WEBRTC

COURSE OUTLINE

- Overview of WebRTC
- WebRTC in a single browser – getUserMedia
- WebRTC signaling
- WebRTC for data exchange
- WebRTC screen sharing
- WebRTC case study – medical application
- WebRTC design considerations

COURSE OVERVIEW

Chapter 1, Getting Started with WebRTC, covers how WebRTC enables audio and video communication for web-based applications. You will also begin by running an example of a WebRTC application inside your browser.

Chapter 2, Getting the User's Media, covers the first step when creating a communication application to get webcam and microphone input. This chapter also covers how to use the Media Capture and Streams API to capture this information from you. We also begin development by building the foundation of our communication example.

Chapter 3, Creating a Basic WebRTC Application, covers an introduction to the first WebRTC API—the RTCPeerConnection. This chapter also lays the groundwork for creating a WebRTC application by peeking inside the complex structure of WebRTC and what we can expect when we begin working with the API.

COURSE OVERVIEW

Chapter 4, Creating a Signaling Server, covers the steps in creating our very own signaling server to help our clients find each other on the Internet. This includes in-depth information on how signaling works in WebRTC and how we will utilize it in our example application.

Chapter 5, Connecting Clients Together, covers the actual usage of our signaling server. It also covers connecting two users successfully using the WebRTC API, Media Capture, and the signaling server that we created in the previous chapter to build our working example.

Chapter 6, Sending Data with WebRTC, covers an introduction to the RTCDataChannel and how it is used to send raw data between two peers. This chapter elaborates on adding a text-based chat for our clients.

COURSE OVERVIEW

Chapter 7, File Sharing, elaborates on the concept of sending raw data by looking at how we can share files between two peers. This will demonstrate the many uses of WebRTC outside of audio and video sharing.

Chapter 8, Advanced Security and Large-scale Optimization, covers advanced topics when delivering a large-scale WebRTC application. We look at theoretical security and performance optimizations used by other companies in the industry.

LOGISTICS

Class Hours:

- Start time is 8:30am
- End time is 4:00pm
- Class times may vary slightly for specific classes
- Breaks mid-morning and afternoon (15 minutes)

Lunch:

- Lunch is noon to 1:15pm
- Yes, 1 hour and 15 minutes
- Extra time for email, phone calls, or simply a walk.

Telecommunication:

- Turn off or set electronic devices to vibrate
- Reading or attending to devices can be distracting to other students
- Try to delay until breaks or after class

Miscellaneous

- Courseware
- Bathroom
- Fire drills

meet the instructor

George Niece

Cloud, DevOps, IoT Consultant with a Linux sysadmin background. Focused on cloud-native application modernization

Twitter
@georgeniece

LinkedIn
Linkedin.com/in/GeorgeNiece

mail
George.Niece@DigitalTransformationStrategies.net

Expertise

- Cloud
- AppDev
- IoT
- Automation
- CI/CD
- Microservices
- Agile

STARTING OFF RIGHT

Now we'll do a short GTKY

Get To Know You

Name

Title or Role

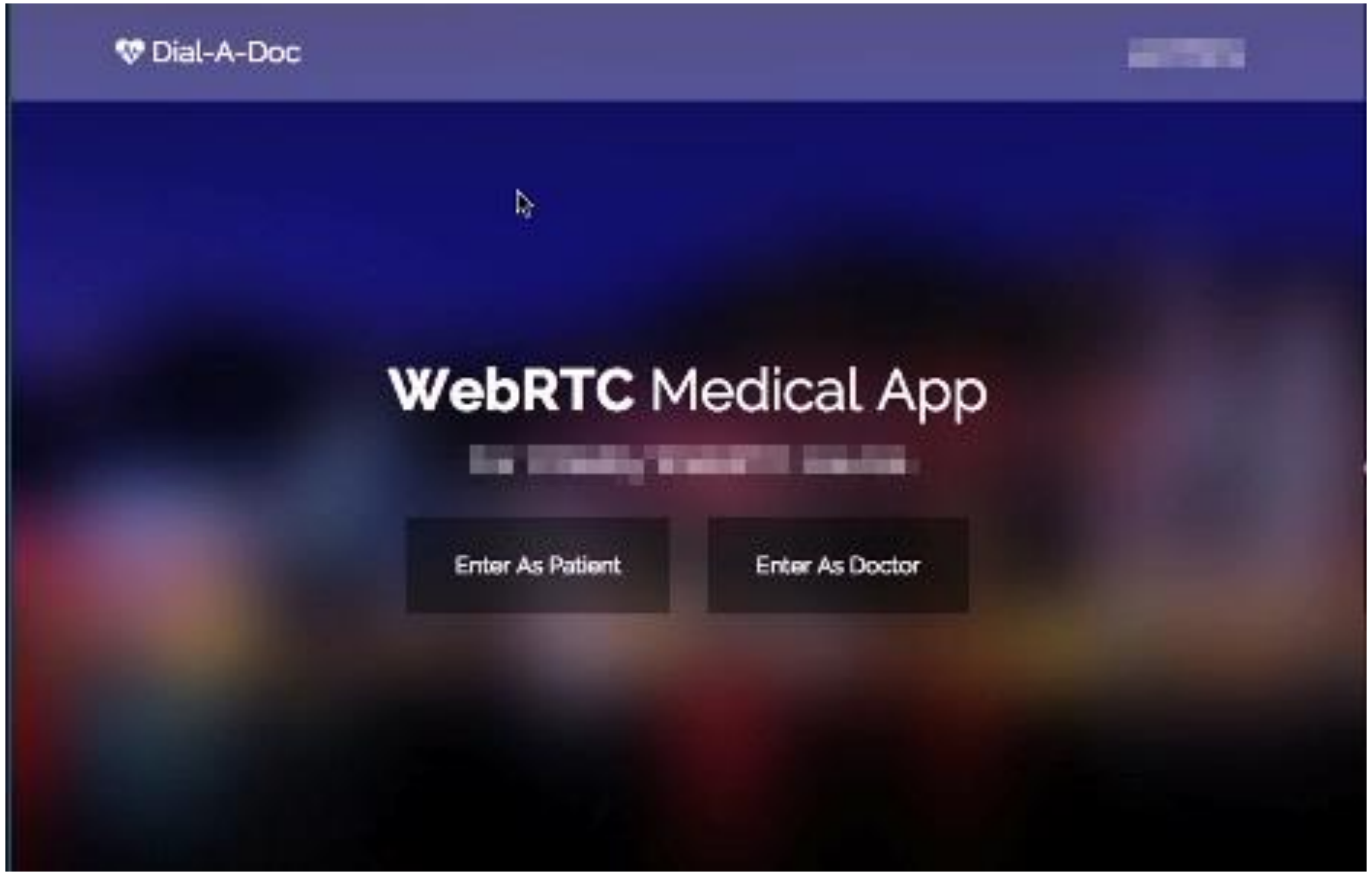
What you do?

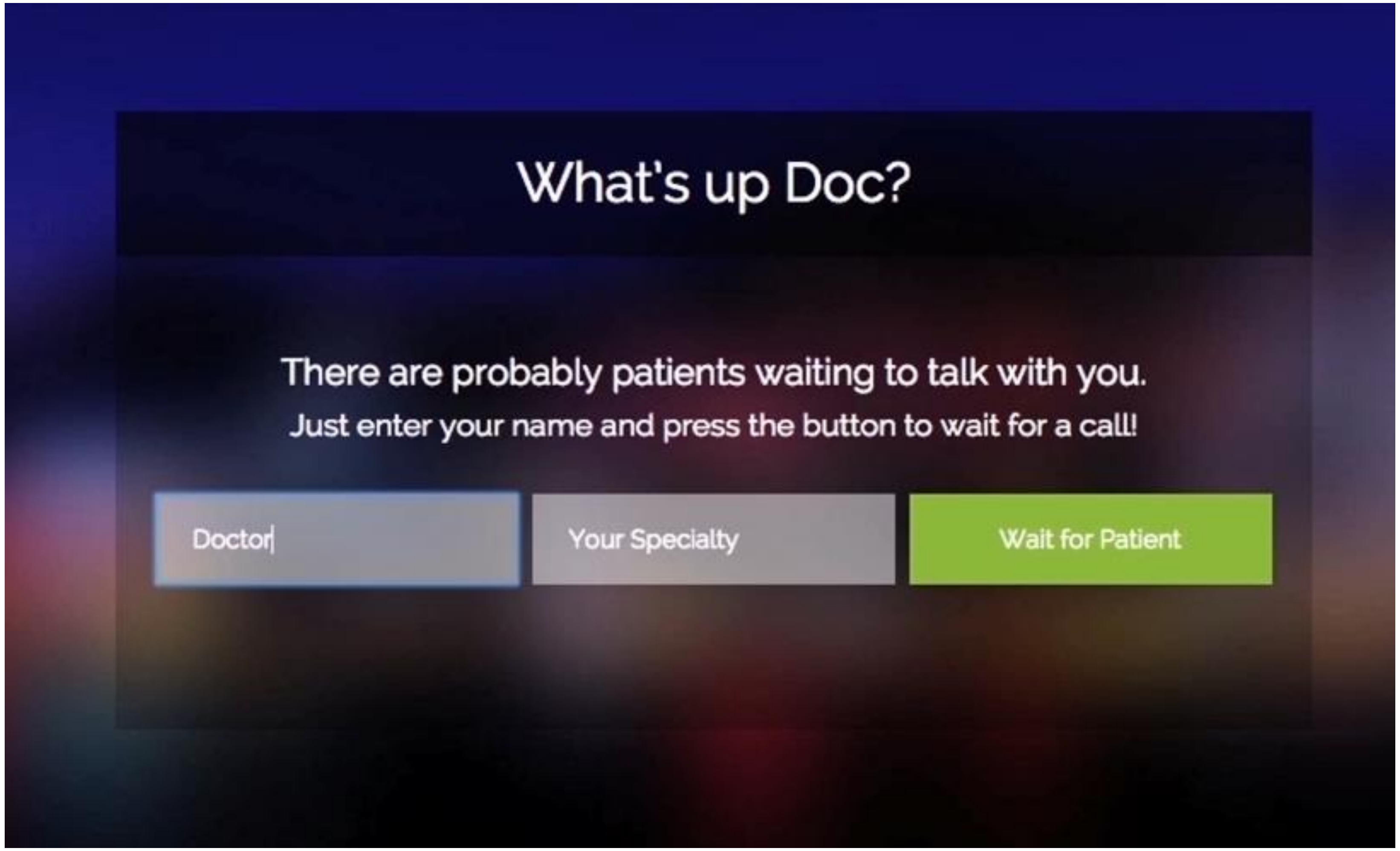
Why are you here?

What is your experience with Node.js and WebRTC

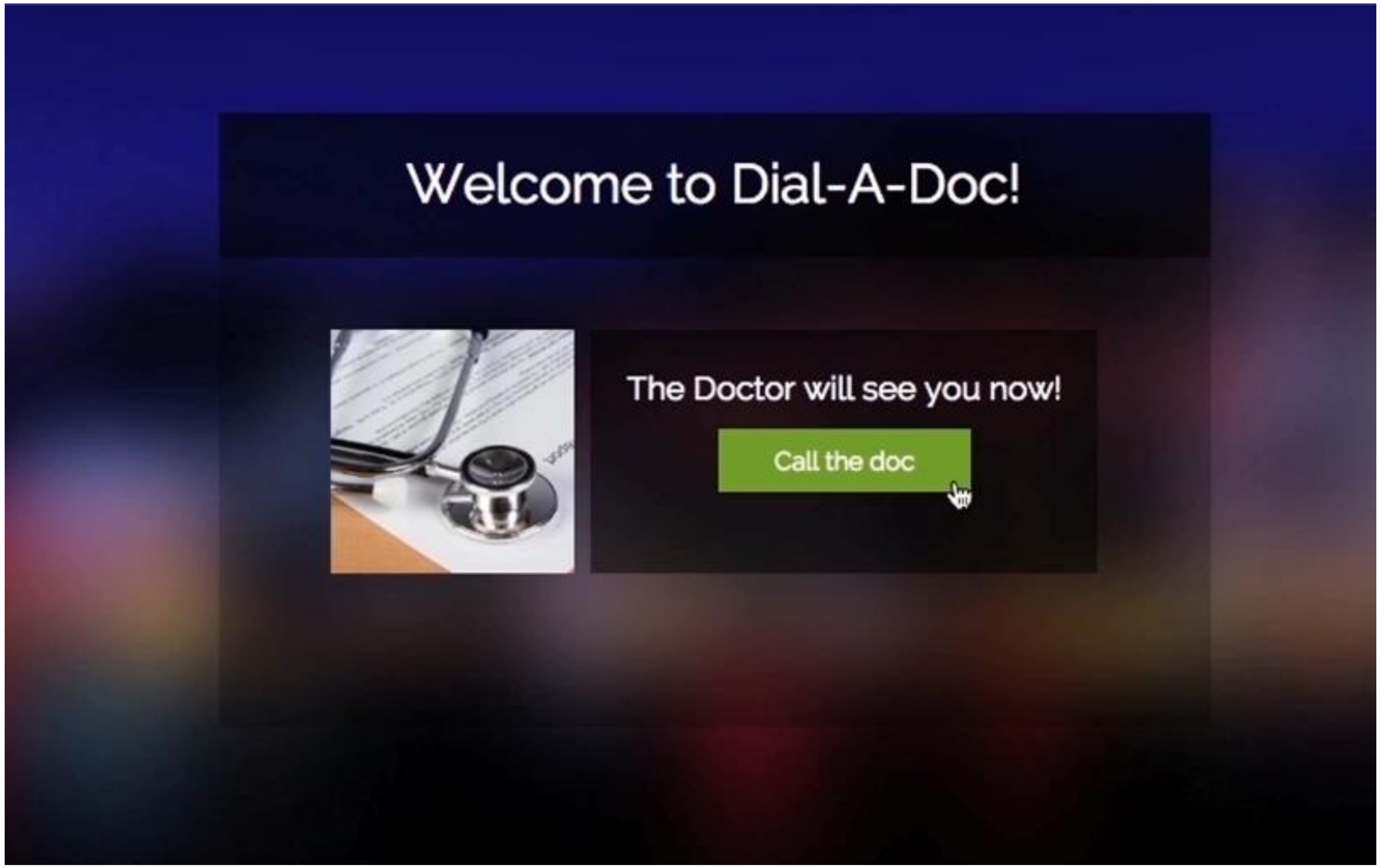
DEMO

THIS IS THE APPLICATION THAT WE WILL BUILD




What's up Doc?

There are probably patients waiting to talk with you.
Just enter your name and press the button to wait for a call!


Welcome to Dial-A-Doc!
The Doctor will see you now!
Call the doc

WHAT IS WEBRTC

WebRTC = Web Real-Time Communication)

Released in May 2011, and still developing and evolving standards.

A set of protocols is standardized by Real-Time Communication in WEB-browsers Working group at **http://tools.ietf.org/wg/rtcweb/** of the IETF (Internet Engineering Task Force)

APIs are standardized by the Web Real-Time Communications Working Groupe at **http://www.w3.org/2011/04/webrtc/** of the W3C (World Wide Web Consortium).

OVERVIEW OF WEBRTC

The biggest accomplishment of WebRTC is bringing high-quality audio and video to the open the Web without the need for third-party software or plugins.

Currently, there are no high-quality, well-built, freely available solutions that enable real-time communication in the browser.

The success of the Internet is largely due to the high availability and open use of technologies, such as HTML, HTTP, and TCP/IP. To move the Internet forward, we want to continue building on top of these technologies.

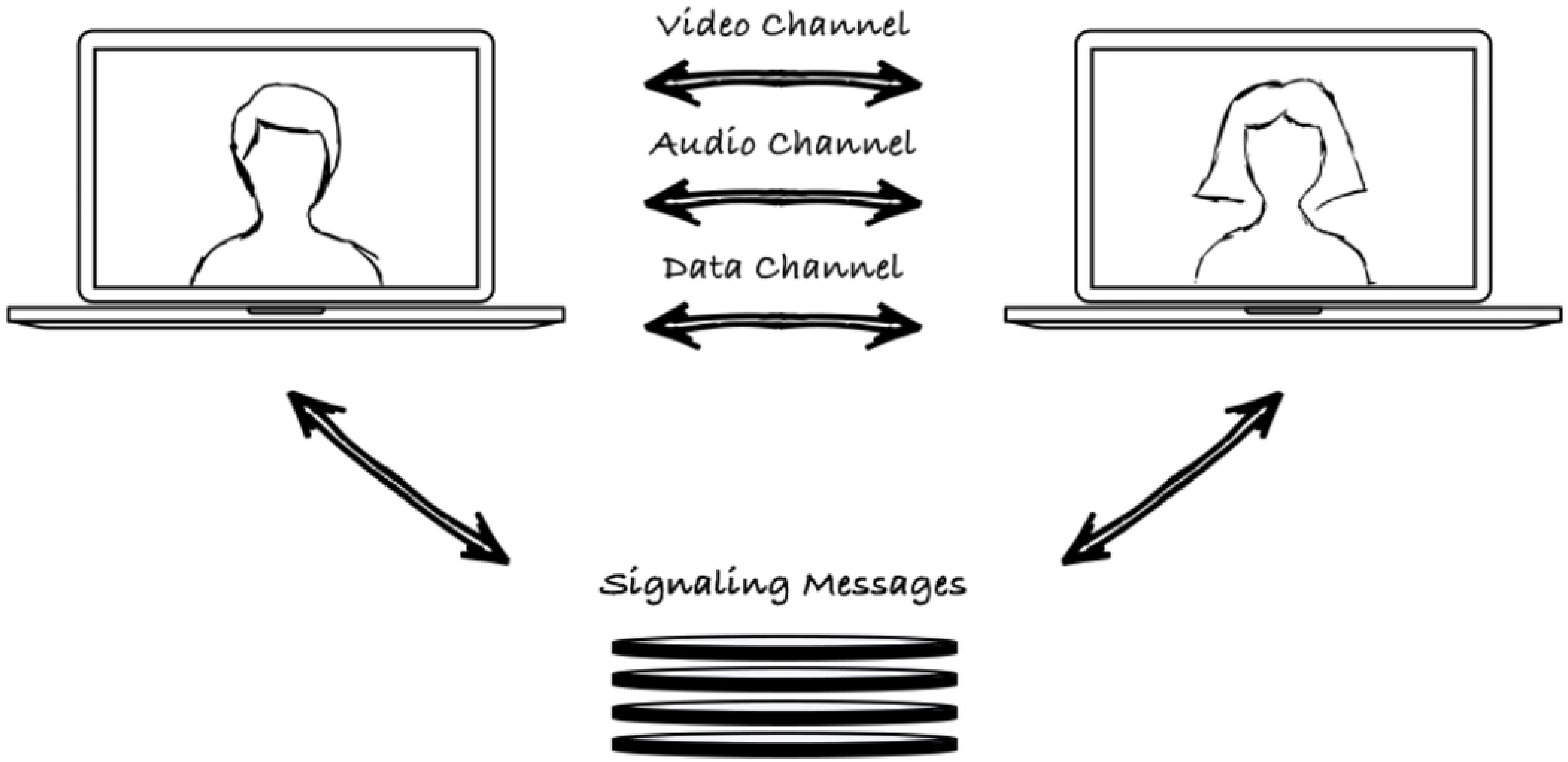
This is where WebRTC comes into play.

WEB RTC HIGH LEVEL BEHAVIOUR

With WebRTC, the heavy lifting is all done for you. The API brings a host of technologies into the browser to make implementation details easy. This includes **camera and microphone capture**, **video and audio encoding and decoding**, **transportation layers**, and **session management**.

THE GREAT PROMISE OF WebRTC

WebRTC AT A HIGH LEVEL



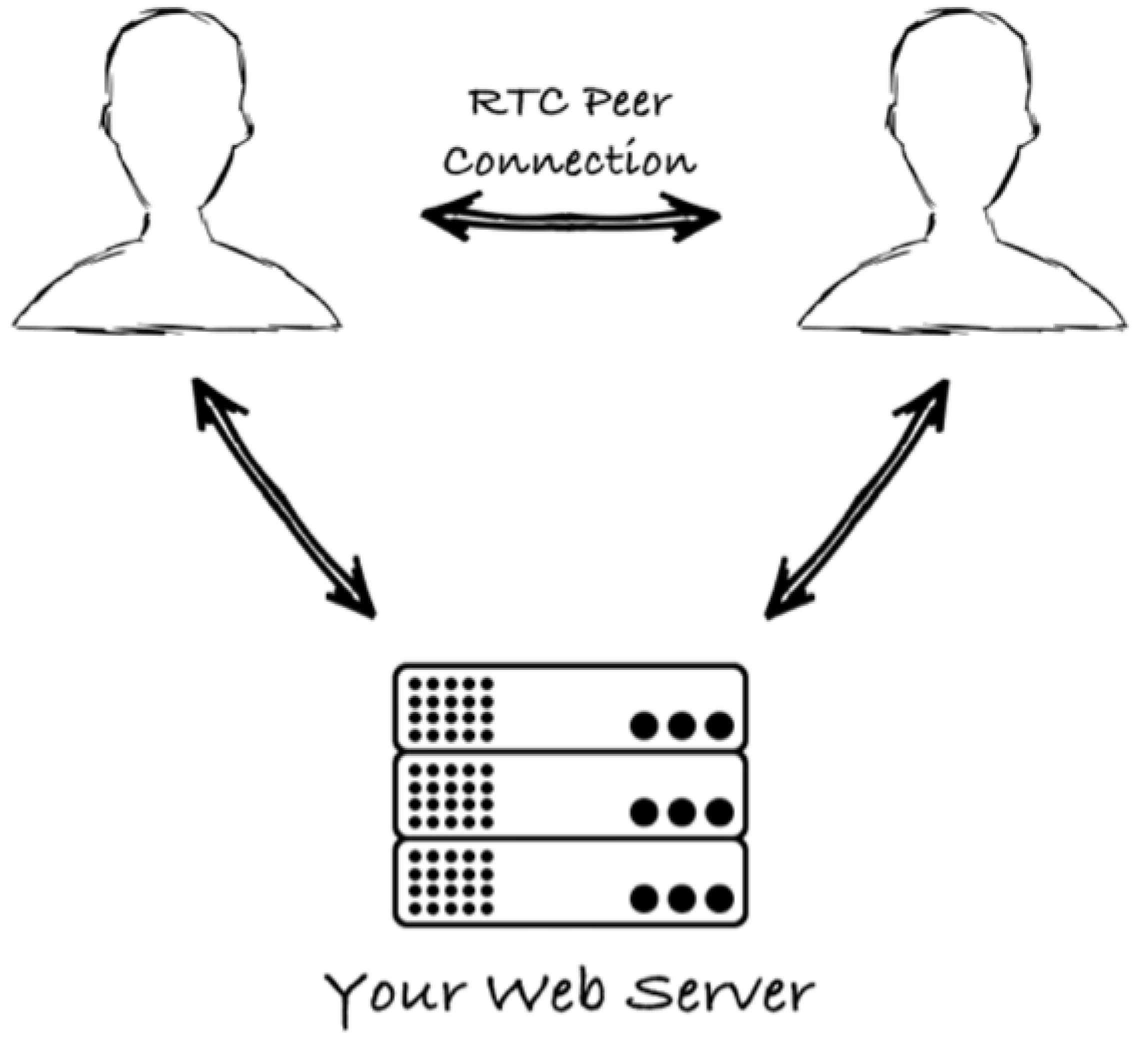
getUserMedia

```
navigator.webkitGetUserMedia
({ audio: true, video: true}, gotStream);
```

gotStream stores the stream for display…

```
video.src = window.URL.createObjectURL(stream);
```

PEER-TO-PEER IS KEY



NOW IS THE TIME TO LEARN WebRTC

POSITIVE: NO PLUGINS

WEBRTC

- **The current status of the audio and video space**
- **The role that WebRTC plays in changing this space**
- **The major features of WebRTC and how they can be used**

AUDIO

- Communicating with audio and video is a fairly common task with a history of technologies and tools.
- For a good example of audio communication, just take a look at a cell phone carrier.
- Large phone companies have established large networks of audio communication technology to bring audio communication to millions of people across the globe.
- These networks are a great example when it comes to showing widespread audio communication at its finest.

VIDEO

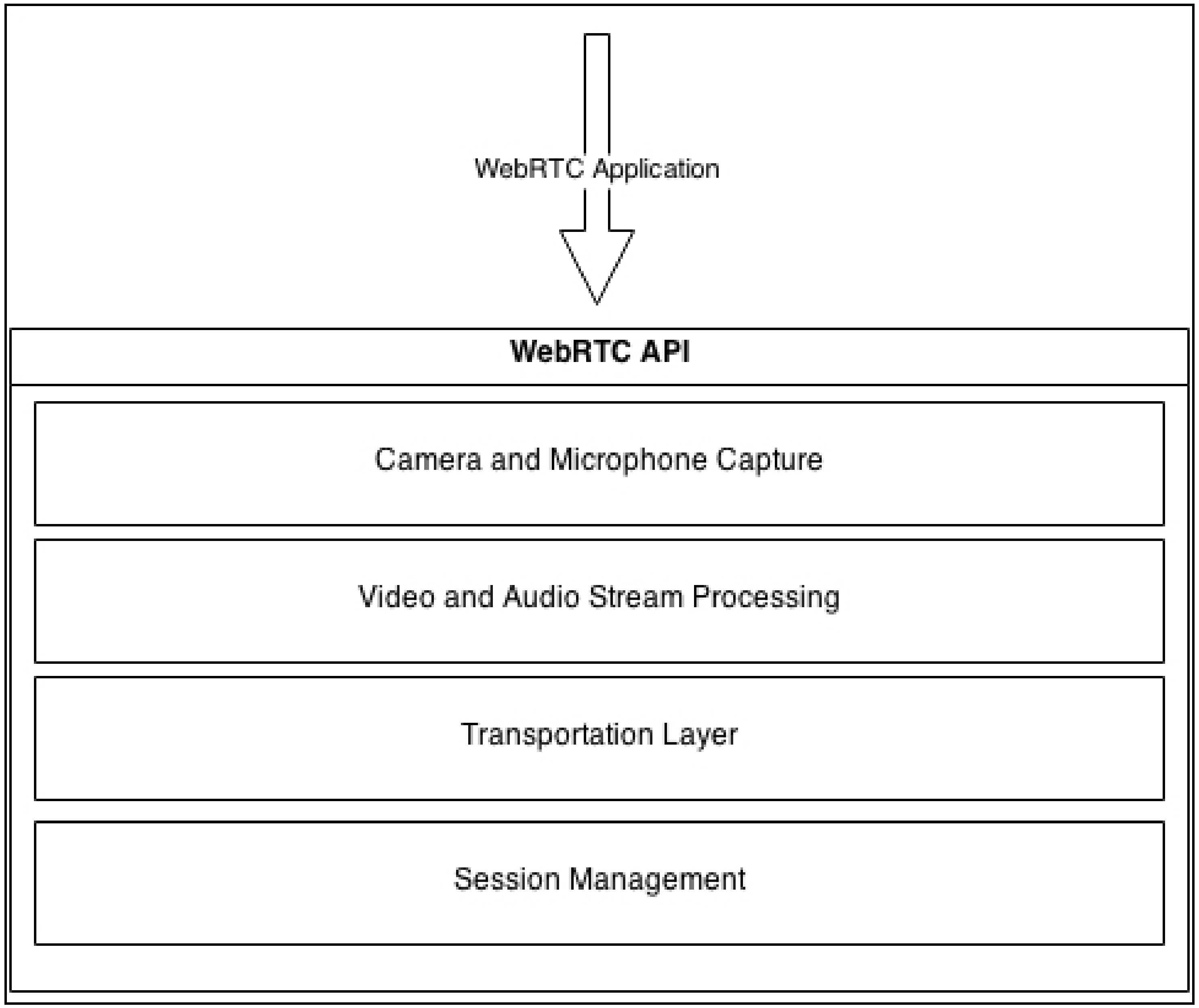
- Video communication is also becoming just as prevalent as audio communication.
- With technologies such as Apple's FaceTime, Google Hangouts, and Skype video calling, speaking to someone over a video stream is a simple task for an everyday user.
- A wide range of techniques have been developed in these applications to ensure that the quality of the video is an excellent experience for the user.
- There have been engineering solutions to problems, such as losing data packets, recovering from disconnections, and reacting to changes in a user's network.

AUDIO + VIDEO

- **The aim of WebRTC is to bring all of this technology into the browser.**
- **Many of these solutions require users to install plugins or applications on their PCs and mobile devices.**
- **They also require developers to pay for licensing, creating a huge barrier and deterring new companies to join this space.**
- **With WebRTC, the focus is on enabling this technology for every browser user without the need for plugins or hefty technology license fees for developers.**
- **The idea is to be able to simply open up a website and connect with another user right then and there.**

AUDIO + VIDEO ON THE WEB

- The biggest accomplishment of WebRTC is bringing high-quality audio and video to the open the Web without the need for third-party software or plugins.
- Currently, there are no high-quality, well-built, freely available solutions that enable real-time communication in the browser.
- The success of the Internet is largely due to the high availability and open use of technologies, such as HTML, HTTP, and TCP/IP.
- To move the Internet forward, we want to continue building on top of these technologies. This is where WebRTC comes into play.


WebRTC Application
WebRTC API
Camera and Microphone Capture
Video and Audio Stream Processing
Transportation Layer
Session Management

WEB STANDARDS

- The great thing about the Web is that it moves so fast.
- New standards are changed or created everyday and it is always improving.
- Browsers have further improved on this concept by allowing updates to be downloaded and installed without the user ever knowing.
- This makes the web developer's job an easier one, but it does mean that you have to keep up with what is going on in the world of the Web, and this includes WebRTC.
- The two organizations that control the standards for WebRTC are the World Wide Web Consortium (W3C) and the Internet Engineering Task Force (IETC).

BROWSER SUPPORT

- **Although the goal of WebRTC is to be ubiquitous for every user, this does not mean that every browser all the same features at the same time.**
- **Different browsers may choose to be ahead of the curve in certain areas, which makes some things work in one browser and not another.**

There are multiple websites that can tell you if your browser supports a specific technology, such as http://caniuse.com/rtcpeerconnection, that tells you which browsers support WebRTC.

BROWSER SUPPORT - DESKTOP

- Google Chrome Version 23 and higher
- Mozilla Firefox Version 22 and higher
- Opera Version 18 and higher
- Internet Explorer can support WebRTC yet using the Chrome Component

BROWSER SUPPORT - ANDROID

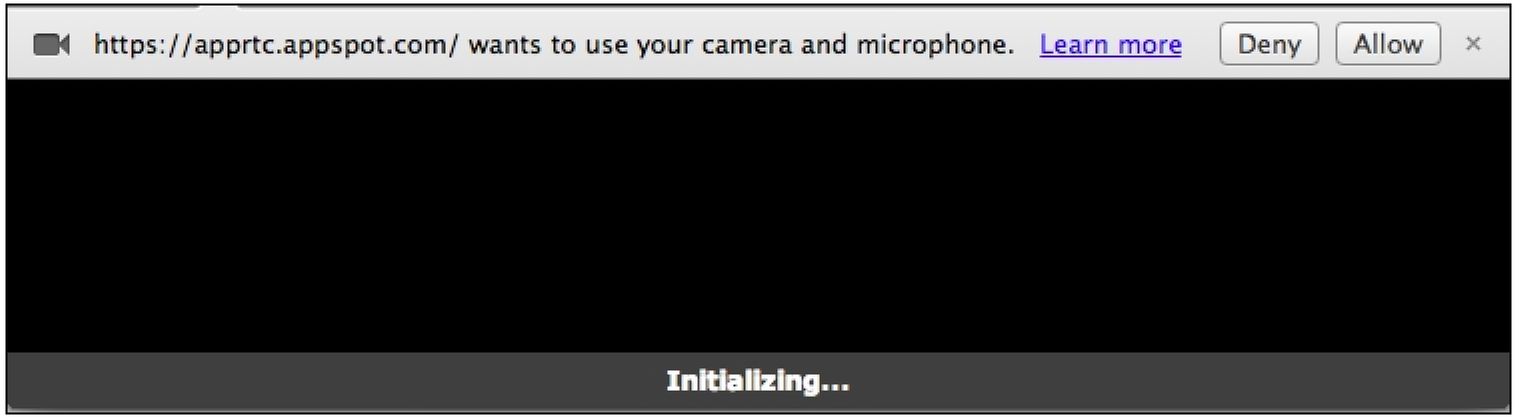
- Google Chrome Version 29 and higher
- Mozilla Firefox Version 24 and higher
- Opera Mobile Version 12 and higher
- Google Chrome OS

BROWSER SUPPORT - IOS

- Since the iPhone and iPad have different rules and limitations, particularly around video, I’d strongly recommend that you test your app on both devices. It’s probably smarter to start by getting it working fully on the iPhone, which seems to have more limitations than the iPad.
- As of this writing, the WebRTC is really only predictably solid on Safari and still has quite a few bugs on IOS for any other browser
- Feel free to try the Mini-Lab in Chrome on an iPhone, if you are skeptical

More background on this here: https://webkit.org/blog/6784/new-video-policies-for-ios

MINI-LAB


https://apprtc.appspot.com/ wants to use your camera and microphone.
Learn more
Deny
Allow
×
Initializing...

- Now that you know which browser to use, we will jump right in and try out WebRTC right now!
- Navigate your browser to the demo application available at https://apprtc.appspot.com/
- Get a “room” number (maybe random) and join with someone else in the class

ENABLEMENT

- Under the hood, WebRTC enables a basic peer-to-peer connection between two browsers.
- This is the heart of everything that happens with WebRTC. It is the first truly peer-to-peer connection inside a browser.
- This also means that anything you can do with peer connections can be easily extended to WebRTC.
- Many applications today use peer-to-peer capabilities, such as file sharing, text chat, multiplayer gaming, and even currencies.
- There are already hundreds of great examples of these types of applications working right inside the browser.

ENABLEMENT

- Most of these applications have one thing in common—they need a low-latency, high-performance connection between two users.
- WebRTC makes use of low-level protocols to deliver high-speed performance that could not be achieved otherwise.
- This speeds up data flow across the network, enabling large amounts of data to be transferred in a short amount of time.

ENABLEMENT

- WebRTC also enables a secure connection between two users to enable a higher level of privacy between them.
- Traffic traveling across a peer connection will not only be encrypted, but will also take a direct route to the other user.
- This means that packets sent in different connections might take entirely different routes over the Internet.
- This gives anonymity to users of WebRTC applications that is otherwise hard to guarantee when connecting to an application server.

SUMMARY

POP QUIZ: WEBRTC

Which of the following is not a feature that the browser provides through WebRTC?

1. Camera and microphone capture
2. Video and audio stream processing
3. Accessing a contact list
4. Session management

1 MINUTE

POP QUIZ: WEBRTC

Which of the following is not a feature that the browser provides through WebRTC?

1. Camera and microphone capture
2. Video and audio stream processing
3. **Accessing a contact list**
4. Session management

2 MINUTE

POP QUIZ: WEBRTC

WebRTC is built to be used by enterprise applications for large monolithic applications?

True or False

1 MINUTE

POP QUIZ: WEBRTC

WebRTC is built to be used by enterprise applications for large monolithic applications?

True or **False**

WebRTC is created to simplify development of streaming applications for small, medium, and enterprise organizations

1 MINUTE

SECTION 2: ACCESSING THE USER’S MEDIA

- **Getting access to media devices**
- **Constraining the media stream**
- **Handling multiple devices**
- **Modifying the stream data**

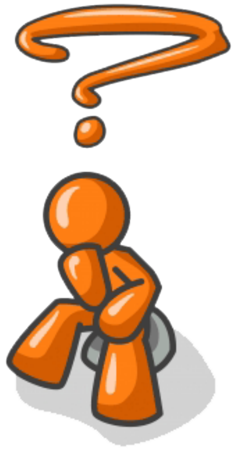
KEY POINTS OF API FUNCTIONALITY

- It provides a stream object that represents a real-time media stream, either in the form of audio or video
- It handles the selection of input devices when there are multiple cameras or microphones connected to the computer
- It provides security through user preferences and permissions that ask the user before a web page can start fetching a stream from a computer's device

KEY POINTS OF API FUNCTIONALITY

- **getUserMedia:** This component allows a web browser to access the camera and microphone
- **PeerConnection:** This component sets up audio/video calls
- **DataChannels:** This component allows browsers to share data through peer-to-peer connections

WHAT IS NODEJS

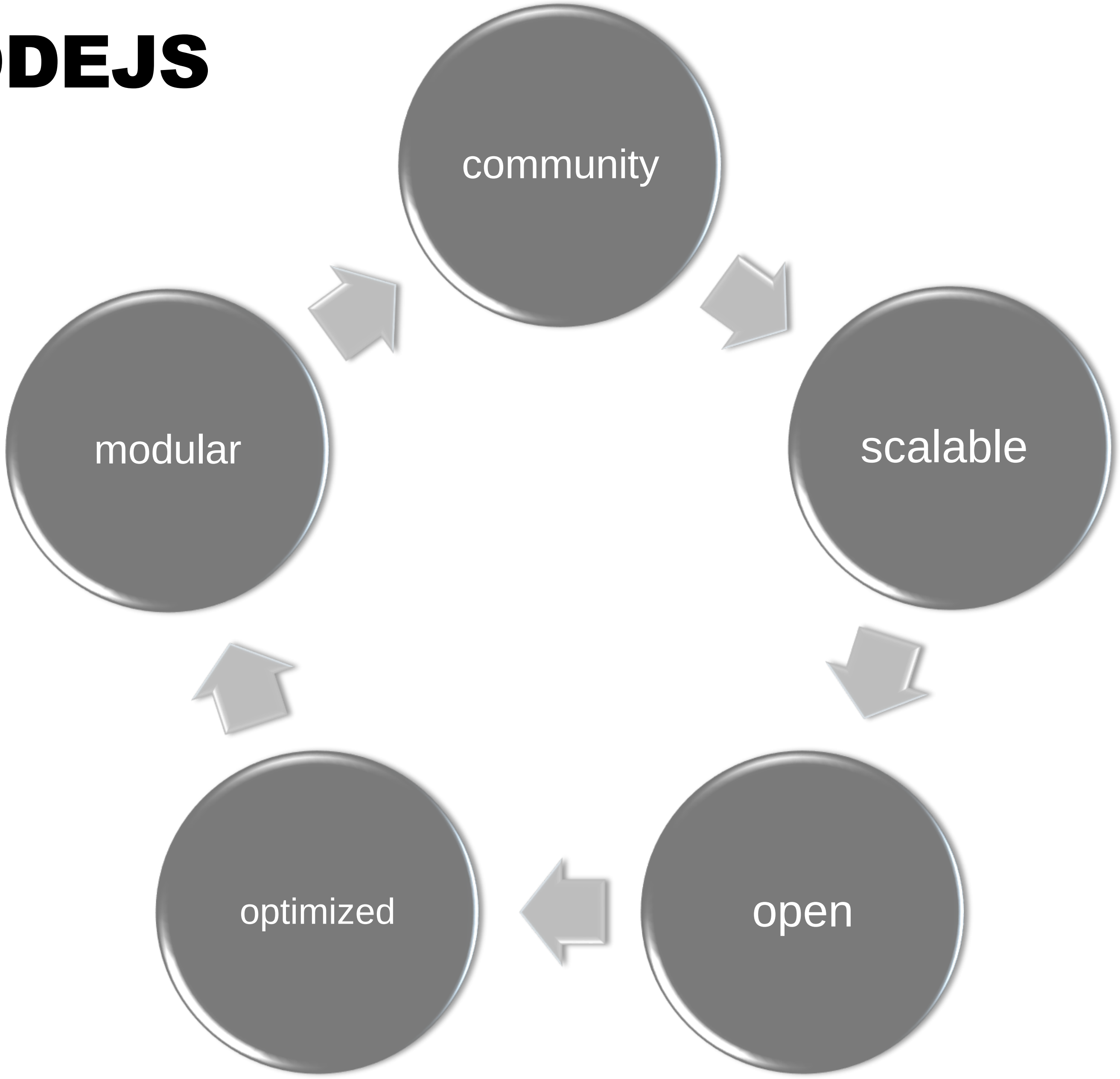


An Open source, Cross platform, Event Based and Non-blocking framework used to develop server side and networking applications

Node.js applications are written in JavaScript and can be run with its own runtime in Windows, Linux, OS X

It is not a web server but it is an alternative way to run your code on computer

WHY NODEJS



TYPE OF APPLICATION YOU CAN BUILD WITH NODEJS

- Messaging middleware
- Web Application frameworks
- Servers for HTML5 multi player games
- Functions-as-a-Service
- Streaming audio/video (RTC)
- Cross-platform programs
- Static file servers

LAB: SETTING UP A STATIC SERVER

1. Create a ***server.js*** file in your project's root directory and add the code from the following slide.
2. Add a folder named ***public*** to your project.
3. Copy/paste the HTML snippet from the following slide into ***./public/index.html*** as well to verify the static server is working. The public folder needs to be in the same directory as the server.js file that you created.
4. Start the server by running ***node server.js*** from the command-line. Be sure to change to your project directory before running this.
5. Open your browser to localhost:8080, and voila our first node static server project is running.
6. That's it! You should be all set with a simple node static server setup now. To give credit where it's due, this setup is leveraging the ***node-static*** Node.js module.

LAB: SETTING UP A STATIC SERVER

1. **Visit the Node.js website at http://nodejs.org/. There should be a big INSTALL button on the home page that will help you with installing Node.js on your OS.**
2. **Once Node.js is installed on your system, you will also have the package manager for Node.js installed npm.**
3. **Open up a terminal or command line interface and type**

   npm install --save node-static

```
C:\nodejs>npm install --save node-static
C:\Users\george.niece\AppData\Roaming\npm\static ->
C:\Users\george.niece\AppData\Roaming\npm\node_modules\node-static\bin\cli.js
+ node-static@0.7.11
added 6 packages from 8 contributors in 0.886s
```

4. **Now you can navigate to any directory that contains the HTML files you would like to host on the server, initially we can create them from the nodejs installation folder.**
5. **Run the static command to start a static web server in this directory. You can navigate to http://localhost:8080 to see your file in the browser!**

LAB: SERVER.JS

```
/**
 * Static HTTP Server - create a static file server instance to serve files */

// modules
var static = require( 'node-static'), port = 8080, http = require( 'http' );

// config
var file = new static.Server( './public', {  cache: 3600, gzip: true } );

// serve
http.createServer( function ( request, response ) {
    request.addListener( 'end', function () {
        file.serve( request, response );
    } ).resume();
} ).listen( port );
```

LAB: ./PUBLIC/INDEX.HTML

```
<!DOCTYPE html>
<html>
<head>
    <title>Simple, Static Node.js Server</title>
</head>
<body>
<h1>Simple, Static Node.js Server</h1>
</body>
</html>
```

CREATE YOUR FIRST MEDIASTREAM PAGE

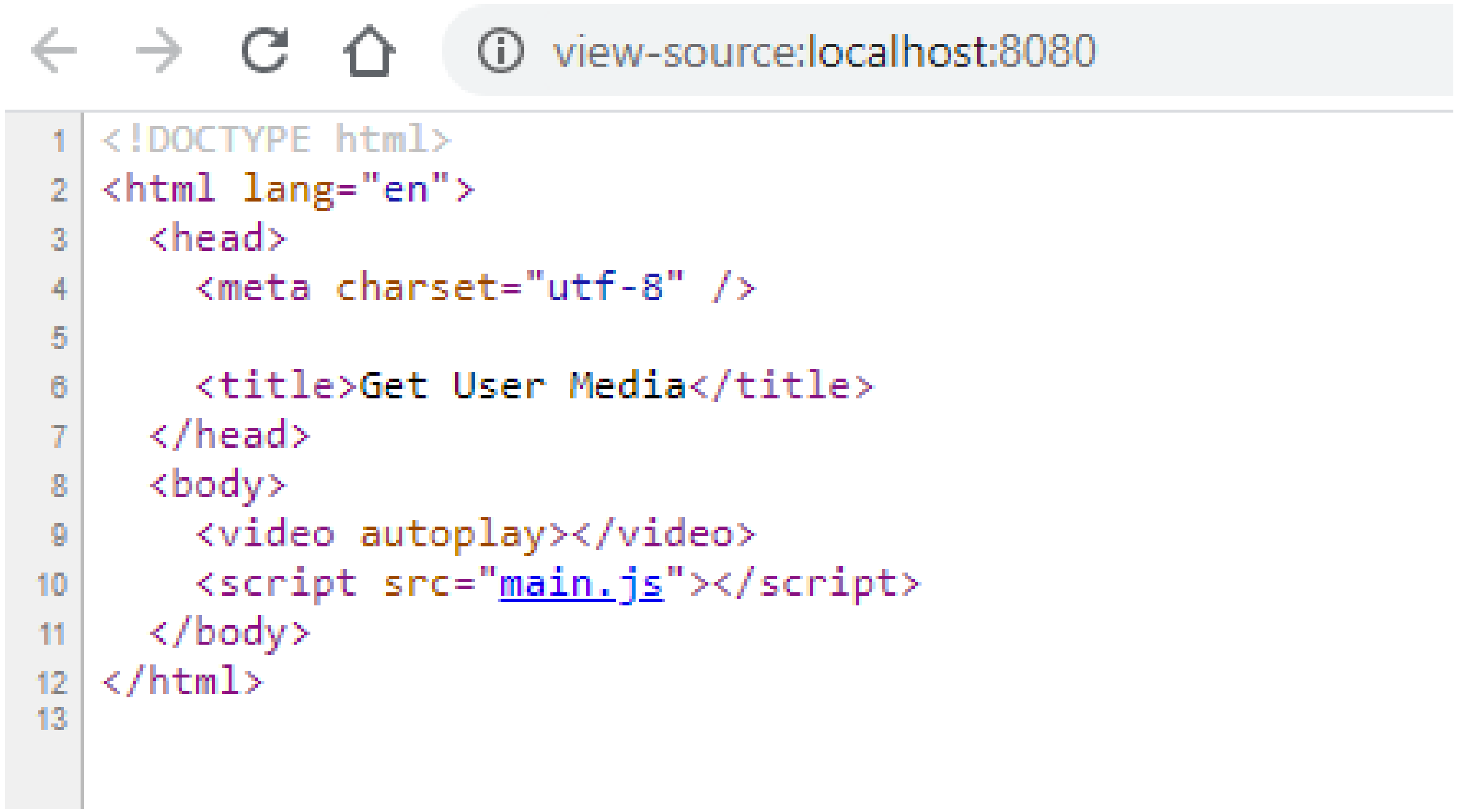
- **Our first WebRTC-enabled page will be a simple one.**
- **It will show a single <video> element on the screen, ask to use the user's camera, and show a live video feed of the user right in the browser. The video tag is a powerful HTML5 feature in itself.**
- **It will not only allow us to see our video on the screen, but can also be used to play back a variety of video sources.**

LAB: CREATE MEDIASTREAM PAGE

We will start by creating a simple HTML page with a video element contained in the body tag. Create a file named index.html and type the following:

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8" />
    <title>Session 2: Get User Media</title>
  </head>
  <body>
    <video autoplay></video>
    <script src="main.js"></script>
  </body>
 </html>
```

LAB: RUN IT!

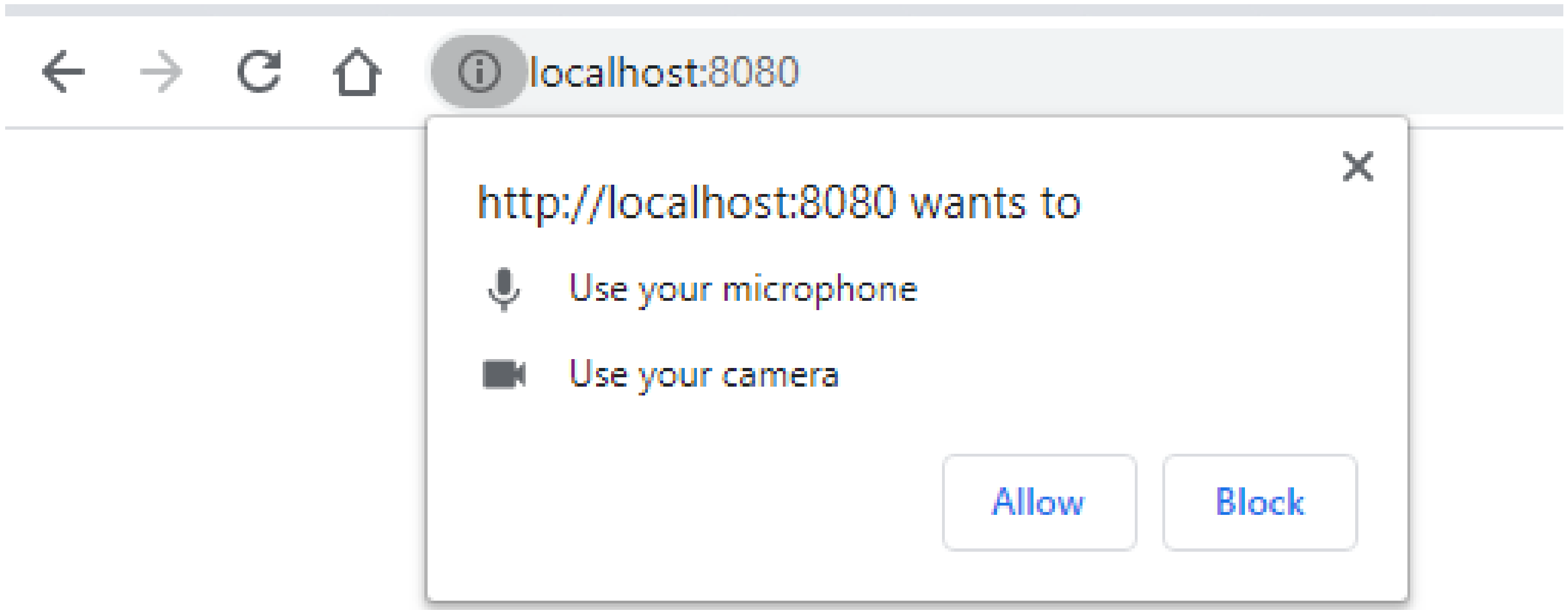


```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8" />

    <title>Get User Media</title>
  </head>
  <body>
    <video autoplay></video>
    <script src="main.js"></script>
  </body>
</html>
```

LAB: MAIN.JS

```
function hasUserMedia() {
  return !!(navigator.getUserMedia || navigator.webkitGetUserMedia || navigator.mozGetUserMedia ||
navigator.msGetUserMedia);
}
if (hasUserMedia()) {
  navigator.getUserMedia = navigator.getUserMedia || navigator.webkitGetUserMedia || navigator.mozGetUserMedia ||
navigator.msGetUserMedia;
  navigator.getUserMedia({ video: true, audio: true }, function (stream) {
    var video = document.querySelector('video');
    video.srcObject = stream;
  }, function (err) {});
} else {
  alert("Sorry, your browser does not support getUserMedia.");
 }
```

LAB: RUN IT!

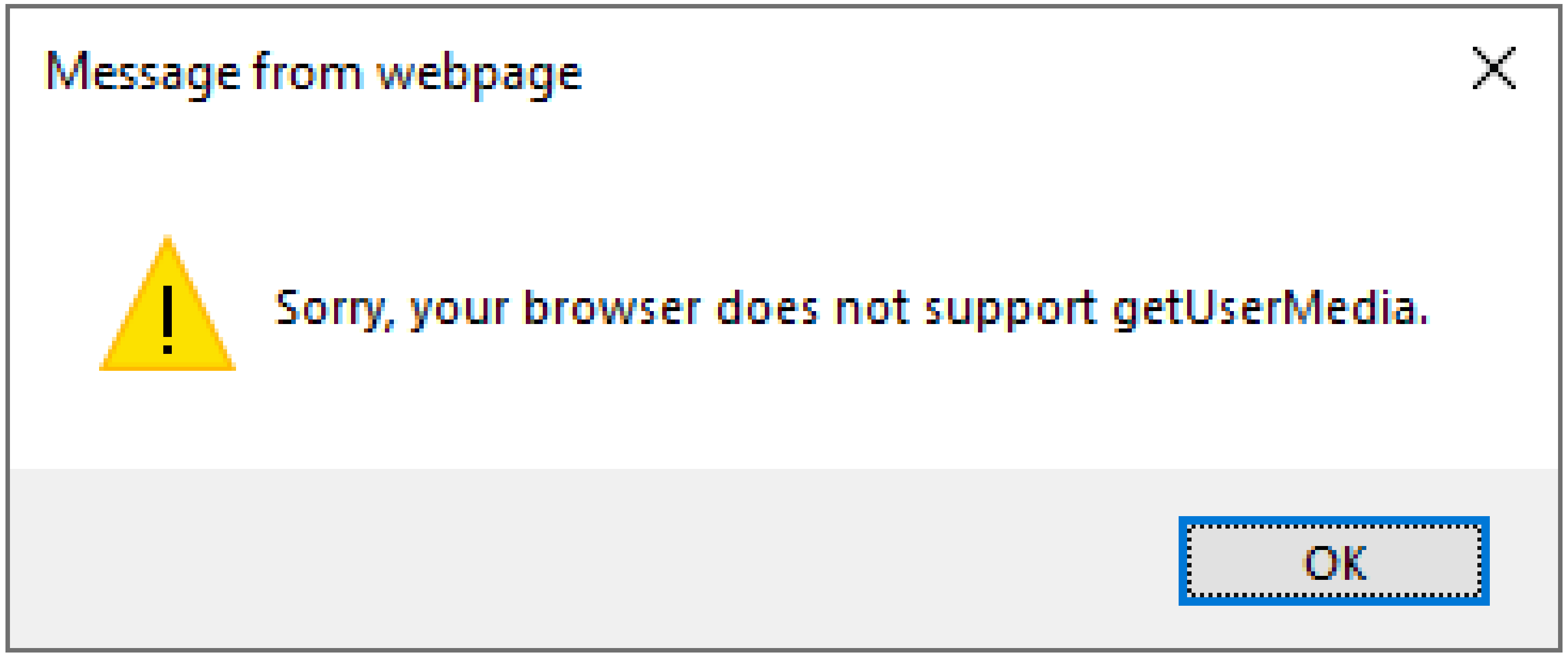

localhost:8080

http://localhost:8080 wants to

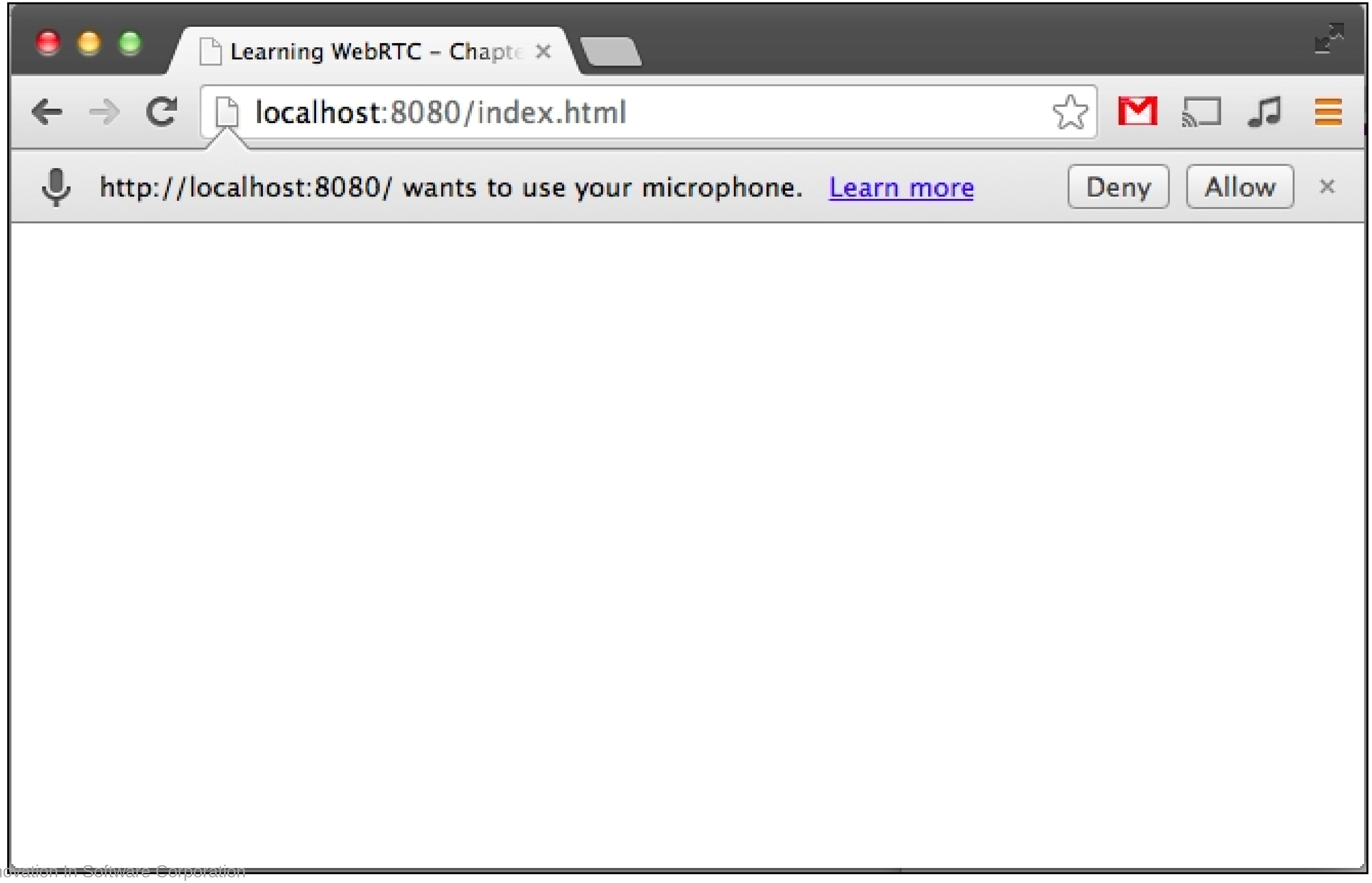
Use your microphone

Use your camera

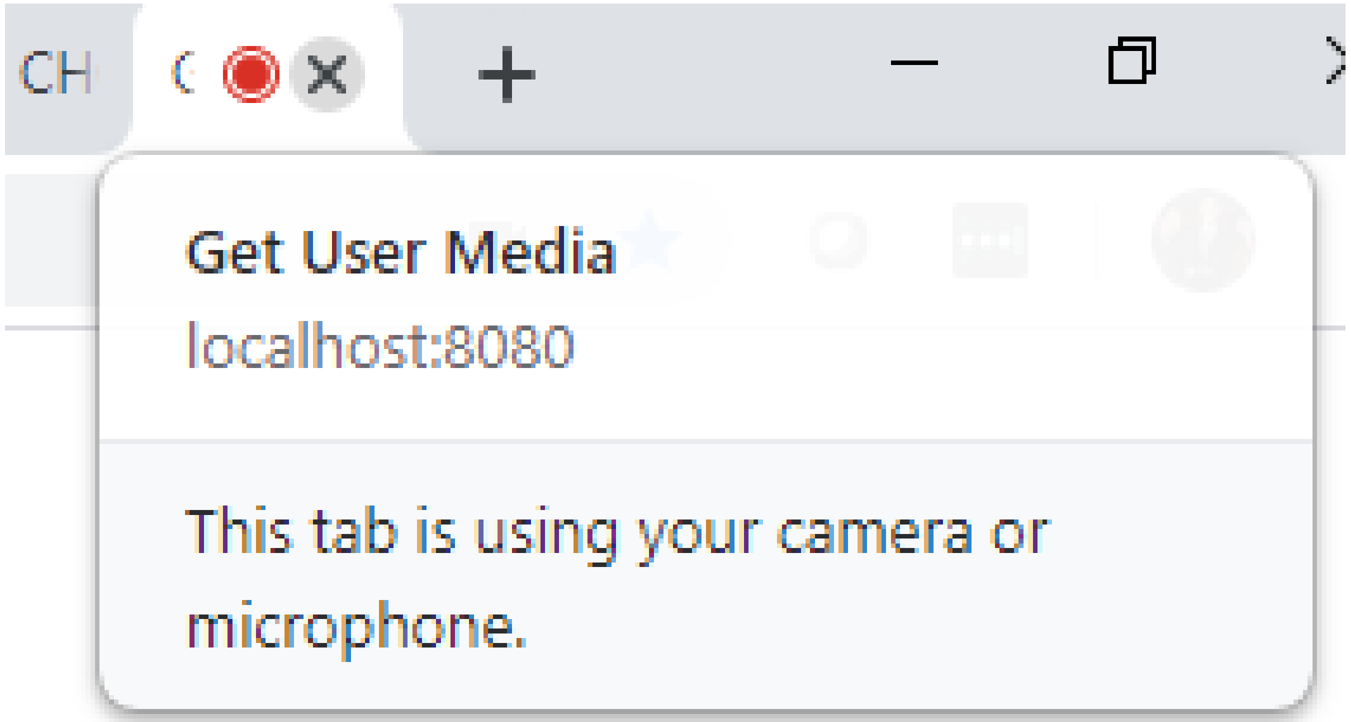
LAB: RUN IT!


Message from webpage

Sorry, your browser does not support getUserMedia.


Learning WebRTC - Chapte
localhost:8080/index.html
http://localhost:8080/ wants to use your microphone.
Learn more
Deny
Allow

GET USER MEDIA IN ACTION


Get User Media

localhost:8080

This tab is using your camera or microphone.

LET’S ANALYZE: MAIN.JS

```
function hasUserMedia() {

  return !!(navigator.getUserMedia || navigator.mediaDevices.webkitGetUserMedia || navigator.mozGetUserMedia ||
navigator.msGetUserMedia);

}

if (hasUserMedia()) {

  navigator.getUserMedia = navigator.getUserMedia || navigator.mediaDevices.webkitGetUserMedia|| navigator.mozGetUserMedia
|| navigator.msGetUserMedia;

  navigator.getUserMedia({ video: true, audio: true }, function (stream) {

    var video = document.querySelector('video');

    video.srcObject = stream;

  }, function (err) {});

} else {

  alert("Sorry, your browser does not support getUserMedia.");
 }
```

CONSTRAIN THE MEDIA STREAM

- Now that we know how to get a stream from the browser, we will cover configuring this stream using the first parameter of the getUserMedia API.
- This parameter expects an object of keys and values telling the browser how to look for and process streams coming from the connected devices.
- The first options we will cover are simply turning on or off the video or audio streams:

```
navigator.getUserMedia({ video: false, audio: true }, function (stream) {
   // Now our stream does not contain any video!
 });
```

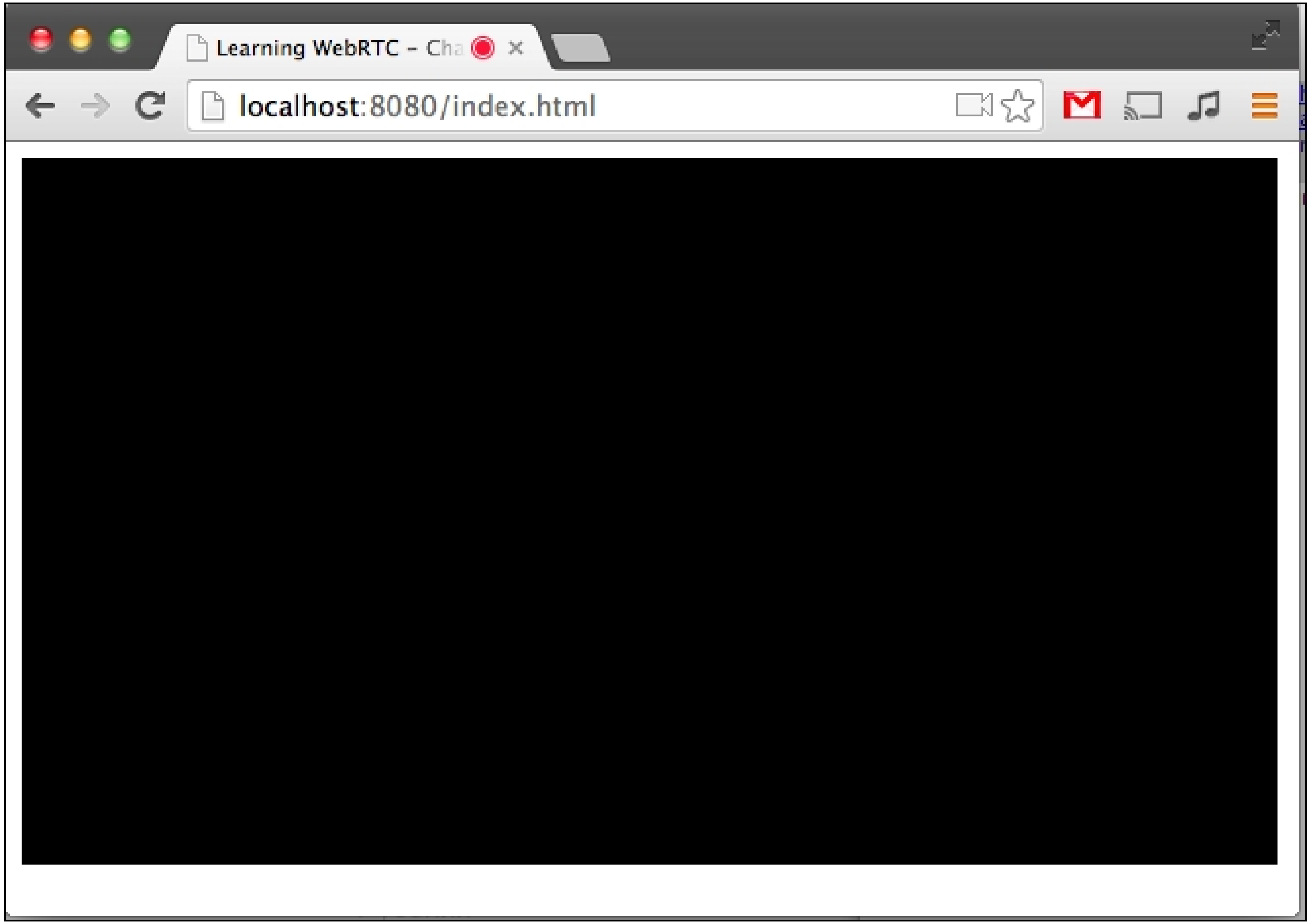
CONSTRAINING THE VIDEO CAPTURE

- **The options for constraining the getUserMedia API not only allows the true or false values, but also allows you to pass in an object with a complex set of constraints.**
  - You can see the full set of constraints provided in the specification detailed at **https://tools.ietf.org/html/draft-alvestrand-constraints-resolution-03**.
- **These allow you to constrain options such as minimum required resolution and frameRate, video aspect ratio, and optional parameters all through the configuration object passed into the getUserMedia API.**

STREAM POLITENESS BEST PRACTICES

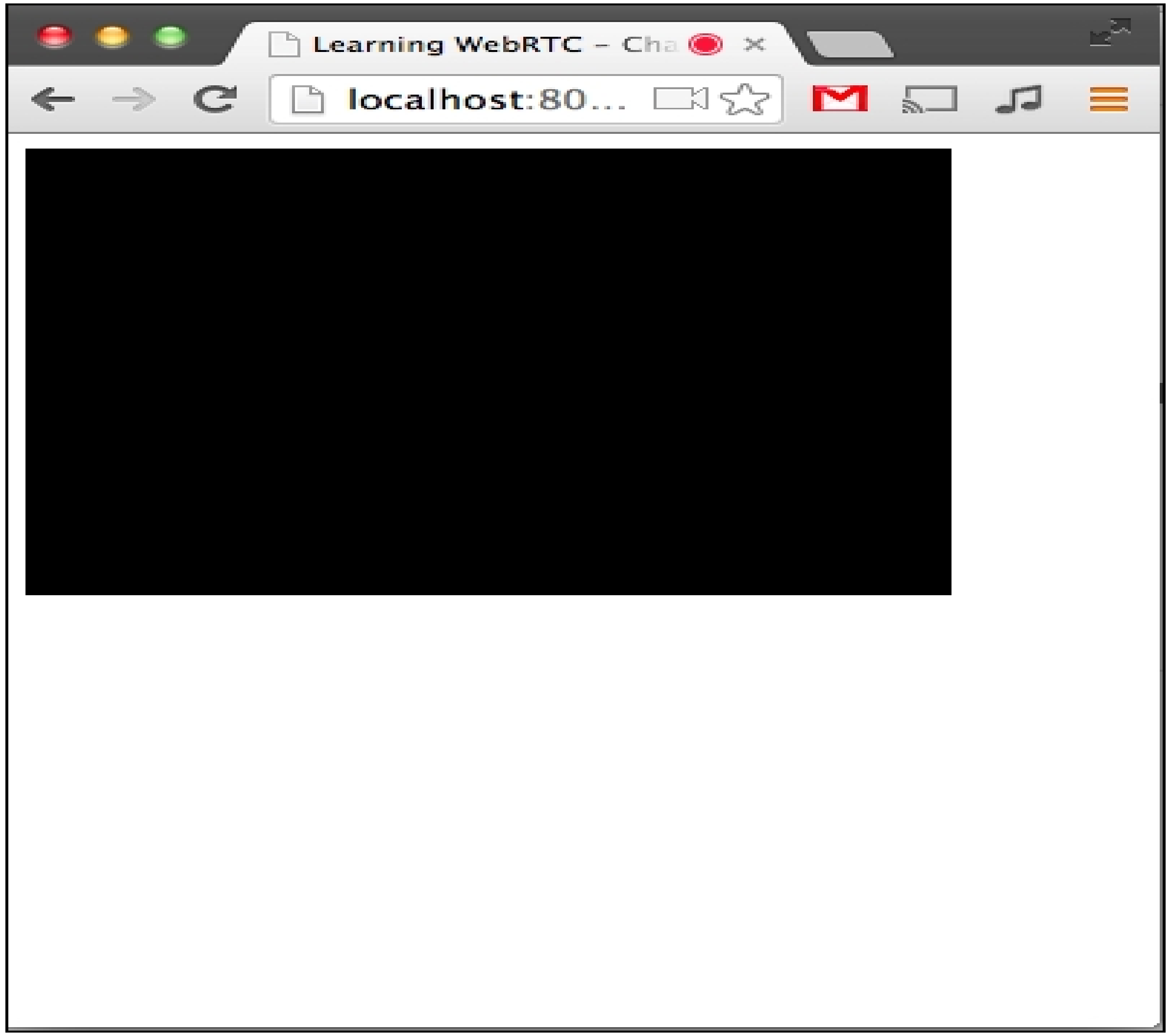
- Asking for a minimum resolution in order to create a good user experience for everyone participating in a video call
- Providing a certain width and height of a video in order to stay in line with a particular style or brand associated with the application
- Limiting the resolution of the video stream in order to save computational power or bandwidth if on a limited network connection

```
navigator.getUserMedia({
    video: {
      mandatory: {
        minAspectRatio: 1.777,
        maxAspectRatio: 1.778
      },
      optional: [
        { maxWidth: 640 },
        { maxHeigth: 480 }
      ]
    },
    audio: false
  }, function (stream) {
    var video = document.querySelector('video');
    video.srcObject = stream;
  }, function (error) {
    console.log("Raised an error when capturing:", error);
   });
```


Learning WebRTC - Cha
localhost:8080/index.html

HTML FOR MULTIPLE MEDIA

```
< !DOCTYPE html >
  <html lang="en">
    <head>
      <meta charset="utf-8" />

      <title>Section 2: Multiple Media</title>
      <style>
        video {
          object - fit: cover;
         }

  @media (min-width: 1000px) {
          video {
          height: 480px;
         }
        }
 </style>
    </head>
    <body>
      <div class="select">
        <label for="audioSource">Audio source: </label><select id="audioSource"></select>
      </div>

      <div class="select">
        <label for="videoSource">Video source: </label><select id="videoSource"></select>
      </div>

      <video autoplay muted playsinline></video>

      <script async src="multiple-sources.js"></script>
    </body>
  </html>
```


Learning WebRTC - Cha
localhost:80...

MULTIPLE DEVICES

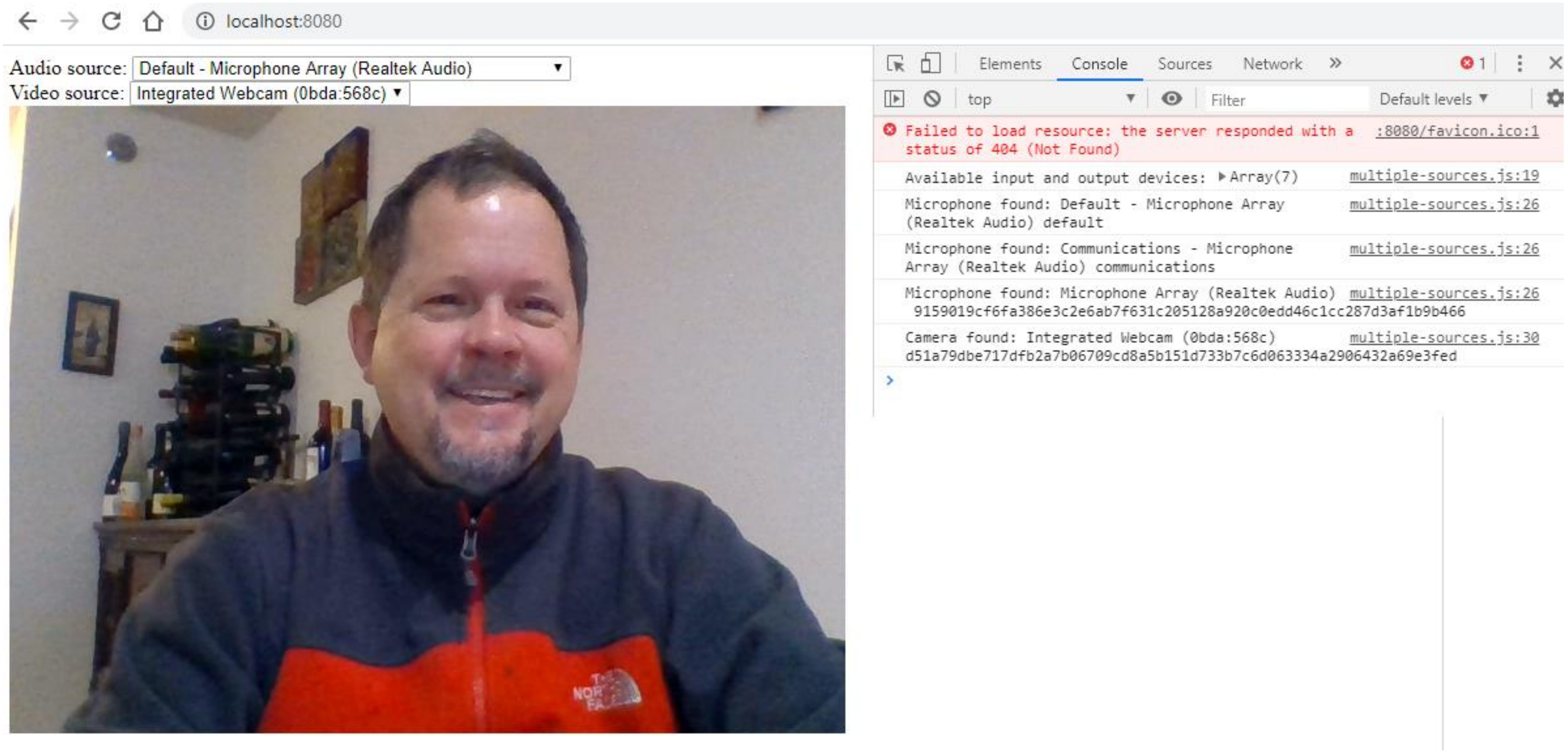
- In some cases, users may have more than one camera or microphone attached to their device.
- This is especially the case on mobile devices that often have a front-facing camera and a rear-facing one.
- In this case, you want to search through the available cameras or microphones and select the appropriate device for your user's needs.
- Fortunately, to do this, an API called MediaSourceTrack is exposed to the browser.

MULTIPLE MEDIA SCRIPTING

```
var videoElement = document.querySelector('video');
var audioSelect = document.querySelector('select#audioSource');
var videoSelect = document.querySelector('select#videoSource');

audioSelect.onchange = getStream;
videoSelect.onchange = getStream;

getStream().then(getDevices).then(gotDevices);

function getDevices() {
    // AFAICT in Safari this only gets default devices until gUM is called :/
    return navigator.mediaDevices.enumerateDevices();
}

function gotDevices(deviceInfos) {
    window.deviceInfos = deviceInfos; // make available to console
    console.log('Available input and output devices:', deviceInfos);
    for (const deviceInfo of deviceInfos) {
        const option = document.createElement('option');
        option.value = deviceInfo.deviceId;
        if (deviceInfo.kind === 'audioinput') {
            option.text = deviceInfo.label || `Microphone ${audioSelect.length + 1}`;
            audioSelect.appendChild(option);
            console.log("Microphone found:", deviceInfo.label, deviceInfo.deviceId);
        } else if (deviceInfo.kind === 'videoinput') {
            option.text = deviceInfo.label || `Camera ${videoSelect.length + 1}`;
            videoSelect.appendChild(option);
            console.log("Camera found:", deviceInfo.label, deviceInfo.deviceId);
        }
    }
}
```

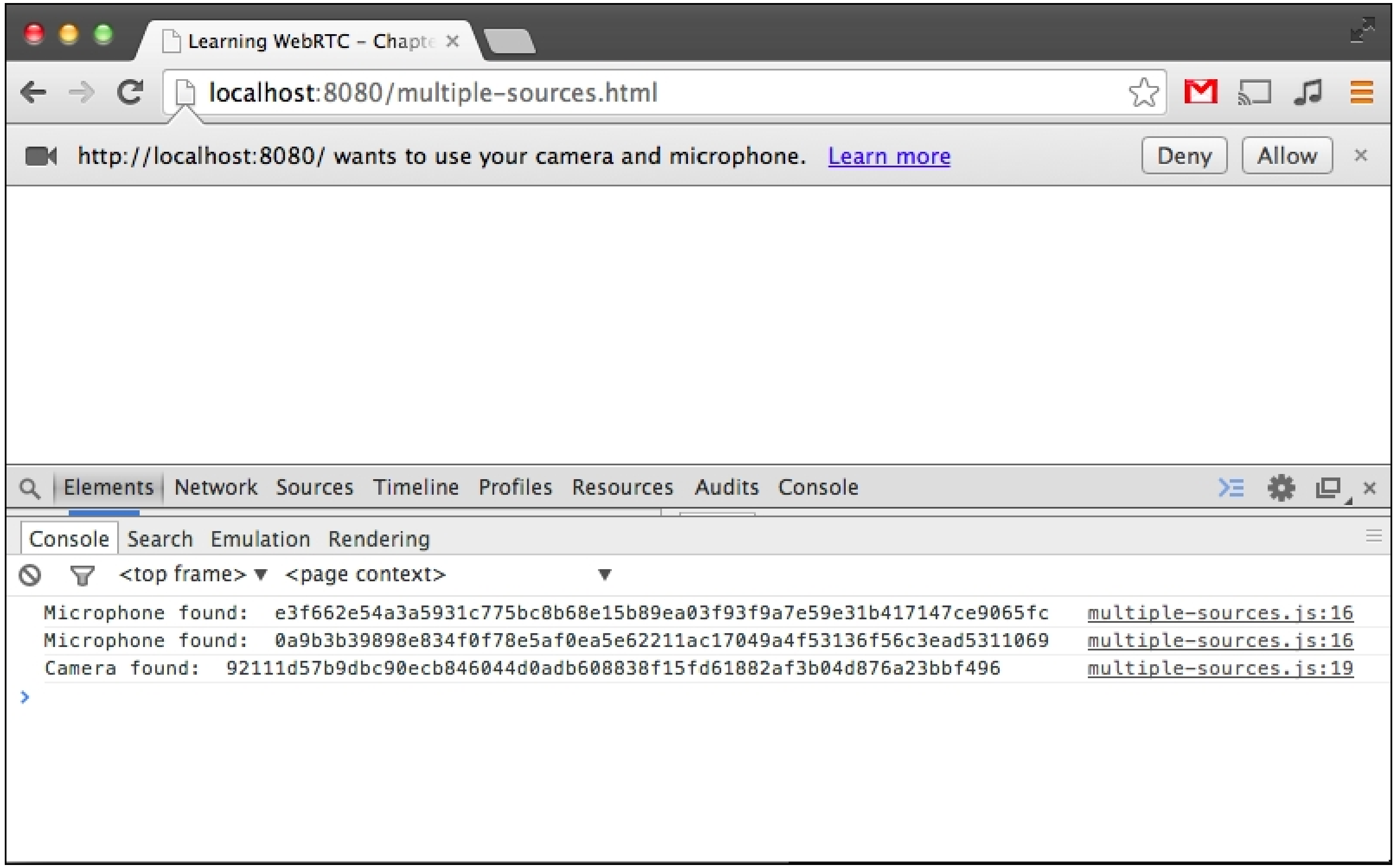
MULTIPLE MEDIA SCRIPTING CONT.

```
function getStream() {
    if (window.stream) {
        window.stream.getTracks().forEach(track => {
            track.stop();
        });
    }
    const audioSource = audioSelect.value;
    const videoSource = videoSelect.value;
    const constraints = {
        audio: { deviceId: audioSource ? { exact: audioSource } : undefined },
        video: { deviceId: videoSource ? { exact: videoSource } : undefined }
    };
    return navigator.mediaDevices.getUserMedia(constraints).
        then(gotStream).catch(handleError);
}

function gotStream(stream) {
    window.stream = stream; // make stream available to console
    audioSelect.selectedIndex = [...audioSelect.options].
        findIndex(option => option.text === stream.getAudioTracks()[0].label);
    videoSelect.selectedIndex = [...videoSelect.options].
        findIndex(option => option.text === stream.getVideoTracks()[0].label);
    videoElement.srcObject = stream;
}

function handleError(error) {
    console.error('Error: ', error);
}
```



Audio source: Default - Microphone Array (Realtek Audio)
Video source: Integrated Webcam (0bda:568c)

```
Failed to load resource: the server responded with a        :8080/favicon.ico:1
status of 404 (Not Found)
Available input and output devices: ▶Array(7)          multiple-sources.js:19
Microphone found: Default - Microphone Array           multiple-sources.js:26
(Realtek Audio) default
Microphone found: Communications - Microphone          multiple-sources.js:26
Array (Realtek Audio) communications
Microphone found: Microphone Array (Realtek Audio)     multiple-sources.js:26
 9159019cf6fa386e3c2e6ab7f631c205128a920c0edd46c1cc287d3af1b9b466
Camera found: Integrated Webcam (0bda:568c)            multiple-sources.js:30
d51a79dbe717dfb2a7b06709cd8a5b151d733b7c6d063334a2906432a69e3fed
```


Learning WebRTC – Chapte
localhost:8080/multiple-sources.html
Deny
Allow
Elements Network Sources Timeline Profiles Resources Audits Console
Console Search Emulation Rendering
<top frame> <page context>

http://localhost:8080/ wants to use your camera and microphone. Learn more

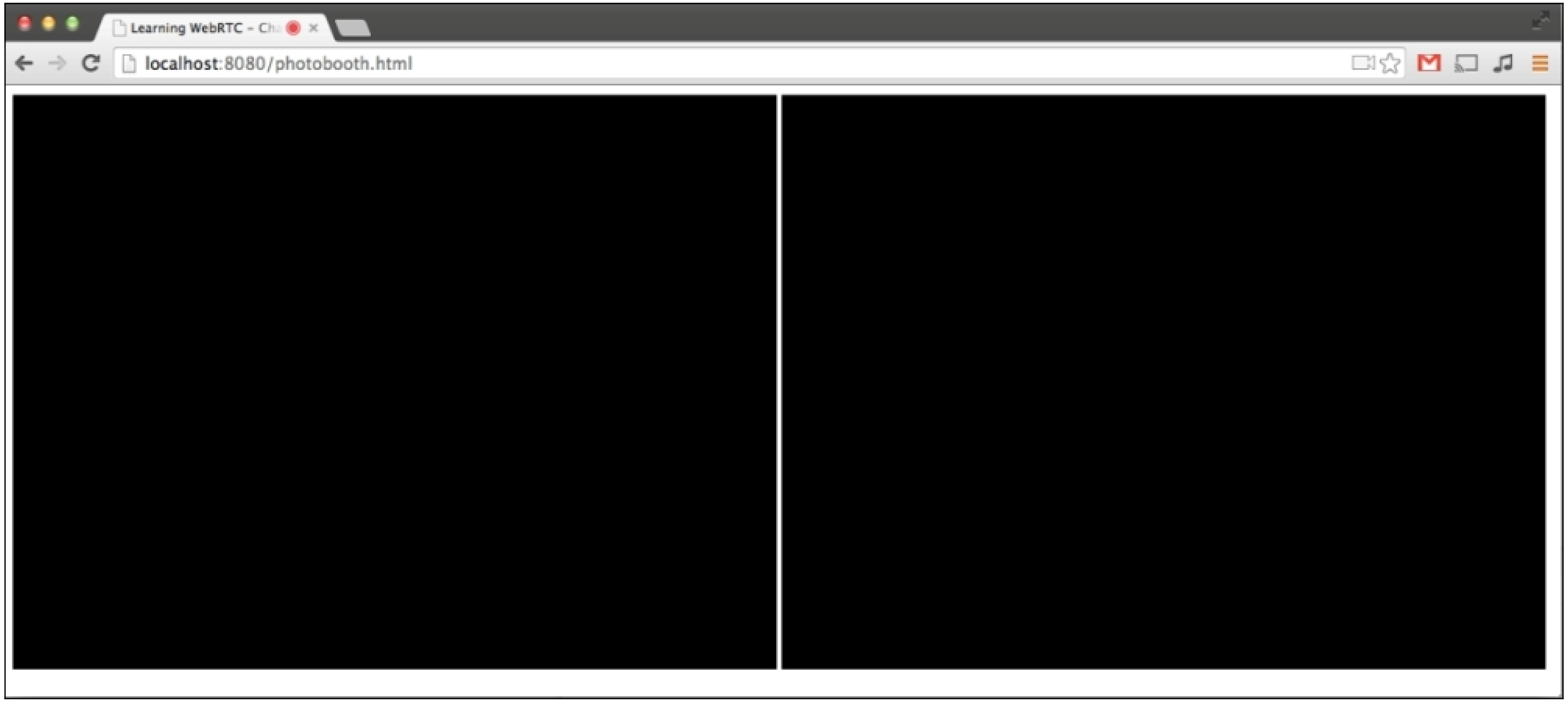
```
Microphone found:  e3f662e54a3a5931c775bc8b68e15b89ea03f93f9a7e59e31b417147ce9065fc    multiple-sources.js:16
Microphone found:  0a9b3b39898e834f0f78e5af0ea5e62211ac17049a4f53136f56c3ead5311069    multiple-sources.js:16
Camera found:  92111d57b9dbc90ecb846044d0adb608838f15fd61882af3b04d876a23bbf496        multiple-sources.js:19
>
```

WEBRTC FOR A PHOTO BOOTH

- **A photo booth application allows you to see yourself on the screen while being able to capture pictures of yourself, much like a real photo booth.**
- **The Canvas API is a set of arbitrary methods to draw lines, shapes, and images on the screen.**
- **This is popularized through the use of Canvas for games and other interactive applications across the Web.**

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8" />
    <title>Learning WebRTC - section 2: Get User Media</title>
    <style>
      video, canvas {
        border: 1px solid gray;
        width: 480px;
        height: 320px;
      }
    </style>
  </head>
  <body>
    <video autoplay></video>
    <canvas></canvas>
    <button id="capture">Capture</button>
    <script src="photobooth.js"></script>
  </body>
</html>
```

```
function hasUserMedia() {

  return !!(navigator.getUserMedia || navigator.webkitGetUserMedia || navigator.mozGetUserMedia ||
navigator.msGetUserMedia);

}

if (hasUserMedia()) {

  navigator.getUserMedia = navigator.getUserMedia || navigator.webkitGetUserMedia || navigator.mozGetUserMedia ||
navigator.msGetUserMedia;

  var video = document.querySelector('video'),

      canvas = document.querySelector('canvas'),

      streaming = false;

  navigator.getUserMedia({

    video: true,

    audio: false

  }, function (stream) {

    video.srcObject = stream;

    streaming = true;

  }, function (error) {

    console.log("Raised an error when capturing:", error);

  });
document.querySelector('#capture').addEventListener('click', function (event) {
    if (streaming) {
      canvas.width = video.clientWidth;
      canvas.height = video.clientHeight;
      var context = canvas.getContext('2d');
      context.drawImage(video, 0, 0);
    }
  });
} else {
  alert("Sorry, your browser does not support getUserMedia.");
 }
```


Learning WebRTC - Ch
localhost:8080/photobooth.html

MODIFY THE MEDIA STREAM

- We can take this project even further.
- Most image sharing applications today have some set of filters that you can apply to your images to make them look even cooler.
- This is also possible on the Web using CSS filters to provide different effects.
- We can add some CSS classes that apply different filters to our <canvas> element

```
<style>
      .grayscale {
        -webkit-filter: grayscale(1);
        -moz-filter: grayscale(1);
        -ms-filter: grayscale(1);
        -o-filter: grayscale(1);
        filter: grayscale(1);
      }

      .sepia {
        -webkit-filter: sepia(1);
        -moz-filter: sepia(1);
        -ms-filter: sepia(1);
        -o-filter: sepia(1);
        filter: sepia(1);
      }

      .invert {
        -webkit-filter: invert(1);
        -moz-filter: invert(1);
        -ms-filter: invert(1);
        -o-filter: invert(1);
        filter: invert(1);
      }
    </style>
```

```
var filters = ['', 'grayscale', 'sepia', 'invert'],
      currentFilter = 0;
  document.querySelector('video').addEventListener('click', function (event) {
    if (streaming) {
      canvas.width = video.clientWidth;
      canvas.height = video.clientHeight;

      var context = canvas.getContext('2d');
      context.drawImage(video, 0, 0);

      currentFilter++;
      if(currentFilter > filters.length - 1) currentFilter = 0;
      canvas.className = filters[currentFilter];
    }
   });
```

- With the access to apply a stream to the canvas, you have unlimited possibilities. The canvas is a low-level and powerful drawing tool, which enables features such as drawing lines, shapes, and text.
- For instance, add the following after the class name is assigned to the canvas to add some text to your images:

```
context.fillStyle = "white";
 context.fillText("Hello World!", 10, 10);
```

SUMMARY

POP QUIZ: WEBRTC

WebRTC requires browser plugins?

True or False

1 MINUTE

POP QUIZ: WEBRTC

WebRTC requires browser plugins?

True or **False**

WebRTC has none of the plugins or additional software installations most video and audio streaming applications require

1 MINUTE

POP QUIZ: WEBRTC

WebRTC is supported in all web browsers?

True or False

1 MINUTE

POP QUIZ: WEBRTC

WebRTC is supported in all web browsers on all platforms?

True or **False**

WebRTC is only supported in HTML5 browsers and not across all platforms

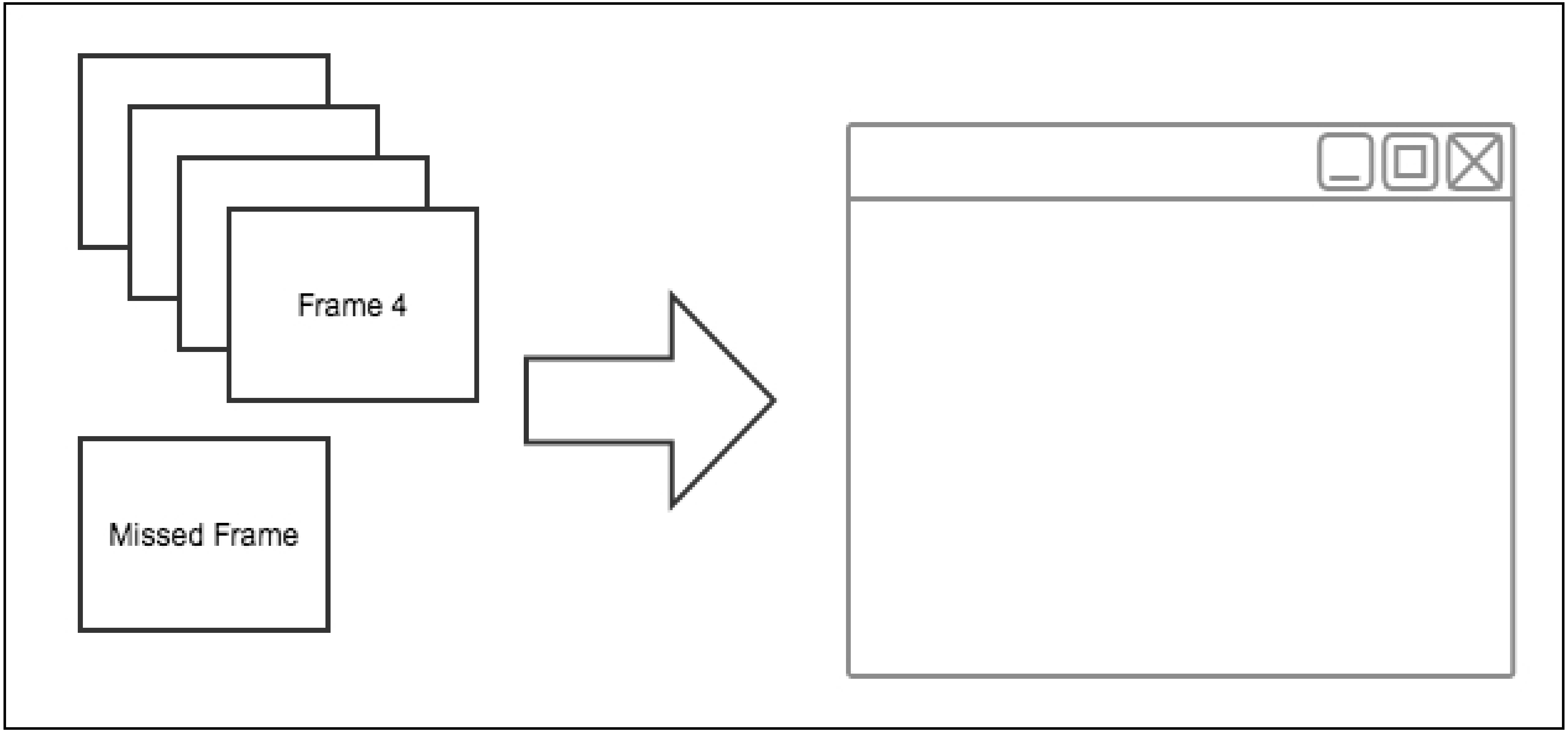
1 MINUTE

SECTION 3: CREATE A BASIC APP

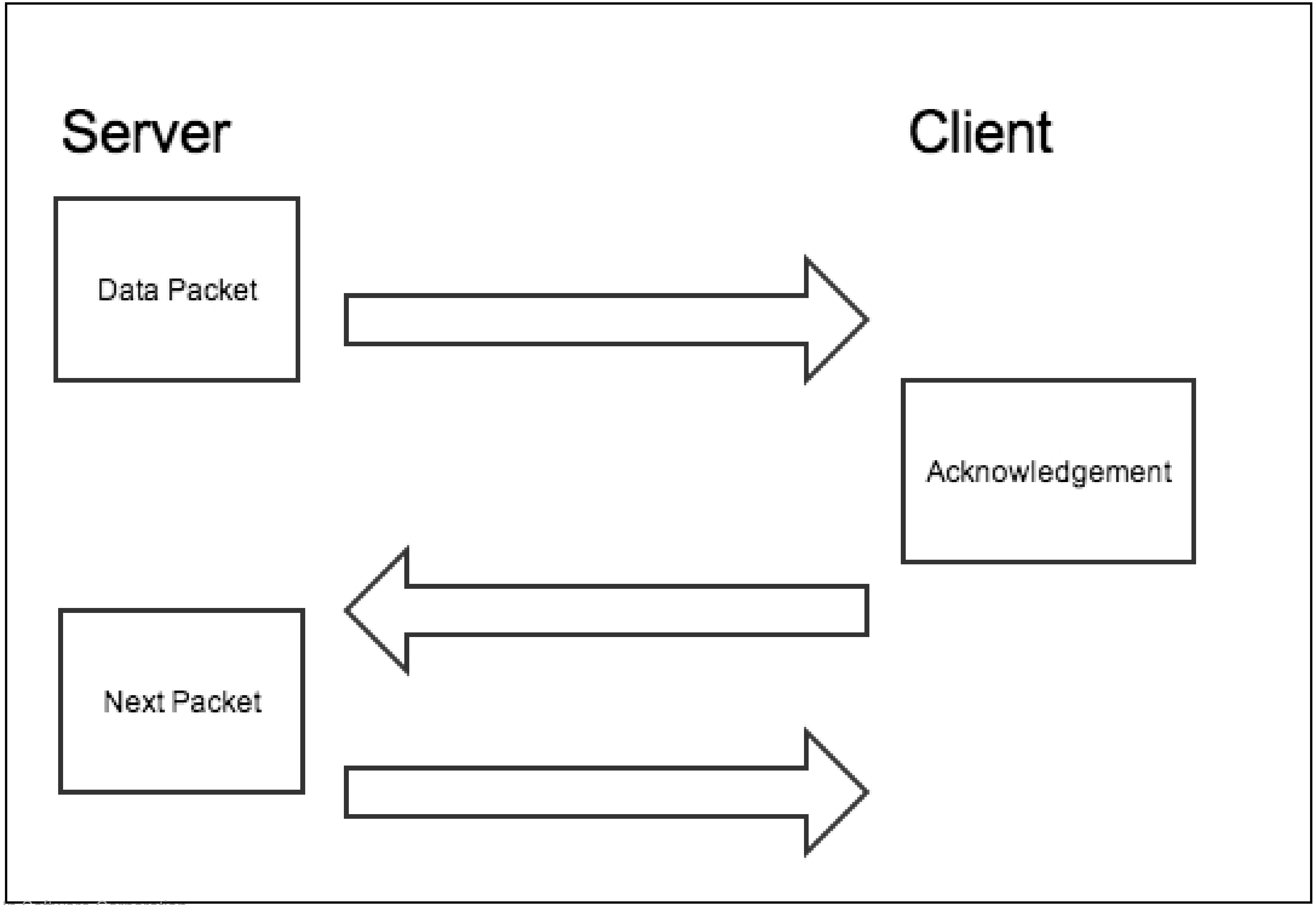
HOW TO CREATE A BASIC APP

- **Understanding UDP transport and real-time transfer**
- **Signaling and negotiating with other users locally**
- **Finding other users on the Web and NAT traversal**
- **Creating an RTCPeerConnection**

- **Real-time transfer of data requires a fast connection speed between both the users.**
- **A typical connection needs to take a frame of both—audio and video—and send it to another user between 40 and 60 times per second in order to be considered good quality.**
- **Given this constraint, audio and video applications are allowed to miss certain frames of data in order to keep up the speed of the connection.**
- **This means that sending the most recent frame of data is more important than making sure that every frame gets to the other side.**


Frame 4
Missed Frame

- Any data sent will be acknowledged as received
- Any data that does not make it to the other side will get resent and halt the sending of any more data
- Data will be unique and no data will be duplicated on the other side


Server
Client
Data Packet
Acknowledgement
Next Packet

- It does not guarantee the order your data is sent in or the order in which it will arrive on the other side

- It does not guarantee that every packet of data will make it to the other side; some may get lost along the way

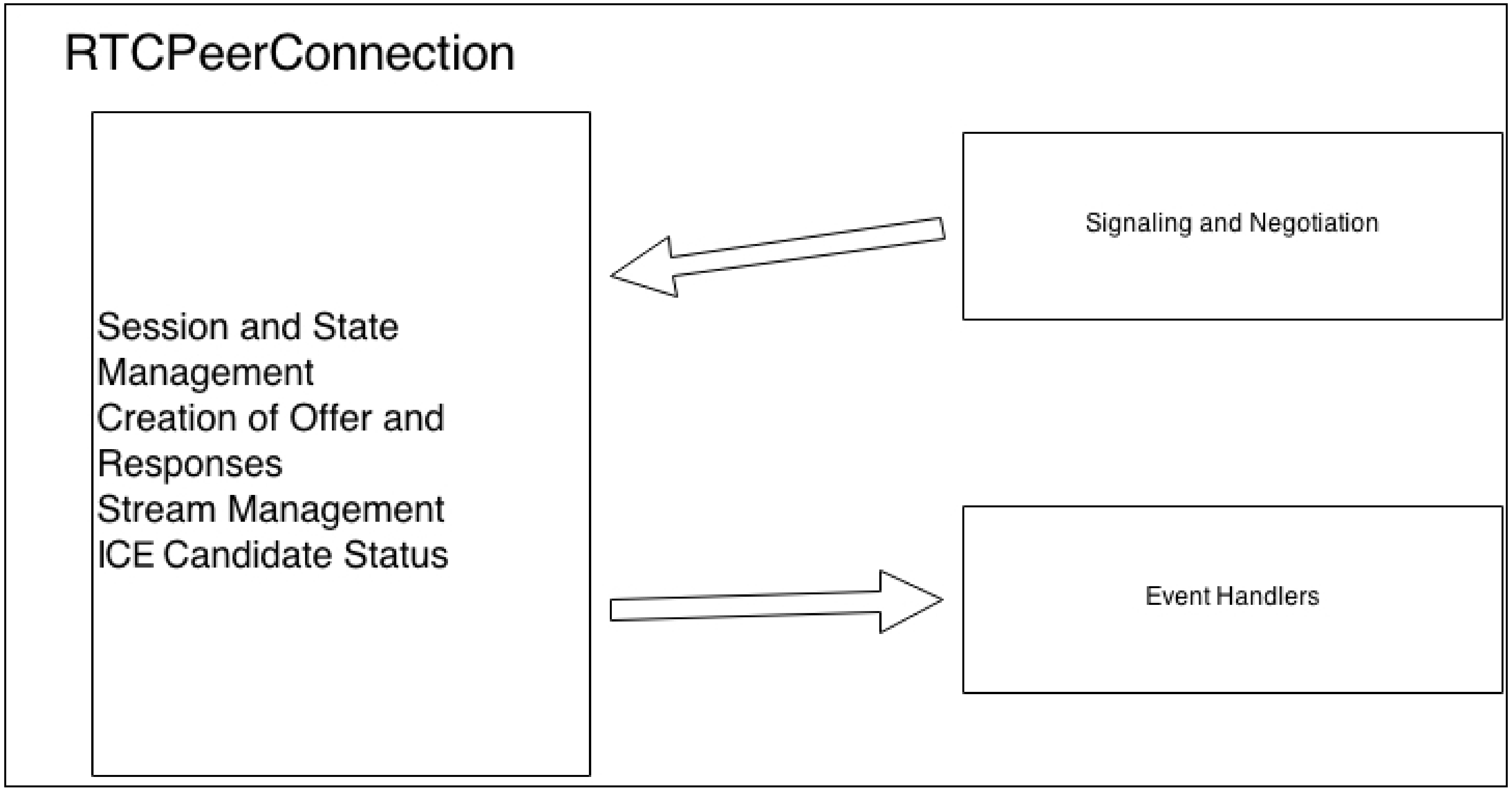
- It does not track the state of every single data packet and will continue to send data even if data has been lost by the other client

WEBRTC API

- The RTCPeerConnection object
- Signaling and negotiation
- Session Description Protocol (SDP)
- Interactive Connectivity Establishment (ICE)

RTC PEER CONNECTION OBJECT

- The RTCPeerConnection object is the main entry point to the WebRTC API.
- It is what allows us to initialize a connection, connect to peers, and attach media stream information.
- It handles the creation of a UDP connection with another user. It is time to get familiar with this name because you will be seeing it a lot.


RTCPeerConnection
Session and State Management
Creation of Offer and Responses
Stream Management
ICE Candidate Status
Signaling and Negotiation
Event Handlers

```
var myConnection = new RTCPeerConnection(configuration);
myConnection.onaddstream = function (stream) {
  // Use stream here
};
```

SIGNALING AND NEGOTIATION

- **Typically, connecting to another browser requires finding where that other browser is located on the Web.**
- **This is usually in the form of an IP address and port number, which act as a street address to navigate to your destination.**
- **The IP address of your computer or mobile device allows other Internet-enabled devices to send data directly between each other; this is what RTCPeerConnection is built on top of.**
- **Once these devices know how to find each other on the Internet, they also need to know how to talk to each other.**
- **This means exchanging data about which protocols each device supports as well as video and audio codecs and more.**

PROCESS OF SIGNALING

- Generate a list of potential candidates for a peer connection.
- Either the user or a computer algorithm will select a user to make a connection with.
- The signaling layer will notify that user that someone would like to connect with him/her, and he/she can accept or decline.
- The first user is notified of the acceptance of the offer to connect.
- If accepted, the first user will initiate RTCPeerConnection with the other user.
- Both the users will exchange hardware and software information about their computers over the signaling channel.
- Both the users will also exchange location information about their computers over the signaling channel.
- The connection will either succeed or fail between the users.

SESSION DESCRIPTION PROTOCOL

<key>=<value>\n

SDP

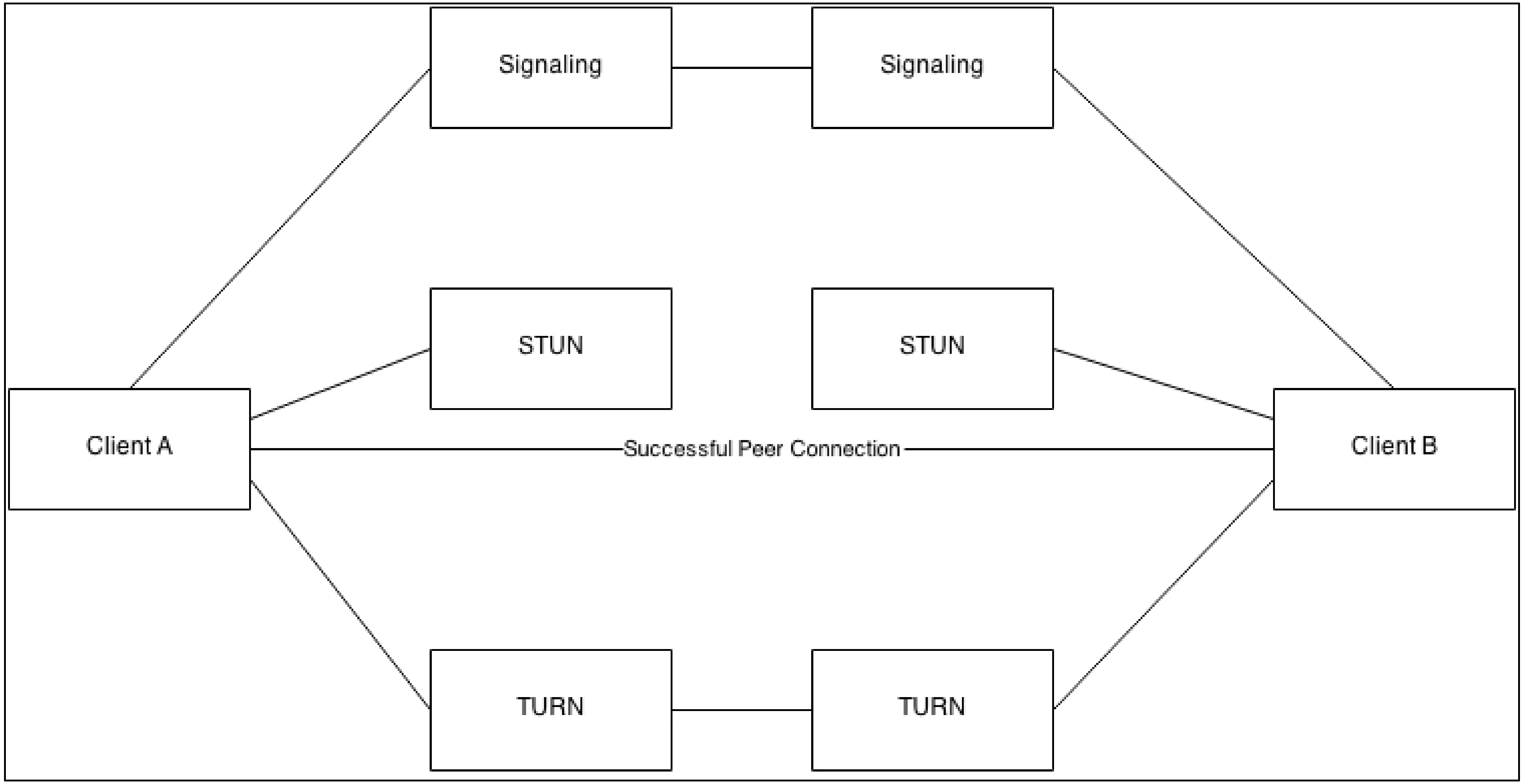
- The SDP will cover the description, timing configuration, and media constraints for a given user.
- The SDP is given by the RTCPeerConnection object during the process of establishing a connection with another user.
- When we start working with the RTCPeerConnection object later in the section, you can easily print this to the JavaScript console.
- This will allow you to see exactly what is contained in the SDP.

SDP

- Overall, the SDP acts as a business card for your computer to other users trying to connect with you.
- The SDP, combined with signaling and negotiation, is the first half of the peer connection.
- In the next few sections, we will cover what happens after both users know how to find each other.

FINDING A CLEAR ROUTE TO ANOTHER USER

- **Session Traversal Utilities for NAT(STUN)**
- **Traversal Using Relays around NAT (TURN)**
- **Interactive Connectivity Establishment (ICE)**


Signaling
Signaling
STUN
STUN
Client A
Successful Peer Connection
Client B
TURN
TURN

SESSION TRAVERSAL UTILITIES FOR NAT

- Using the STUN protocol requires having a STUN-enabled server to connect to.
- Currently, in Firefox and Chrome, default servers are provided directly from the browser vendors.
- This is great for getting up-and-running quickly and testing things out.

TRAVERSAL USING RELAYS AROUND NAT

- In some cases, a firewall might be too restrictive and not allow any STUN-based traffic to the other user.
- This may be the case in an enterprise NAT that utilizes port randomization to allow thousands of more devices than you would typically find.
- In this case, we need a different method of connecting with another user. The standard for this is called TURN.

INTERACTIVE CONNECTIVITY ESTABLISHMENT

- **Now that we have covered STUN and TURN, we can learn how it is all brought together through another standard called ICE.**
- **It is the process that utilizes STUN and TURN to provide a successful route for peer to peer connections.**
- **It works by finding a range of addresses available to each user and testing each address in sorted order, until it finds a combination that will work for both the clients.**

BUILDING A BASIC APP

- Creating a RTCPeerConnection
- Creating the SDP offer and response
- Finding ICE candidates for peers
- Creating a successful WebRTC connection

CREATING A RTCPEERCONNECTION

- **The first step we will take is to create a few functions that handle support across multiple browsers.**
- **These will be able to tell us whether the current browser supports the functionality we need to use to make our application work.**
- **It will also normalize the API, making sure we can always use the same function, no matter what browser we may be running in.**

HTML DRIVER FILE

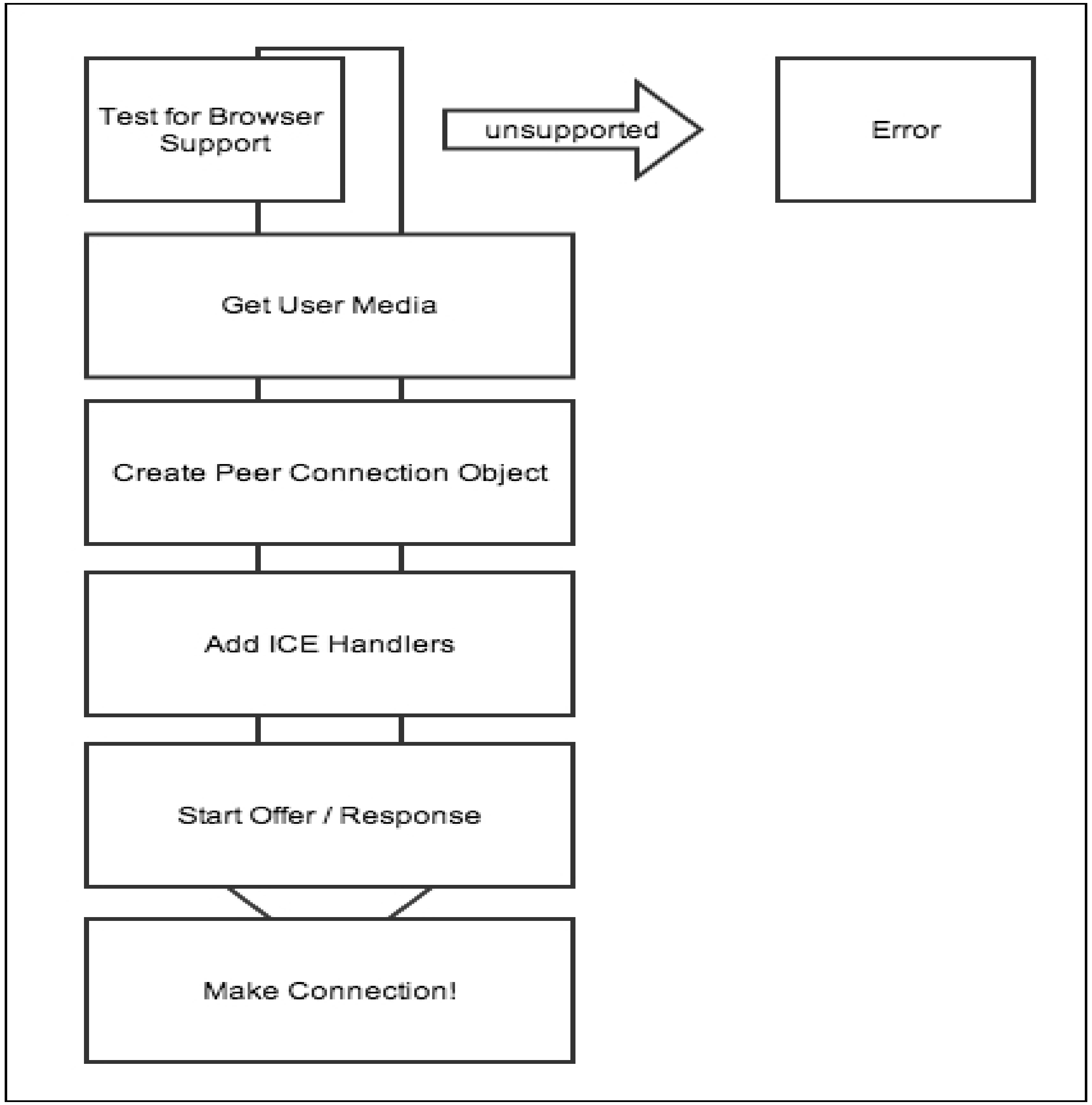
```html
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8" />
    <title>Section 3: Creating a RTCPeerConnection</title>
  </head>
  <body>
    <div id=""container"">
      <video id=""yours"" autoplay></video>
      <video id=""theirs"" autoplay></video>
    </div>
    <script src=""main.js""></script>
  </body>
</html>
```

HTML DRIVER FILES

- The html and head tags of this page should be familiar if you are used to creating HTML5 web pages.
- This is the standard format for any HTML5-compliant page.
- There are a lot of different boilerplate templates for creating a page, and this one is the one I feel is the simplest while still getting the job done.
- There is nothing that will drastically change the way our application works as long as the video elements are there, so if you need to make changes to this file, feel free to do so.

ABSTRACTING BROWSER

```
function hasUserMedia() {
  navigator.getUserMedia = navigator.getUserMedia || navigator.webkitGetUserMedia || navigator.mozGetUserMedia || navigator.msGetUserMedia;
  return !!navigator.getUserMedia;
}

function hasRTCPeerConnection() {
  window.RTCPeerConnection = window.RTCPeerConnection || window.webkitRTCPeerConnection || window.mozRTCPeerConnection;
  return !!window.RTCPeerConnection;
}
```



PEER CONNECTION FLOW

MEDIA STREAM CREATION

- **First we need to get the media stream from the user.**
- **This ensures that the stream is ready and the user has agreed to share their camera and microphone.**
- **Next, we create the peer connection.**
  - This starts off the process in the disconnected state.
  - This is where we can configure the WebRTC connection with the ICE servers that we would like to use.
  - At this moment, the browser is sitting idly and waiting for the connection process to start.

CROSSING THE STREAMS

```
var yourVideo = document.querySelector("#yours"),
    theirVideo = document.querySelector("#theirs"),
    yourConnection, theirConnection;

if (hasUserMedia()) {
  navigator.getUserMedia({ video: true, audio: false }, function (stream)
{
    yourVideo.srcObject = stream;
    theirVideo.srcObject = stream;   // We will set this, since it is just us
if (hasRTCPeerConnection()) {
      startPeerConnection(stream);
    } else {
      alert(""Sorry, your browser does not support WebRTC."");
    }
  }, function (error) {
    alert(""Sorry, we failed to capture your camera, please try again."");
  });
} else {
  alert("Sorry, your browser does not support WebRTC.");
}
```

```
function startPeerConnection(stream) {
  var configuration = {
    // Uncomment this code to add custom iceServers
    //"iceServers": [{ "url": "stun:stun.1.google.com:19302" }]" }]
  };
  yourConnection = new webkitRTCPeerConnection(configuration);
  theirConnection = new webkitRTCPeerConnection(configuration);
};
```

```
function startPeerConnection(stream) {
  var configuration = {
    // Uncomment this code to add custom iceServers
    //""iceServers"": [{ ""url"": ""stun:stun.1.google.com:19302"" }]
  };
  yourConnection = new webkitRTCPeerConnection(configuration);
  theirConnection = new webkitRTCPeerConnection(configuration);
// Begin the offer
  yourConnection.createOffer(function (offer) {
    yourConnection.setLocalDescription(offer);
    theirConnection.setRemoteDescription(offer);

    theirConnection.createAnswer(function (offer) {
      theirConnection.setLocalDescription(offer);
      yourConnection.setRemoteDescription(offer);
    });
  });
};
```

```
function startPeerConnection(stream) {
  var configuration = {
    // Uncomment this code to add custom iceServers
    //""iceServers"": [{ ""url"": ""stun:127.0.0.1:9876"" }]
  };
  yourConnection = new webkitRTCPeerConnection(configuration);
  theirConnection = new webkitRTCPeerConnection(configuration);

     // Setup ice handling
  yourConnection.onicecandidate = function (event) {
    if (event.candidate) {
      theirConnection.addIceCandidate(new RTCIceCandidate(event.candidate));
    }
   };
theirConnection.onicecandidate = function (event) {
    if (event.candidate) {
      yourConnection.addIceCandidate(new RTCIceCandidate(event.candidate));
    }
      };

  // Begin the offer
  yourConnection.createOffer(function (offer) {
    yourConnection.setLocalDescription(offer);
    theirConnection.setRemoteDescription(offer);

    theirConnection.createAnswer(function (offer) {
      theirConnection.setLocalDescription(offer);
      yourConnection.setRemoteDescription(offer);
    });
  });
};
```

```
// Setup stream listening
  yourConnection.addStream(stream);
  theirConnection.onaddstream = function (e) {
    theirVideo.srcObject = e.stream;
  };
```

```
<style>
   body {
     background-color: #3D6DF2;
     margin-top: 15px;
   }

   video {
     background: black;
     border: 1px solid gray;
   }
```

```
#container {
    position: relative;
    display: block;
    margin: 0 auto;
    width: 500px;
    height: 500px;
  }

  #yours {
    width: 150px;
    height: 150px;
    position: absolute;
    top: 15px;
    right: 15px;
  }
```

```
#theirs {
        width: 500px;
        height: 500px;
      }
    </style>
```

```
<!DOCTYPE html>
<html lang=""en"">
 <head>
  <meta charset=""utf-8"" />

  <title>Section 3: Creating a RTCPeerConnection</title>

  <style>
   body {
    background-color: #3D6DF2;
    margin-top: 15px;
   }

   video {
    background: black;
    border: 1px solid gray;
   }
```

```
#container {
    position: relative;
    display: block;
    margin: 0 auto;
    width: 500px;
    height: 500px;
  }

  #yours {
    width: 150px;
    height: 150px;
    position: absolute;
    top: 15px;
    right: 15px;
  }
```

```
#theirs {
        width: 500px;
        height: 500px;
      }
    </style>
  </head>
  <body>
    <div id=""container"">
      <video id=""yours"" autoplay></video>
      <video id=""theirs"" autoplay></video>
    </div>

    <script src=""main.js""></script>
  </body>
</html>
```

```
function hasUserMedia() {
  navigator.getUserMedia = navigator.getUserMedia || navigator.webkitGetUserMedia ||
navigator.mozGetUserMedia || navigator.msGetUserMedia;
  return !!navigator.getUserMedia;
}

function hasRTCPeerConnection() {
  window.RTCPeerConnection = window.RTCPeerConnection ||
window.webkitRTCPeerConnection || window.mozRTCPeerConnection;
  return !!window.RTCPeerConnection;
}

var yourVideo = document.querySelector("#yours"),
    theirVideo = document.querySelector("#theirs"),
    yourConnection, theirConnection;

i
```

```
if (hasUserMedia()) {
  navigator.getUserMedia({ video: true, audio: false }, function (stream) {
    yourVideo.srcObject = stream;
    theirVideo.srcObject = stream;

    if (hasRTCPeerConnection()) {
      startPeerConnection(stream);
    } else {
      alert(""Sorry, your browser does not support WebRTC."");
    }
  }, function (error) {
    console.log(error);
  });
} else {
  alert(""Sorry, your browser does not support WebRTC."");
}
```

```
function startPeerConnection(stream) {
  var configuration = {
    ""iceServers"": [{ ""url"": ""stun:stun.1.google.com:19302"" }]
  };
  yourConnection = new webkitRTCPeerConnection(configuration);
  theirConnection = new webkitRTCPeerConnection(configuration);

  // Setup stream listening
  yourConnection.addStream(stream);
  theirConnection.onaddstream = function (e) {
    theirVideo.srcObject = e.stream;
  };

  // Setup ice handling
  yourConnection.onicecandidate = function (event) {
    if (event.candidate) {
      theirConnection.addIceCandidate(new RTCIceCandidate(event.candidate));
    }
  };
  theirConnection.onicecandidate = function (event) {
    if (event.candidate) {
      yourConnection.addIceCandidate(new RTCIceCandidate(event.candidate));
    }
  };

  // Begin the offer
  yourConnection.createOffer(function (offer) {
    yourConnection.setLocalDescription(offer);
    theirConnection.setRemoteDescription(offer);

    theirConnection.createAnswer(function (offer) {
      theirConnection.setLocalDescription(offer);
      yourConnection.setRemoteDescription(offer);
    });
  });
};
```

SUMMARY