More on Java 8 Optional Functionality

by Dale Wyttenbach · May. 14, 18 · Java Zone · Tutorial

A recent article on a proposed "isEmpty" method addition to the Java 8 Optional API piqued my curiosity. I'm not using Java 8 on a day-to-day basis, so I don't know its features very well yet. Several years ago (probably not long after Java 8 was released), I wrote a program to play with lambda expressions. The Optional API was something I had yet to investigate, until today. In order to learn about Optional, I dusted off that old lambda program (Person.java and PersonTest.java in this GitHub repository) and added additional JUnit test cases to it.

The Basics of Optional

The first test cases I added to PersonTest were:

```
@org.junit.Test
public void testOptionalNotPresent() {
    Assert.assertEquals(String.format("Optional
}
```

These don't require much explanation, I think. I did scratch my head for a while trying to figure out what this Optional thing is doing, until I realized that I needed to use "Optional.ofNullable" instead of "Optional.of" in my implementation of Person class getEmailAddress method:

```
public Optional<String> getEmailAddress() {
    return Optional.ofNullable(emailAddress);
}
```

It's still not clear to me why anyone would want to create Optional objects which are NOT nullable. I mean that's the whole point, isn't it?

Empty Strings

I was also curious about the Optional API's use of the word "empty" and whether that applied only to null values, or if empty String objects qualified as "empty" as well. As it turns out, Optional lets you decide for yourself. I found an excellent answer to this question at StackOverflow. (See top answer "You could use a filter" in the referenced SO article)

```
@org.junit.Test
public void testOptionalEmptyString_Allowed() {

    final String emailAddr = "";
    person1.setEmailAddress(emailAddr);
    person1.setAllowEmpty(true);
    Optional<String> email1 = person1.getEmailA

    email1.ifPresent(value -> Assert.assertEqua

    Assert.assertEquals(String.format("Optional

}
```

The "Not Allowed" test case above tests the default behavior of the Person class, which is that empty email addresses are treated like null (not present). In the "Allowed" version of the test case, the Person object is mutated to allow empty strings (treated as present). In order to accomplish this my implementation of Person getEmailAddress method changed to:

```
public Optional<String> getEmailAddress() {

    return allowEmpty ? Optional.ofNullable(ema

}
```

As you can see, this is where I lifted the "Use a filter" answer from the SO article.

```
      .flatMap(Address::getCity)
      .ifPresent(value -> Assert.fail("Should r
}

@org.junit.Test
public void testOptional_Monad_Found() {
    OptMap<String, Person> personMap = new OptM
    Person person = new Person(new Address(new

    personMap.put(PERSON_NAME, person);

    personMap.find(PERSON_NAME)
      .flatMap(Person::getAddress)
      .flatMap(Address::getCity)
      .ifPresent(PersonTest::process);
}
```

In order to accomplish the above, I had to quickly whip up a couple of new classes, Address and City. I also lifted the OptMap idea from Pierre-Yves' excellent article. My biggest stumbling block in getting these new test cases to compile was needing to nTest "process" method static:

```
tic void process(City city) {
.assertEquals(CITY_NAME, city.name);
}
```

- https://github.com/wytten/java8-optional
- https://stackoverflow.com/questions/28322464/in-java-8-transform-optionalstring-of-an-empty-string-in-optional-empty

Topics: JAVA 8 LAMBDA EXPRESSIONS , JAVA 8 OPTIONAL , JAVA , TUTORIAL

JavaScript, WebRTC, and Okta

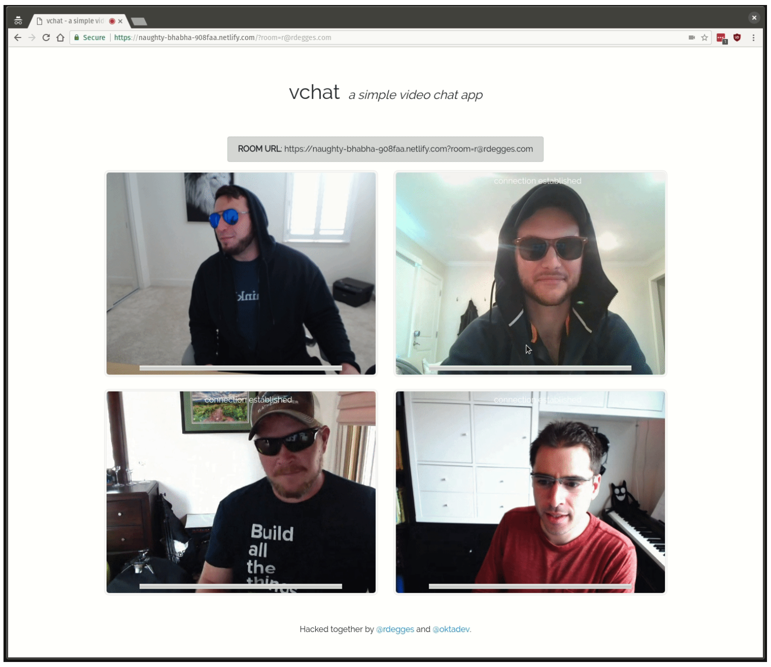
by Randall Degges MVB · May 13, 18 · Web Dev Zone · Tutorial

“I love writing authentication and authorization code.” ~ No Developer Ever. Try Okta instead

As recently as seven short years ago, building video applications on the web was a massive pain. Remember the days of using Flash and proprietary codecs (*which often required licensing*)? Yuck. In the last few years, video chat technology has dramatically improved and Flash is no longer required.

Today, the video chat landscape is much simpler thanks to WebRTC: an open source project built and maintained by Google, Mozilla, Opera, and others. WebRTC allows you to easily build real-time communication software in your browser and is being standardized at the W3C and IETF levels. Using WebRTC, you can build real-time video chat applications in the browser that actually work *well*! It’s pretty amazing.

Today, I thought it’d be fun to walk you through the process of using WebRTC and Okta to build a simple video chat service that allows users to create a chatroom and share the link around to anyone they want who can then join the room and chat with them


vchat a simple video chat app
ROOM URL: https://naughty-bhabha-908faa.netlify.com?room=r@rdegges.com
connection established
Hacked together by @rdegges and @oktadev.

NOTE: Want to play around with the chat app in real-time? You can do so here: https://naughty-bhabha-908faa.netlify.com You can also view the source code for the app we'll be building on GitHub.

Create the Web Page

The first thing you'll do is create a simple HTML web page for the app.

When building web applications, I like to start by first creating my markup and CSS, then going back for a second pass and adding in application logic.

Create a new folder somewhere on your computer called `chatapp` , then create an `index.html` file with the following contents:

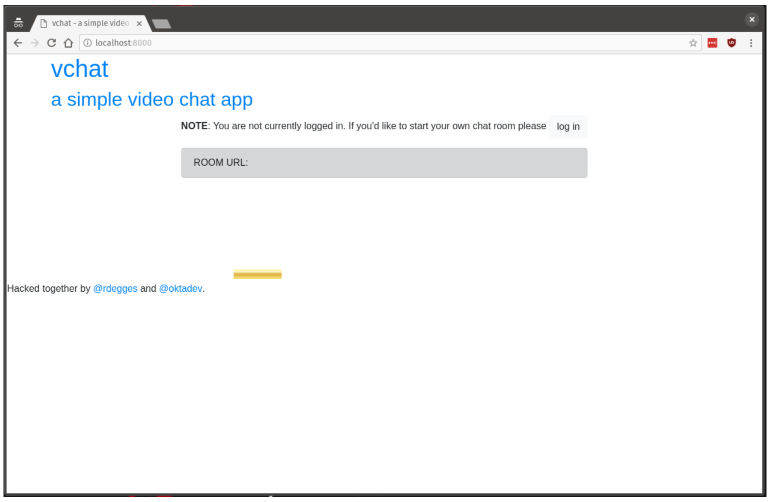
```
<!DOCTYPE html>
```

```
    </header>

    <div id="okta-login-container"></div>


    <div class="row">
      <div class="col"></div>
      <div class="col-md-auto align-self-cent
        <p id="login"><b>NOTE</b>: You are nc
          chat room please <button type="butt
        <div id="url" class="alert alert-dark
          <span id="roomIntro">ROOM URL</spar
        </div>
      </div>
      <div class="col"></div>
    </div>

    <div id="remotes" class="row">
      <div class="col-md-6">
        <div class="videoContainer">
          <video id="selfVideo"></video>
          <meter id="localVolume" class="volu
        </div>
      </div>
    </div>
  </div>

```

The key elements present in this minimalistic HTML page are:

- An `okta-login-container` div, which will eventually hold our login form
- A login notice and Room URL placeholder that will notify a user whether they need to log in, and what chat room they are currently in
- A div that will eventually contain all of the video feeds from various participants

If you open this page up in your browser, you'll notice that it looks pretty bad. But don't worry, you'll make it look pretty soon enough!



vchat

a simple video chat app

NOTE: You are not currently logged in. If you'd like to start your own chat room please log in

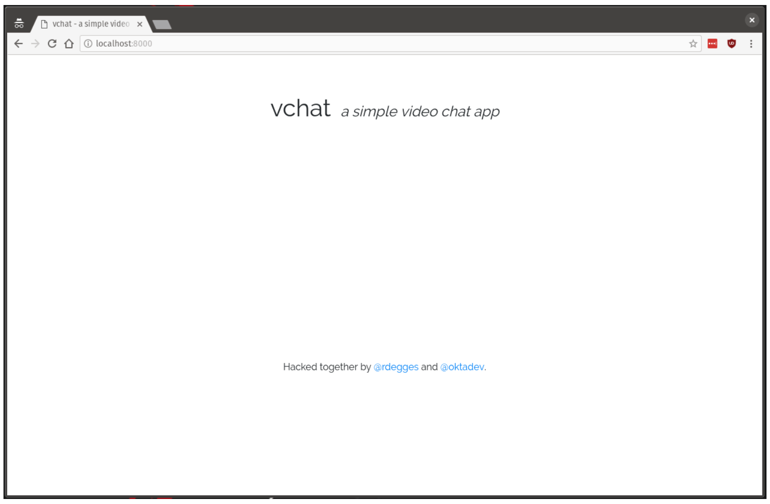
ROOM URL:

Hacked together by @rdegges and @oktadev.

Next, you'll want to create a folder called `static`, which contains another folder named `css`. Then, you'll need to copy the following CSS into a new `style.css` file inside that folder:

```
h2 {
  font-style: italic;
}

header {
  text-align: center;
  margin: 4em;
}

header h1, header h2 {
  display: inline;
}

header h1 a, header h2 a, header h1 a:hover, he
  color: inherit;
  text-decoration: none;
}

header h2 {
  font-size: 24px;
  padding-left: .5em;
}

#remotes {
  visibility: hidden;
}
```

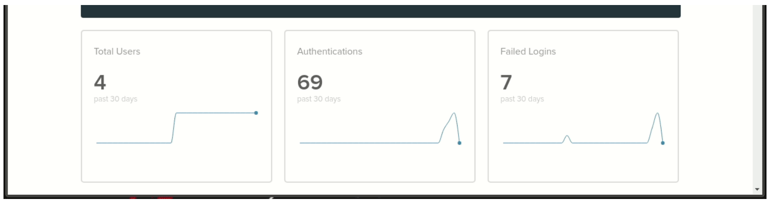
```

.videoContainer {
  object-fit: cover;
  margin: 0 auto;
  padding: 0;
}

.videoContainer video {
  width: 100%;
  height: 100%;
  border-radius: 10px;
  border: 5px double #f2f2f2;
}

.volume {
  position: absolute;
  left: 15%;
  width: 70%;
  bottom: 20px;
  height: 10px;
  display: none;
}

.connectionstate {
  position: absolute;
  top: 10px;
  width: 100%;
  text-align: center;
  color: #fff
}

.col-md-6 {
    <link href="https://fonts.googleapis.com/cs
    <link rel="stylesheet" href="/static/css/st
</head>
```


vchat - a simple video
localhost:8000
vchat a simple video chat app
Hacked together by @rdegges and @oktadev.

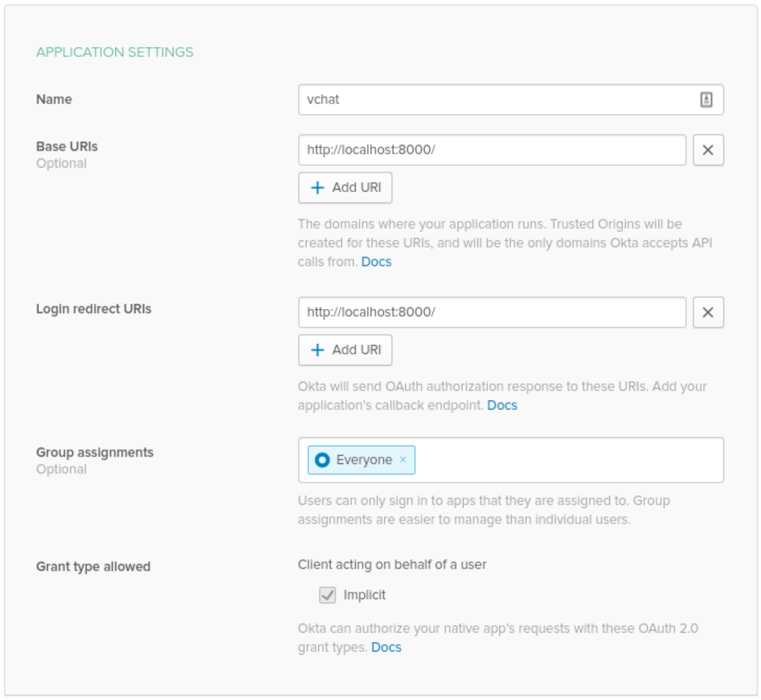
Set Up Okta

Now that you've got a simple web page with some styling on it, let's set up the user login component using Okta. If you don't already have an Okta developer account, go create one now then come back (it should only take a second): https://developer.okta.com/signup/

Once you've got your Okta account and you're logged into the Okta dashboard, you'll need to create a new Okta application (this is how Okta knows what type of app you're building and what type of authentication to allow).



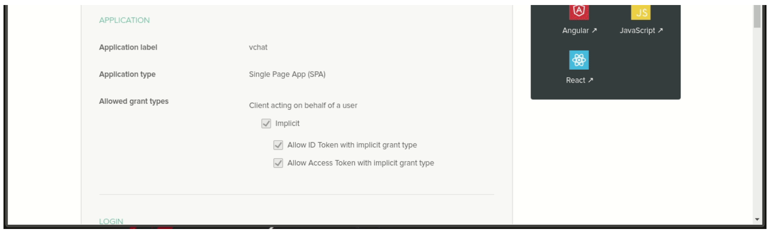
Once you've reached the **Create New Application** page fill out the **Application Settings** form with the following information:


APPLICATION SETTINGS

Name	vchat
Base URIs Optional	http://localhost:8000/ + Add URI The domains where your application runs. Trusted Origins will be created for these URIs, and will be the only domains Okta accepts API calls from. Docs
Login redirect URIs	http://localhost:8000/ + Add URI Okta will send OAuth authorization response to these URIs. Add your application's callback endpoint. Docs
Group assignments Optional	Everyone × Users can only sign in to apps that they are assigned to. Group assignments are easier to manage than individual users.
Grant type allowed	Client acting on behalf of a user ☑ Implicit Okta can authorize your native app's requests with these OAuth 2.0 grant types. Docs

When done, click **Done**. Your Okta Application is now *almost* ready to go.

The next thing you'll need to do is add your local computer as a **Trusted Origin** — trusted origins are


APPLICATION

Application label vchat

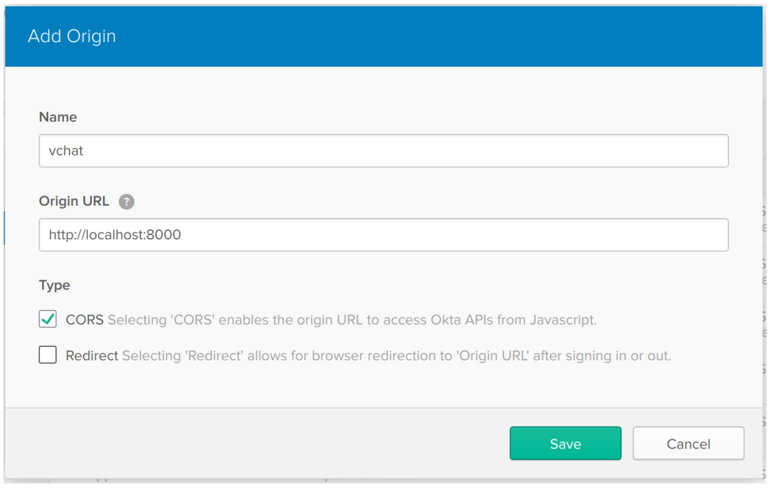
Application type Single Page App (SPA)

Allowed grant types Client acting on behalf of a user

- [✓] Implicit
  - [✓] Allow ID Token with implicit grant type
  - [✓] Allow Access Token with implicit grant type

LOGIN

Once you’ve reached the **Add Origin** screen, enter the following information which tells Okta to allow you to use the Okta authentication service from your local test environment:


Add Origin

Name: vchat

Origin URL: http://localhost:8000

Type

- [✓] CORS Selecting 'CORS' enables the origin URL to access Okta APIs from Javascript.
- [ ] Redirect Selecting 'Redirect' allows for browser redirection to 'Origin URL' after signing in or out.

Finally, now that you’ve configured your Okta Application and all necessary security rules, you should go create some user accounts in Okta that you can log in with later. You can do this by clicking on the **Users** tab followed by the **Add Person** button:

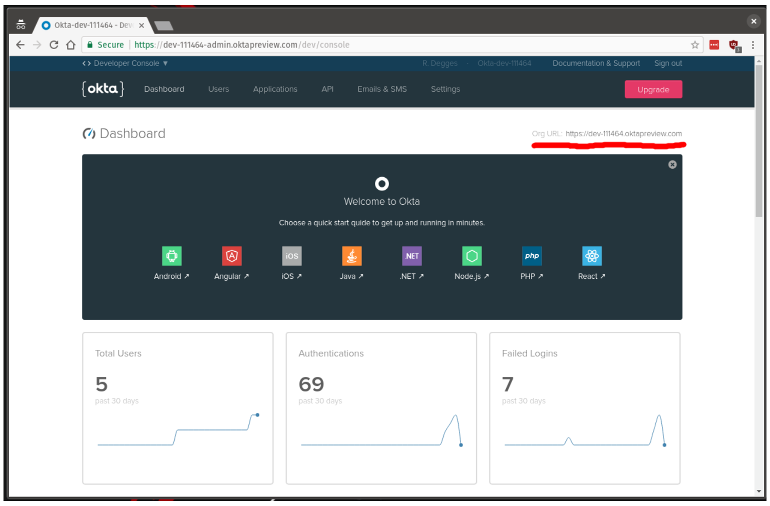
Authentication

Now that you've got Okta configured, you need to plug Okta into your web app so users can log into your video chat app.

While there are many different ways to integrate with Okta, for a single-page app like the one you're building today you'll want to use the Okta Sign-In Widget.

The Okta Sign-In Widget is a pure JS library you can drop into any web page that handles user authentication for you.

Before adding the widget's JS code (below), you'll want to visit your Okta dashboard and grab the **Org URL** value from the top-right portion of the page.



You'll also need to view the Application you created and import the Okta Sign-In widget dependencies as well as initialize the widget at the bottom of the page in a script tag. Be sure to substitute `{{OKTA_ORG_URL}}` and `{{CLIENT_ID}}` with the appropriate values for your app.

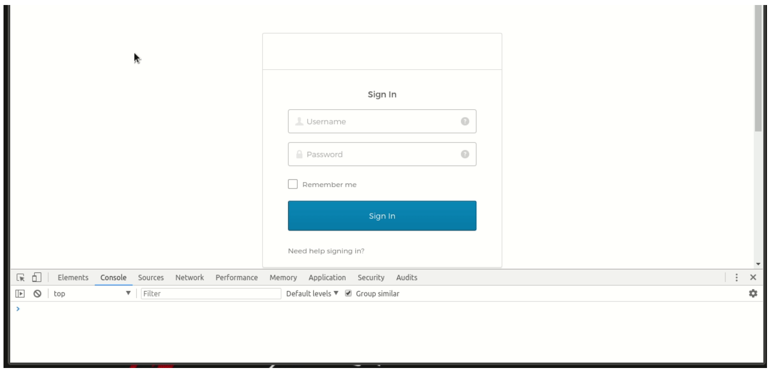
```
<!-- snip -->

<head>
  <title>vchat - a simple video chat app</title
  <link rel="stylesheet" href="https://maxcdn.b
84xqQ1aoWXA+058RXPxPg6fy4IWvTNh0E263XmFcJlSAwiG
  <link href="https://fonts.googleapis.com/css?
  <script src="https://ok1static.oktacdn.com/as
cript"></script>
  <link href="https://ok1static.oktacdn.com/ass
 rel="stylesheet"/>
  <link href="https://ok1static.oktacdn.com/ass
stylesheet"/>
  <link rel="stylesheet" href="/static/css/styl
</head>

<!-- snip -->
```

```
    responseType: ["token", "id_token"],
    display: "page"
  }
});

// Render the login form.
function showLogin() {
  okta.renderEl({ el: "#okta-login-container"
    alert("Couldn't render the login form, sc
  });
}

// Handle the user's login and what happens r
function handleLogin() {
  // If the user is logging in for the first
  if (okta.token.hasTokensInUrl()) {
    okta.token.parseTokensFromUrl(
      function success(res) {
        // Save the tokens for later use, e.g
        okta.tokenManager.add("accessToken",
        okta.tokenManager.add("idToken", res[

        console.log("user just logged in");
      }, function error(err) {
        alert("We weren't able to log you in,
        console.log("user not logged in");
        showLogin();
      });
    }
  }

  handleLogin();
</script>
```

The code above initializes the Okta widget, handles user authentication, and logs some messages to the developer console to help you understand what's going on.

The `okta` object you create above controls the widget's functionality: by creating a new instance of the `OktaSignIn` class and giving it your app-specific details, you're essentially telling Okta where your OpenID Connect server is and how to use it (Okta uses the OpenID Connect protocol behind the scenes to power this login widget).

The `handleLogin` function you see above is what controls the session management in your app. If a user has just logged in (as part of the OIDC flow) then the user's access and ID tokens will be stored in HTML local storage so your app can remember who the user is. If the user was already logged in but is viewing the page, a message will be echoed to the console. And if the user is not logged in at all, then the login form will be rendered (via



Configure State Management

The next thing you will need to do is configure state management for the app. But before we dive into that, let's talk about how the app is going to work.

The video chat app you're building will give each registered user their own dedicated chat room that they can use at any time and that can be shared with any external person. When another person joins one of these dedicated chat rooms they'll be instantly put into the video chat session without needing an account on the site.

To make this functionality work in a simple manner, you'll structure the app such that each registered user will have a dedicated chat room whose URL is `{{YOUR_SITE_URL}}?room={{email}}`. This way, if my email address is `r@rdegges.com` then I'll have a dedicated chat room my friends can join that is `{{YOUR_SITE_URL}}?room=r@rdegges.com` — easy to

To get started, create a `hasQueryString` function that will be helpful in determining whether or not the user is on the homepage of the app or in a specific chat room:

```
// Determine whether or not we have a querystri
function hasQueryString() {
    return location.href.indexOf("?") !== -1;
}
```

Next, define two helper functions: `getRoom` and `getRoomURL` which will determine the chat room name (from the querystring) as well as the fully qualified room URL as well. These will be helpful later on when writing the video chat code:

```
// Determine the room name and public URL for t
function getRoom() {
  var query = location.search && location.searc

  if (query) {
    return (location.search && decodeURICompone
  }

  return okta.tokenManager.get("idToken").claim
}
```

- Notify users who aren't logged in (but are in a video chat room) that they can log in if they want to

```
// Handle the user's login and what happens nex
function handleLogin() {
  // If the user is logging in for the first ti
  if (okta.token.hasTokensInUrl()) {
    okta.token.parseTokensFromUrl(
      function success(res) {
        // Save the tokens for later use, e.g.
        okta.tokenManager.add("accessToken", re
        okta.tokenManager.add("idToken", res[1]

        // Redirect to this user's dedicated rc
        window.location = getRoomURL();
      }, function error(err) {
        alert("We weren't able to log you in, s
      }
    );
  } else {
    okta.session.get(function(res) {

      // If the user is logged in...
      if (res.status === "ACTIVE") {

        // If the user is logged in on the home
      // we'll prompt them for login immediatel

      if (hasQueryString()) {
        document.getElementById("login").style.

      } else {
        showLogin();
      }
    });
  }
}
```

By using the simple helper functions to handle redirects, you're *almost* able to accomplish everything you need in terms of state management.

But, there's one tiny thing left to do: you need to make sure that the `login` button redirects any users to the homepage of the app so they can view the login form. To do this, simply define an `onclick` handler on the `button` element in the page:

```
<p id="login">
  <b>NOTE</b>: You are not currently logged in.

</p>
```

And with that final change, the app's state

Time to move onto the fun stuff: real-time video with WebRTC.

Use WebRTC to Enable Real-Time Video Chat

To get real-time video chat working in this app we'll be using the fantastic SimpleWebRTC library. This library provides some excellent APIs that wrap the underlying WebRTC APIs making them much simpler to work with.

To get started with SimpleWebRTC, you first need to include the required adapter library in the `head` section of the web app:

```
<head>
  <title>vchat - a simple video chat app</title
  <link rel="stylesheet" href="https://maxcdn.b
  <link href="https://fonts.googleapis.com/css?
  <script src="https://ok1static.oktacdn.com/as
  <link href="https://ok1static.oktacdn.com/ass
  <link href="https://ok1static.oktacdn.com/ass
  <script src="https://webrtc.github.io/adapter
  <link rel="stylesheet" href="/static/css/styl
</head>
```

To do this, all you'll need to do is replace the `videoContainer` div with this updated HTML:

```
<div class="videoContainer">
  <video id="selfVideo" oncontextmenu="return f
  <meter id="localVolume" class="volume" min="-
</div>
```

The `oncontextmenu` attribute simply tells the browser to not do anything when the video element is right-clicked. The extra `min` , `max` , `high` , and `low` attributes on the volume meter are the reasonable defaults recommended by SimpleWebRTC, so that's what we'll go with.

Next, you need to define a `loadSimpleWebRTC()` function which will:

- Prompt the user to allow camera/mic access to the web page
- Render a video feed on the screen
- Automatically join the appropriate chat room based on the URL the user is visiting
- Handle any errors that arise (video feeds being dropped when someone disconnects, etc.)

To do this, you need to define the function in your JS code:

```
// Dynamically load the simplewebrtc script so
```

```
      autoRequestMedia: true,
      debug: false,
      detectSpeakingEvents: true,
      autoAdjustMic: false
    });

    // Set the publicly available room URL.
    document.getElementById("roomUrl").innerTex

    // Immediately join room when loaded.
    webrtc.on("readyToCall", function() {
      webrtc.joinRoom(getRoom());
    });

    function showVolume(el, volume) {
      if (!el) return;
      if (volume < -45) volume = -45; // -45 to
      if (volume > -20) volume = -20; // a gooc
      el.value = volume;
    }

    // Display the volume meter.
    webrtc.on("localStream", function(stream) {
      var button = document.querySelector("forn
      if (button) button.removeAttribute("disat
      document.getElementById("localVolume").st
    });
      var remotes = document.getElementById("re

      if (remotes) {
        var outerContainer = document.createEle
        outerContainer.className = "col-md-6";

        var container = document.createElement(
        container.className = "videoContainer";
        container.id = "container_" + webrtc.ge
        container.appendChild(video);

        // Suppress right-clicks on the video.
        video.oncontextmenu = function() { retu

        // Show the volume meter.
        var vol = document.createElement("meter
        vol.id = "volume_" + peer.id;
        vol.className = "volume";
        vol.min = -45;
        vol.max = -20;
        vol.low = -40;
        vol.high = -25;
        container.appendChild(vol);

        // Show the connection state.
```

```
                break;
              case "connected":
              case "completed": // on caller si
                vol.style.display = "block";
                connstate.innerText = "connecti
                break;
              case "disconnected":
                connstate.innerText = "disconne
                break;
              case "failed":
                connstate.innerText = "connecti
                break;
              case "closed":
                connstate.innerText = "connecti
                break;
            }
          });
        }

        outerContainer.appendChild(container);
        remotes.appendChild(outerContainer);

        // If we're adding a new video we need
```

```
      }
    });

    // If a user disconnects from chat, we neec
    webrtc.on("videoRemoved", function(video, p
      console.log("user removed from chat", pee
      var remotes = document.getElementById("re
      var el = document.getElementById("contair

      if (remotes && el) {
        remotes.removeChild(el.parentElement);
      }
    });

    // If our volume has changed, update the me
    webrtc.on("volumeChange", function(volume,
      showVolume(document.getElementById("local
    });

    // If a remote user's volume has changed, u
    webrtc.on("remoteVolumeChange", function(pe
        connstate.innerText = "connection faile

        fileinput.disabled = "disabled";
      }
    });

    // remote p2p/ice failure
    webrtc.on("connectivityError", function (pe
      var connstate = document.querySelector("#
      console.log("remote fail", connstate);
      if (connstate) {
        connstate.innerText = "connection faile

        fileinput.disabled = "disabled";
      }
    });
  }
}
```

While this is a lot of code to take in, it's not all complex.

The first thing we do in the above function is dynamically load the SimpleWebRTC library (this is

For instance, if the user is attempting to view the `?room=test` chat room, they'll be dropped into the chat room named `test`. By simply handling this logic via querystring parameters, we're able to avoid using any server-side logic.

Here's the snippet of code which handles the room joining:

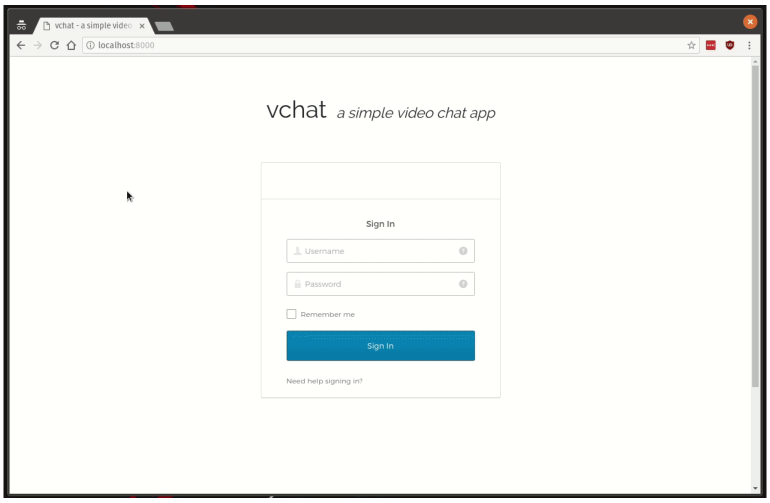
```
script.onload = function() {
  var webrtc = new SimpleWebRTC({
    localVideoEl: "selfVideo",
    // the id/element dom element that will hol

    remoteVideosEl: "",
    autoRequestMedia: true,
    debug: false,
    detectSpeakingEvents: true,
    autoAdjustMic: false
  });

  // Set the publicly available room URL.
  document.getElementById("roomUrl").innerText


  // Immediately join room when loaded.
  webrtc.on("readyToCall", function() {
    webrtc.joinRoom(getRoom());
  });

  // snip
```

And finally, you've now got to go back and modify your `handleLogin` function so that it calls your new `enableVideo` function when appropriate:

```
// Handle the user's login and what happens nex
function handleLogin() {
  // If the user is logging in for the first ti
  if (okta.token.hasTokensInUrl()) {
    okta.token.parseTokensFromUrl(
      function success(res) {
        // Save the tokens for later use, e.g.
        okta.tokenManager.add("accessToken", re
        okta.tokenManager.add("idToken", res[1]

        // Redirect to this user's dedicated ro
        window.location = getRoomURL();
      }, function error(err) {
        alert("We weren't able to log you in, s
      }
    );
  } else {
    okta.session.get(function(res) {

      // If the user is logged in, display the
      if (res.status === "ACTIVE") {
```

```
   // "room" so we should display our passiv
   // we'll prompt them for login immediatel
   if (hasQueryString()) {
    document.getElementById("login").style.
    enableVideo();
   } else {
    showLogin();
   }
  });
 }
}
```

By calling `enableVideo` when appropriate above, everything should now be working nicely! You can test this out by opening your page and giving things a go:



```
  <head>
    <title>vchat - a simple video chat app</tit
    <link rel="stylesheet" href="https://maxcdr
    <link href="https://fonts.googleapis.com/cs
    <script src="https://ok1static.oktacdn.com/
    <link href="https://ok1static.oktacdn.com/a
    <link href="https://ok1static.oktacdn.com/a
    <script src="https://webrtc.github.io/adapt
    <link rel="stylesheet" href="/static/css/st
  </head>
  <body>
    <div class="container">
      <header>
        <h1><a href="/">vchat</a></h1>
        <h2><a href="/">a simple video chat app
      </header>

      <div id="okta-login-container"></div>

      <div class="row">
        <div class="col"></div>
        <div class="col-md-auto align-self-cent
          <p id="login"><b>NOTE</b>: You are nc
          <video id="selfVideo" oncontextmenu
          <meter id="localVolume" class="volu
        </div>
      </div>
    </div>
  </div>

  <footer>
    <p>Hacked together by <a href="https://tw
      and <a href="https://twitter.com/oktade
  </footer>

  <script>
    var okta = new OktaSignIn({
      baseUrl: "https://dev-111464.oktapreviе
      clientId: "0oaejf8gmll1TiDRz0h7",
      authParams: {
        issuer: "https://dev-111464.oktaprevi
        responseType: ["token", "id_token"],
        display: "page"
      }
    });

    // Render the login form.
    function showLogin() {
      okta.renderEl({ el: "#okta-login-contai
        alert("Couldn't render the login form
      return okta.tokenManager.get("idToken")
    }

    // Retrieve the absolute room URL.
    function getRoomURL() {
      return location.protocol + "//" + locat
    }

    // Determine whether or not we have a que
    function hasQueryString() {
      return location.href.indexOf("?") !== -
    }

    // Handle the user's login and what happe
    function handleLogin() {
      // If the user is logging in for the fi
      if (okta.token.hasTokensInUrl()) {
        okta.token.parseTokensFromUrl(
          function success(res) {
            // Save the tokens for later use,
            okta.tokenManager.add("accessToke
            okta.tokenManager.add("idToken",

            // Redirect to this user's dedica
          // If the user is logged in, displa

          if (res.status === "ACTIVE") {

            // If the user is logged in on th

            if (!hasQueryString()) {
              window.location = getRoomURL();

            }

            return enableVideo();
          }

          // If we get here, the user is not

          // If there's a querystring in the
          // "room" so we should display our
          // we'll prompt them for login imme

          if (hasQueryString()) {
            document.getElementById("login").

            enableVideo();
          } else {
```

```
      document.getElementById("remotes").styl

      loadSimpleWebRTC();

    }

    // Dynamically load the simplewebrtc scri

    // kickstart the video call.

    function loadSimpleWebRTC() {
      var script = document.createElement("sc

      script.src = "https://simplewebrtc.com/

      document.head.appendChild(script);


      script.onload = function() {

        var webrtc = new SimpleWebRTC({

          localVideoEl: "selfVideo",
          // the id/element dom element that

          remoteVideosEl: "",

          autoRequestMedia: true,

          debug: false,

          detectSpeakingEvents: true,
          webrtc.joinRoom(getRoom());
        });

        function showVolume(el, volume) {
          if (!el) return;
          if (volume < -45) volume = -45; //
          if (volume > -20) volume = -20; //
          el.value = volume;
        }


        // Display the volume meter.
        webrtc.on("localStream", function(str
          var button = document.querySelector
          if (button) button.removeAttribute(
          document.getElementById("localVolum
        });

        // If we didn't get access to the cam
        webrtc.on("localMediaError", functior
          alert("This service only works if y
            var outerContainer = document.cre
            outerContainer.className = "col-m

            var container = document.createEl
            container.className = "videoConta
            container.id = "container_" + web
            container.appendChild(video);

            // Suppress right-clicks on the v
            video.oncontextmenu = function()


            // Show the volume meter.
            var vol = document.createElement(
            vol.id = "volume_" + peer.id;
            vol.className = "volume";
            vol.min = -45;
            vol.max = -20;
            vol.low = -40;
            vol.high = -25;
            container.appendChild(vol);
```

```
                    switch (peer.pc.iceConnectior

                      case "checking":
                        connstate.innerText = "cc

                        break;
                      case "connected":
                      case "completed": // on cal
                        vol.style.display = "bloc
                        connstate.innerText = "cc

                        break;
                      case "disconnected":
                        connstate.innerText = "di

                        break;
                      case "failed":
                        connstate.innerText = "cc

                        break;
                      case "closed":
                        connstate.innerText = "cc

                        break;
                    }
```

```
       if (!(remoteVideos % 2)) {
         var spacer = document.createEle
         spacer.className = "w-100";
         remotes.appendChild(spacer);
       }
     }
   });

   // If a user disconnects from chat, w
   webrtc.on("videoRemoved", function(vi
     console.log("user removed from chat
     var remotes = document.getElementBy
     var el = document.getElementById("c
     if (remotes && el) {
       remotes.removeChild(el.parentElem
     }
   });
```

```
        });


        // If there is a P2P failure, we neec
        webrtc.on("iceFailed", function(peer)
          var connstate = document.querySelec
          console.log("local fail", connstate

          if (connstate) {
            connstate.innerText = "connectior
            fileinput.disabled = "disabled";
          }
        });


        // remote p2p/ice failure
        webrtc.on("connectivityError", functi
          var connstate = document.querySelec
          console.log("remote fail", connstat

          if (connstate) {
            connstate.innerText = "connectior
            fileinput.disabled = "disabled";
```

```
</html>
```

With just a little bit of effort you were able to build a real-time video chat app using WebRTC to power video chat and Okta to power user authentication.

If you'd like to see a slightly more organized version of this project, you can check out my GitHub repo which contains the cleaned up source here: https://github.com/rdegges/chatapp

If you enjoyed this article but didn't follow through the tutorial, one thing you might want to do is create an Okta developer account (they're free) to handle any user management, authentication, authorization, single-sign on, social login, etc. It's a really powerful tool and quite useful when building web apps (especially those without server-side backends).

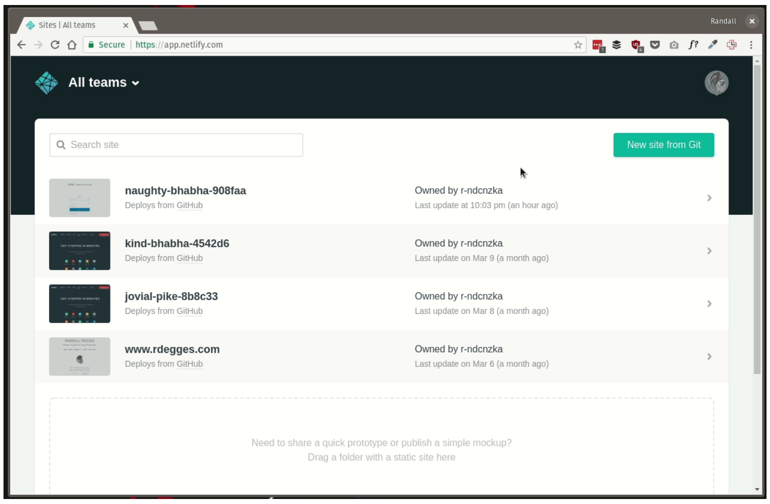
If you'd like to see more articles like this, tweet us @oktadev and let me know! We've also got a ton of other interesting developer articles you can find on the Okta developer blog.

Bonus: Deploy Your Site Live with Netlify

If you're anything like me then you hate following tutorials only to discover that the app you were building only runs on localhost. If you'd like to deploy your new Okta + WebRTC app live to the world in a simple way, I'd highly recommend using Netlify.

a new Netlify app (you can have as many as you want, one for each website you run). Simply click the **New site from Git** button at the top right of your dashboard, then select your hosting provider (GitHub in my case), find your project, then configure it.

Here's what this process looks like for me:



naughty-bhabha-908faa
Deploys from GitHub
Owned by r-ndcnzka
Last update at 10:03 pm (an hour ago)

kind-bhabha-4542d6
Deploys from GitHub
Owned by r-ndcnzka
Last update on Mar 9 (a month ago)

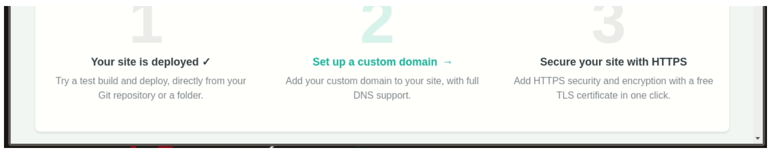
jovial-pike-8b8c33
Deploys from GitHub
Owned by r-ndcnzka
Last update on Mar 8 (a month ago)

www.rdegges.com
Deploys from GitHub
Owned by r-ndcnzka
Last update on Mar 6 (a month ago)

Need to share a quick prototype or publish a simple mockup?
Drag a folder with a static site here

In just a few short seconds I'm able to sync my GitHub repo with Netlify so that it auto-deploys my website live to its CDN (over SSL) — pretty amazing, right? From this point forward, each time you push a new commit to your repo (depending on how you've configured Netlify) it will automatically deploy your static site to its CDN.

And... once you've gotten your site all setup in Netlify, the next thing you'll want to do is optimize it! Netlify has some great functionality that will automatically


1
Your site is deployed ✓
Try a test build and deploy, directly from your Git repository or a folder.
2
Set up a custom domain →
Add your custom domain to your site, with full DNS support.
3
Secure your site with HTTPS
Add HTTPS security and encryption with a free TLS certificate in one click.

With just a few clicks you're able to speed up your site dramatically by compressing all your images, JS, and CSS.

If you'd like to use a real URL instead of the Netlify default project URL, you can easily do that through the **Domain management**tab which allows you to do things like setup your own DNS, generate SSL certs for free, etc.

Overall, it's a really nice service that I've come love for all my projects.

Anyhow: I hope this tutorial was useful and helpful! Please leave a comment if you have any questions or feedback below and I'll do my best to reply.

"I love writing authentication and authorization code." ~ No Developer Ever. Try Okta instead

---

Topics: WEB DEV, JAVASCRIPT, VIDEO CHAT, OKTA, WEBRTC, TUTORIAL