Okta Customer Identity for Developers Lab Guide

Okta
Education Services

Table of Contents

Module 1: Introducing Okta

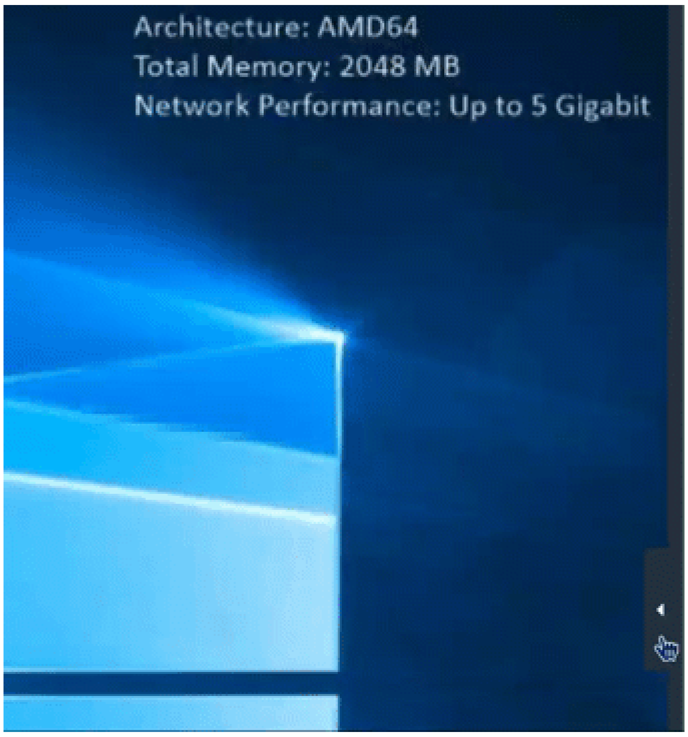
Lab 1.1: Access Your Okta Org

🎯 **Objective**: Sign in to your virtual machine and authenticate to your Okta organization.

⏱ **Duration**: 15 minutes

Access Your Credentials

In your VM, click on the arrow to the right of the Desktop to expand the **Credentials** panel that will reveal your Okta org credentials.


Architecture: AMD64
Total Memory: 2048 MB
Network Performance: Up to 5 Gigabit

Keep this panel open for the following step.

Access Your Okta Admin Dashboard

1. Launch the Chrome browser.
2. In your VM's **Credentials** panel, click the icon in the `Link` section. This will automatically paste your Okta org URL into the Chrome browser.
3. Click into the URL address bar in the Chrome browser and hit `Enter` to navigate to your Okta org.
4. Once the Okta Sign-In page loads, click into the `Username` field.
5. In your VM's **Credentials** panel, click the icon in the `Login` section. This will automatically paste your admin account's username into the **Username** field.
6. In the Okta interface, click `Next`
7. In your VM's **Credentials** panel, click the icon in the `Password` section. This will automatically paste your admin account's password into the **Password** field.
8. In the Okta interface, click `Verify`
9. Once logged in to Okta, click the `Admin` button to enter the Admin Dashboard.

You may now collapse the Credentials side panel in your VM.

✅ Checkpoint

At this point, you have access to your lab environment to complete the rest of the labs.

Lab 1.2: Create Okta Groups

🎯 **Objective** Create Okta Groups -- one for Okta Ice Franchisees and one for Okta Ice customers. Create a Group rule for automatically adding certain users to the Customers group.

🎬 **Scenario** Franchisees and customers require distinct groups for application access and access policies.

⏱️ **Duration** 10 minutes

Create a Franchisee Group

1. Ensure you are logged in as your Okta Super Admin account `oktatraining` and that you are on the `Admin` dashboard.
2. Navigate to `Directory > Groups`
3. Click `Add Group`
4. In the `Name` field, enter `Franchisees`
5. In the `Description` field, enter `All Franchisees`
6. Click `Save`

Create a Customer Group

1. Click `Add Group` again.
2. This time, enter `Customers` in the `Name` field and `All Customers` in the `Description` field.
3. Click `Save`

Create a Group Rule for the Customer Group

Now we will create a rule so that any user created that has the `userType` `customer` will automatically be added to the Customer group. This will be helpful when we import existing users with the Users API in Module 6.

1. On the top of the **Groups** page, click the `Rules` tab.
2. Click **Add Rule**
3. Name the rule `Add customer userType to Customers Group`
4. Set the **IF** section to read: IF `User attribute` `userType` `Equals` `customer`
5. In the **THEN Assign to** section, type and select the `Customers` group.
6. Click `Save`

Activate the Group Rule

You should now see the `Add customer userType to Customers Group` rule listed on the **Group Rules** page.

Notice, however, that the `Status` is `Inactive`, so we'll need to activate it:

1. Click the drop down next to `Inactive`
2. Select `Activate`

✅ Checkpoint

You now have two Okta Groups that you will use to manage access to applications.

Lab 1.3 Create Okta Users

🎯 **Objective**: Create some End User accounts and assign them to Okta Groups for testing configurations.

🎬 **Scenario**: We'll need some End Users to test out the Franchisee and Customer experience.

⏱️ **Duration**: 15 minutes

Navigate to the People Directory

1. Ensure you are signed in as your Super Admin account, `oktatraining`.
2. In the Admin Dashboard, select `Directory` > `People`.

Create and Add a Test User to the Franchisee Group

1. Click the `Add Person` button.
2. Enter the following details:

Field	Value
First name	`Kay`
Last name	`West`
Username	`kay.west@oktaice.com`
Primary email	`kay.west@oktaice.com`
Group	`Franchisees`
Activation	`Activate now`
I will set the password	`CHECKED`
Enter password	`Tra!nme4321`
User must change password on first login	`UNCHECKED`

Last, click the `Save and Add Another` button.

Create and Add a Test User to the Customer Group

Enter the following details:

Field	Value
First name	`Soraya`
Last name	`Esfeh`
Username	`soraya.esfeh@oktaice.com`
Primary email	`soraya.esfeh@oktaice.com`
Groups	`Customers`
Activation	`Activate now`
I will set the password	`CHECKED`
Enter password	`Tra!nme4321`
User must change password on first login	`UNCHECKED`

Last, click the `Save` button.

✅ Checkpoint

You now have a test user in the Franchisee group and a test user in the Customer group.

Lab 1.4: Create Okta Application Integrations

🎯 **Objective** Create Okta application integrations for two existing applications. Assign one application to the Customers group and assign one application to the Franchisee group.

🎬 **Scenario** Customers and Franchisees require access to a different set of applications.

⏱️ **Duration** 15 minutes

Enable Cross-Origin Resource Sharing (CORS)

In Okta, **CORS** allows JavaScript hosted on your website to make a request using an `XMLHttpRequest` to the Okta API. While we aren't hosting the Okta Sign-In Widget when we use the Redirect Model, our sample applications ***do*** call out to Okta's API to close an Okta session and log a user out. Every website origin must be explicitly permitted as a **Trusted Origin** in your Okta org.

1. Ensure you are logged in to the Admin Dashboard as `oktatraining`
2. In the Admin menu, navigate to `Security` > `API`
3. Click on the `Trusted Origins` tab.
4. Click the `Add Origin` button
5. Enter `Okta Ice Portal` into the **Origin name** field.
6. Enter `http://localhost:8080` into the **Origin URL** field.
7. Under **Choose Type**, select `Cross-Origin Resource Sharing (CORS)`
8. Click `Save`

Navigate to Applications

In the Admin menu, navigate to `Applications` > `Applications`

Create an Application Integration for the Rewards App

The Rewards app is an existing application in this project. We're going to set up an integration in Okta so our customers can access this application. We'll talk about the code implementation in subsequent modules.

1. Click `Create App Integration`
2. Select the `OIDC - OpenID Connect` radio button.
3. Select the `Single-Page Application` radio button.
4. Click `Next`
5. Name this application `Customer Rewards`
6. In the **Sign-in redirect URIs** field, enter `http://localhost:8080/redirect/rewards.html`

7. Under **Assignments** click the radio button option for `Limit access to select groups`

8. Type in and select `Customers`

9. Click `Save`

Configure the Rewards Application `appClientId`

The Rewards application makes use of Okta's AuthJS SDK, which we will learn about more in the next module. For now, you'll simply need to configure the ClientID and Okta Org URL to make this integration work.

1. Copy the `Client ID` that was displayed after you saved your integration.

2. Here in VSCode, notice that `rewards.html` has opened for you automatically and the line containing the `appClientID` variable is highlighted. Currently there is a placeholder value of `"XXXXXXXXXXXXXXXXXXXXXX"`

3. Double click the placeholder value to highlight its contents and press `CTRL`+`V` to paste your `Client ID` value so that it is enclosed in the double quotes.

Configure the Rewards Application `baseOktaURL`

1. Notice that `rewards.html` is still open, but now the line containg the the `baseOktaURL` variable is highlighted. It contains the placeholder value `https://oktaice######.oktapreview.com`

2. Highlight the placeholder value **excluding** the quotation marks. Ensure your cursor is now between the quotation marks.

3. Open your VM's **Credentials** panel by clicking on the arrow to the right hand side of the VM screen.

4. In your VM's **Credentials** panel, click the icon in the `Link` section. This will automatically paste your Okta org URL into `rewards.html` where your cursor was placed. Ensure it was pasted in the correct location and that the URL is enclosed in double quotes.

5. Save your `rewards.html` file.

Create an Application Integration for the CRM App

We will complete the same steps for the Franchisee app. The one difference is we will assign this app to the Franchisee group. See how many steps you can complete without referring to the instructions!

1. Back in Okta, click `← Back to Applications`

2. Click `Create App Integration`

3. Select the `OIDC - OpenID Connect` radio button.

4. Select the `Single-Page Application` radio button.

5. Click `Next`

6. Name this application `Franchisee CRM`

7. In the **Sign-in redirect URIs** field, enter `http://localhost:8080/redirect/crm.html`
8. Under **Assignments** click the radio button option for `Limit access to select groups`
9. Type in and select `Franchisees`
10. Click `Save`

Configure the CRM Application `appClientId`

1. Copy the `Client ID` that was displayed after you saved your integration.
2. Here in VSCode, notice that `crm.html` has opened for you automatically and the line containing the `appClientID` variable is highlighted. Currently there is a placeholder value of `"XXXXXXXXXXXXXXXXXXXXXX"`
3. Double click the placeholder value to highlight its contents and press `CTRL`+`V` to paste your `Client ID` value so that it is enclosed in the double quotes.

Configure the CRM Application `baseOktaURL`

1. Notice that `crm.html` is still open, but now the line containg the the `baseOktaURL` variable is highlighted. It contains the placeholder value `https://oktaice######.oktapreview.com`
2. Highlight the placeholder value **excluding** the quotation marks. Ensure your cursor is now between the quotation marks.
3. Open your VM's **Credentials** panel by clicking on the arrow to the right hand side of the VM screen.
4. In your VM's **Credentials** panel, click the icon in the `Link` section. This will automatically paste your Okta org URL into `crm.html` where your cursor was placed. Ensure it was pasted in the correct location and that the URL is enclosed in double quotes.
5. Save your `crm.html` file.
6. **Log out** of your Okta org since you will be logging in as a different user in the subsequent steps.

Start the Web Server

1. Open a new terminal in VSCode
2. Notice that the terminal automatically opens to the project directory.
3. Issue the command `python -m http.server 8080`

Test out the Rewards App

Before following the directions below, that ensure you are **logged out** of any Okta sessions. **DO NOT use an Incognito browser window** for the steps that follow.

1. In Chrome, visit http://localhost:8080
2. Click the `Rewards App (Redirect)` link. You should be redirected to Okta to sign in.

3. Log in as `soraya.esfeh@oktaice.com`/`Tra!nme4321`

You should be redirected back to the Rewards app and you should see some information printed to the page that we will discuss later. For now you can see Soraya's email address at the top as well as your app's Client ID at the bottom.

Try to Access the CRM App

1. Still logged in as Soraya, click `Return to Portal`
2. Click on `CRM App (Redirect)`
3. The CRM application does not load because Soraya is not authorized to access this application. The CRM app is only assigned to users in the `Franchisees` group and she is not a member of that group.
4. Click the `Close Okta Session` button.
5. Click `Return to Portal`

Log in as a Franchisee Partner

Now we're going to log in as Kay West, who is a member of the Franchisees group.

1. Click on `Rewards App (Redirect)`
2. Enter `kay.west@oktaice.com` as the login and click `Next`
3. Notice that this user did not have an existing session with Okta before trying to access an application that is not assigned to them. In this case, we are not even prompted for a password to authenticate and we are told that the application is not assigned to the user. The user still does not have a session.
4. Click `Return to Portal`
5. Click on `CRM App (Redirect)`
6. Log in as `kay.west@oktaice.com`/`Tra!nme4321`

You should be redirected back to the CRM app and you should see see Kay's email address at the top as well as your app's Client ID at the bottom.

End Your Okta Session and Shut Down the Web Server

1. Click the `Close Okta Session` button.
2. In the terminal window in VSCode where you launched the web server, press `CTRL`+`C` to stop the web server.

✅ Checkpoint

You now know how to set up an application integration in Okta.

🎉 End of Module 1 Labs

You may close this workspace project, ensuring all changes were saved.

Module 2: Customizing the Okta-Hosted Sign In Widget

Lab 2.1 Configure a Custom Domain

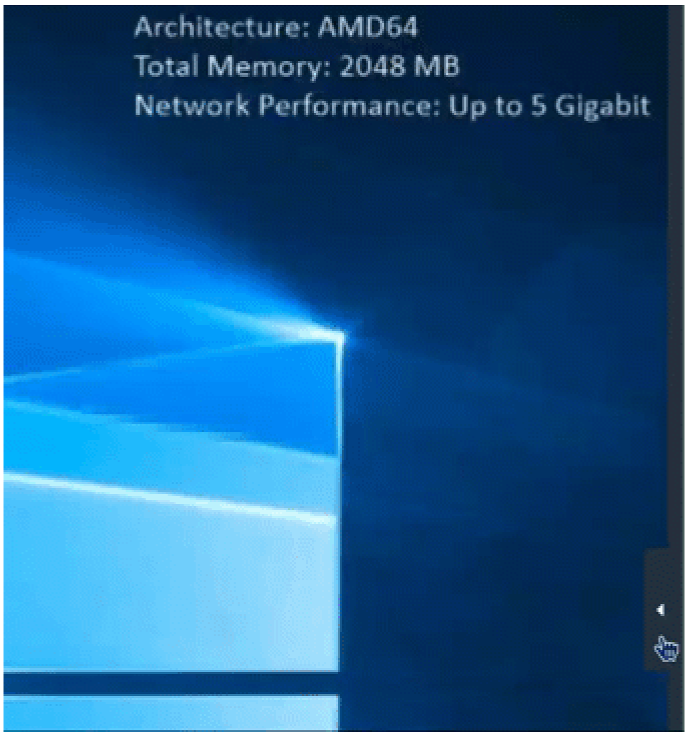
🎯 **Objective:** Configure a custom domain in Okta and configure your DNS to support the custom URL.

🎬 **Scenario** Continuing with the white label experience, Okta Ice wants to change the Okta URL to its own domain. In this lab, you will configure Okta and your DNS to change the Okta URL end-users will use.

⏱️ **Duration:** 20 minutes

Configure a Subdomain

1. Log in to your Okta org as `oktatraining`.
2. Click `Customizations` > `Domain`.
3. Click `Edit` and then click the `Get started` button.
4. To prepare for the next step, expand the **Credentials** panel in your VM.



Architecture: AMD64
Total Memory: 2048 MB
Network Performance: Up to 5 Gigabit

Fill Out Your Fully-Qualified Domain

The fully-qualified domain you will use for this step consists of a **subdomain** and a **domain** in the format `oktaice######.coffee-ice.com`

1. In the **Add domain** form on Okta, click into the `Domain` field.

2. In the **Credentials** panel in your VM, click the icon next to `Subdomain` to paste your subdomain into this field.

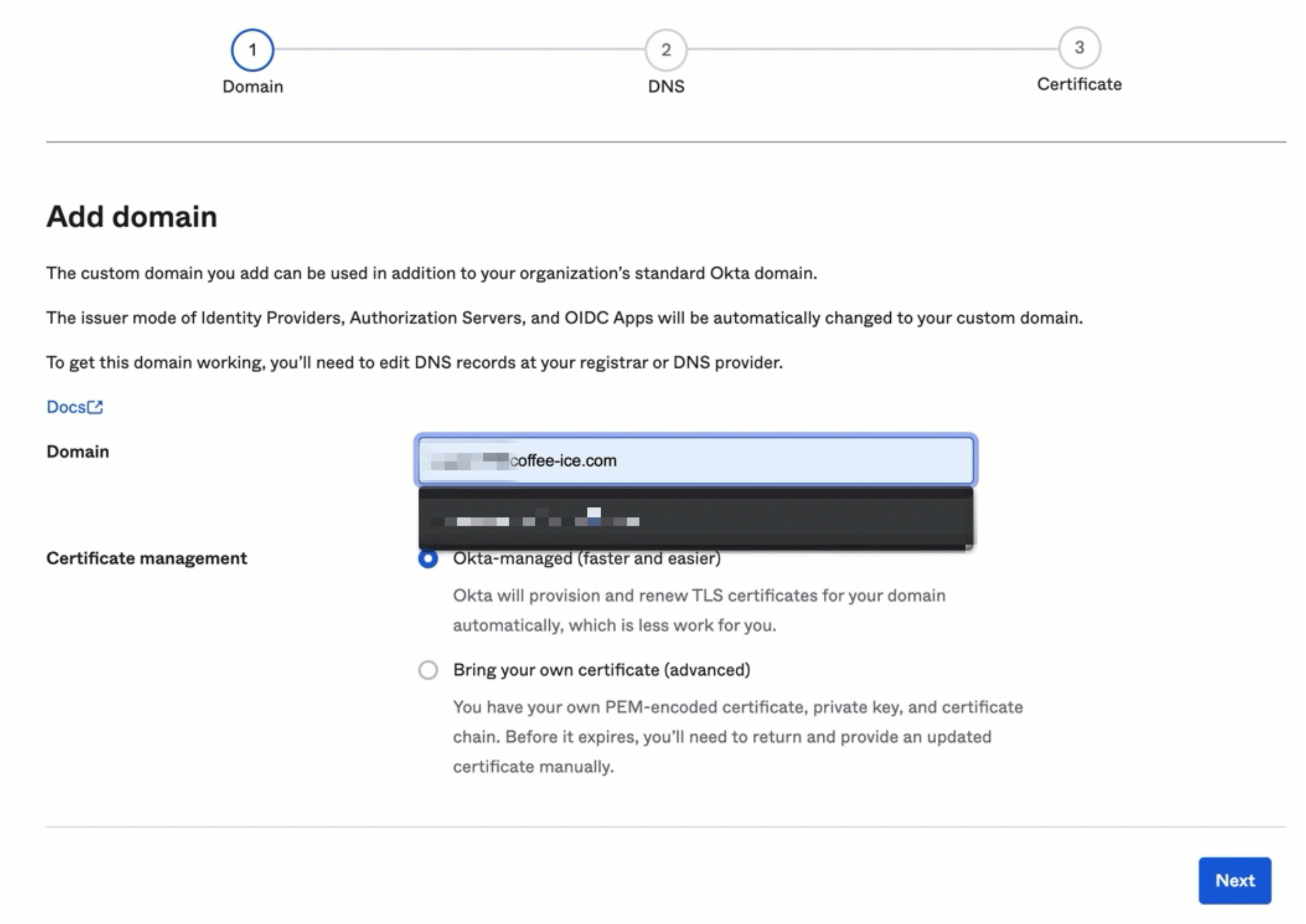
3. Click back into the `Domain` field in the **Add domain** form.

4. Append `.coffee-ice.com` after your subdomain. It should now look something like `oktaice######.coffee-ice.com`

5. Under **Certificate management**, ensure `Okta-managed` is selected.

6. Click `Next`.

Okta will now provide you with a `TXT` value and a `CNAME` value, which we will use in the next step to verify that we own this domain.



Add domain

The custom domain you add can be used in addition to your organization's standard Okta domain.

The issuer mode of Identity Providers, Authorization Servers, and OIDC Apps will be automatically changed to your custom domain.

To get this domain working, you'll need to edit DNS records at your registrar or DNS provider.

Docs

Domain: [redacted]coffee-ice.com

[redacted]

Certificate management: [✓ Okta-managed (faster and easier)]

Okta will provision and renew TLS certificates for your domain automatically, which is less work for you.

Bring your own certificate (advanced)

You have your own PEM-encoded certificate, private key, and certificate chain. Before it expires, you'll need to return and provide an updated certificate manually.

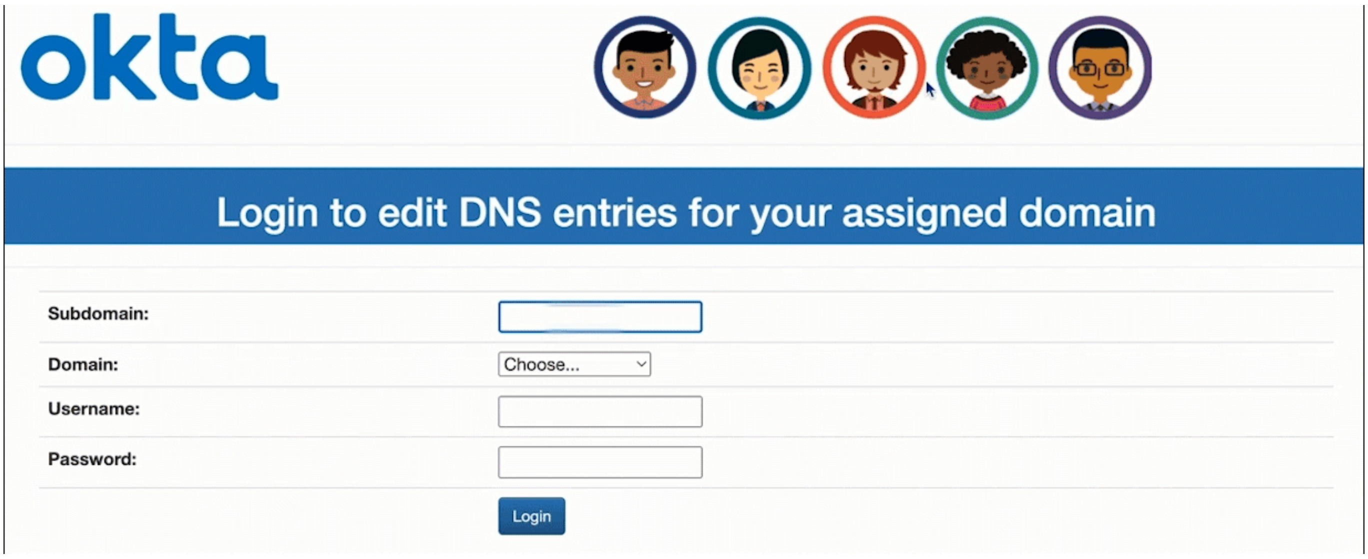
Log In to the DNS Configuration Tool

In a new tab, navigate to our DNS configuration tool at https://oktaice.store/ciam/ and log in with the following information (replacing ####### with your assigned subdomain number from the **Credentials** panel in your VM)

Field Name	Value
Subdomain	`oktaice######`
Domain	`coffee-ice.com`
Username	`oktatraining`
Password	`Tra!nme4321`

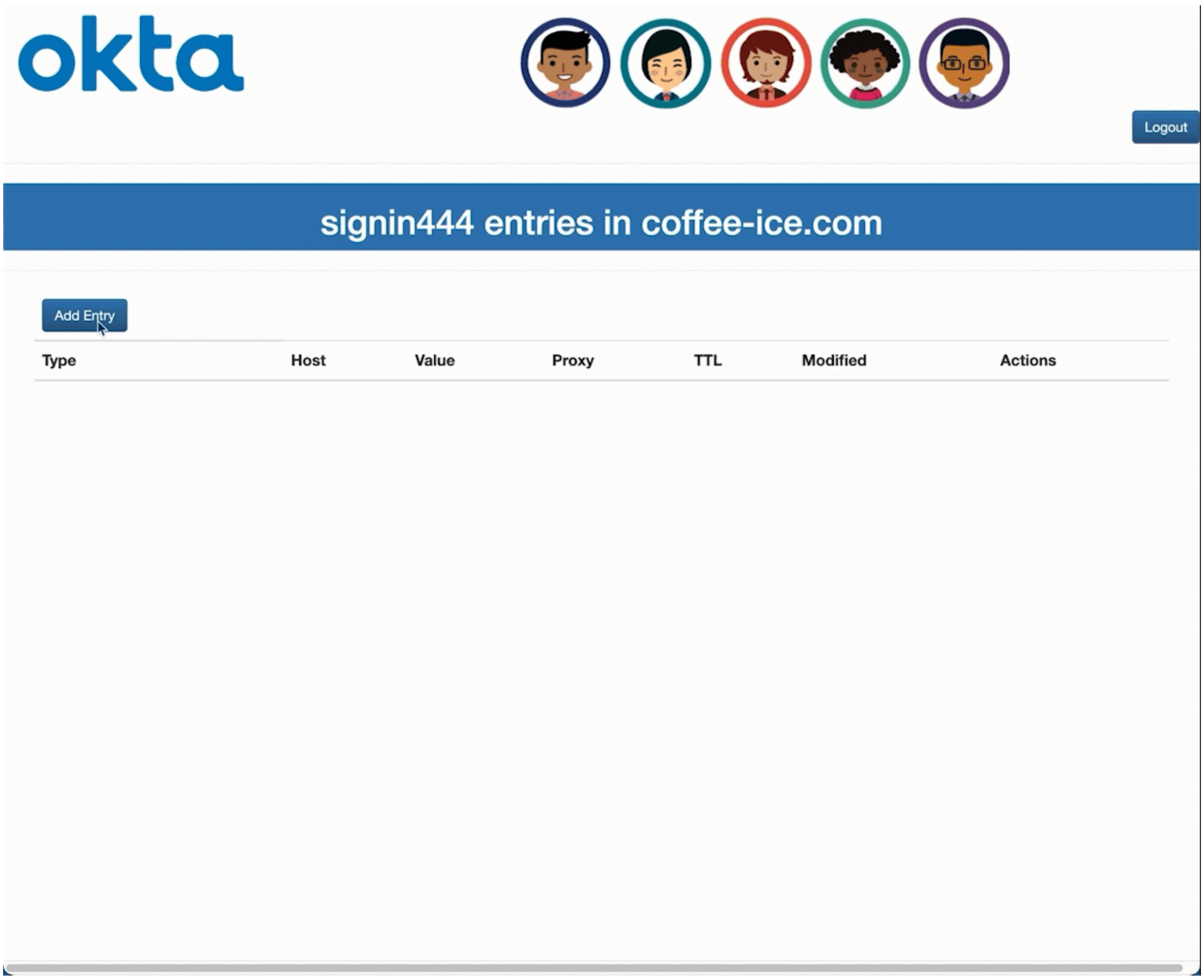
When you log in, verify that you have logged in to modify the DNS entries for the correct subdomain and domain.

There should not be any entries. If there are, delete them.



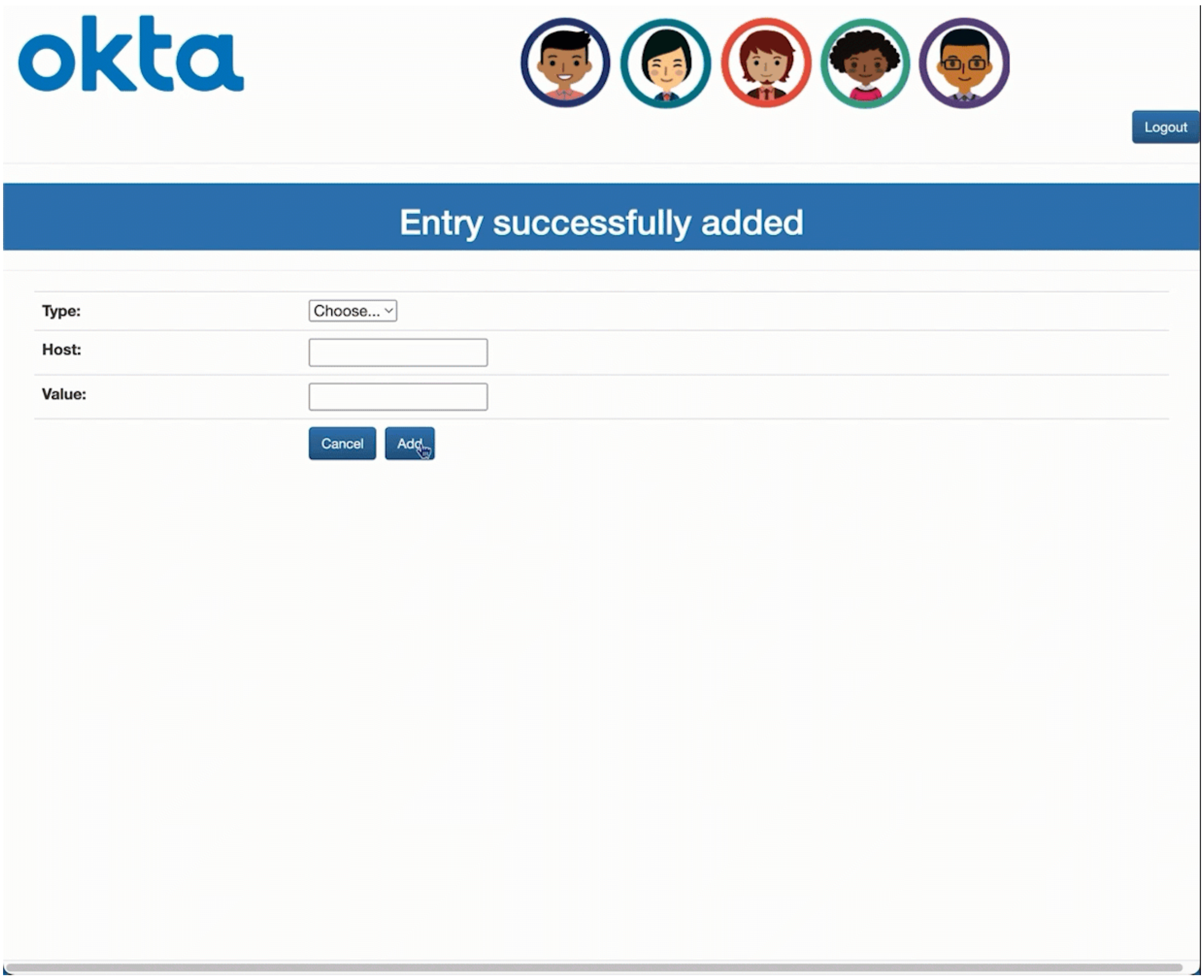
Subdomain:	
Domain:	Choose...
Username:	
Password:	

1. Click `Add Entry`
2. In the `Type` section, choose `TXT`
3. Switch back to your Okta tab
4. Copy the **Host** value next to in the **TXT** row (e.g.,`_acme-challenge.oktaice######.coffee-ice.com`)
5. Switch back to the DNS configuration tool
6. Paste this text into the `Host` field
7. Switch back to your Okta tab
8. Copy the **Value** entry in the **TXT** row (a unique value generated by Okta)
9. Switch back to the DNS configuration tool
10. Paste this text into the `Value` field
11. Click `Add`



Add the CNAME Entry

1. In the `Type` section, choose `CNAME`
2. Switch back to your Okta tab
3. Copy the **Host** entry in the **CNAME** row (e.g., `oktaice######.coffee-ice.com`)
4. Switch back to the DNS configuration tool
5. Paste this text into the `Host` field
6. Switch back to your Okta tab
7. Copy the **Value** entry in the **CNAME** row (Your Okta org URL)
8. Switch back to the DNS configuration tool
9. Paste this text into the `Value` field
10. Click `Add`
11. Click `Cancel`
12. You should now see the two entries you added
13. You can now log out of the DNS configuration tool and close the tab



Entry successfully added

Type:	Choose...
Host:	
Value:	

Issue an Okta-Managed Certificate

Navigate back to the Okta tab and click `Next`. Okta will take a moment to verify the DNS 01 challenge:

- If verification fails, go back and check that you entered both the TXT and CNAME records correctly
- If verification is successful, Okta will issue a TLS certificate for your custom domain. Click `Finish`.

You will see a status of **Pending** for your custom domain as it can take several minutes to half an hour for the domain to become ready.

✅ Checkpoint

At this point, you have configured a custom domain for your Okta org. Setting up a custom domain also allows you to customize the Okta-hosted SIW.

Lab 2.2: Customize the Okta Sign-In Page With the Branding UI

🎯 **Objective:** Customize the Okta Sign-In Page with Okta Ice branding.

🎬 **Scenario** Okta Ice needs a branded Sign-In Page for a complete white label experience.

⏱️ **Duration:** 20 minutes

⚠ **Prerequisite:** Lab 2-1

Access the Sign-in Page Code Editor

1. If you aren't already, sign in to your Okta org as your Super Admin acount `oktatraining`
2. From the Admin dashboard, click `Customizations` > `Sign-in page code editor`

You will see the default Sign-in page code, which we'll walk through next.

The Default Sign-in Page Code

The code from the default Okta-hosted sign-in Page has been copied here so we can explore it more in depth. You can customize the Okta-hosted sign-in page, not only by modifying the code in the Sign-in Page Code Editor, but by configuring settings in Okta's Branding UI. Your Okta org must have Branding enabled to use these macros in the Sign In Page code editor.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0"
/>
    <meta name="robots" content="noindex,nofollow" />
    <!-- Styles generated from theme -->
    <link href="{{themedStylesUrl}}" rel="stylesheet" type="text/css" />
    <!-- Favicon from theme -->
    <link rel="shortcut icon" href="{{faviconUrl}}" type="image/x-icon" />

    <title>{{pageTitle}}</title>
    {{{SignInWidgetResources}}}
  </head>
  <body>
    <div
      class="login-bg-image tb--background"
      style="background-image: {{bgImageUrl}}"
    ></div>
    <div id="okta-login-container"></div>

    <!--
        "OktaUtil" defines a global OktaUtil object
        that contains methods used to complete the Okta login flow.
     -->
    {{{OktaUtil}}}

    <script type="text/javascript">
      // "config" object contains default widget configuration
      // with any custom overrides defined in your admin settings.
      var config = OktaUtil.getSignInWidgetConfig();

      // Render the Okta Sign-In Widget
```

```
        var oktaSignIn = new OktaSignIn(config);
        oktaSignIn.renderEl(
          { el: "#okta-login-container" },
          OktaUtil.completeLogin,
          function (error) {
            // Logs errors that occur when configuring the widget.
            // Remove or replace this with your own custom error handler.
            console.log(error.message, error);
          }
        );
      </script>
    </body>
  </html>
```

Examine the Sign-in Page Macros

Throughout the default Sign-in Page code, you will see entries surrounded by curly braces `{{}}` such as the one highlighted here. These are called **macros**. Macros generate code that can be applied in the Sign-In Page code editor. The data these macros depend upon are configurable in Okta's Branding UI.

Sign-in Page Macro: `themedStylesUrl`

The macro `{{themedStylesUrl}}` generates a CSS file that defines the primary and secondary colors for this page.

Sign-in Page Macro: `faviconUrl`

The `{{faviconUrl}}` macro inserts the URL for the favicon. The favicon can be changed in the Okta Branding UI.

Sign-in Page Macro: `pageTitle`

The `{{pageTitle}}` macro inserts the page title set in the Okta Branding UI.

Sign-in Page Macro: `SignInWidgetResource`

The `SignInWidgetResource` inserts the JavaScript and CSS files required to use the Okta Sign-In Widget.

Sign-in Page Macro: `bgImageUrl`

The `{{bgImageUrl}}` macro inserts the URL to the background image configured in your Okta organization. You can upload and set this background image in the Okta Branding UI.

Sign-in Page Macro: `OktaUtil`

Defines a global `OktaUtil` JavaScript object that contains methods used to complete the Okta sign-in flow. When an application uses the Okta-hosted sign-in page to sign a user in, information (called request context) is available about the target application and the request.

Sign-In Widget JavaScript

Next we'll walk through this bit of JavaScript that is responsible for rendering the Okta Sign-In Widget (SIW) on the sign in page. The SIW itself is a JavaScript library.

SIW JavaScript: `config` Variable

The `config` variable gets the configuration of the Sign-In Widget, This configuration, at minimum, defines the `issuer`, `clientId`, and `redirectUri`

SIW JavaScript: Creating an `OktaSignIn` Object

This line instantiates a new `OktaSignIn` object with the configurations we have stored in `config`. This will be used to render the SIW in the following lines.

SIW JavaScript: Rendering the Okta SIW

The `OktaSignIn` object is then used to call the `renderEl()` method to render the SIW. Let's take a look at the method signature to break down what's being passed to this method:

```
 renderEl(renderOptions: RenderOptions, successFn?: RenderSuccessCallback,
errorFn?: RenderErrorCallback): Promise<RenderResult>
```

Render Options

This parameter takes render options for the SIW in the form of JSON data. It must include the `el` or `$el` property. In this case, we're passing a CSS class `#okta-login-container`, which will render the SIW in a `div` of this class.

Success Callback

This parameter expects a function, which gets called upon successfully configuring and rendering the widget.

Error Callback

This paramater expects a function, which gets called if there is an error when configuring and rendering the widget. Currently, we are simply logging the error to console.

Access the Okta Branding UI

Before we make any changes to the default Sign-in Page code, let's see what we can customize the SIW without touching any code at all. We'll do this with Okta's Branding UI.

Navigate to `Customize` > `Branding`

Under the `Theme` section, you can define:

- Primary and Secondary colors
- Logo
- Favicon

- Background

Customize the Primary and Secondary Colors

In the default Sign-in Page code, the primary color is applied to the `Sign In` button and the secondary color is applied to the background color of the Sign-in page.

Change the Primary color to `#2d75bb`

Change the Secondary color to `#c7e5f5`

Customize the Logo

Next, we're going to upload a custom logo to be displayed at the top of the SIW.

1. Click the pencil icon next to the `Logo` item
2. Select `Upload new image`
3. Open `C:\ClassFiles\02-customizing-okta-hosted-siw\img\ice-logo.png`

Customize the Favicon

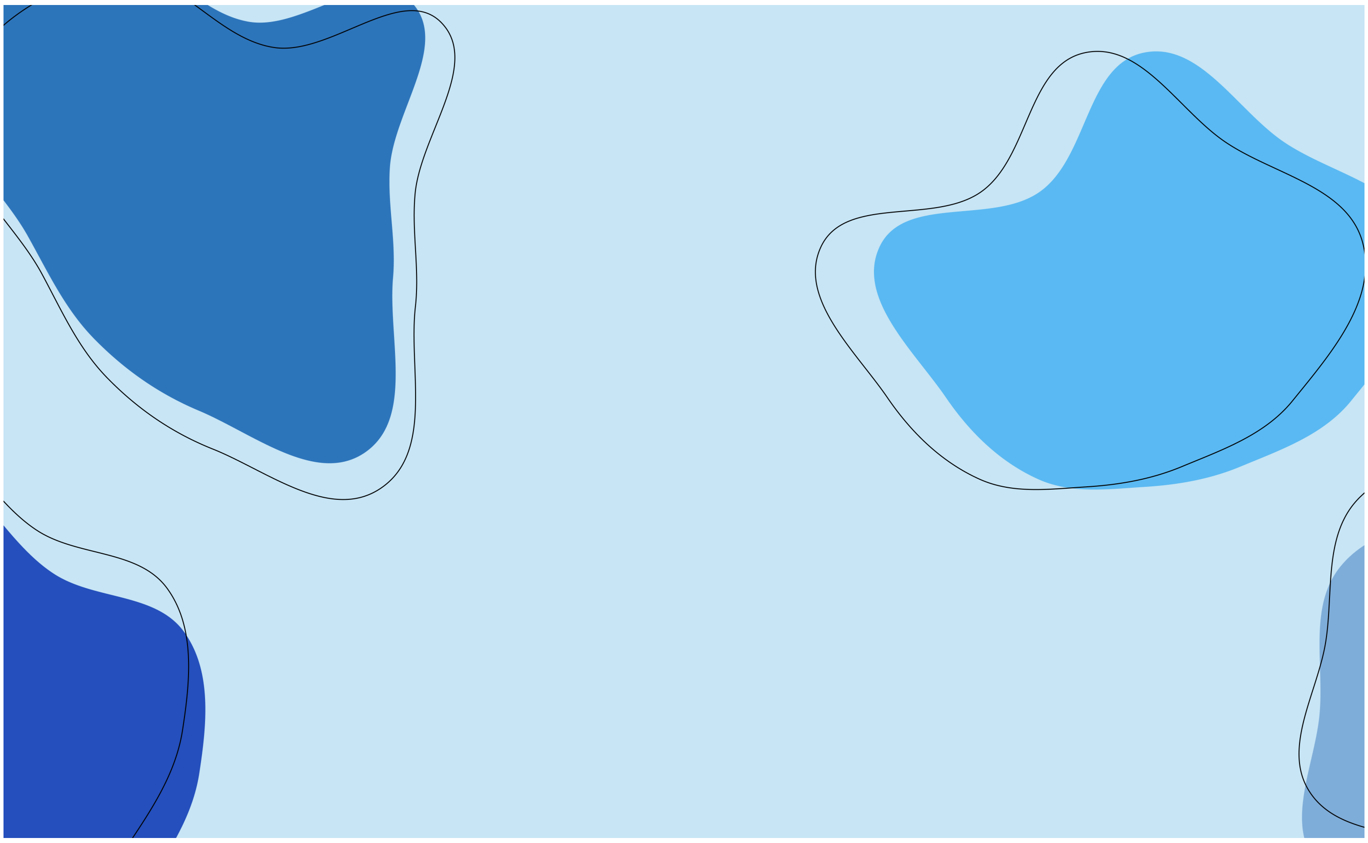
Next, we're going to upload a custom favicon.

1. Click the pencil icon next to the `Favicon` item
2. Select `Upload new image`
3. Open `C:\ClassFiles\02-customizing-okta-hosted-siw\img\favicon.png`

Customize the Background Image

Next, we're going to upload a custom background image.

1. Click the pencil icon next to the `Background` item
2. Select `Upload new image`
3. Open `C:\ClassFiles\02-customizing-okta-hosted-siw\img\ice-cream-bg.png`



Click `Save Changes`

Now that we have customized our theme, we need to apply it to our Sign-in page.

1. In the section labeled `Pages`, click `Edit` on the item labelled `Okta-hosted Sign-In Page`
2. Click the style labeled `Solid background` and observe the change in the preview panel. This style uses our custom logo, applies the primary color to the `Sign In` button, and uses the secondary color as the solid background for the page.
3. Click the style labelled `Image background` and observe the change in the preview panel. This style is the same as the above, but it applies our image background to the page.
4. Click `Save and Publish`

Preview the Sign-In Page

We're going to navigate back to the Sign-in page code editor to preview the Sign In Page and add some additional customizations.

Leave this page open for the next lab

1. Navigate to `Customizations` > `Sign-in page code editor`
2. Click `Preview`

You should now see a customized Sign-in Page, which we've achieved without changing any code.

Leave this page open for the next lab.

✅ Checkpoint

We have seen how we could customize the SIW without touching the Sign-In Page code through Okta's Branding UI. For some use cases, this level of customization is enough! For other use cases, we might want to customize the Sign-In Page beyond what Okta's Branding UI can offer. In the next lab, we use the integrated Sign-In Page code editor to achieve this.

Lab 2.3: Customize the Okta Sign-In Page Using the Sign-In Page Code Editor

🎯 **Objective:** Further customize the Sign-In Page with the Sign-In Page Code Editor.

🎬 **Scenario** Okta Ice wants to further customize the Sign-In Page beyond what is offered by the Okta Branding UI.

⏱️ **Duration:** 20 minutes

⚠️ **Prerequisite:** Lab 2-1 and 2-2

Note about this lab:

In this lab, you will be adding code to the default Okta Sign-In Page html. For the best learning experience, you are highly encouraged to type the code in manually.

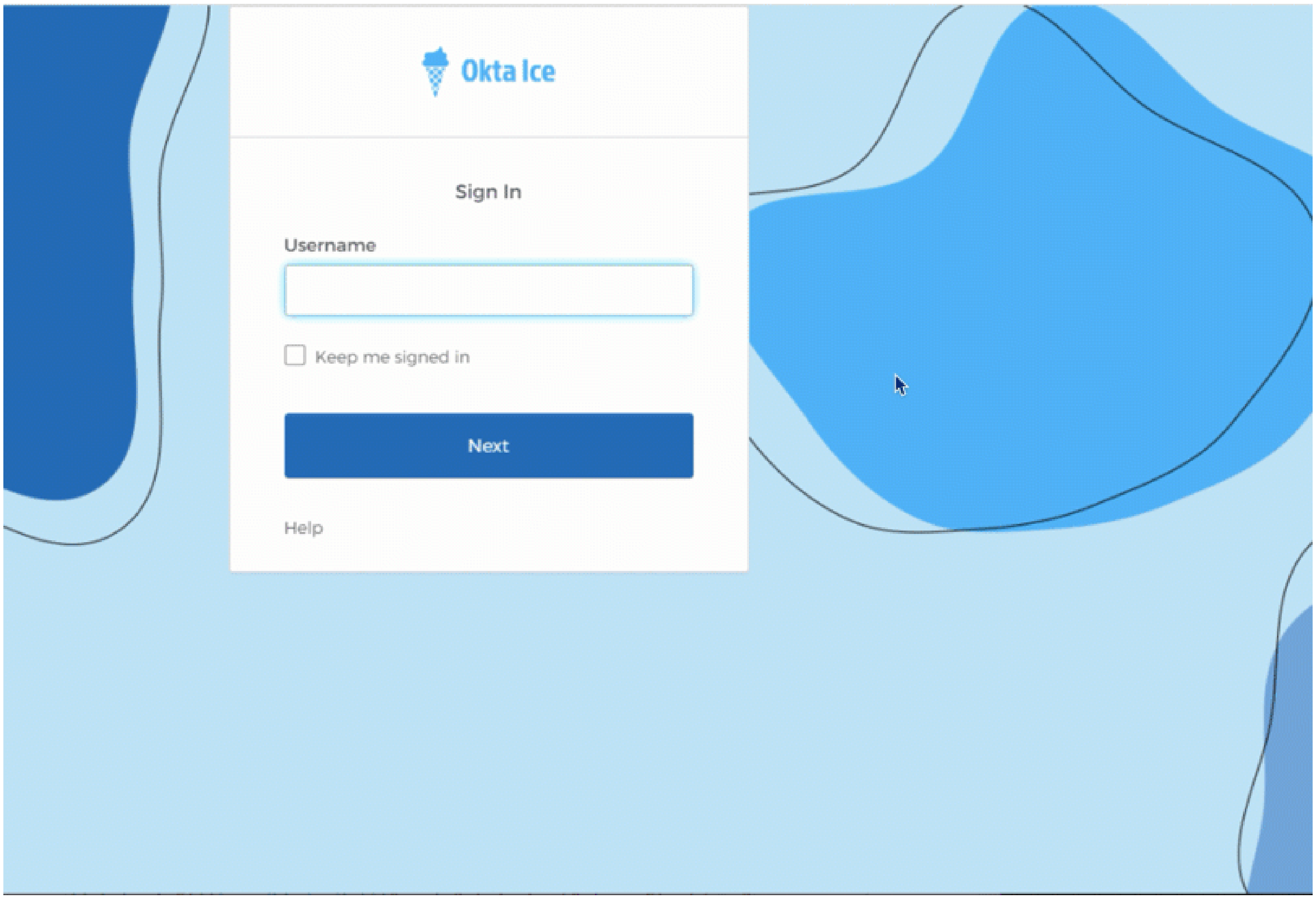
To provide a differentiated experience for learners of all types, we've included an `Insert Code` link after each coding segment. This inserts the code to the correct line(s) automatically. After you insert the code, you will still have to click `Next` to advance to the next lab step.

Inspect the Sign-In Page Elements

You should still have the Sign-In Page preview open from the last lab. If you do not have this page open, ensure you are signed in as your `oktatraining` and:

1. Navigate to `Customizations` > `Sign-in page code editor`
2. Click `Preview`

On the preview of your Sign-In Page, right click on the page and select `Inspect` and ensure that you are viewing the `Elements` tab and the `Style` sub-tab.

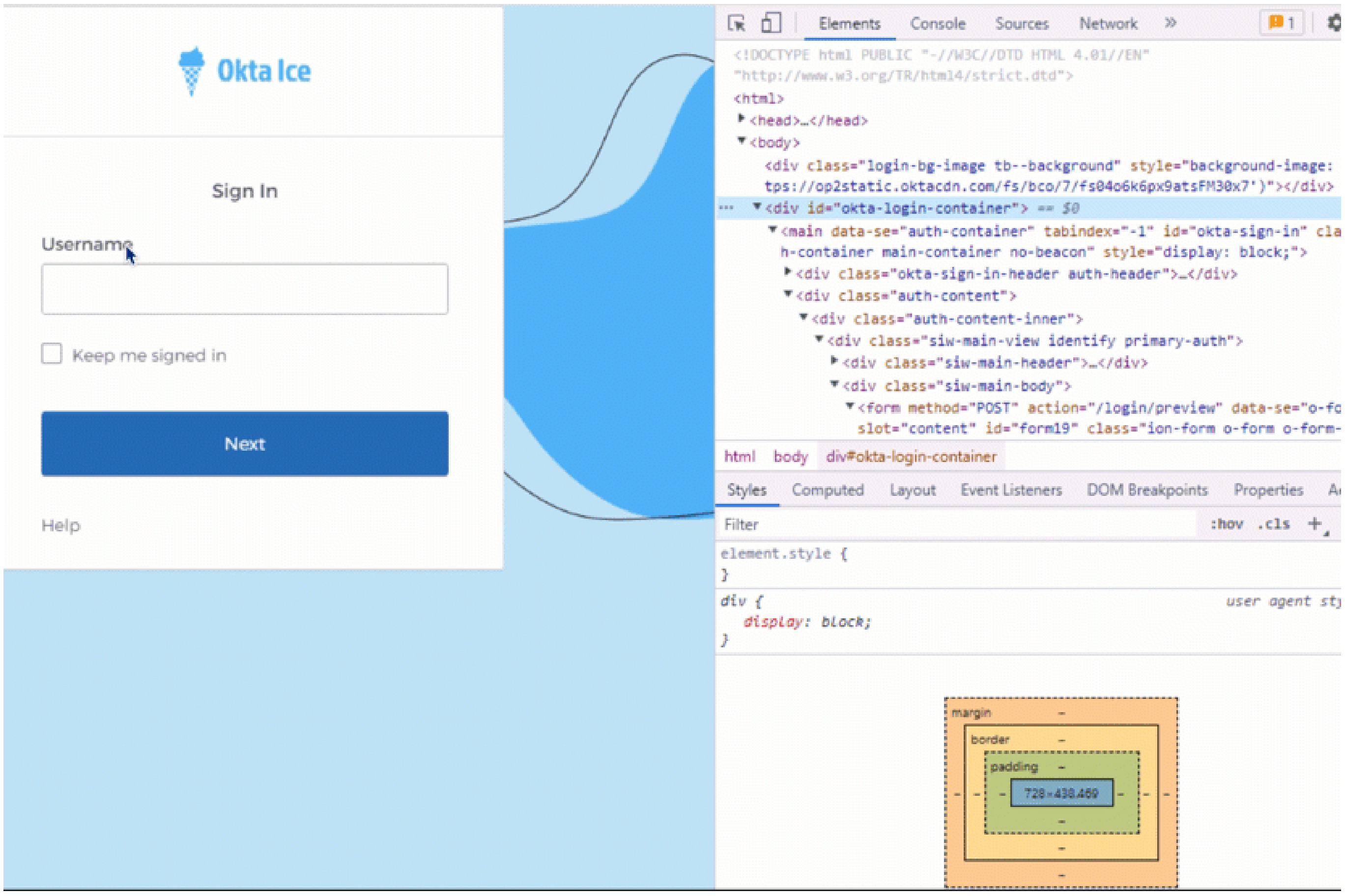


Alternatively, you can use the keyboard shortcut `SHIFT` + `CONTROL` + `C`

Inspect the Form Label Element

One of the elements we may want to customize, is the font color of the text in the SIW. Let's inspect the `Username` form label in the SIW by right-clicking on the element and selecting `Inspect`

Notice that this updates what we see in `Elements` and `Style` and we can see that the CSS selector that defines the text color for this element is `#okta-sign-in *`



```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<html>
 ▸<head>…</head>
 ▾<body>
    <div class="login-bg-image tb--background" style="background-image:
    tps://op2static.oktacdn.com/fs/bco/7/fs04o6k6px9atsFM30x7')"></div>
... ▾<div id="okta-login-container"> == $0
     ▾<main data-se="auth-container" tabindex="-1" id="okta-sign-in" cla
      h-container main-container no-beacon" style="display: block;">
       ▸<div class="okta-sign-in-header auth-header">…</div>
       ▾<div class="auth-content">
         ▾<div class="auth-content-inner">
           ▾<div class="siw-main-view identify primary-auth">
             ▸<div class="siw-main-header">…</div>
             ▾<div class="siw-main-body">
               ▾<form method="POST" action="/login/preview" data-se="o-fo
                slot="content" id="form19" class="ion-form o-form o-form-
```

```
element.style {
}
div {                                              user agent sty
   display: block;
}
```

Update the Font Color of General Text in the SIW

We now know that the text color of the general text elements in the SIW is defined by the `#okta-sign-in *` selector so now we can add some embedded CSS to the Sign-In Page code. We'll make changes here first before checking the changes in the embedded Sign-In Page Code Editor.

Notice that this lab step automatically points to a particular line in the HTML code. This is where you will be adding some embedded CSS to change the font color of the general text in the SIW to a blue hue that matches the Okta Ice logo.

Type in the following embedded CSS **or** use the `Insert Code` link to add it automatically:

```
<style>
  #okta-sign-in * {
    color: #0e2d5b;
  }
</style>
```

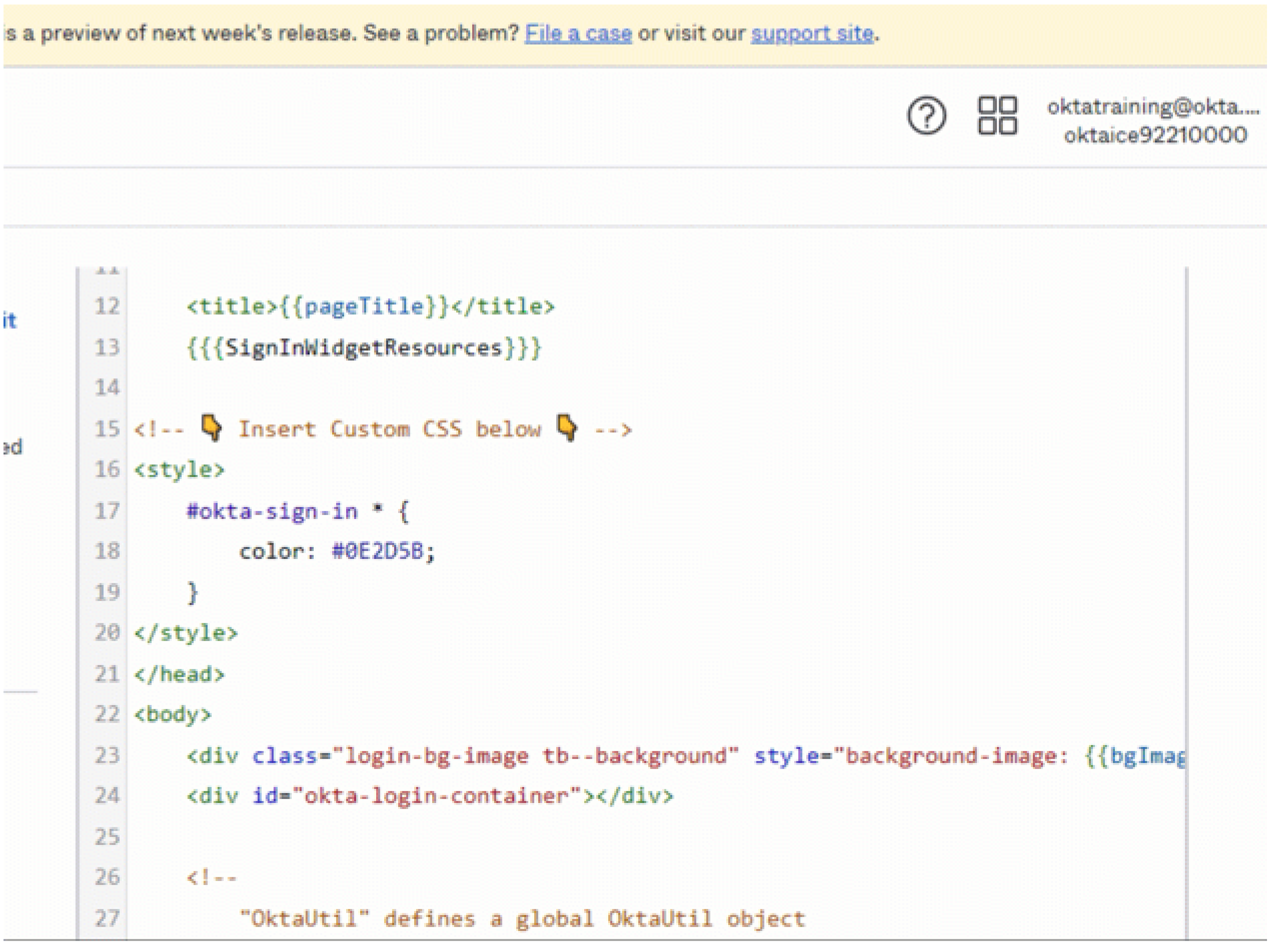
Finally, save your `HTML` file.

Preview Font Color Change in the Sign-In Page Code Editor

Notice that this step has automatically opened the the HTML file for you here in VSCode.

1. Click into this HTML file and press `CTRL`+`A` to select the entire contents.

2. Press `CTRL`+`C` to copy the code.
3. Go back to your Okta Sign-In Page Code Editor tab in Chrome.
4. Click inside of the Code Editor box and press `CTRL`+`A` to highlight the entire contents and press `BACKSPACE` to delete it.
5. Press `CTRL`+`V` to paste in your updated code.
6. Scroll up and verify that your embedded CSS now appears in the code.
7. Click the `Preview` button and observe the font color change.


s a preview of next week's release. See a problem? File a case or visit our support site.

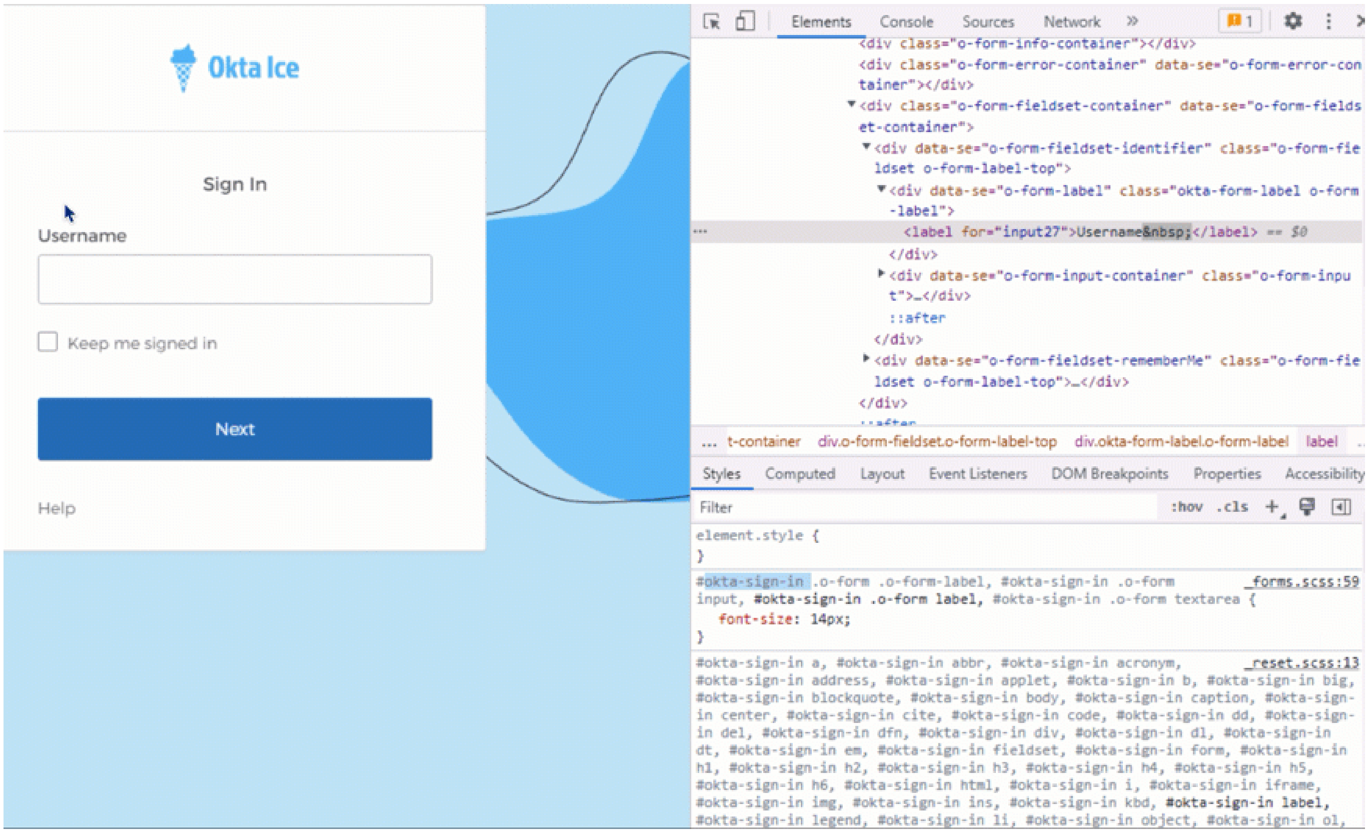
oktatraining@okta....
oktaice92210000

```
    <title>{{pageTitle}}</title>
    {{{SignInWidgetResources}}}

<!-- 👇 Insert Custom CSS below 👇 -->
<style>
    #okta-sign-in * {
        color: #0E2D5B;
    }
</style>
</head>
<body>
    <div class="login-bg-image tb--background" style="background-image: {{bgImag
    <div id="okta-login-container"></div>

    <!--
        "OktaUtil" defines a global OktaUtil object
```

Inspect the SIW Form Header

Notice that our font color change was not applied to the `Sign In` header on above the form in the SIW. Let's inspect the Sign-In Page code again. In your latest preview of the Sign-In Page, right click on on the `Sign-In` text and select `Inspect`

When we do this, we see that the font color for this element is specified in the selector `#okta-sign-in.auth-container h2`



```
<div class="o-form-info-container"></div>
<div class="o-form-error-container" data-se="o-form-error-con
tainer"></div>
▼<div class="o-form-fieldset-container" data-se="o-form-fields
et-container">
  ▼<div data-se="o-form-fieldset-identifier" class="o-form-fie
  ldset o-form-label-top">
    ▼<div data-se="o-form-label" class="okta-form-label o-form
    -label">
      <label for="input27">Username&nbsp;</label> == $0
    </div>
    ▶<div data-se="o-form-input-container" class="o-form-inpu
    t">…</div>
    ::after
  </div>
  ▶<div data-se="o-form-fieldset-rememberMe" class="o-form-fie
  ldset o-form-label-top">…</div>
</div>
```

... t-container div.o-form-fieldset.o-form-label-top div.okta-form-label.o-form-label label

```
element.style {
}
#okta-sign-in .o-form .o-form-label, #okta-sign-in .o-form        _forms.scss:59
input, #okta-sign-in .o-form label, #okta-sign-in .o-form textarea {
   font-size: 14px;
}
#okta-sign-in a, #okta-sign-in abbr, #okta-sign-in acronym,       _reset.scss:13
#okta-sign-in address, #okta-sign-in applet, #okta-sign-in b, #okta-sign-in big,
#okta-sign-in blockquote, #okta-sign-in body, #okta-sign-in caption, #okta-sign-
in center, #okta-sign-in cite, #okta-sign-in code, #okta-sign-in dd, #okta-sign-
in del, #okta-sign-in dfn, #okta-sign-in div, #okta-sign-in dl, #okta-sign-in
dt, #okta-sign-in em, #okta-sign-in fieldset, #okta-sign-in form, #okta-sign-in
h1, #okta-sign-in h2, #okta-sign-in h3, #okta-sign-in h4, #okta-sign-in h5,
#okta-sign-in h6, #okta-sign-in html, #okta-sign-in i, #okta-sign-in iframe,
#okta-sign-in img, #okta-sign-in ins, #okta-sign-in kbd, #okta-sign-in label,
#okta-sign-in legend, #okta-sign-in li, #okta-sign-in object, #okta-sign-in ol,
```

Update the SIW Form Header Font Color

We're going to add some more embedded CSS to change the color of the form header to match the Okta Ice dark blue of the other text in the SIW.

Notice that this step highlights the line where you should be adding this code, which is just under the CSS we added in the previous step.

As always, we recommend manually typing in the following code segment, but you may alternatively use the `Insert Code` link.

```
#okta-sign-in.auth-container h2 {
  color: #0e2d5b;
}
```

Finally, Click here to `Save` your `HTML` file.

Preview Font Color Change of Form Header

Notice that this step has automatically opened the the HTML file for you here in VSCode.

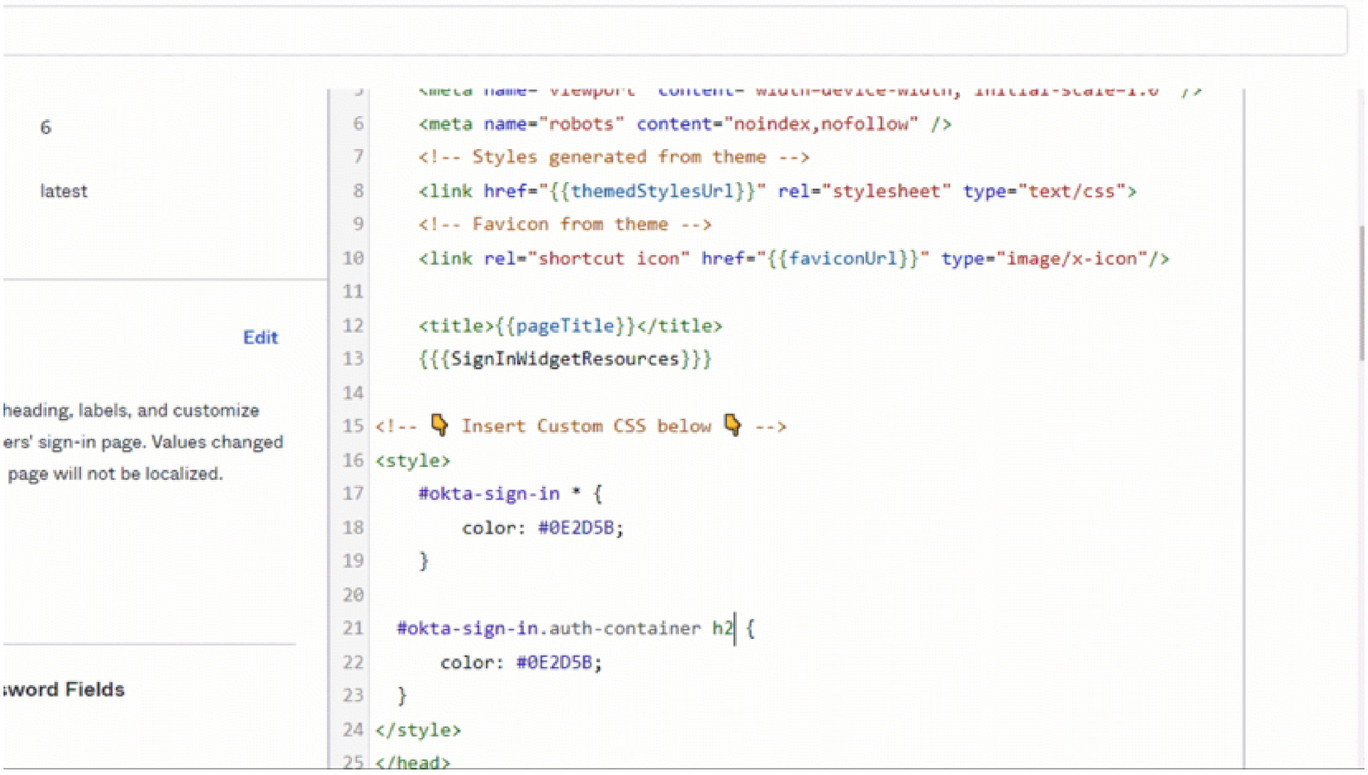
1. Click into this HTML file and press `CTRL`+`A` to select the entire contents.
2. Press `CTRL`+`C` to copy the code.
3. Go back to your Okta Sign-In Page Code Editor tab in Chrome.

4. Click inside of the Code Editor box and press CTRL+A to highlight the entire contents and press BACKSPACE to delete it.

5. Press CTRL+V to paste in your updated code.

6. Scroll up and verify that your updated embedded CSS now appears in the code.

7. Click the `Preview` button and observe the font color change of the header.

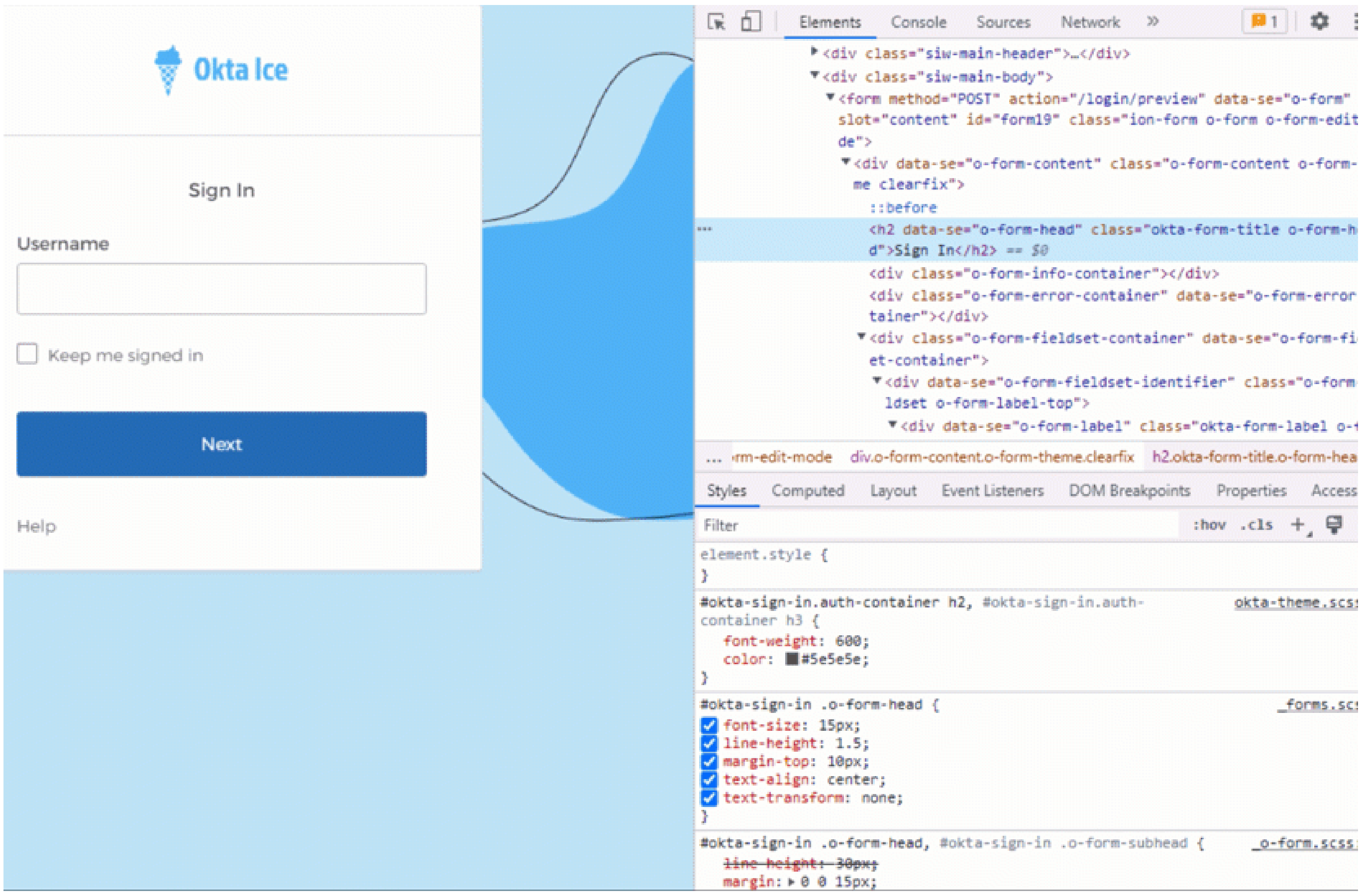


6
latest
Edit
heading, labels, and customize
ers' sign-in page. Values changed
page will not be localized.
word Fields

```
    <meta name="robots" content="noindex,nofollow" />
    <!-- Styles generated from theme -->
    <link href="{{themedStylesUrl}}" rel="stylesheet" type="text/css">
    <!-- Favicon from theme -->
    <link rel="shortcut icon" href="{{faviconUrl}}" type="image/x-icon"/>

    <title>{{pageTitle}}</title>
    {{{SignInWidgetResources}}}

<!-- 👇 Insert Custom CSS below 👇 -->
<style>
    #okta-sign-in * {
        color: #0E2D5B;
    }

  #okta-sign-in.auth-container h2 {
      color: #0E2D5B;
  }
</style>
</head>
```

Inspect the SIW Header

Next, let's find out what selector we can access to change the background color of the SIW header area. In your latest preview of the Sign-In Page, right click into the header area of the SIW. Here we see the selector `#okta-sign-in.auth-container .okta-sign-in-header`



```
▶<div class="siw-main-header">…</div>
▼<div class="siw-main-body">
  ▼<form method="POST" action="/login/preview" data-se="o-form"
   slot="content" id="form19" class="ion-form o-form o-form-edit
   de">
    ▼<div data-se="o-form-content" class="o-form-content o-form-
     me clearfix">
       ::before
       <h2 data-se="o-form-head" class="okta-form-title o-form-h
       d">Sign In</h2> == $0
       <div class="o-form-info-container"></div>
       <div class="o-form-error-container" data-se="o-form-error
       tainer"></div>
      ▼<div class="o-form-fieldset-container" data-se="o-form-fi
       et-container">
        ▼<div data-se="o-form-fieldset-identifier" class="o-form
         ldset o-form-label-top">
          ▼<div data-se="o-form-label" class="okta-form-label o-
... rm-edit-mode  div.o-form-content.o-form-theme.clearfix  h2.okta-form-title.o-form-hea

element.style {
}
#okta-sign-in.auth-container h2, #okta-sign-in.auth-          okta-theme.scs
container h3 {
   font-weight: 600;
   color: ■#5e5e5e;
}
#okta-sign-in .o-form-head {                                   _forms.sc
✓ font-size: 15px;
✓ line-height: 1.5;
✓ margin-top: 10px;
✓ text-align: center;
✓ text-transform: none;
}
#okta-sign-in .o-form-head, #okta-sign-in .o-form-subhead {    _o-form.scss
   ~~line-height: 30px;~~
   margin: ▶ 0 0 15px;
```

Update the SIW Header Background Color

Notice that this step automatically opens our HTML file and selects the line where we can add to our code. Let's add some custom CSS that will change the background color of the SIW header to a dark blue. We'll use the `#okta-sign-in.auth-container .okta-sign-in-header` selector to do this.

As always, we recommend manually typing in the following code segment, but you may alternatively use the `Insert Code` link.

```
#okta-sign-in.auth-container .okta-sign-in-header {
  background: #0e2d5b;
}
```

Finally, Click here to `Save` your `HTML` file.

Preview the SIW Header Background Color Change

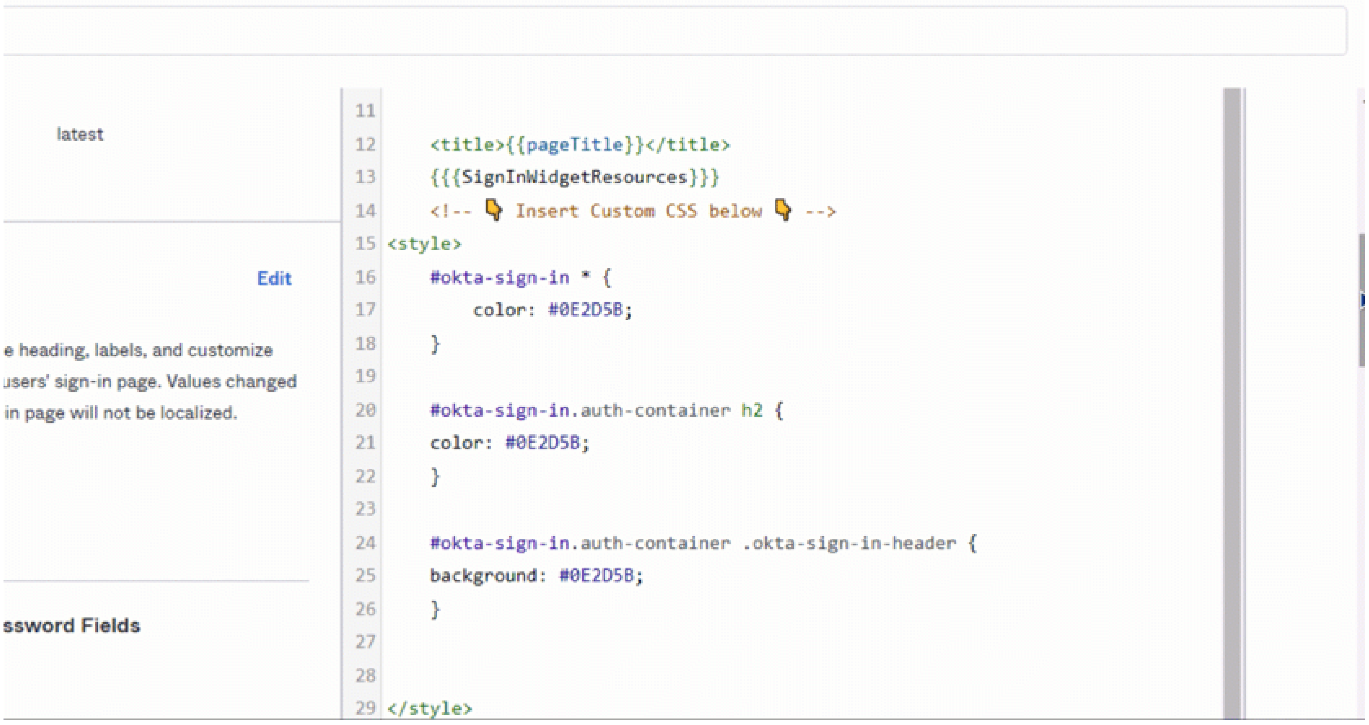
Notice that this step has automatically opened the the HTML file for you here in VSCode.

1. Click into this HTML file and press `CTRL`+`A` to select the entire contents.
2. Press `CTRL`+`C` to copy the code.
3. Go back to your Okta Sign-In Page Code Editor tab in Chrome.
4. Click inside of the Code Editor box and press `CTRL`+`A` to highlight the entire contents and press `BACKSPACE` to delete it.

5. Press CTRL+V to paste in your updated code.

6. Scroll up and verify that your updated embedded CSS now appears in the code.

7. Click the `Preview` button and observe that the background color of the SIW header has changed.

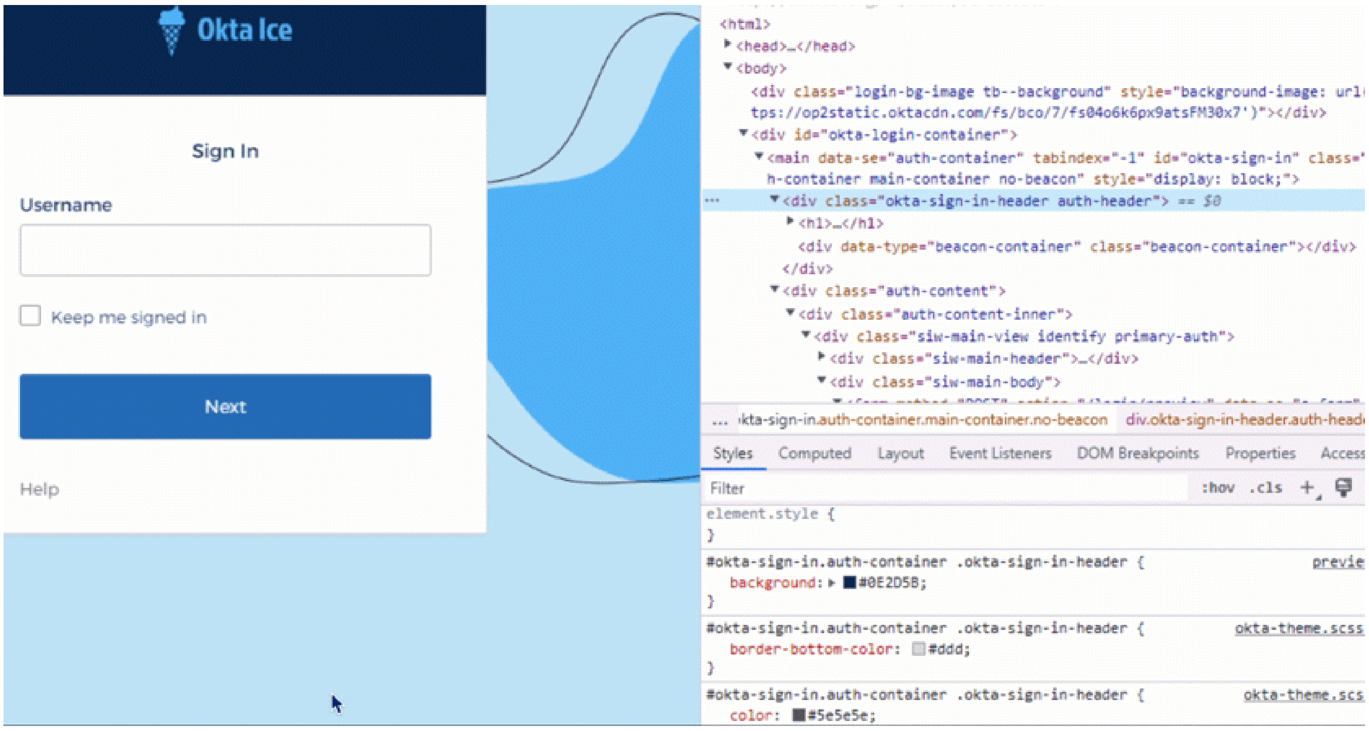

latest
Edit
e heading, labels, and customize
users' sign-in page. Values changed
in page will not be localized.
ssword Fields

```
    <title>{{pageTitle}}</title>
    {{{SignInWidgetResources}}}
    <!-- 👇 Insert Custom CSS below 👇 -->
<style>
    #okta-sign-in * {
        color: #0E2D5B;
    }

    #okta-sign-in.auth-container h2 {
    color: #0E2D5B;
    }

    #okta-sign-in.auth-container .okta-sign-in-header {
    background: #0E2D5B;
    }


</style>
```

Inspect Link Element

Let's change the color of the `Help` so it stands out a bit more and matches our branding. In your latest preview of the Sign-In Page, right click on the `Help` link at the bottom of the SIW and select `Inspect`

Here we find the relevant selectors: `#okta-sign-in.auth-container .link:link` and `#okta-sign-in.auth-container .link:visited`



```
<html>
 ▶<head>…</head>
 ▼<body>
    <div class="login-bg-image tb--background" style="background-image: url(
    tps://op2static.oktacdn.com/fs/bco/7/fs04o6k6px9atsFM30x7')"></div>
  ▼<div id="okta-login-container">
    ▼<main data-se="auth-container" tabindex="-1" id="okta-sign-in" class="
     h-container main-container no-beacon" style="display: block;">
···   ▼<div class="okta-sign-in-header auth-header"> == $0
        ▶<h1>…</h1>
         <div data-type="beacon-container" class="beacon-container"></div>
       </div>
      ▼<div class="auth-content">
        ▼<div class="auth-content-inner">
          ▼<div class="siw-main-view identify primary-auth">
            ▶<div class="siw-main-header">…</div>
            ▼<div class="siw-main-body">
```

```
... kta-sign-in.auth-container.main-container.no-beacon   div.okta-sign-in-header.auth-head

element.style {
}
#okta-sign-in.auth-container .okta-sign-in-header {                    previe
   background: ▶ ■#0E2D5B;
}
#okta-sign-in.auth-container .okta-sign-in-header {              okta-theme.scss
   border-bottom-color: □#ddd;
}
#okta-sign-in.auth-container .okta-sign-in-header {               okta-theme.scs
   color: ■#5e5e5e;
```

Update the SIW Link Color

We're going to add to our embedded CSS at the indicated line of the HTML file. Not only are we going to define the link color, but we'll define the color for a visited link as well. We'll define both as a light blue.

Notice that this step highlights the line where you should be adding this code, which is just under the CSS we added in the previous step.

As always, we recommend manually typing in the following code segment, but you may alternatively use the `Insert Code` link.

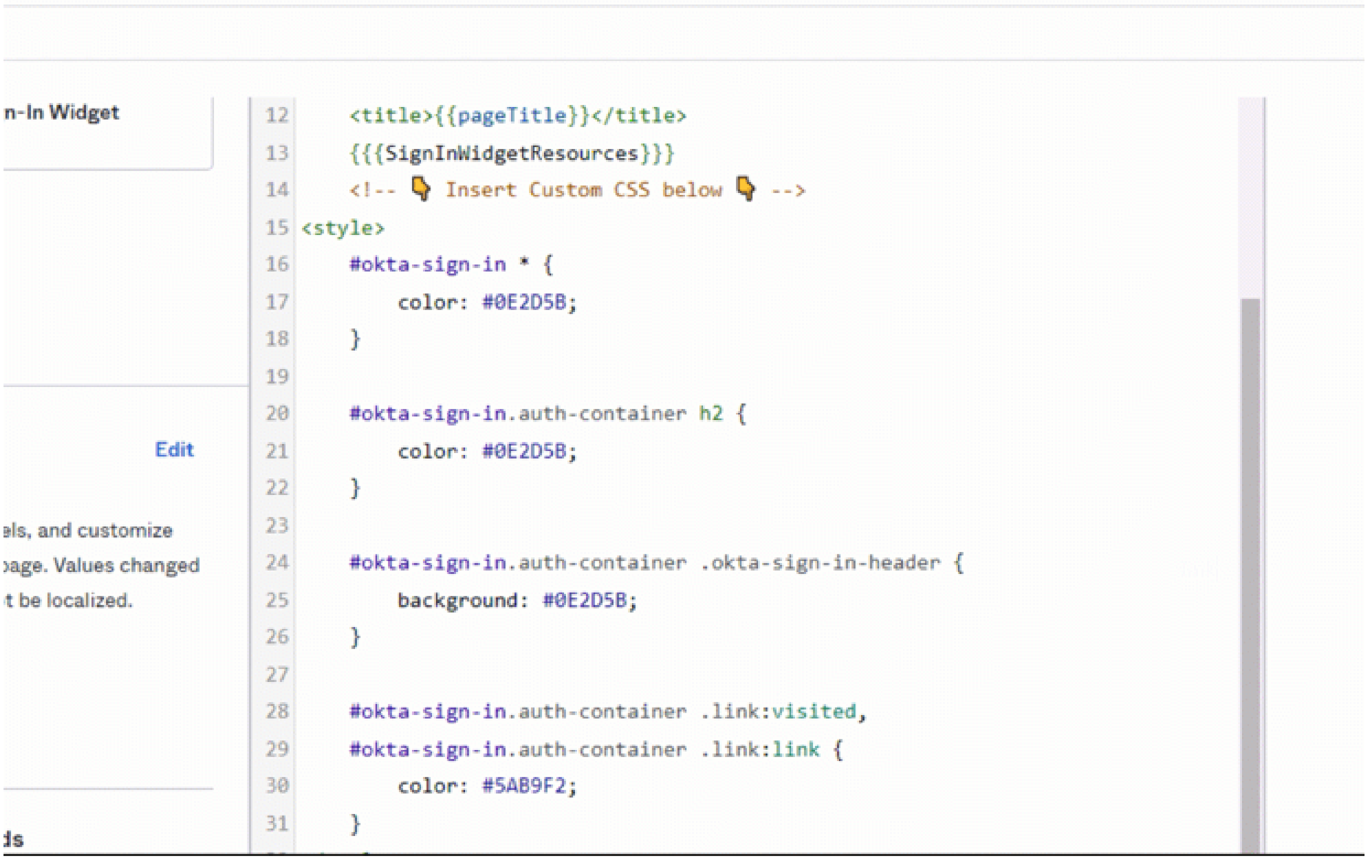
```
#okta-sign-in.auth-container .link:visited,
#okta-sign-in.auth-container .link:link {
  color: #5ab9f2;
}
```

Finally, Click here to `Save` your `HTML` file.

Preview the Link Color Change

Notice that this step has automatically opened the the HTML file for you here in VSCode.

1. Click into this HTML file and press `CTRL`+`A` to select the entire contents.
2. Press `CTRL`+`C` to copy the code.
3. Go back to your Okta Sign-In Page Code Editor tab in Chrome.
4. Click inside of the Code Editor box and press `CTRL`+`A` to highlight the entire contents and press `BACKSPACE` to delete it.
5. Press `CTRL`+`V` to paste in your updated code.
6. Scroll up and verify that your updated embedded CSS now appears in the code.
7. Click the `Preview` button and observe that the color of the `Help` link in the SIW has changed to white.



```
    <title>{{pageTitle}}</title>
    {{{SignInWidgetResources}}}
    <!-- 👇 Insert Custom CSS below 👇 -->
<style>
    #okta-sign-in * {
        color: #0E2D5B;
    }

    #okta-sign-in.auth-container h2 {
        color: #0E2D5B;
    }

    #okta-sign-in.auth-container .okta-sign-in-header {
        background: #0E2D5B;
    }

    #okta-sign-in.auth-container .link:visited,
    #okta-sign-in.auth-container .link:link {
        color: #5AB9F2;
    }
```

Save and Publish Changes

If there's time remaining, feel free to continue tinkering and finding out what else you can customize here. When you're done, click the `Save and Publish` button in the Sign-In Page Code Editor.

✅ Checkpoint

At this point, you've investigated additional customizations you can make to the Sign-In Page by using the embedded Sign-In Page Code Editor.

Lab 2.4: Configure the Redirect Apps to Use the Custom Domain

🎯 **Objective:** Configure the redirect applications we set up previously to use the custom domain when redirecting users to the Okta-hosted Sign In Widget.

🎬 **Scenario** When users authenticate into an Okta Ice app that uses the Okta-hosted Sign In Widget, they should be redirected to the widget through the custom URL rather than the Okta URL.

⏱ **Duration:** 10 minutes

⚠ **Prerequisite:** Lab 2-1

Configure the Authorization Server Issuer URI

We're going to configure the default Okta Authorization Server to to use either our Okta org URL or our custom URL as an Issuer URI -- depending on what our client application expects.

1. Ensure you're logged into the Okta Admin Dashboard as `oktatraining`
2. In the Admin menu, navigate to `Security` > `API`

3. Within the **Authorization Servers** table, local the `default` entry and click the blue pencil icon.
4. On the following screen, click `Edit`
5. In the `Issuer` section, select `Dynamic` from the drop down menu.
6. Click `Save`
7. **Logout** of Okta.

Copy Redirect Applications

Copy the `redirect` directory to this workspace:

1. Open a new terminal in VSCode. Notice it open the terminal in the root directory of the workspace.

2. Issue the command:

```
mkdir redirect; cp ../01-introducing-okta/redirect/* redirect
```

Edit the Customer Rewards App

1. Notice that `rewards.html` has been opened here for you and that the value of `baseOktaURL` is highlighted.

2. Change the value of `baseOktaURL` so that it corresponds to your newly configured custom URL (e.g., https://oktaice########.coffee-ice.com)

3. Save `rewards.html`

Edit the Franchisee CRM App

1. Notice that `crm.html` has been opened here for you and that the value of `baseOktaURL` is highlighted.

2. Change the value of `baseOktaURL` so that it corresponds to your newly configured custom URL (e.g., https://oktaice########.coffee-ice.com)

3. Save `crm.html`

Test Signing in with a Custom Domain

1. Click [here to launch the web server](command:codetour.sendTextToTerminal?["python -m http.server 8080"]).

2. Open the Chrome browser and navigate to http://localhost:8080

3. Click on `Customer Rewards (Redirect)`

4. Notice that you are now redirected to the Okta-hosted Sign In Widget via your custom URL!

✅ Checkpoint

At this point, you've set up the Rewards and CRM applications to use the custom domain when redirecting to the Okta-hosted Sign In Widget for authentication.

🎉 End of Module 2 Labs

You may close this workspace project, ensuring all changes were saved.

Module 3: Exploring Authentication Protocol Flows

Lab 3.1 Project Setup

🎯 Objective: In order to have working copies of our previously configured redirect apps in our portal, we'll need to copy them to this workspace. This will be necessary to see how SSO works as we configure additional applications in the portal.

⏱ Duration: 1 min

⚠ Prerequisite: Lab 1.4

Copy Redirect Applications

Copy the `redirect` directory to this workspace:

1. Open a new terminal in VSCode. Notice it open the terminal in the root directory of the workspace.
2. Issue the command:

```
mkdir redirect; cp ../01-introducing-okta/redirect/* redirect
```

✅ Checkpoint

At this point, you have copies of the apps you configured in `Lab 1.4` in the current workspace. This will be necessary to see how SSO works as we configure additional applications in the portal.

Lab 3.2: Deploy Authentication with the Embedded Sign In Widget

🎯 Objective: Create an an Okta application integration and deploy authentication using the embedded Sign-In Widget.

⏱ Duration: 15 min

A Note on Cross-Origin Resource Sharing (CORS)

Recall that we have already enabled CORS for the Okta Ice Portal in Module 1. This is necessary when deploying authentication using the Embedded Widget model. In Okta, **CORS** allows JavaScript hosted on your website to make a request using an `XMLHttpRequest` to the Okta API. Every website origin must be explicitly permitted as a **Trusted Origin** in your Okta org.

Enable Interaction Code Grant Type

In order to use the Embedded Widget for Sign-In, the **Interaction Code** grant type must be turned on in our Authorization Server. We will do that in this step.

1. Ensure you are logged in to the Admin Panel as `oktatraining`
2. In the Admin menu, navigate to `Security` > `API`
3. In the **Authorization Servers** table, click the blue pencil icon in the `default` row.
4. Click on the `Access Policies` tab.
5. In the `Default Policy Rule` row, click the blue pencil icon.
6. Ensure that the `Interaction Code` option is selected.
7. Click `Update Rule`

Create an Application Integration for the Customer Polling App

The Polling app is an existing application in this project. We're going to set up an integration in Okta so our customers can access this application.

1. In the Admin menu, navigate to `Applications` > `Applications`
2. Click `Create App Integration`
3. Select the `OIDC - OpenID Connect` radio button.
4. Select the `Single-Page Application` radio button.
5. Click `Next`
6. Name this application `Customer Polling`
7. Under **Grant Type** select both `Authorization Code` and `Interaction Code`.
8. In the **Sign-in redirect URIs** field, enter `http://localhost:8080/embedded/polling.html`
9. Under **Assignments** click the radio button option for `Limit access to select groups`
10. Type in and select `Customers`
11. Click `Save`

Configure the Polling Application `appClientID`

1. Copy the `Client ID` that was displayed after you saved your integration.
2. Paste the `Client ID` into the `appClientID` variable.

Configure the Polling Application `baseOktaURL`

1. Update the value of `baseOktaURL` to match your assigned Okta org URL. Recall that you can access this URL from the `Link` section of the **Credentials** panel in your VM.
2. `Save` your `polling.html` file.

Import the Okta Sign-In Widget Library

We're going to import Okta's Sign In Widget JavaScript and CSS library from a CDN (Content Delivery Network) by adding the followng two lines to the `<head>` section of the Customer Polling app:

```
<script
  src="https://global.oktacdn.com/okta-signin-widget/5.8.3/js/okta-sign-
in.min.js"
  type="text/javascript"
></script>
<link
  href="https://global.oktacdn.com/okta-signin-widget/5.8.3/css/okta-sign-
in.min.css"
  type="text/css"
  rel="stylesheet"
/>
```

The libraries imported here are the same ones used by the Okta-hosted Sign-In Widget that we saw in Module 2.

Embedding the Sign In Widget directly to a page in your application allows for full customization.

✅ Checkpoint

At this point, you have configured an application using the Embedded Sign-In Widget model of deploying authentication.

Lab 3.3 Test Web SSO

🎯 Objective: Now that we have deployed authentication on two applications assigned to the Customers group, we can test out web SSO.

⏱ Duration: 10 min

Log Out of Okta

1. If you are still logged in on your Okta org, log out.

Start the Web Server

1. Open a new terminal in VSCode
2. Notice that the terminal automatically opens to the project directory.
3. Issue the command `python -m http.server 8080`

Test the Polling App

1. In Chrome, visit http://localhost:8080
2. Click the `Polling App (Embedded Widget)` link.

3. Notice that you are not redirected to Okta for authentication. Instead, the Sign-In Widget loads on the page of your website.

4. Log in as `soraya.esfeh@oktaice.com`/`Tra!nme4321`

You should now see the ID Token (which is formatted as a JSON web token or **JWT**) value and its claims, including:

- `sub` (subject of the JWT): Soraya's Okta User ID
- `name`: Soraya's name
- `email`: Soraya's email address
- `ver`: Okta API version
- `iss` (issuer of the JWT): The URL of your Okta Authorization server
- `aud` (audience): Your app's Client ID
- `iat` (issued at time): Time the JWT was issued. Expressed in Unix time.
- `exp` (expiration time): Time the JWT expires. Expressed in Unix time.
- `jti` (JWT ID): Unique identifier used to prevent the JWT from being replayed

Test the Rewards App

1. Click on `Return to Portal`

2. Click the `Rewards App (Redirect)` link.

3. Notice that you are not prompted to authenticate again as you have an existing session.

You should see the ID Token value and its claims. Much of this content is the same as you saw on the Polling App. However, notice that the `jti` (JWT ID) and the `aud` (audience) are different. You should now see that the `aud` is this Client ID of the Rewards app.

✅ Checkpoint

At this point, you have seen how web SSO works between two applications accessible to members of the Customer group.

🎉 End of Module 3 Labs

You may close this workspace project, ensuring all changes were saved.

Module 4: Exploring the Okta Data Model

Lab 4.1: Get an API token and set up the Postman Environment

🎯 Objective: Create an Okta service account for administrative tasks and associations with API tokens. Set up the Postman environment to make API requests to Okta.

⏱️ Duration: 15 min

Create an Okta Service Account

1. Ensure you are signed in as your Super Admin account, `oktatraining`.

2. In the Admin Dashboard, select `Directory` > `People`.

3. Click the `Add Person` button.

4. Enter the following details:

Field	Value
First name	`Okta`
Last name	`Service`
Username	`okta.service@oktaice.com`
Primary email	`okta.service@oktaice.com`
Activation	`Activate now`
I will set the password	`CHECKED`
Enter password	`Tra!nme4321`
User must change password on first login	`UNCHECKED`

Last, click the `Save` button.

Assign Administrator Permissions

1. Navigate to `Security` > `Administrators`

2. Click `Add Administrator`

3. In the `Admin` field, type and select the user `Okta Service`

4. In the `Role` field, select `Super Administrator`

5. Click `Save Changes`

Sign In to Service Account

1. Log out of the **oktatraining** account.

2. Log back in with the credentials: `okta.service@oktaice.com` / `Tra!nme4321`

3. Click on the `Admin` button.

Create an API Token

1. Ensure you are still logged in to the Admin Dashboard as **Okta Service**

2. Navigate to `Security` > `API`.

3. On the `Tokens` tab, click `Create Token`.

4. Enter `Postman` as the token name and click `Create Token`.

5. A token value is generated and displayed in a popup modal. Copy this value by clicking the clipboard button.
6. Paste the value in the text file here.
7. Click here to save the text file.

Import the Postman Environment

1. In the VM, launch the Postman app (an orange circular icon).
2. Click the `Import` button on the top left of the application.
3. Click `Link`.
4. Paste the URL `https://developer.okta.com/docs/api/postman/example.oktapreview.com.environment`
5. Click `Continue` and then `Import` to confirm.

Rename the Postman Environment

1. At the top of the Postman window, there's a drop down that says `No Environment` Click this drop down and select the environment you just imported: `${yourOktaDomain}`
2. Click the eyeball icon next to the environment name.
3. Click `Edit`.
4. Rename the environment `oktaice######.oktapreview.com`, replacing `######` with your unique Okta org number.

Configure the Postman Environment Variables

In the next steps, you will define certain environment variables so that you can make calls to the Core Okta API in the next lab. When defining varialbles, update the CURRENT VALUE column.

Configure the `url` Environment Variable

Update the `CURRENT VALUE` of `url` to your Okta org's url, e.g. `http://oktaice######.oktapreview.com`

Configure the `apikey` Environment Variable

Copy the API Token from the text file opened above and paste it into the `CURRENT VALUE` column of the `apikey` variable

Configure the `email-suffix` Environment Variable

Enter `okta-ice.com` in in the `CURRENT VALUE` column for the `email-suffix` variable.

Persist and Update all Environment Variables

1. Click `Persist All`
2. Scroll down and click `Update`
3. You may now close the environment variable tab.

✅ Checkpoint

At this point, Postman is now configured to make API calls to Okta. In the next lab, you will import the Okta Users API Collection. A collection is a group of saved API requests.

Lab 4.2: Create an Okta User via the Users API

🎯 Objective: Import the Okta Users API collection into Postman and create an Okta user via API request.

⏱ Duration: 15 min

⚠ Prerequisite: Lab 4.1

Import the Okta Users API Collection Into Postman

1. Within the Chrome browser in your VM, visit https://developer.okta.com/docs/reference/api/users/
2. Click `Run in Postman`
3. Click `Postman for Windows`
4. If prompted, click `Open Postman`
5. After a moment, you should see a toast notification saying that the **Users (Okta API)** collection was successfully imported.
6. Click on the `Collections` tab and you should now see the **Users (Okta API)** collection with 42 requests.

You should now see the **Users (Okta API)** collection under **Collections**

Open the `Create Activated User with Password` Request

1. Click on the **Users (Okta API)** collection to expand it.
2. Click on the **Create User** subfolder to expand it.
3. Click on the **Create Activated User with Password** request to open it in a tab.

Examine the Request Params

Let's take a moment to examine this request and its parameters. Ensure you are on the `params` tab.

1. First hover over the `{{url}}` environment variable at the top of the request window. Verify that you see your Okta URL.
2. Next, look at the URI (relative to our base Okta URL) of the request endpoint: `/api/v1/users` This is the relative URI for all requests in this collection, though some requests may require additional URI

parameters or query strings.

3. Observe that this request has a boolean `activate` URI parameter. When `activate` is set to true, this request will create a user with the status of `ACTIVE`
4. Finally, notice that this request uses the HTTP `POST` method. This method is used when creating a new web resource, in this case, an Okta User.

Examine Request Header

1. Click on the `Headers` tab.
2. Notice that your API Token is being passed via the `Authorization` header via the `{{apikey}}` environment variable.
3. The `Accept` header communicates what type of data our client (Postman) can accept back in a response from Okta. Here, we accept `application/json` data.
4. The `Content-Type` header communicates the type of data we are sending in our request to Okta. This is also `application/json` data.

Examine the Request Body

1. Click on the `Body` tab.
2. Here you will see `JSON` formatted data that represents a user. In this case, the `User` object is comprised of a `Profile` object and a `Credentials` object.
3. The `Profile` object consists of the four User profile attributes that are required by default on Okta: `firstName`, `lastName`, `email`, and `login` (username).
4. The `Credentials` object consists of a `password`.

Update the Request Body

The `Profile` object in this request only includes the four profile attributes that are required by default on Okta. We can include additional optional attributes as well!

We're going to update the `Profile` object's default attributes and add a `nickName` and `userType` attribute. Recall that we created a **Group Rule** in `Lab 1.2` that will automatically add a user with `userType` **customer** to our **Customers** group.

The `Credentials` object will be left unchanged.

Your updated request body will be:

```
{
  "profile": {
    "firstName": "Samus",
    "lastName": "Aran",
    "email": "samus.aran@{{email-suffix}}",
    "login": "samus.aran@{{email-suffix}}",
    "userType": "customer",
    "nickName": "Sammy"
  },
  "credentials": {
    "password": { "value": "{{password}}" }
  }
}
```

(You can ignore the **Insert Code** link above and simply copy and paste the JSON into the request body in Postman)

Update the `password` Environment Variable

1. Click the eyeball icon to the right of your Postman environment drop down.
2. Click `Edit`
3. Scroll down to the `password` variable and enter `Tra!nme4321` in the `CURRENT VALUE` column.
4. Click `Persist All` and then scroll down and click `Update`
5. Close the Environment variable tab and return to the **Create Activated User with Password** request tab.

Send the Request

1. Click `Send`
2. Verify that you get a `200 OK` response (visible at the bottom of Postman). If not, go back and ensure you entered the JSON in your request body correctly and that all of your environment variables have been set correctly.

Examine the Response

1. Click on the `Body` tab in the Response area (bottom half of the Postman window).
2. Notice that the response we get back from Okta is `JSON`
3. The first entry is the User's unique `id`.
4. The user has a status of `ACTIVE`
5. If you scroll down, you will find the `Profile` object you specified.
6. You will also see a `Credentials` object, but the password is not echoed back.
7. Finally, the `_links` section exposes additional Okta Users API endpoints relevant to this user's lifecycle. Notice that all of these links have the user's `id` as a URI parameter in them.

Set the `userId` Environment Variable

You just saw that some API endpoints require a user's `id` as a URI parameter. Next, we're going to call an endpoint that will list all the Groups a user is a member of.

We'll store the `id` in the `userId` environment variable:

1. Scroll back to the top of the response body.
2. Highlight the value to the right of the first `id` entry (excluding the quotes).
3. Right click on the highlighted value.
4. Select `Set: {environment name}` and then `userId`

Get Groups for User

1. In the `Users (Okta API)` collection, click on the `Get Groups for User` request to open it.
2. Notice that this endpoint uses the `{{userId}}` environment variable, which will pass in our user's `id` as a URI parameter.
3. Notice that the HTTP method used for this request is `GET`. We're requesting information about this user.
4. Click `Send` to send the request. You should get a `200 OK` response. If not, ensure you have correctly set the `userId` environment variable.
5. Examine the reponse body. The first entry in the JSON data should be the **Everyone** group.
6. Scroll down to locate the second entry in the JSON data, which should be the **Customers** group.

✅ Checkpoint

At this point, you have created an activated user with a password using the Users API. This has allowed you to see how a User is represented in Okta.

Lab 4.3: Update an Okta User Profile Via the Users API

🎯 Objective: Update an existing Okta User using a request from the Okta Users API collection in Postman.

⏱️ Duration: 15 min

⚠️ Prerequisite: Lab 4.2

Access Okta's Users API Documentation

Although the `Users (Okta API)` collection does not have a request to perform updates on a User's profile, the endpoint is documented here: https://developer.okta.com/docs/reference/api/users/#update-user

We'll use this documentation to craft our request in Postman.

From the documentation, we see that the endpoint we need to issue our request to is `/api/v1/users/{{userId}}` and that we should use the `POST` method for a partial update. A partial update means we only provide the data for the profile attributes we want to update.

Alternatively, the `PUT` method can be used for a complete update. This would mean that any attributes that are not specified in the request body will be deleted from the user's profile.

For our purposes, we will use the `POST` method for a partial update.

Create the Request

1. In Postman, click the big orange `+ New` button.
2. In the window that pops up, click `HTTP REQUEST`
3. Enter `Update User` as the **Request name**
4. Select `Users (Okta API)` in the **Select a collection or folder to save to** section.
5. Click `Save to Users (Okta API)`
6. In the new request tab that opens, click the drop down menu and change the HTTP method from `GET` to `POST`
7. For the request URL, enter `{{url}}/api/v1/users/{{userId}}`
8. Click `Save`

Set the Authorization Header

1. Click on `Headers`
2. In the next blank entry, enter `Authorization` in the `Key` column
3. Enter `SSWS {{apikey}}` in the `Value` column
4. Click `Save`

Set the Request Body to JSON

1. Click on the `Body` tab of the request.
2. Click the `raw` radio button
3. Change the drop down value from `Text` to `JSON`
4. Click `Save`

Create the Request Body JSON Payload

In order to update the User, we'll need to craft a JSON payload that specifies what profile attributes need to be updated.

In this case, we can imagine that the user's `lastName` needs to be updated. Since we are performing a partial update, we only need to specify this attribute in the Profile object of our JSON payload:

```
{
  "profile": {
    "lastName": "Nara"
  }
}
```

Ignore the Insert code link above

Click `Save` and then `Send` to issue the request.

Examine the Response Body

1. You should get a `200 OK` response. If not, ensure you have followed all the prerequisite steps correctly, including setting the `userId` environment variable in `Lab 4.2`
2. Locate the Profile object within the JSON in the Response body.
3. Notice that the `lastName` is now updated, but all other profile attributes remain the same.

✅ Checkpoint

You now have an understanding of how a partial update of a user's profile is performed by the Users API, which is used by Okta's Management SDKs.

🎉 End of Module 4 Labs

You may close this workspace project, ensuring all changes were saved.

Module 5: Implementing Self-Service Registration

Lab 5.1 Modify the Default User Profile Requirements.

🎯 **Objective** Update the required attributes in the Default User Profile to minimize sign up friction.

🎬 **Scenario** Okta Ice wants to minimize customer self-registration by only requiring a username and password. Additionally, Okta Ice does not want to enforce the default email-formatted username.

⏱️ **Duration** 15 minutes

Access the User Profile Editor

1. Ensure you are logged in to the Admin dashboard as `oktatraining`
2. Click on `Directory` > `User` from the Admin menu
3. Click on `Profile Editor`
4. Click `User (default)` under the **Profile** column.

Update Which Attributes Are Required

Recall that Okta requires four User Profile attributes by default: `login` (Username), `firstName`, `lastName`, and `email`

Let's change this!

1. In the **Attributes** section, click on the `i` icon that is in the **First name** `firstName` row.
2. Uncheck the `Attribute required` box.
3. Click the `Save Attribute` button.
4. Click on the `i` icon in the **Last name** `lastName` row.
5. Uncheck the `Attribute required` box.
6. Click the `Save Attribute` button.

Update Username Requirements

By default, Okta requires usernames to be in the format of an email. Let's disable that requirement.

1. Click on the `i` icon in the **Username** `login` row.
2. Next to **Format restrictions**, select `None` from the drop down menu.
3. Do not save the changes just yet as we still have more to do in this section.

Update Permissions on Attributes

In order to allow users to set their own username and email address during self-service registration, they will need write access to these attributes.

1. Still in the `Username` editing window, find the to **User permission** section.
2. Click the `Read-Write` radio button.
3. Click the `Save Attribute` button.
4. Click the `i` icon in the **Primary email** `email` row.
5. In the **User permission** section, click the `Read-Write` radio button.
6. Click the `Save Attribute` button.

✅ Checkpoint

At this point, you have updated the default User Profile and attribute permissions. Now you are ready to enable self-service registration for Okta Ice's customer applications!

Lab 5.2 Enable Self-Service Registration

🎯 **Objective** Configure Self-Service Registration for customer applications.

🎬 **Scenario** Okta Ice would like to allow customers to self-register to access customer applications.

⏱ **Duration** 15 minutes

Create Profile Enrollment Policy

1. Navigate to `Security` > `Profile Enrollment`
2. Click `Add Profile Enrollment Policy`
3. Name this policy `Customer Apps Enrollment Policy`
4. Click `Save`

Enable Self-Registration in the Profile Enrollment Policy

1. Click the `Pencil icon` next to the policy you just created (`Customer Apps Enrollment Policy`)
2. Notice that `Self-service registration` is already `Allowed` in this policy. Keep it this way and stay on this page.
3. In the **Profile Enrollment** section, click `Edit`
4. Next to **Add the user to group**, enter and select `Customers`

Add Customer Applications to the Profile Enrollment Policy

1. Click `Manage Apps`
2. Click `Add App to this Policy`
3. Next to `Customer Rewards`, click `Apply`
4. Next to `Customer Polling`, click `Apply`
5. Click `Close`

Specify Fields for the Enrollment Form

The Enrollment Form will appear during Self-Service Registation

1. Click the `← Back to Profile Enrollment Policy` link at the top of the page.
2. Scroll down to the **Profile enrollment form** section.
3. Click `Add form input`
4. Select `Username (login)`
5. Delete `First Name` and `Last Name`

✅ Checkpoint

At this point, you have configured self-service registration for the the customer applications.

Lab 5.3 Customize an Email Template

🎯 **Objective** Customize the email template used to generate the email customers receive when they sign up for Okta Ice's customer apps.

🎬 **Scenario** Okta Ice would like to apply their own branding to the email that customers receive after registration. They also need to modify the HTML code so that customers are greeted by their username.

⏱️ **Duration** 15 minutes

Navigate to the Branding UI

1. Ensure you are logged in to the Okta Admin dashboard as `oktatraining`
2. From the Admin menu, go to `Customizations` > `Branding`

Edit the Base Email Style

1. Under **Communication**, find the **Base email style** section
2. Click the `Edit` button.
3. Select `Solid background` -- this will replace the Okta logo with the logo we uploaded in Module 2. It will also set the email background color to the secondary color we set in Module 2.
4. At the top of the page, click `Save and publish`

Navigate to the Emails UI

1. From the Admin menu, go to `Customizations` > `Emails`

Open the Default Email Editor

Now we're going to customize the content of the **Email Factor Verification** email that users receive when they sign up for customer applications.

1. In the **Emails** UI, scroll down to the **Other** section.
2. Select **Email Factor Verification**
3. Click `Edit`

Examine the Email Factor Verification Template

For your convenience, we've displayed an editable copy of the default HTML code used to generate the Email Factor Verification emails here in VSCode.

1. Examine the highlighted line. This generates the salutation. Notice that it references the profile attribute `user.profile.firstName`
2. Recall that we've updated the Default User Profile so that customers do not have to provide a first name. This means the email generated will look like:

```
Hi ,
```

This doesn't look very professional at all, so let's fix this by greeting the user by their `login` (username) instead.

Edit the Email Factor Verification Template

Let's edit the highlighted line so that it references the user's `login` profile attribute:

```
Hi $!{StringTool.escapeHtml($!{user.profile.login})},
```

You can do this by manually editing the line, or by clicking the **Insert Code** link above.

Finally, click here to save your changes.

Apply the Changes in the Email UI

1. Highlight and copy the entire contents of the `custom-email-template.html` file that is open in VSCode.
2. Switch back to your Chrome browser.
3. Highlight the contents of **Message** by clicking in the field and pressing `CTRL`+`A`
4. Delete the default HTML code.
5. Paste the updated HTML code by pressing `CTRL`+`V`
6. Scroll all the way to the bottom and click `Save`
7. Optionally, click on `English (en)` to preview a sample email.

Import And Run Your Previously Configured Web Apps

We will now copy the web applications we previously configured to this workspace, so that we can test out self-service registration and the customized Email Factor Verification email.

1. Click here to copy the previously configured applications.
2. Click here to launch the web server.

Explore Self Service Registration Availability

1. Log out of `oktatraining`
2. In Chrome, navigate to http://localhost:8080
3. Click on `Polling App (Embedded Widget)`
4. Notice there is now a `Sign up` link
5. Click on `Return to Portal`
6. Click on `Rewards App (Redirect)`

7. Notice that there is a `Sign up` link here as well.

8. Click the `Back` button in Chrome to return to the index page.

9. Click `CSRM App (Redirect)

10. Notice that there is no `Sign up` link here since this is a Franchisee app, and only applied our Profile Enrollment Policy to customer apps.

11. Once again, press the `Back` button in the Chrome browser to return to the index page.

Test Self Service Registration

1. Click on `Rewards App (Redirect)`

2. Click on `Sign up`

3. For the **Email**, enter an email you have access to, so that you will be able to receive the Email Factor Verification email. If you have a Gmail account, we recommend using an email alias like `<your-gmail-username>+oktalab@gmail.com`. So if your email address was `samusaran@gmail.com`, you would enter `samusaran+oktalab@gmail.com`

4. Enter whatever you like for the **Username**

5. Click the `Sign Up` button.

Set Up a Password

You'll now be asked to set up a password.

1. Set and confirm your password as `Tra!nme4321`

2. Click `Next`

Verify Your Email

Since you must verify our email upon self-service registration, you get to see your custom **Email Factor Verification** email!

1. In a new Chrome tab, login to the email you used to register your account and locate the **Confirm your email address** email.

2. Highlight and copy the numeric verification code.

3. Switch back to the tab where you initiated the self-registration process.

4. Paste the numeric verification code into the **Enter Code** field.

5. Click `Verify`

6. If you are prompted to set up any additional optional authenticators, click `Set up later`

You should now be redirected to the Rewards app.

Test Web SSO

It may seem like you are only registered for the Rewards app. However, recall that our Profile Enrollment Policy adds self-registered users to the **Customers** group. So, you should have access to the Polling app too!

1. Click on `Return to Portal`
2. Click on `Polling App (Embedded Widget)`

Because you have an existing session and are a member of the **Customers** group, you are authenticated for the Polling App without having to register for this application or enter your credentials again!

Close Your Okta Session

1. In the Polling App in the Chrome browser, click the `Close Okta Session` button.
2. You may now close this browser tab.

✅ Checkpoint

At this point, you have customized the email template used to generate the **Email Factor Verification** customers receive when they sign up for applications. You have also verified that self-service registration works for customer applications, and that these users can access all customer applications via web SSO.

Lab 5.4 Customize Implement a Registration Inline Hook

🎯 **Objective** Implement an Okta Inline Hook to customize the Profile Enrollment flow during Self-Service Registration.

🎬 **Scenario** During their testing phase, Okta Ice would like to limit self-service registration to their customer applications to people with certain email domain names.

⏱️ **Duration** 20 minutes

Preparation and Background

The Okta registration inline hook allows you to integrate your own custom code into Okta's Profile Enrollment flow. The hook is triggered after Okta receives the registration or profile update request. Your custom code can:

- Allow or deny the registration attempt, based on your own validation of the information the user has submitted
- Set or override the values that are populated in attributes of the user's Okta profile

This custom code must be accessible publicly on the Internet rather than our localhost.

Our custom code that will handle the profile enrollment flow during self-service registration is a Nodejs app hosted on Glitch at: https://es-okta-hooks.glitch.me/

Examine `registrationHooks.js` - Routing

While our custom code for the inline hook is hosted on Glitch the contents of `registrationHooks.js` is copied and opened here so we can walk through it.

`registrationHooks.js` defines a route with the endpoint `/domain`. The anonymous function in this route gets called when the application receives a `POST` request to `https://es-okta-hooks.glitch.me/okta/hooks/registration/domain`

This is the URL Okta will need to call to via the inline hook during self-service registration.

Examine `registrationHooks.js` - Payload Validation

When our application on Glitch receives a `POST` request to `domain`, the request data is validated. We check if there is a JSON object named `data` and if that object has a `userProfile` object. The `data.userProfile` object appears in SSR requests from Okta.

If `data` or `data.userProfile` are null, the application returns a response that tells Okta to `DENY` registration. Let's take a closer look at how that response is formed.

Examine `registrationHooks.js` - Response to Invalid Payload

When Okta sends a request to your customer code via the Registration Inline Hook, Okta expects a JSON response that contains:

- an array of one or more **commands** to be executed by Okta

and/or

- an **error** object to indicate problems with the registration request

We can see in the code highlighted above that, if we have an invalid payload, we construct a `commands` array that contains one command of **type** `com.okta.action.update` and the **value** of that command is `"registration": "DENY"`

Additionally, we create an `error` object specifying that there was an invalid request payload. These two objects will be sent in the response to Okta.

This tells Okta to deny the registration request with the specified error.

Examine `registrationHooks.js` - Parse Email

What happens if the payload our app receives is valid? We get the `email` from the `data.userProfile` object and parse out certain components of the email address to handle whether we will allow or deny the registration. This decision is handled by switch cases later on in the script.

- `emailName`: Everything to the left of the `@` symbol in the email address
- `emailDomain`: Everything to the right of the `@` symbol in the email address
- `emailPrefix`: We'll use a predetermined prefix that we'll append to email addresses (e.g. `allow.gmail.com`) to demonstrate how we can limit registration to specific email domains.
- `parsedEmail`: The complete email address without the `emailPrefix` so that we can still receive Okta activation emails when we register with an email like `user@allow.gmail.com` (the activation

email will be sent to `user@gmail.com`)

Examine `registrationHooks.js` - Allow Case

If a user attempts to register with an email address with the prefix `allow` in the domain, two commands are added to the `commands` array:

- The first command is of **type** `command.okta.action.update` with a **value** of `registration: "ALLOW"`
- The second command is of **type** `com.okta.user.profile.update` with a **value** of `email: parsedEmail`

Note that the command that allows registration is not required, as it is given by default. It is provided here for clarity. The second command that updates the user's email address is provided since we are using `allow.email.com` as a stand-in for our allowed email domain. Because we want to make sure the Okta activation email goes to our actual email address at `email.com`, we issue a command to update the user's profile with the actual email address without this special `allow` prefix.

Finally, there is no **error** object for this case since it is not relevant.

Examine `registrationHooks.js` - Default Deny Case

If a user attempts to register with any other email address, the `commands` array will contain a single command to deny the registration, similar to the case when Okta sent an invalid payload.

In addition to this, an `error` object is created. The `errorCauses.errorSummary` string will be in the Sign-In Widget. The other information in this `error` object is used for logging purposes.

Both the `commands` array and `error` object are sent in the response to Okta later in this script.

Create the Registration Inline Hook

1. Ensure you are signed in to the Okta Admin dashboard as your Super Admin account, `oktatraining`.
2. In the Admin menu, select `Workflows` > `Inline Hooks`
3. Click `Add Inline Hook` and select `Registration` from the drop down menu.
4. For `Name`, enter `Email Domain Registration Hook`
5. For `URL`, enter `https://es-okta-hooks.glitch.me/okta/hooks/registration/domain`
6. For `Authentication key`, enter `x-api-key`
7. For `Authentication secret`, enter `NOTUSED` (this is just a placeholder value for our class demonstration purposes)
8. Click `Save`

Enable the Registration Inline Hook

📝 **Note:** You can associate only one inline hook at a time with your Profile Enrollment policy.

1. In the Admin menu, go to `Security` > `Profile Enrollment`.
2. Click the pencil icon next to `Customer Apps Enrollment Policy`
3. In the **Profile Enrollment** section, click `Edit`
4. In the **Inline Hook** section, select `Email Domain Registration Hook` from the drop down menu.
5. Click `Save`
6. **Log out** of Okta.

Test the Registration Inline Hook

1. Visit the Okta Ice Portal at http://localhost:8080

> > If your Okta Ice Portal app is no longer running, run in with this command:

```
python -m http.server 8080
```

2. Click on `Rewards App (Redirect)`
3. Click the `Sign up` link

In the next steps, we are going to test the following cases (where `example.com` is any domain):

- **`allow.example.com`** - Registration will be allowed, and the response to Okta will include a debugContext message.
- **any other domain** - The response to Okta will include a command to deny the registration, an error object that will be displayed in the Sign-In widget, and a debugContext message.

Test the Registration Inline Hook - Allow Case

1. In the `Email` field, enter an email address you can check so that it fits the `allow` case.

> > For example, if your email is `user@gmail.com`, you would enter **`user@allow.gmail.com`**

2. For `Username`, enter `testuser`
3. Click `Sign Up`
4. You will now see that your registration is allowed since you are prompted to set your password. Set your password to `Tra!nme4321`
5. Check your email and enter the numeric code you receive in the `Enter Code` field.

You're now registered and logged in to the Customer Rewards app.

Test the Registration Inline Hook - Default Deny Case

Now we'll test the default (deny) case, which should display a custom error in the Okta Sign-In widget

1. Click `Close Session` in the Okta Ice Portal app
2. Click `Return to Portal`
3. Click `Rewards App (Rewards)`
4. Click the `Sign up` link
5. In the `Email` field, enter your email address without any modifications.
6. In the `Username` field, enter `iamerror`
7. Click `Sign Up`

You will now see that your registration is denied with a custom error displayed in the Okta Sign-In widget:

```
Invalid email domain: gmail.com (message from inline hook)
```

✅ Checkpoint

At this point, you have created and enabled a Registration Inline Hook in a self-service registration Profile Enrollment Policy in Okta. This hook calls out to our custom Nodejs hosted on Glitch and determines whether or not a user can register for a customer application, depending on the value of their email address. Only users who register with `allow.example.com` email addresses will be permitted to register for customer applications.

Lab 5.5: Implement a User Account Update Event Hook

🎯 Objective: Implement a User account update event hook

🎬 **Scenario** Okta Ice would like to forward user account update event information to their own external logging service.

⏱ Duration: 20 min

Preparation and Background

Event hooks are outbound calls from Okta that can be used to notify your own software systems of events occurring in your Okta org.

Setting up an event hook in your Okta org requires the following generic steps:

1. Implement your external web service to receive event hook calls from Okta.
2. Register the endpoint of your external service with Okta and configure event hook parameters.
3. Verify to Okta that you control the endpoint.

4. Begin receiving ongoing delivery of event notifications.

Examine `eventHooks.js`

While our custom code for the inline hook is hosted on Glitch the contents of `eventHooks.js` is copied and opened here so we can walk through it.

`eventHooks.js` defines two routes with the endpoint `/user-profile/update`. One handles a `POST` request, and the other handles a `GET` request.

Let's take a look at the `GET` route first.

Examine `eventHooks.js` - `GET /user-profile/update`

When setting up an Okta event hook, Okta will issue a one-time `GET` request to the endpoint we specify. To verify that we truly own that endpoint, Okta requires a response with a request header named `verification` with whatever value Okta sent in its request header named `x-okta-verification-challenge`.

For example, if Okta sends a request with the header

```
{ "x-okta-verification-challenge": "topsecret" }
```

Then, our application should send a response with the header:

```
{ "verification ": "topsecret" }
```

Examine the highlighted segment in `eventHooks.js`. This segment handles `GET` requests to the `user-profile/update` endpoint. It extracts the value from the request header `x-okta-verification-challenge` and returns that value in a response header named `verification`. This meets the criteria Okta expects when verifying that we own the endpoint.

Examine `eventHooks.js` - `POST /user-profile/update`

When Okta logs an event our event hook is registered to, Okta will sent to our a request to our endpoint. The `body` of that request will contain JSON data in the following format:

```
{
"eventType": "com.okta.event_hook",
"eventTypeVersion": "1.0",
"cloudEventsVersion": "0.1",
"source": "<base-url-of-okta-org>/api/v1/eventHooks/#########",
"eventId": "################",
"data": {
    "events": [
        {  "uuid": "##################",
           "published": "#################",
            "eventType": "user.account.update_profile",
            "version": "0",
            "displayMessage": "Update user profile for Okta",
            "severity": "INFO",
            <...additional entries...>
        }
    ]
},
"eventTime": "2022-11-15T05:54:21.133Z",
"contentType": "application/json"
}
```

Examine the highlighted segment in `eventHooks.js` This handles the `POST` request to `/user-profile/update`. It will first verify that we received a valid request payload by checking that the `body` of the request contains a JSON object `data` with an entry named `events`. If we received a valid payload, the `eventType` entry located in the `events` JSON array is stored to a variable called `eventType`. In the example payload above, this would be `"user.account.update_profile"`.

The event type (embedded in a `description` string), requesting URL, and the entire request body, are then sent to the **Hook Viewer**, which stands in for our logging system. We'll look at the Hook Viewer when we start logging events.

Create the User Account Update Event Hook

1. Ensure you are signed in to the Okta Admin dashboard as your Super Admin account, `oktatraining`.
2. In the Admin menu, select `Workflows` > `Event Hooks`
3. Click `Create Event Hook`
4. For `Name`, enter `User Account Update Event Hook`
5. For `URL`, enter `https://es-okta-hooks.glitch.me/okta/hooks/event/user-profile/update`
6. Leave the `Authentication` fields blank as we will not use them for the purposes of this lab.
7. Next to `Subscribe to events`, type in and select `User's Okta profile updated`
8. Click `Save & Continue`

Verify the User Account Update Event Hook

At this stage, Okta will ask you to **verify** the endpoint that Okta will post event data to. This is where Okta will make a `GET` request with the `x-okta-verification-challenge` header and our application will send a `verfication` header in the response.

Click the `Verify` button.

If verification fails, double check that you have entered the correct `URL` in the previous step.

Open the Hook Viewer

Open a new tab and visit https://es-okta-hooks.glitch.me/

We will use this page to view logged events. Keep in mind that the endpoint we are using in this example is shared among all classmates, so you will see events logged from other orgs here.

Test the User Account Update Event Hook

1. Switch to your Okta tab
2. In the Admin menu, navigate to `Directory` > `People`
3. Click on `Kay West`
4. Click on the `Profile` tab.
5. Click `Edit`
6. Update the **Last Name** to `East`.
7. Expand the VM's **Credentials** panel.
8. Scroll down and click `Save`

Check the Hook Viewer

1. Switch back to the Hook Viewer tab
2. Verify that an event was logged from your Okta org. Your org URL will be listed as the `source`

✅ Checkpoint

At this point, you have created and registered an Event Hook for logging updates to user's accounts to an external service using these four steps:

1. Implement your external web service to receive event hook calls from Okta.
2. Register the endpoint of your external service with Okta and configure event hook parameters.
3. Verify to Okta that you control the endpoint.
4. Begin receiving ongoing delivery of event notifications.

🎉 End of Module 5 Labs

You may close this workspace project, ensuring all changes were saved.

Module 6: Migrating and Managing Users

Lab 6.1 Migrate Users and Hashed Passwords with Okta's Users API

🎯 **Objective** Migrate users with Okta's Users API

🎬 **Scenario** Okta Ice has an existing data store of customers that they would like to migrate to Okta. They want customers to be able to sign in with their existing passwords. Since the passwords are hashed and salted, they can import these in bulk using Okta's Users API.

⏱ **Duration** 20 minutes

Examine the Create User with Imported Hashed Password API Endpoint

Okta's Users API has an endpoint that allows us to create a user with an imported hashed password.

It is similar to the `Create User` endpoint we used in Lab 4.2, except we provide some additional details in the `credentials.password` entry of the JSON request payload:

```
"credentials": {
    "password" : {
      "hash": {
        "algorithm": {{algorithm-name}},
        "workFactor": {{workfactor-value}},
        "salt": {{salt-value}},
        "value": {{hashed-password}}
      }
    }
  }
```

Notice that, instead of just providing a password value, we provide the hashing algorithm, a workfactor (if relevant to the hashing algorithm), the salt value, and the hashed value of the password.

Let's create this API request in Postman.

Create a New API Request in Postman

Since the Create User with imported Hashed Password API request is not included in the Okta Users API collection, we will have to create it. We'll do this by duplicating a similar request, renaming it, and modifying the body of the request.

1. Open Postman
2. Ensure your Environment is set to the environment you created in Module 3. It should have the same name as your Okta org.
3. Expand the `Users (Okta API)` collection
4. Expand the `Create User` folder
5. Right click on `Create Activated User with Password` and select `Duplicate`
6. In the new API request tab that opens, hover your mouse over the title (`Create Activated User with Password Copy`) and the pencil icon that appears.
7. Rename this API request `Create User with Imported Hashed Password`

8. Click `Save`

Modify the Request Body of the Create User with Imported Hashed Password API Request

1. Notice that this is a `POST` request. We are going to be sending data via the request body.

2. Delete the contents of the request body.

3. Copy the following JSON and paste it into the request body:

```
{
  "profile": {
    "firstName": "Hashem",
    "lastName": "Pesar",
    "email": "hashem.pesar@{{email-suffix}}",
    "login": "hashem.pesar@{{email-suffix}}",
    "userType": "customer"
  },
"credentials": {
    "password" : {
      "hash": {
        "algorithm": "BCRYPT",
        "workFactor": 12,
        "salt": "S77ORXjoSbjRMdvG/Yvc0u",
        "value": "x6WB1d4z5BjpeuyANyVWZqBbIfCq24q"
            }
        }
    }
}
```

This will import our existing user with a username of `hashem.pesar@oktaice.com` and their existing password that was previously hashed using the bcrypt algorithm with the salt provided. Notice that we also specify a `userType` of `customer` so that our existing Group Rule (Lab 1.2) will assign this user to the `Customers` group. This will give this user access to the customer apps.

Create the User with an Imported Hashed Password

1. Click `Send` in Postman.

2. Examine the JSON response from Okta.

You'll notice Okta echoes back the user profile details. If you look at the `credentials` entry, you should see:

```
"credentials": {
        "password": {},
        "provider": {
            "type": "IMPORT",
            "name": "IMPORT"
        }
```

This indicates that the user credentials were imported from an external source via the API.

Test the Imported User

1. Click [here to copy the previously configured applications](command:codetour.sendTextToTerminal?["mkdir redirect; mkdir embedded; cp ../03-exploring-authentication-protocol-flows/redirect/* redirect; cp ../03-exploring-authentication-protocol-flows/* embedded"]).
2. Click [here to launch the web server](command:codetour.sendTextToTerminal?["python -m http.server 8080"]).
3. Navigate to http://localhost:8080
4. Click `Rewards App (Redirect)`
5. Enter `hashem.pesar@oktaice.com` as the **Username**
6. Click `Next`
7. Enter `Tra!nme4321` as the password
8. Click `Verify`
9. Once you are redirected to the Rewards App, click `Close Okta Session`

✅ Checkpoint

At this point, you have successfully imported an existing customer with their hashed password into Okta! In a real-world scenario, you would import users in bulk.

Okta employs built-in rate limit controls designed to protect the Okta service from the negative impacts that high traffic levels can create. This enables Okta to maintain service uptime and stability. As a result, during heavy usage periods an Okta tenant might experience traffic spikes that cause rate limits to go into effect. To avoid having rate limits impact your migration it’s suggested that you work with Okta support to plan your user migration during a time when rate limits can be temporarily adjusted and identify what the available options are for doing so.

🎉 End of Module 6 Labs

You may close this workspace project, ensuring all changes were saved.

Module 7: Securing Your Environment with Policies and MFA

Lab 7.1 Configure Passwordless Authentication with Email Magic Link

🎯 **Objective** Configure passwordless authentication with Email Magic Link

🎬 **Scenario** Okta Ice would like to provide customers with the option of using a passwordless method of authentication when accessing the Rewards and Polling apps.

⏱️ **Duration** 15 minutes

Import And Run Your Previously Configured Web Apps

We will now copy the web applications we previously configured to this workspace, so that we can test out passwordless authentication and, later, multifactor authentication and self-service account recovery.

1. Click [here to copy the previously configured applications](command:codetour.sendTextToTerminal?["mkdir redirect; mkdir embedded; cp ../03-exploring-authentication-protocol-flows/redirect/* redirect; cp ../03-exploring-authentication-protocol-flows/* embedded"]).

2. Click [here to launch the web server](command:codetour.sendTextToTerminal?["python -m http.server 8080"]).

Enable the Authentication and Recovery Options

1.In the Admin menu, go to `Security` > `Authenticators`

2. Click the `Actions` drop-down in the **Email** row of the list of authenticators.

3. Click `Edit`

4. In the **Used for** section, select `Authentication and recovery`

5. Click `Save`

This means that email can now be used as an authenticator and not just for account recovery. We'll still need to configure a Sign On policy to use email as an authenticator.

Disable Unused Authenticators

For the purposes of this class, we'll only be using the Password and Email (Magic Link) authenticators. So, let's disable the ones we aren't using.

1. On the Authenticators page, click on the `Enrollment` tab.
2. Click the `Edit` button.
3. Change **Okta Verify** to `Disabled`
4. Change **Phone** to `Disabled`
5. **Email** should be `Optional` and **Password** should be `Required`

Configure the Email Magic Link Sign On Policy

1. Ensure you are logged in to the Admin Dashboard as `oktatraining`

2. In the Admin menu, go to `Security` > `Authentication Policies`

3. Click `Add a Policy`

4. Name the policy `Email Magic Link`

5. Click `Save`

6. Click `Add Rule`

7. Name the rule `Customers Email Magic Link Rule`

8. Next to **User's group membership includes**, click the drop-down and select `At least one of the following groups`

9. In the `Enter groups to include` field that pops up below, search for and select the `Customers` group

10. Next to **User must authenticate with**, select `Any 1 factor type` from the drop-down.

11. Next to **Possession factor constraints are**, uncheck `Exclude phone and email authenticators`

12. Click `Save`

Configure Customer Apps to Use the Email Magic Link Sign On Policy

1. Back on the `Email Magic Link` policy page, click the `Applications` tab.

2. Click `Add App`

3. Click the `Add` button next to `Customer Rewards App`

4. Click the `Add` button next to `Customer Polling App`

5. Click `Close`

6. Log out of Okta.

Test the Email Magic Link Sign On Policy

1. Navigate to http://localhost:8080
2. Click on `Rewards App (Redirect)`
3. Enter the username `testuser`, which was previously registered with your personal email address in Module 5.
4. You now have the choice to authenticate with Email or Password. Click `Email`
5. Click `Send me an email`
6. Click `Enter a verification code instead`
7. Check your personal email for an email from your Okta Org and copy the numeric code into the `Enter Code` field
8. Click `Verify` and you will be logged in to the Customer Rewards application!
9. Click `Close Okta Session`
10. Click `Return to Portal`

📝 **Note**: In order for the clickable Email Magic Link to work, the link must be clicked in the same device that the authentication process was initiated. Since we're using a VM and you are likely checking your email on a personal device, we circumvent this by using the verification code instead.

✅ Checkpoint

At this point, you have configured and enabled passwordless authentication for Okta Ice's customers. You have done this by creating a Sign On Policy that specifies that any user in the customer group can sign in with any one factor. Since this policy was applied to the Customer Rewards and Customer Polling apps, customers can choose to authenticate with email or password when accessing this application.

Lab 7.2: Configure MFA with Two Factor Types

🎯 **Objective** Configure multifactor authentication (MFA) with two factor types.

🎬 **Scenario** Okta Ice would like to secure the Franchisee CRM application with MFA. They'd like to require franchisees to authenticate with a knowledge factor (password) and a possession factor (email).

⏱️ **Duration** 15 minutes

Make Yourself a Franchisee

1. Log in to your Okta org as `oktatraining` and access the Admin dashboard.
2. From the Admin menu, select `Directory` > `People`
3. Click on `Kay West`
4. Click on `Profile`
5. Click `Edit`
6. Update the **Primary email** to your personal email address
7. Click `Save`

Create the Franchisee Two Factor Okta Sign-On Policy

1. In the Admin menu, go to `Security` > `Okta Sign-On Policy`.
2. Click `Add a Policy`
3. Name the policy `Franchisee Two Factor`
4. Click `Save`
5. Click `Add Rule`
6. Name the rule `Franchisee Two Factor Rule`
7. Next to **User's group membership includes**, click the drop-down and select `At least one of the following groups`
8. In the `Enter groups to include` field that pops up below, search for and select the `Franchisees` group
9. Next to **User must authenticate with**, select `Password + Another factor` from the drop-down.
10. Next to **Possession factor constraints are**, uncheck `Exclude phone and email authenticators`
11. Click `Save`

Configure Franchisee CRM App to Use Two Factor Sign On Policy

1. Back on the `Franchisee Two Factor` policy page, click the `Applications` tab.

2. Click `Add App`

3. Click the `Add` button next to `Franchisee CRM App`

4. Click `Close`

5. Log out of Okta.

Test the Two Factor Sign On Policy

1. Navigate to http://localhost:8080
2. Click on `CRM App (Redirect)`
3. Enter the **Username** `kay.west@oktaice.com` and click `Next`
4. Enter the **Password** `Tra!nme4321!` and click `Verify`
5. You are now prompted for a email as a second factor of authentication.
6. Click `Send me an email`
7. Click `Enter a verification code instead`
8. Check your personal email for an email from your Okta Org and copy the numeric code into the `Enter Code` field
9. Click `Verify` and you will be logged in to the Customer Rewards application!
10. Click `Close Okta Session`
11. Click `Return to Portal`

📝 **Note**: In order for the clickable Email Magic Link to work, the link must be clicked in the same device that the authentication process was initiated. Since we're using a VM and you are likely checking your email on a personal device, we circumvent this by using the verification code instead.

✅ Checkpoint

At this point, you have configured and enabled MFA using two authentication factors -- Knowledge (password) and Possession (email) You have done this by creating a Sign On Policy that specifies that any user in the Franchisee group must sign in with a password and an additional factor. Since this policy was applied to the Franchisee CRM app, any Franchisee must authenticate with a password and an additional factor. Since we've only configured email as an additional factor, that is the only other factor available in this case.

🎉 End of Module 7 Labs

You may close this workspace project, ensuring all changes were saved.

Module 8: Authenticating to Okta from External IdPs

Lab 8.1 Authenticate with AD FS as an External IdP

🎯 **Objective** Set up AD FS as an external IdP.

🎬 **Scenario** Okta Ice wants to allow customers to be able to authenticate via additional external IdPs. For our proof of concept that doesn't require getting developer keys from social IdPs (e.g., Facebook or Twitter), we'll use AD FS.

⏱ **Duration** 20 minutes

Review the Partner Organizational Unit in AD

The Partner OU contains users that are not synced to Okta via Active Directory. You will use these users to test using AD FS as an external IdP integrated through Okta.

1. In your VM, open `Active Directory Users and Computers` by clicking the icon in your Windows toolbar.
2. On the left panel of the **Active Directory Users and Computers** window, select`oktaice.local` > `Partners`

You should see that the `Partners` organizational unit (group) contains six users. One of these users is Diane Smith. We'll be using that account to test our local AD FS configuration before moving on to configuring AD FS as an external IdP through Okta.

Validate your AD FS Configuration

In this section, you confirm that your AD FS can authenticate users from Active Directory.

1. Open Chrome and navigate to https://adfs.oktaice.local/adfs/ls/idpinitiatedsignon
2. Click `Sign in`
3. Sign in with `diane.smith@oktaice.local` as the **username** and `Tra!nme4321` as the **password**.
4. Click `Sign in`
5. Verify that sign in is successful. This verifies that AD FS is correctly configured and running on your machine.
6. Sign out and close your browser.

Run the Windows Certificate Export Wizard

AD FS uses a signing certificate to sign its SAML assertions. In the next two steps, you will run the Windows Certificate Export Wizard and download the signing certificate.

In a future step, we will upload this certificate to Okta. Okta will use the signing certificate during the app initiated sign-on to verify that the SAML assertion is generated by AD FS.

1. Open `AD FS Management` by clicking on the From the Windows toolbar.
2. In the left panel of the **AD FS Management** window, select `AD FS` > `Service` > `Certificates`
3. In the `Certificates` panel, right-click the certificate under `Token-signing` and select `View Certificate`
4. Click the `Details` tab.

5. Click the `Copy to File...` button.

The **Windows Certificate Export Wizard** is launched.

Download the AD FS Signing Certificate

1. In the **Windows Certificate Export Wizard** window, continue clicking `Next`, taking the default settings, until you reach the step where the wizard asks you what you would like to name your file.

2. Click the `Browse...` button.

3. Navigate to the `Desktop` folder

4. Enter `adfs.cer` as the **File name** and click `Save`

5. Click `Finish`

6. A dialog box will pop up indicating that the file export was successful. Click `OK` to close this dialog box.

7. Click `OK` in the **Certificate** window to close it.

8. Minimize the **AD FS Management** window, as we will return to it in a subsequent step.

9. Navigate to your Desktop and confirm that the `adfs.cer` file was saved there.

Add a New SAML 2.0 Identity Provider (IdP)

We'll use our service account to add an AD FS as a SAML 2.0 IdP.

1. Open Chrome and log in to the Okta Admin Dashboard as `oktatraining`
2. From the Admin menu, go to `Security` > `Identity Providers`
3. Click `Add identity provider`
4. Select `SAML 2.0 IdP`
5. Click `Next`

Configure the SAML 2.0 IdP

1. Name this configuration `AD FS`

2. In the `IdP username` field, select `idpuser.subjectNameId`

3. In the **JIT Settings** section, select `Update attributes for existing users`

4. Under **Group Assignments**, select `Assign to specific groups`

5. In the `Specific Groups` field, type and select `Customers`

6. Scroll down to the **SAML Protocol Settings** section

7. For `IdP Issuer URI`, enter `http://adfs.oktaice.local/adfs/services/trust`

8. For `IdP Single Sign-On URL`, enter `https://adfs.oktaice.local/adfs/ls`

9. For `IdP Signature Certificate`, click `Browse files...`

10. In the file browser window that opens, select `All files` from the drop down in the bottom right corner.

11. Navigate to the `Desktop` folder and double click on the `adfs.cer` file to upload it.

12. Back in the browser, click `Finish`

Configure IdP Routing Rules

We're going to create a routing rule that tells Okta to use AD FS as our IdP for any user that logs in with an `oktaice.local` email address.

1. Click the `Routing rules` tab
2. Click `Add Routing Rule`
3. Name the rule `AD FS Rule`
4. Next to **User matches** select `Domain list on login`
5. In the field that appears beneath, type in `oktaice.local` and click `add oktaice.local...`
6. In the **Use this identity provider** section, keep `Use specific IdP(s)` selected.
7. In the **IdPs** section at the bottom, remove `Okta` by clicking the `X` and then type and select `AD FS`
8. Click `Create rule`
9. Finally, click `Activate` to activate the rule.

Download the Metadata

1. Click on the `Identity providers` tab

2. In the table of identity providers, click the arrow next to the `AD FS` entry to expand it.

3. Click `Download metadata` and the `metadata.xml` file will be saved to your `Downloads` folder.

4. **Log out of Okta**

Launch the Add Relying Party Trust Wizard

1. Maximize the AD FS Management window you minimized in your VM.
2. In the left panel, open `AD FS` > `Relying Party Trusts`
3. In the right panel, click `Add Relying Party Trust`

The **Add Relying Party Trust Wizard** is launched.

Register Okta as a Relying Party in AD FS

1. In the **Add Relying Party Trust Wizard** window, ensure `Claims aware` is selected and click `Start`
2. In the **Select Data Source** step, select `Import data about the relying party from a file` and click `Browse`
3. In the file browser window that opens, navigate to the `Downloads` folder and double-click the `metadata.xml` file you downloaded from Okta.
4. Click `Next`

5. Enter `Okta Ice` as the **Display Name** and click `Next`
6. Continue clicking `Next`, taking the default values, until the Wizard reaches the last step.
7. Click `Close`

A window labelled **Edit Claim Issuance Policy for Okta Ice** should pop up. We'll configure the AD FS Claim rule in this window in the next step.

Configure an AD FS Claim Rule

Claim rules define what user data will be sent from AD FS to Okta via SAML assertion.

1. In the **Edit Claim Issuance Policy for Okta Ice** window, click `Add Rule...`
2. Select `Send LDAP Attributes as Claims` in the **Claim rule template** drop-down and click `Next`
3. **Name** the claim rule `Attributes to Okta`
4. Select`Active Directory` under the **Attribute store** drop-down.
5. Under **Mapping of LDAP attributes to outgoing claim types** add these additional attributes:

LDAP Attribute	Outgoing Claim Type
`Given-Name`	`firstName`
`Surname`	`lastName`
`E-Mail-Addresses`	`email`
`E-Mail-Addresses`	`Name ID`

Finally, click `Finish` and then click `OK` in the **Edit Claim Issuance Policy for Okta Ice** window.

Create a Relying Party Identifier in AD FS for Okta Ice

The Relying party identifier is used to trigger a login without passing by a select Relying Party form in AD FS.

1. Return to `AD FS` > `Relying Party Trusts` in the **AD FS Management** window.

2. Right-click `Okta Ice` and then click `Properties`

3. Click the `Identifiers` tab.

4. Enter `OktaIce` as `Relying party identifier` and click `Add` to add it to the `Relying party identifier` text box.

5. Click `OK`

Test Authentication with AD FS

1. Click [here to copy the previously configured applications](command:codetour.sendTextToTerminal?["mkdir redirect; mkdir embedded; cp ../03-exploring-authentication-protocol-flows/redirect/* redirect; cp ../03-exploring-authentication-protocol-flows/embedded/* embedded"]).

2. Click [here to launch the web server](command:codetour.sendTextToTerminal?["python -m http.server 8080"]).

3. Navigate to http://localhost:8080

4. Click `Rewards App (Redirect)`

5. Enter `diane.smith@oktaice.local` as the **Username**

6. Click `Next`

7. Enter `Tra!nme4321` as the password

8. Click `Verify`

The user should now be logged in via AD FS as an IdP. This should also create a new Okta-managed user via Just-in-Time (JIT) provisioning with the profile attribute mapping we set up in AD FS.

Finally, click `Close Okta Session`

Verify JIT Provisioning

1. Log back in to the Okta Admin Dashboard as `oktatraining`
2. Click `Directory` > `People`
3. Search for `Diane Smith` and click on the user
4. Confirm that Okta displays the message **Profile sourced by SAML 2.0 IdP**.
5. Click the `Profile` tab.
6. Confirm that the user profile now contains the user email and name. This confirms that the Just in Time provisioning (JIT) is working.

✅ Checkpoint

At this point, you have Okta integrated with AD FS as Identity Provider using SAML. This integration allows users located under the Partner OU to log into Okta using via AD FS. Okta leverages Just-In-Time (JIT) provisioning to automatically recognize and create users federated via integration with AD FS.

🎉 End of Module 8 Labs

You may close this workspace project, ensuring all changes were saved.