars.

```
<script id="plantilla" type="text/x-handlebars-template">
  {{#each mensajes}}
  <li>
    <span class="desde">{{desde}}:</span> {{texto}}
  </li>
  {{/each}}
</script>
```

El cual presenta los mensajes recibidos. Se tiene un bucle de los mensajes, mostrando el remitente y el mensaje.

Script.js

Este archivo es el que contiene toda la codificación de la aplicación. Se inicia por declarar variables globales.

```
var mensajes = []
```

```
var usuarioID, nombre, con
var plantilla = Handlebars.compile($('#plantilla').html())
```

En donde la variable mensajes es array de todos los mensajes que se envían entre los peers, el usuarioID y nombre son los datos con los cuales se registra el usario. Con es la variable de la conexición entre peers y la plantilla es en donde se mostraran los mensajes.

La siguiente linea es la creación del Peer, con ayuda de la librería PeerJS, ingresando el host, puerto, la ruta del archivo peerjs, el grado de debug y los datos de configuración. Este último permite establecer el servidor ICE.

```
var peer = new Peer({
      host: 'localhost',
      port: 9000,
      path: '/peerjs',
      debug: 3,
      config: {
        'iceServers': [{
          url: 'stun:stun1.l.google.com:19302'
        },
          {
            url: 'turn:numb.viagenie.ca',
            credential: 'muazkh',
            username: 'webrtc@live.com'
          }
        ]
      }
    })
```

En el momento de abrirse la conexión se dispara las siguienes líneas, permitiendo mostrar el ID del cliente.

```
peer.on('open', function () {
  $('#id').text(peer.id)
})
```

Para las siguientes líneas es para la obtención de las propiedades de audio y video del navegador, agregandoselo al obhjecto navigator. Se obtiene la propiedad getUserMedia del navegador que corresponda.

Entonces se obtiene el video con la función video, se pone el audio y video en true. Y se obtiene el video de la cámara local.

```
navigator.getUserMedia =   navigator.getUserMedia ||
                           navigator.webkitGetUserMedia ||
                           navigator.mozGetUserMedia

function video (callback) {
  navigator.getUserMedia({
    audio: true,
    video: true
  }, callback, function (error) {
    console.log(error)
    console.log('Ha ocurrido un problema. Prueba de nuevo')
  })
}
video(function (stream) {
  window.localStream = stream
  streamRecibido(stream, 'mi-camara')
})
```

La función encargada de la obtención del video en streaming es streamRecibido:

```
function streamRecibido (stream, elemento) {
  var video = $('#' + elemento + ' video')[0]
  video.src = window.URL.createObjectURL(stream)
  window.peerStream = stream
}
```

En el momento en que el usuario ingresa el sus datos, nombre y ID, y presione el botón de Entrar, se dispará el evento de click. La cual obtiene los datos ingrados y crea la conexión entre los peers. Una vez que seestablece la conexicón el evento data se dispara cada vez que se envía un mensaje al usuario local. Y se muestra el chat, ocultando el login.

```
$('#login').click(function () {
  nombre = $('#nombre').val()
  usuarioID = $('#usuarioID').val()
  if (usuarioID) {
    con = peer.connect(usuarioID, {
      metadata: {
        'username': nombre
      }
    })
    con.on('data', handlerMensaje)
  }

  $('#chat').removeClass('hidden')
  $('#conectar').addClass('hidden')
})
```

Cada vez que el usuario local se intenta conectar con el usuario remoto, el evento connection se dispara. Permitiendonos establecer la conexición con el usuario remoto. De la conexión establecida se

escucha el evento data en donde el usuario local envia un mensaje al usuario remoto. Después se oculta el ID del usuario y se muestra muestra el nombre del usuario conectado.

```
peer.on('connection', function (conexion) {
    con = conexion
    usuarioID = conexion.peer
    con.on('data', handlerMensaje)

    $('#usuarioID').addClass('hidden').val(usuarioID)
    $('#peerConectado').removeClass('hidden')
    $('#conectado').text(conexion.metadata.username)
  })
```

La función handlerMensaje es la encargada de mostrar los mensajes, ingresando el mensaje al arreglo mensajes y se obtiene la plantilla y se agrega a la página.

```
function handlerMensaje (data) {
  mensajes.push(data)
  var html = plantilla({
    'mensajes': mensajes
  })
  $('#mensajes').html(html)
}
```

En el momento de presionar el botón de Enviar o puldar la tecla de Enter, se llama a la función de enviarMensaje que es la encargada de obtener los datos del usuario que enviará em mensaje y se llama la función handlerMensaje.

```
function enviarMensaje () {
  var texto = $('#mensaje').val()
  var data = {
    'desde': nombre,
    'texto': texto
  }
  con.send(data)
  handlerMensaje(data)
  $('#mensaje').val('')
}
```

Cuando algún usuario quiera realizar una videollamada se hace uso del método call proporcioando por el objeto peer. El resultado es un objeto en el que podemos escuchar el evento strem y el cual se activa en el momemto en que el usuario remoto está disponible para el usuario que realizó la videollamada.

```
$('#videollamada').click(function () {
  var call = peer.call(usuarioID, window.localStream)
  call.on('stream', function (stream) {
    window.peerStream = stream
    streamRecibido(stream, 'camara-remota')
  })
})
```

Cuando el usuario local hace una llamada a uno remoto, el evento call se dispara. Es aquí en donde se ejecuta la función videoLlamada para aceptar la llamada.

En la función videoLlamada el método answer es invocado para seguir con la llamada, aceptando como argumento el video del usuario local. En el momento en que el usuario remoto está disponile

para la transmisión de video el evento stream es disparado.

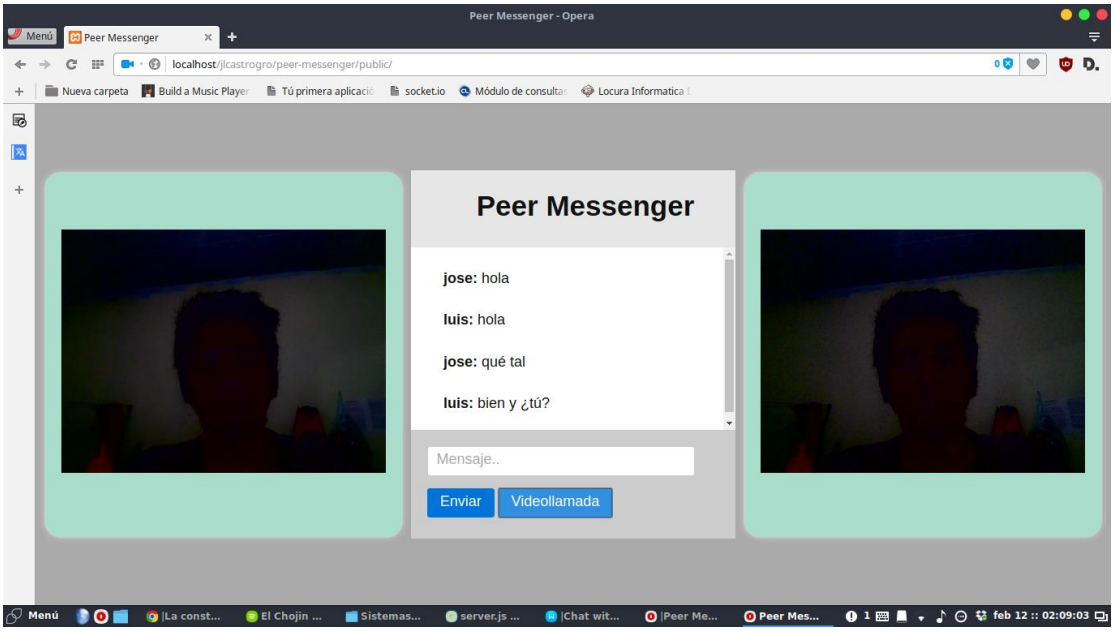
```
function videoLlamada (call) {
      call.answer(window.localStream)
      call.on('stream', function (stream) {
        window.peerStream = stream
        streamRecibido(stream, 'camara-remota')
      })
  }
```

Por parte del servidor, se creo un pequeño servidor con ayuda de nodejs, haciendo uso de la módulo de peer, escuchando en el puerto 9000 y sirviendo el archivo peers.

```
var server = require('peer').PeerServer
server({port: 9000, path: '/peerjs'})
```

Resultados

Para la parte de resultados se hicieron pruebas en una misma máquina abriendo distintas instancias del cliente, como se muestra a continuación.



Conclusiones

La tecnología de WebRCT tiene gran potencial, ya que hace uso de estandares para la transmisión de datos con ayuda de elementos de HTML5. Se pueden encontrar implementaciones ya de esta tecnología como WebTorrent que es la transmisión de torrent a través de la web sin plugins o extensiones.

Para esta práctica intente establecer una conexión verdaderamente remota, cosa que no lo logré. WebRCT necesita utilizar un canal de transmisión seguro, como es HTTPS, para poder establecer una conexión entre peers, esto como medida de seguridad.