XModule Certification Bootcamp Lab Guide

Part of the certification process requires experience using XModule and its elements to prove your ability to create content for app users. The labs included in this guide walk you through XModule concepts, elements, architecture, and coding practices to help you on your journey.

Before attending this class you should have completed the following:

- Watched the following videos:
  - Modo Labs Overview: https://i.modolabs.com/highered/training/_/intro_app_center
  - Application Managers: https://i.modolabs.com/highered/training/_/intro_app_manager
- Complete the W3 Schools Javascript Exercises to refresh your knowledge of Javascript: https://www.w3schools.com/js/exercise_js.asp?filename=exercise_js_variables1

Requirements:

- Laptop and external screen (provide) / tablet / additional laptop to view this lab guide
- Basic knowledge of the Modo App platform
- Coding basics and some Javascript basics (we will be coding in the AWS Console using NodeJS 18 for this class)
- An activated Modo App Center Account and access to Modo support center (can be requested through Max Denmat or by emailing support@modolabs.com)

Assumptions:

- You have two screens:
  - Screen one (your working screen) has Modo App center open and you are logged and have the following tabs open:
    - Your student guide open for access to your Modo Admin Center and important information and links
    - Modo App Center opened to your assigned Modo Application
    - i.modolabs.com
    - Your Okta Login
    - the AWS Environment
  - Screen two has this lab guide open
- Google Chrome browser is being used
- You have an external mouse - laptop trackpads will cause delays in learning
- Are paying attention to the lecture to make sure we are on the same page

Acronyms Used:

- IdP - Identity Provider
- JSON - JavaScript Object Notation
- JWK - JSON Web Key (more information / specification)
- JWKS - JSON Web Key Sets (more information / specification)
- JWT (pronounced jot) - JSON Web Token (more information)

Table of Contents

Lab 1 - Modo App Familiarization **2**
Task 1: Create a Location and a Persona 2
Steps: 2
Task 2: Create a Publish Module with Hello Modo screen 2
Steps: 3
Task 3: Add the Hello Modo Publish screen as the default home module 5
Steps: 6
Task 4: Add a Theme 7
Steps: 11
Lab 2 - Using XModule Resources and AWS **13**
Task 1: Using the XModule Sandbox to test elements 13
Steps: 14
Task 2: Hosting XModule JSON in npoint.io 21
Steps: 22
Task 3: Accessory Icons, Accessory Buttons, and (we don't need no stinkin') Badges 24
Steps: 25
Lab 3 - JWTs, Authentication, and the Modo Communicate Directory **30**
Task 1: Configuring an AWS Lambda Function and your Modo App to show HTTP Request header values 31
Steps: 32
Task 2: Adding header values and changing the Modo JWT header 39
Steps: 40
Task 3: Decoding the JWT and verifying the signature 43
Steps: 46
Task 4: Passing and configuring authentication properties for the JWT 51
Steps: 52
Task 5: Authentication and the Modo Communicate Directory 58
Steps: 60
Lab 4 - RelativePath - AJAX Content and other link types **62**
Task 1: Create a multipage XModule 64
Steps: 68
Task 2: Loading Pages Via AJAX 85
Steps: 88
Task 3: Load page with Geolocation 89
Steps: 90
Lab 5 - Forms **93**
Task 1: Standard Form submission with Progressive Disclosure 94
Steps: 95
Task 2: Progressive Disclosure with AJAX 96
Steps: 100
Task 3: Assisted Select / Assisted Multiselect with and without AJAX 103
Steps: 104
(Optional) Task 4: Save selected favorite foods to communicate database 105

Lab 6 - Events and Actions **107**
Task 1: Directory Search 108
Steps: 109
Task 2: Filtering results using a dropdown 111
Steps: 113
Task 3: Adding multiple events to each click 115
Steps: 117
Lab 7: Full Page Layout **120**
Task 1: Column types, contentContainerWidth, and headers 120
Steps: 121
Task 2: Using the responsiveVisibility property 124
Steps: 124
Lab 8: Miscellaneous topics **126**
Task 1: Collage screen XComponent blocks 127
Steps: 127
Task 2: Redirects and Cookies 128
Steps: 129
Task 3: Other screen-level properties 132
Steps: 132
Task 4: Request Logger 139
Steps: 139
Task 5: Barcode Button 139
Steps: 140
Lab 9: Final Project! **140**
Appendix **141**
Lab 3: All Tasks 141
Lab 4: Task 1 144
Lab 4: Task 2 146
Lab 5: Task 1 - POST Method 149
Lab 5: Task 1 - GET Method with banner 153
Lab 5: Task 2 158
Lab 5: Task 3 160
Lab 6: Task 1 162
Lab 6: Task 2 164
Lab 6: Task 3 166
Lab 7: Task 1 176
Lab 7: Task 2 182
Lab 8: Task 1 183
Lab 8: Task 2 186

Lab 1 - Modo App Familiarization

Estimated Time: 20 minutes

This lab provides a common starting point for all users. You must complete these tasks to preview any content in your Modo application and to ensure we have a similar experience as we build our modules. We could call this lab "**Hello Modo!**" in honor of the traditional "Hello World" programming tasks for other programming languages.

Task 1: Create a Location and a Persona

Context: Every application needs a default location and a default persona (experience) for the application to function. The persona makes up a major part of the URL structure. These must be created before a Screen can be created.

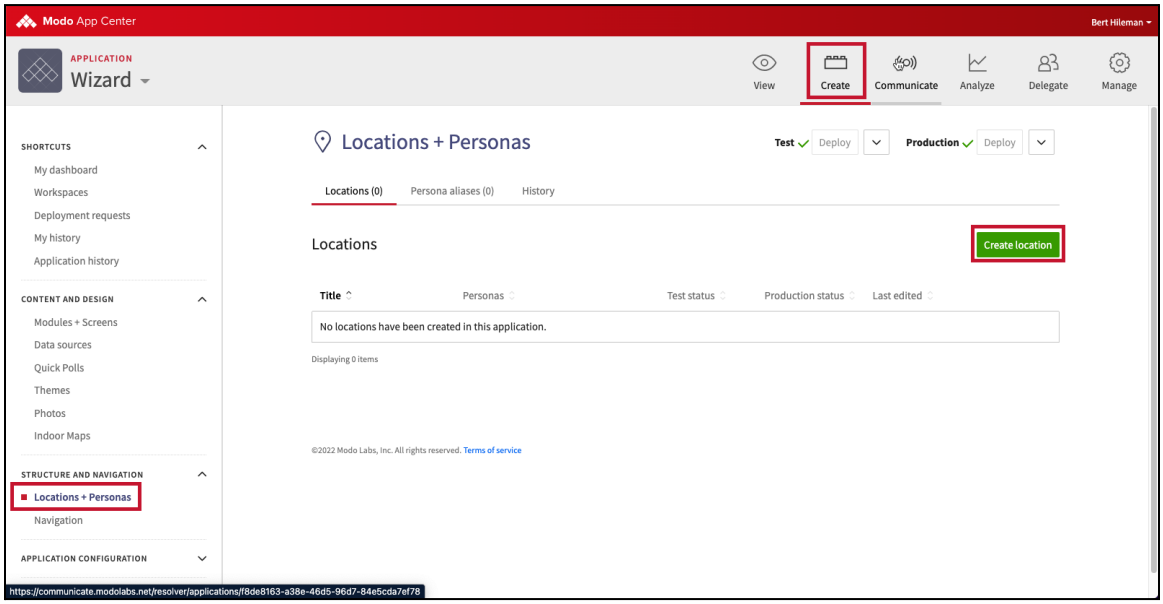
Goal: Create a default location and persona for the app

Task Outline:

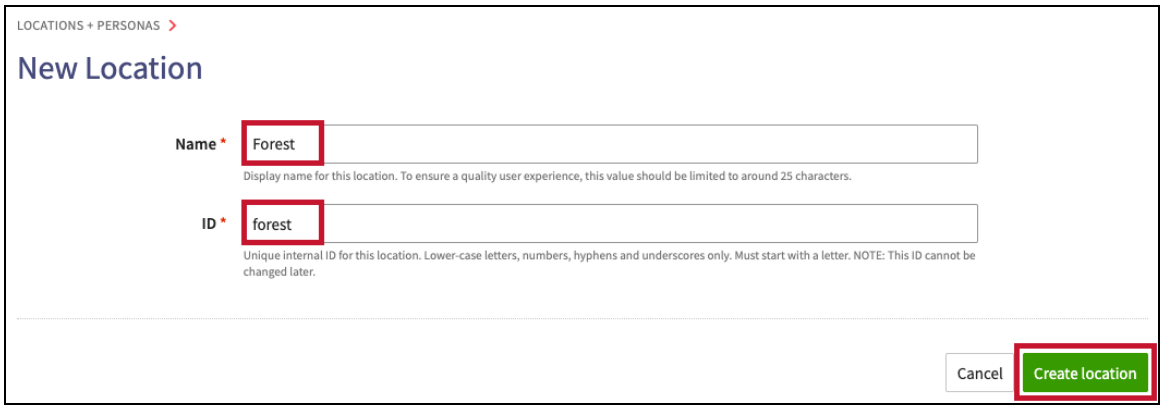
- Create a new location named **forest**, and a persona using your assigned character name.

Steps:

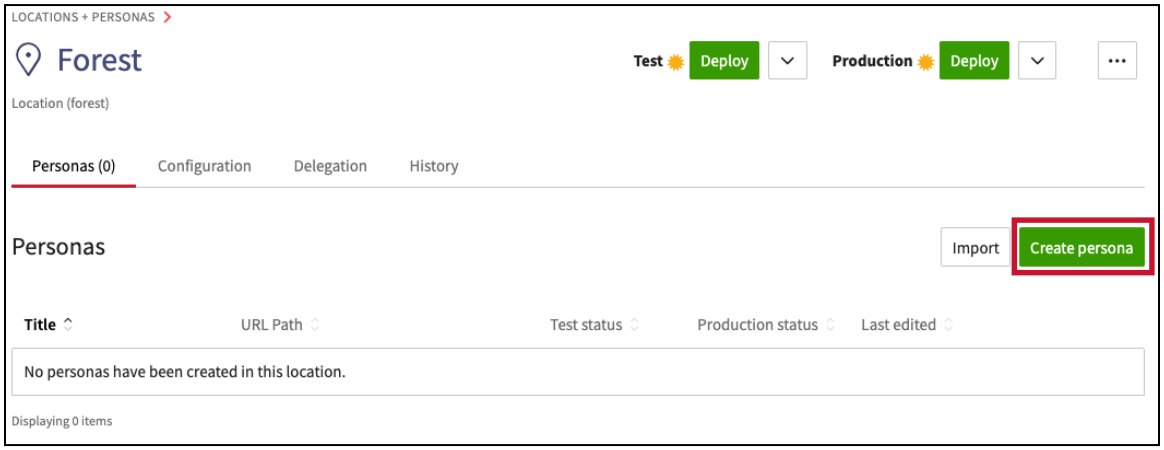
1. Under **Create**, click **Locations + Personas** > **Create location**.



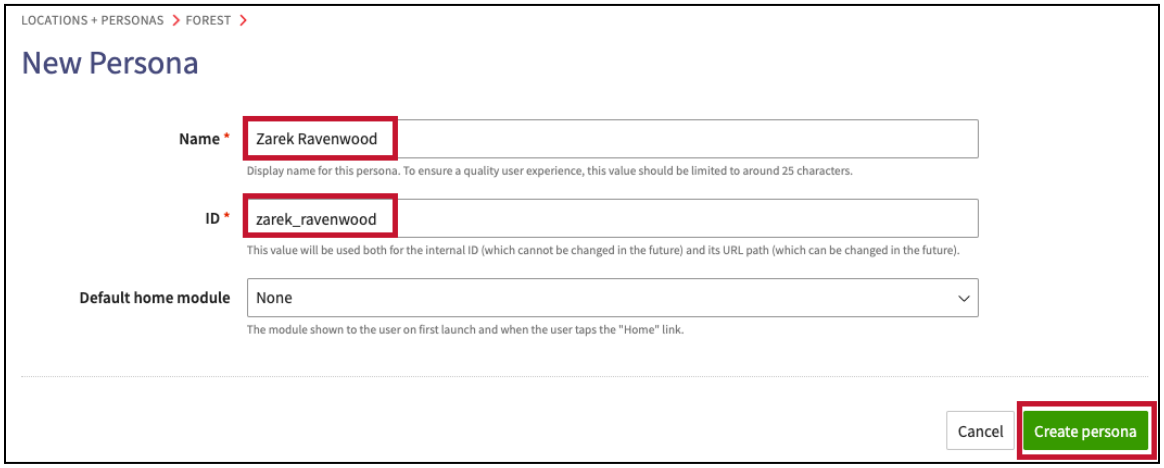
2. Type **Forest** for **Name** and make sure **forest** is automatically added for **ID** then click **Create location**.



3. Click **Create persona**.



4. Type your assigned character name for **Name**, make sure the **ID** field is a lowercase and underscore separated version of the same then click **Create persona**.



Congratulations! We have successfully created a location and persona and are ready to add the first Module and Screen to the application so we can test it.

Task 2: Create a Publish Module with Hello Modo screen

Context: Every application needs a starting page. It doesn't matter what kind of page you use. Publish modules and screens are the easiest type of screen to make. These pages feature static content interspersed with data driven dynamic content. If you ever start with a blank Modo application, this may be the quickest way for you to get started.

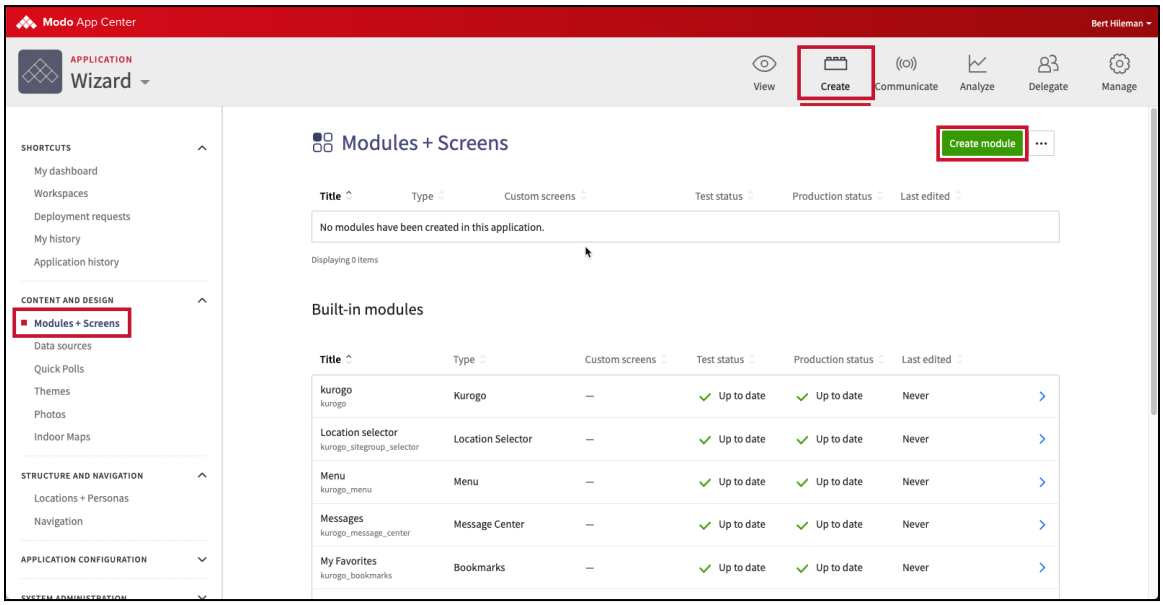
Goal: Create a publisher module and screen that says “Hello Modo!”

Task Outline:

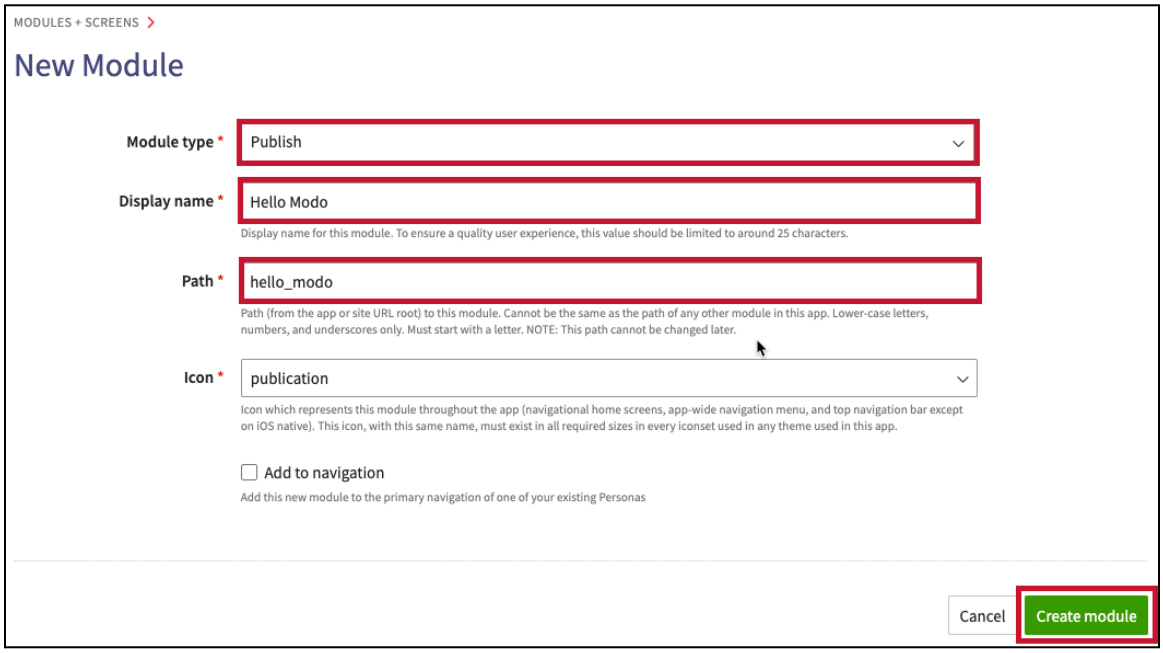
- Create a Publish module titled **Hello Modo** and a collage screen titled **Hello Modo** with a Text/HTML block on the page that says **Hello Modo!**

Steps:

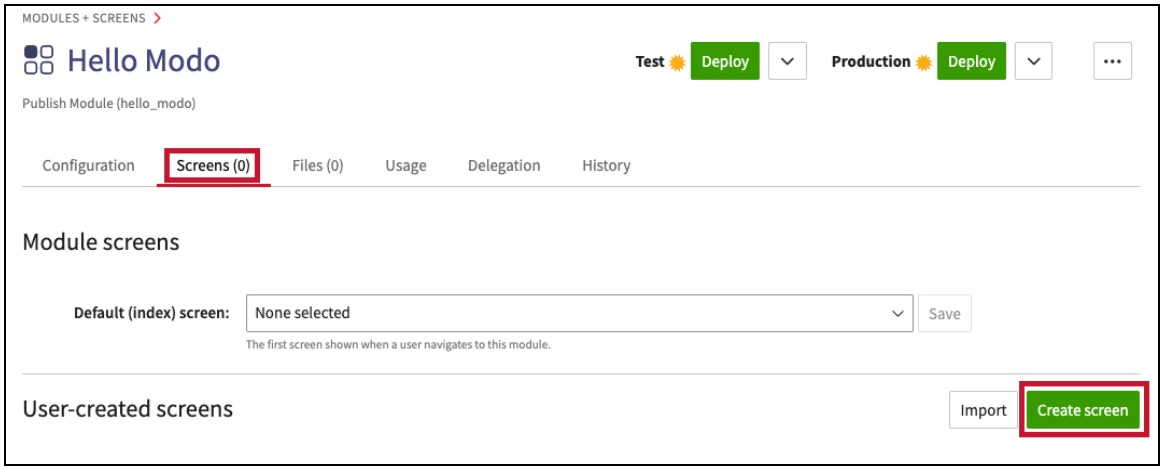
1. Under **Create**, click **Modules + Screens** > **Create module**.



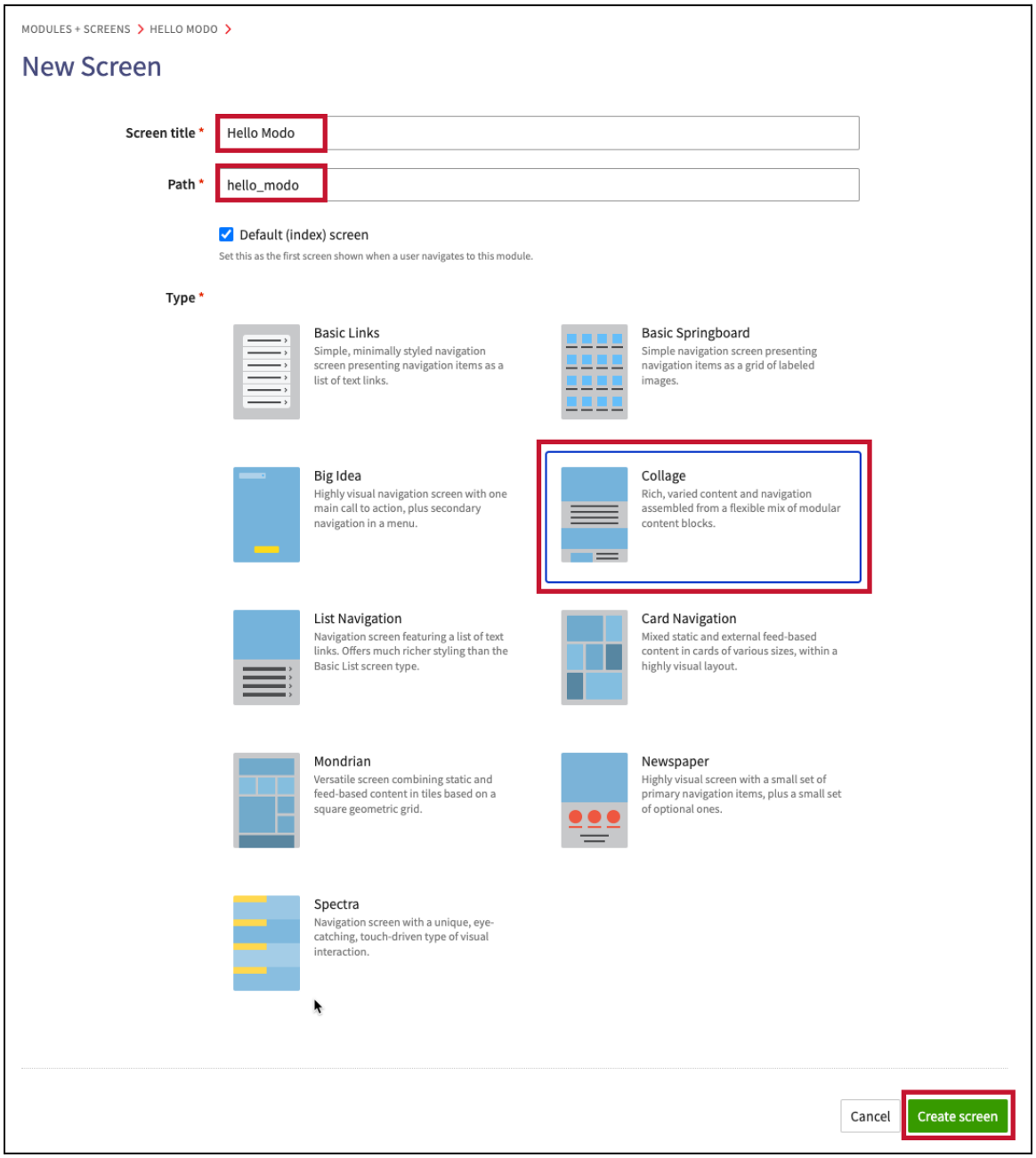
2. Select **Publish** for **Module type** and complete the form by doing the following:
    a. Type **Hello Modo** for **Display name**
    b. Type **hello_modo** for **Path**
    c. Select **publication** for **Icon** (should be the default setting)
    d. Click **Create module**.



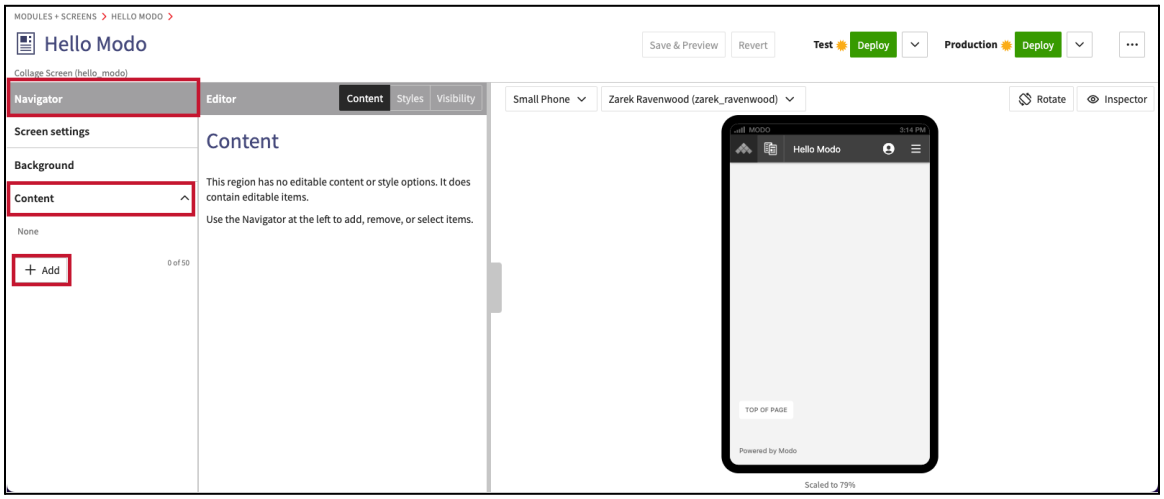
3. Click **Screens** then click **Create screen**.



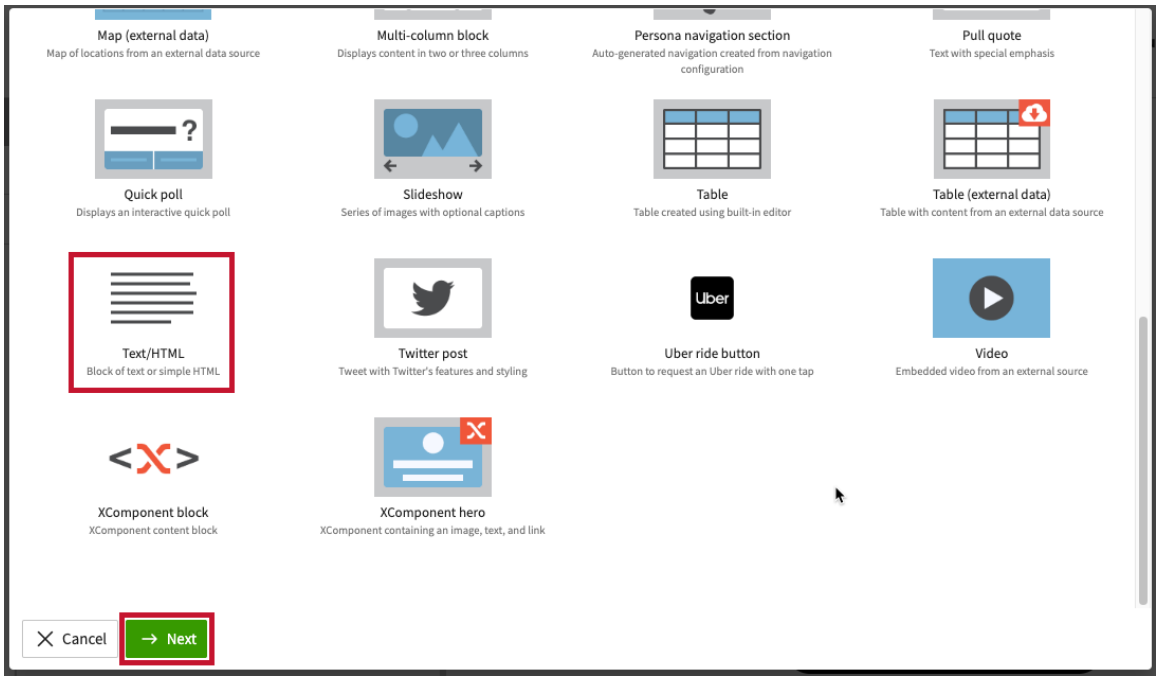
4. Type **Hello Modo** for **Screen title**, make sure **hello_modo** is automatically added for **Path**, click **Collage** for **Type,** then click **Create screen**.



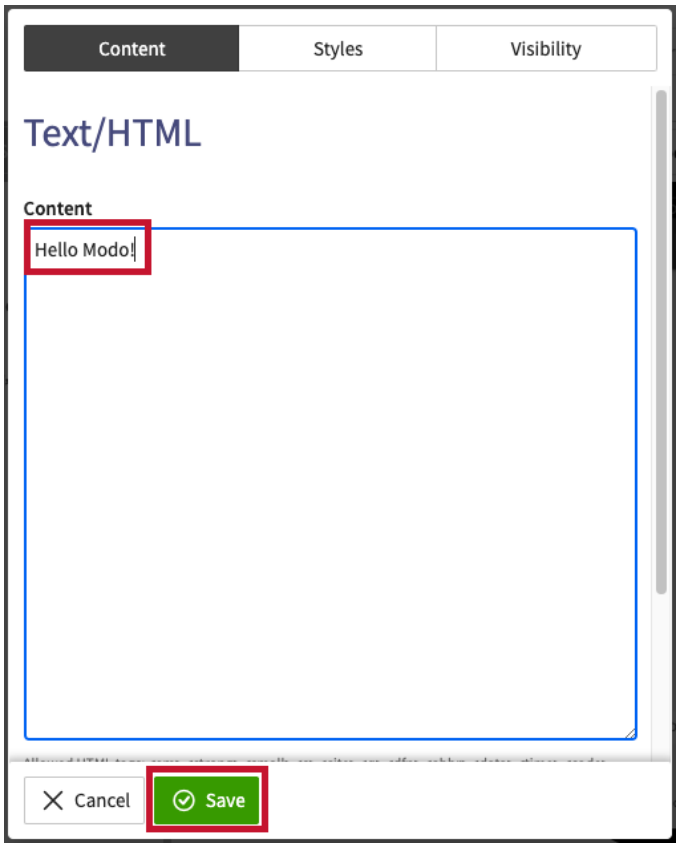
5. In the **Navigator** section of Screen builder, click to expand **Content**, then click **Add**.



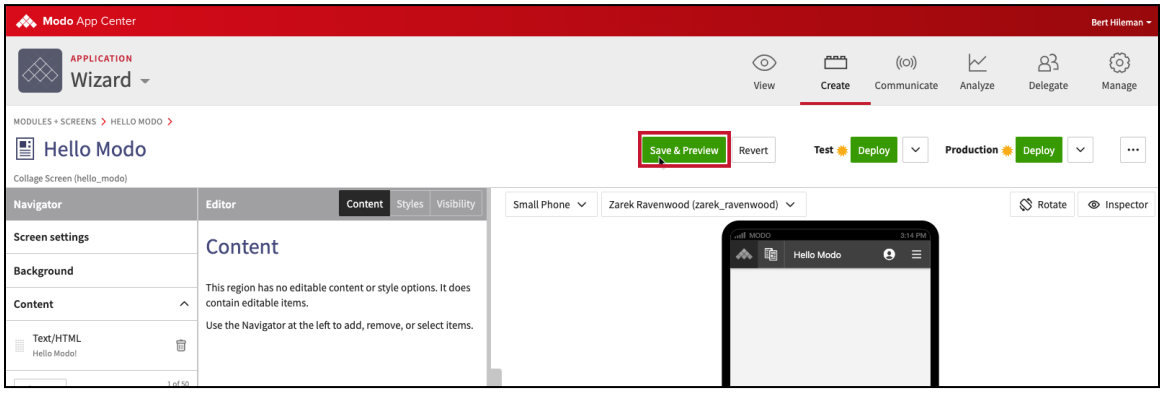
6. Scroll down and click **Text/HTML** to select it from the list of collage content blocks then click **Next**.



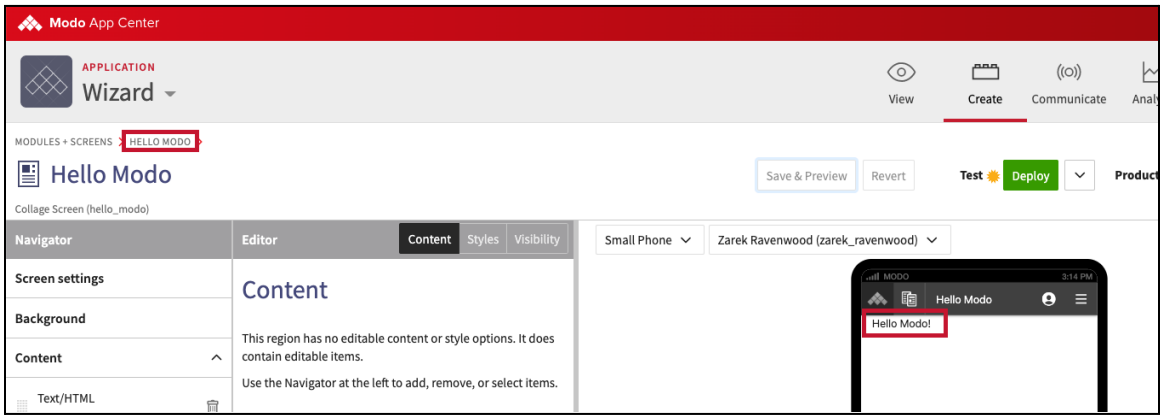
7. Type **Hello Modo!** for **Content** then click **Save**.



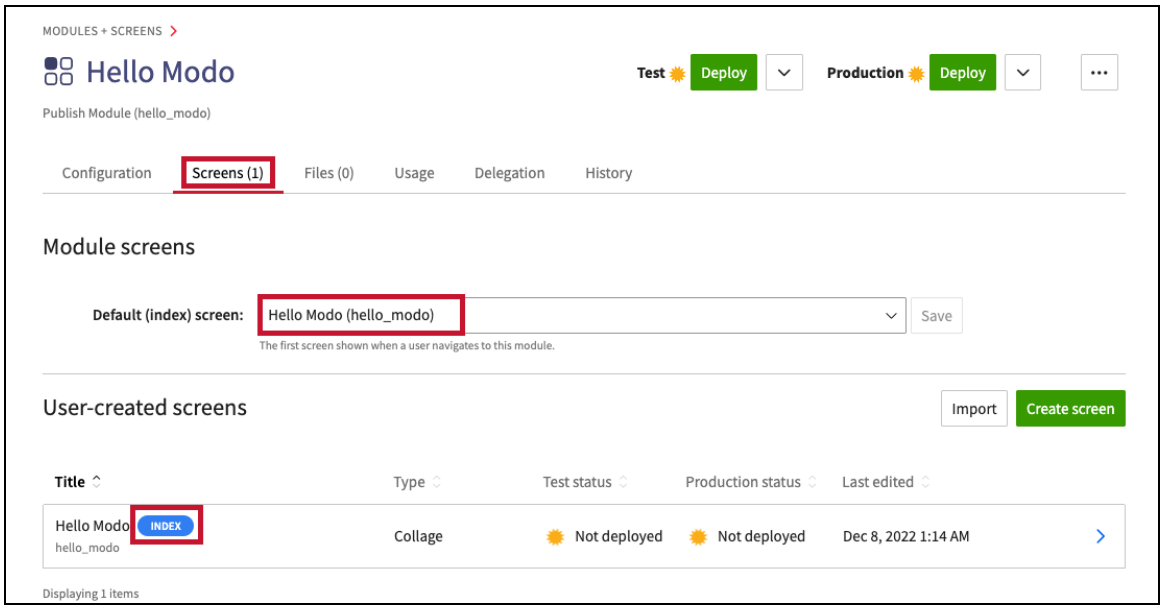
8. Click Save & Preview to view the changes you just made in the Screen builder preview window.



9. Verify that your text appears in the Screen builder preview then click the **Hello Modo** breadcrumb to return to the Hello Modo module you created.



10. Click **Screens** and make sure the **Hello Modo** screen is set as the **Default (index) screen**.



Congratulations! You can quickly create publish screens to show static content to your users. This low-code/no-code platform makes it easy for non-technical managers to provide content to your Modo Application.

Task 3: Add the Hello Modo Publish screen as the default home module

Context: Every persona needs a default screen that opens when it is selected. The application cannot be deployed without making a selection.

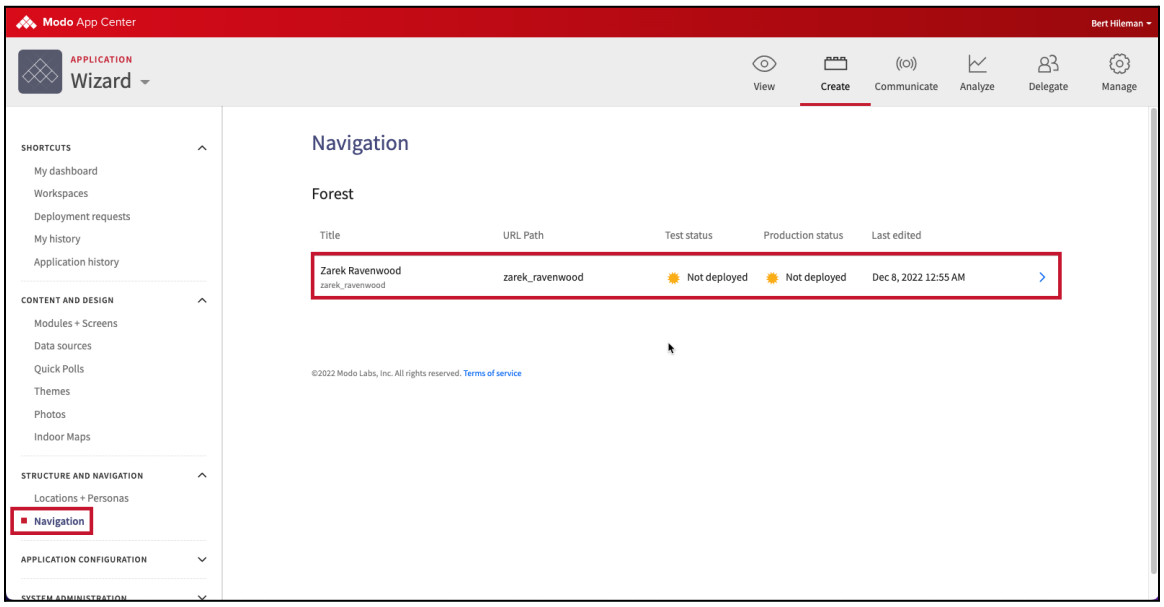
Goal: Set a default screen for the persona and deploy our changes to test.

Task Outline:

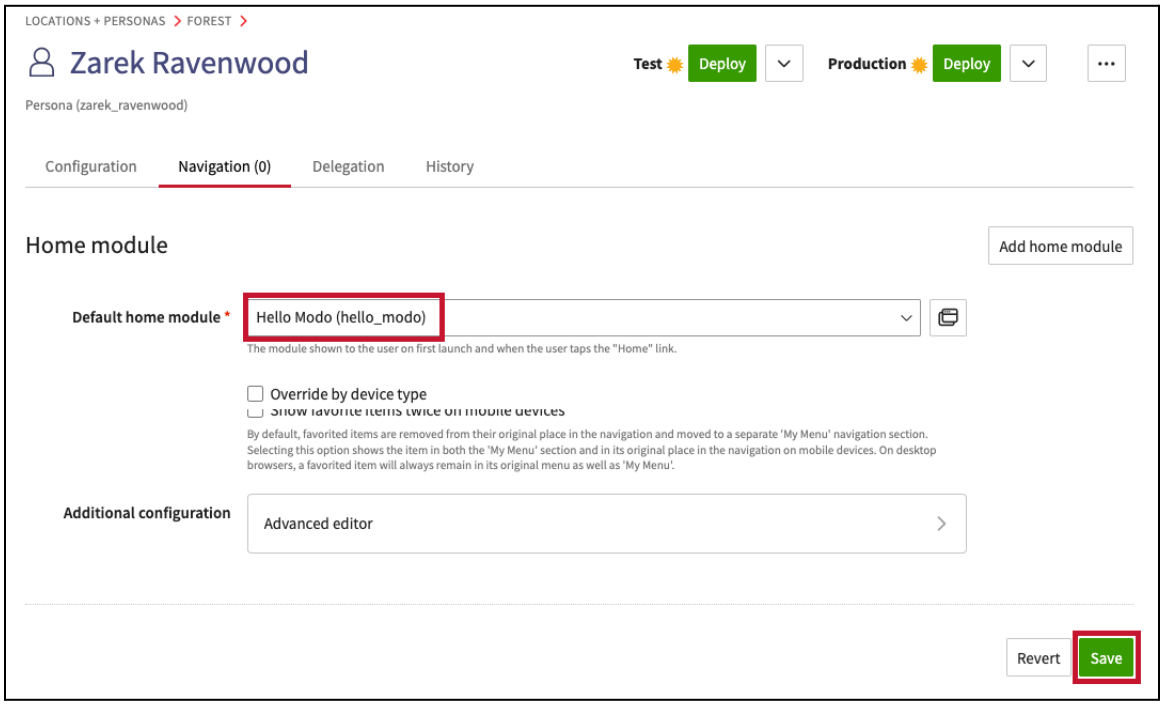
- Add the Hello Modo publish module as the default home screen for your persona

Steps:

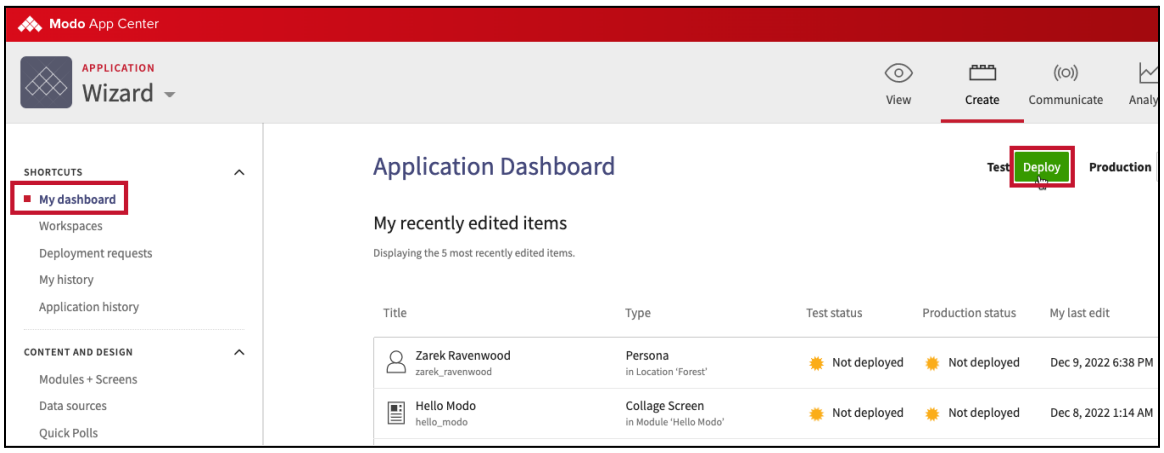
1. Under **Create**, click **Navigation** then click your character name to edit their navigation item.



2. Select **Hello Modo** for **Default home module** then scroll to the bottom of the page and click **Save**.



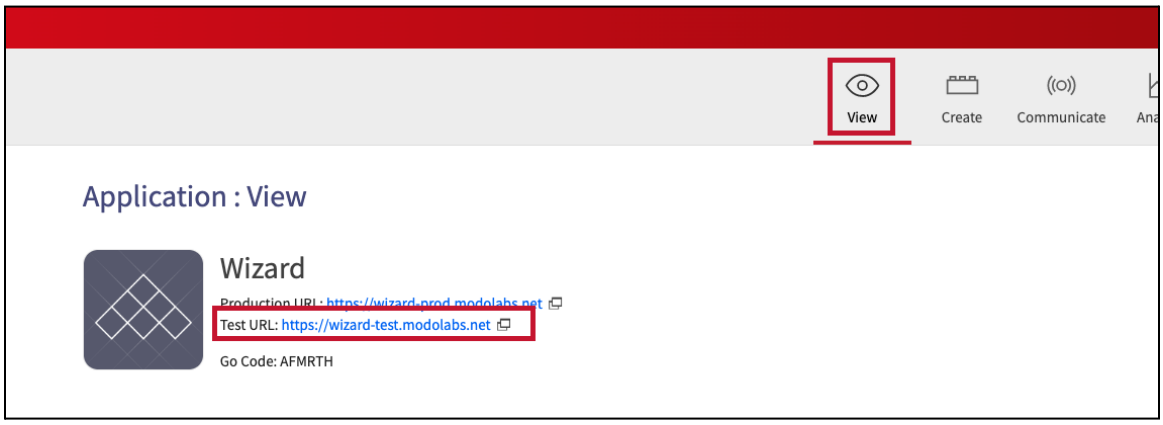
3. Click **My dashboard** then click **Deploy** next to **Test**.



4. Make sure all the boxes are checked then click **Deploy to Test**.

Once we have deployed everything, we are ready to preview the app!

5. Click **View** then click the **Test URL** link.



6. Verify that you see Hello Modo! on the page.

Congratulations! You have successfully added every element necessary to deploy a Modo Application and have something you can preview. Following the steps above provides an application that can be used for testing out XModules.

Task 4: Add a Theme

Context: Themes control the overall look and feel of your Modo application. Since we focused on XModule development we will ignore the theming process and just import a theme into our app to control the look and feel. Ideally the theme matches your customer's brand guidelines for logos and colors. Modo provides design guidelines to help you build your theme. Those guidelines can be found here: https://showcase.modolabs.com/showcase_2022/sc_design_tips/_/create_a_balanced_theme. Time with our design team is also available to help you drill down more deeply into controlling the look and feel of your Modo app.

For this lab, you have the opportunity to choose from the 3 themes below. Find your favorite theme and follow the steps below to add the theme to your application.

Goal: Import a theme to see how it affects the app and the way items are displayed.

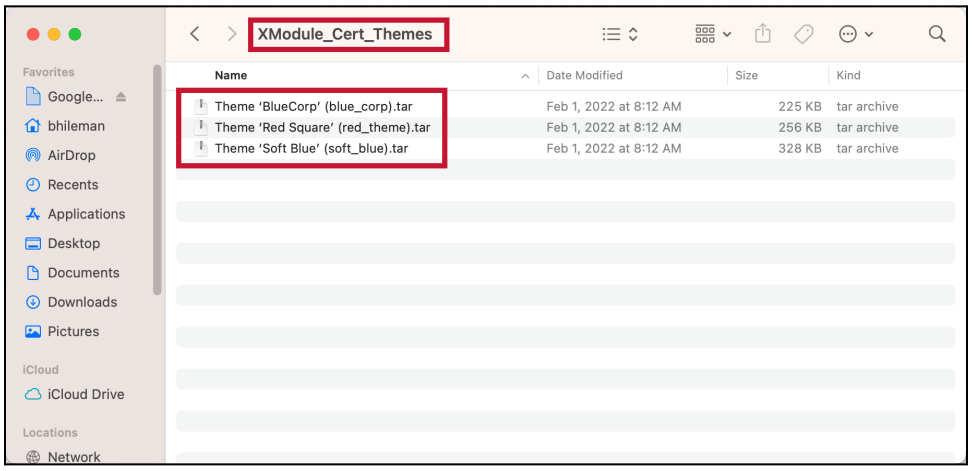
Note: We provide 4 themes for creating your Modo application. The use of a theme that is different from the theme we imported leads to differences in the screenshots used in this lab guide versus what you will see on your screens. Please focus on the content of what should be displayed instead of having a 1:1 match between our screenshots and what you see on your display.

Task Outline:

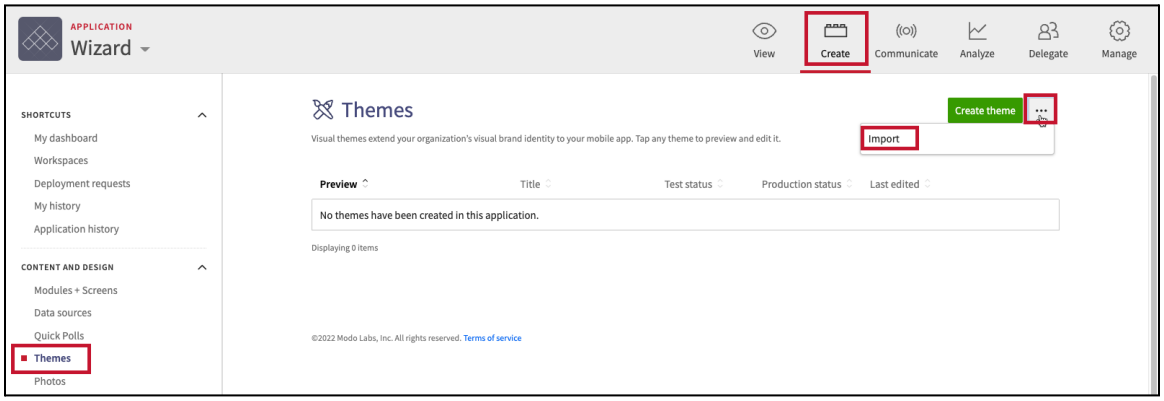
- Select and install one of the three themes from the following URL: https://static.modolabs.com/xmodule_certification/XModule_Cert_Themes.zip

Steps:

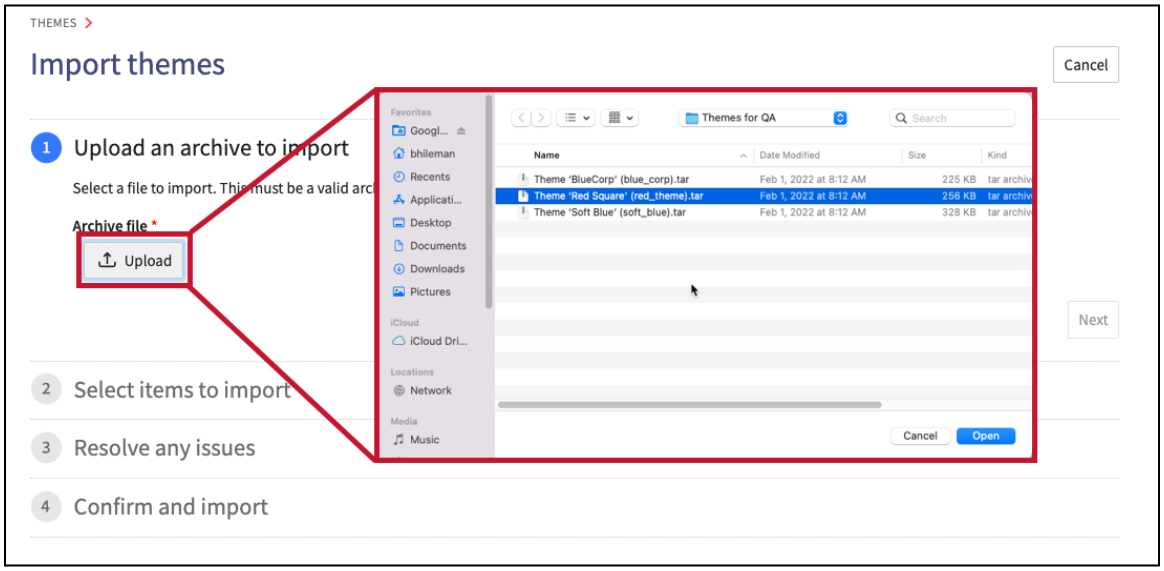
1. Click the following link to download a zip file containing 3 themes: https://static.modolabs.com/xmodule_certification/XModule_Cert_Themes.zip.
2. Extract the zip files to make the theme archive files accessible in your file explorer.



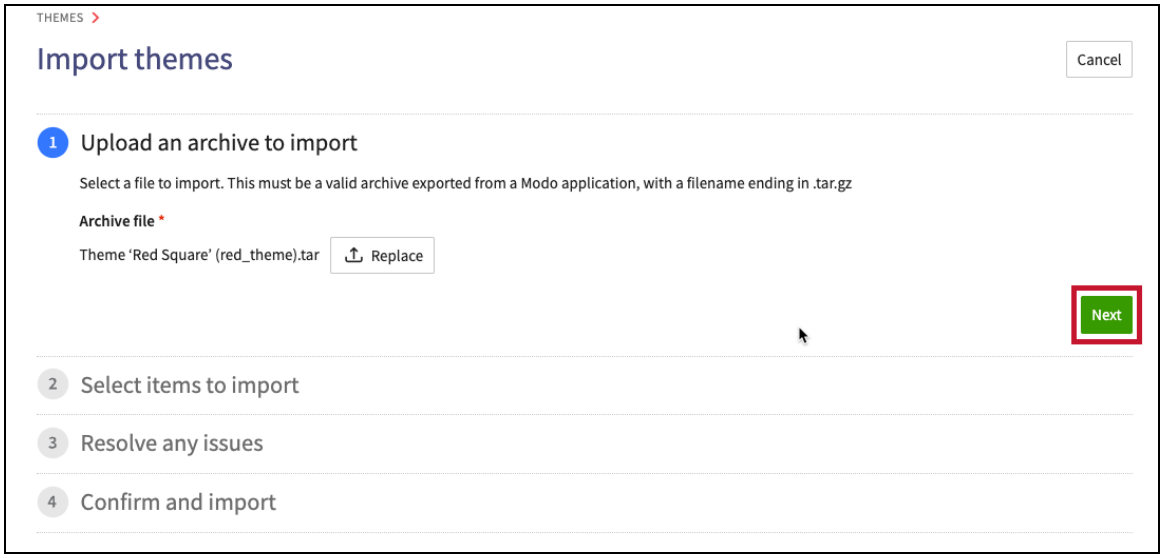
3. Under **Create**, click **Themes** > **…** > **Import** to open the theme importer.



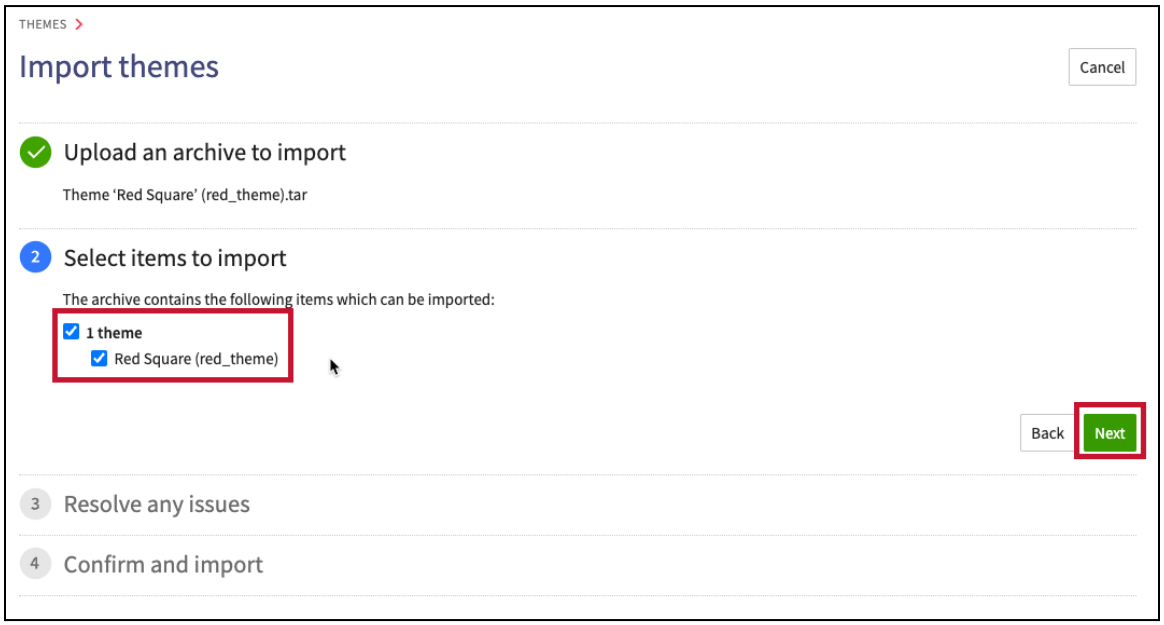
4. Click the **Upload** button then navigate to the Theme archive you want to import then Click **Open.**



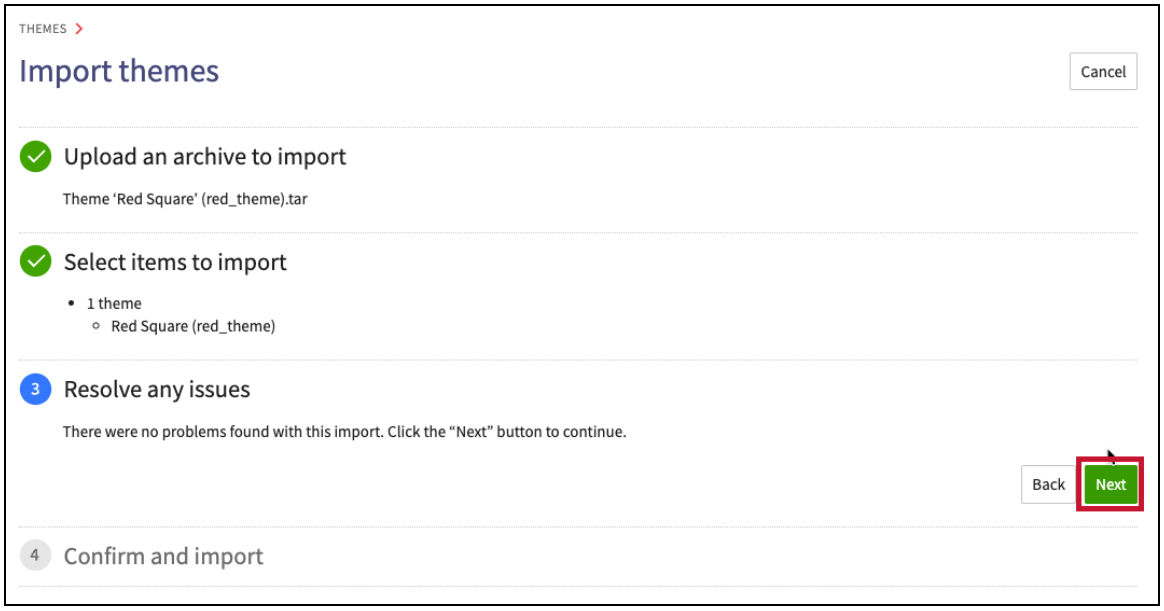
5. Click **Next** to start the upload process.



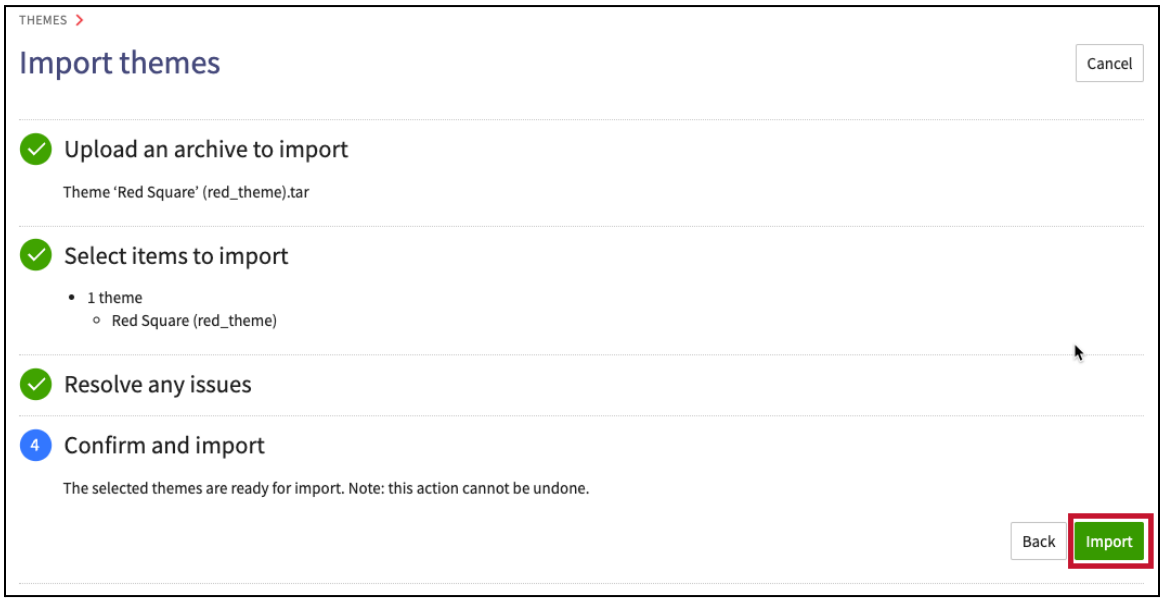
6. Verify the theme name then click **Next**.



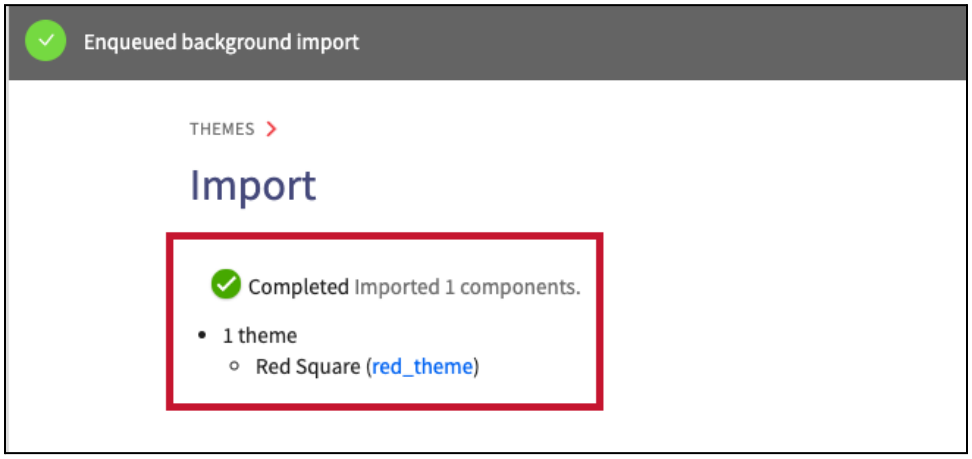
7. There should not be any issues unless you are uploading a theme a second time and already have a theme with that same identifier so click **Next**.



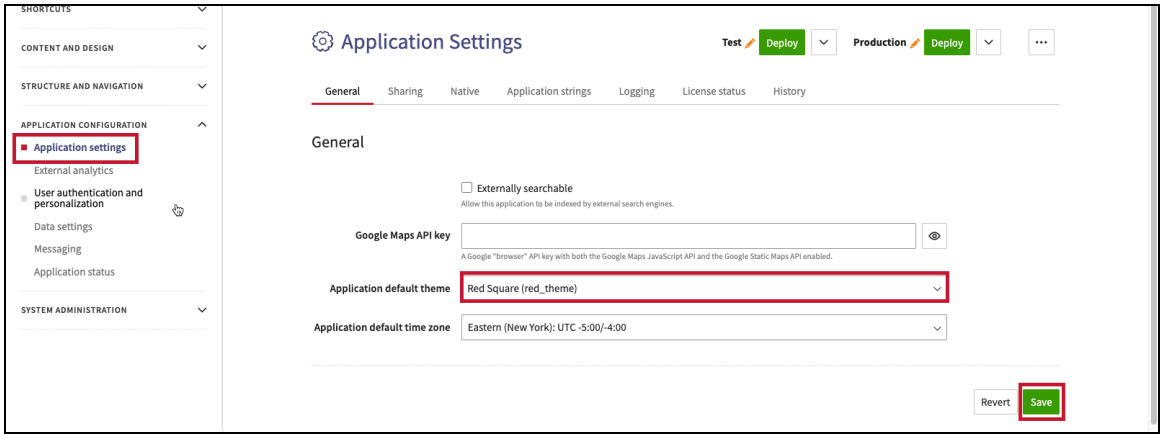
8. Click **Import**.



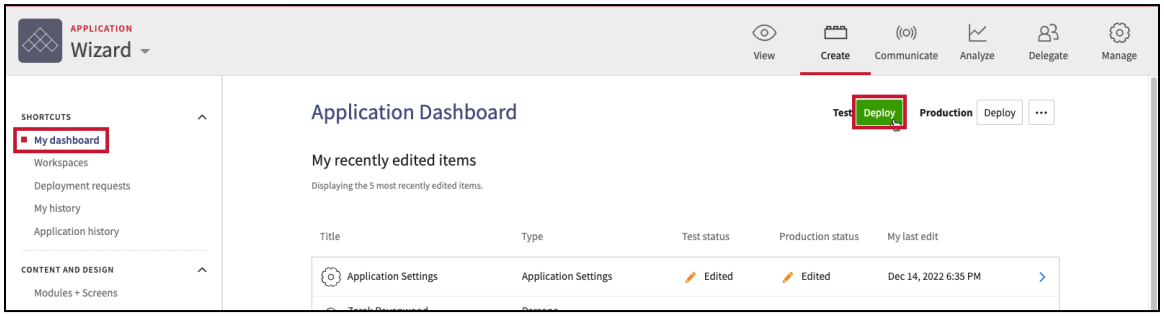
9. Verify that the theme has completed the import process.



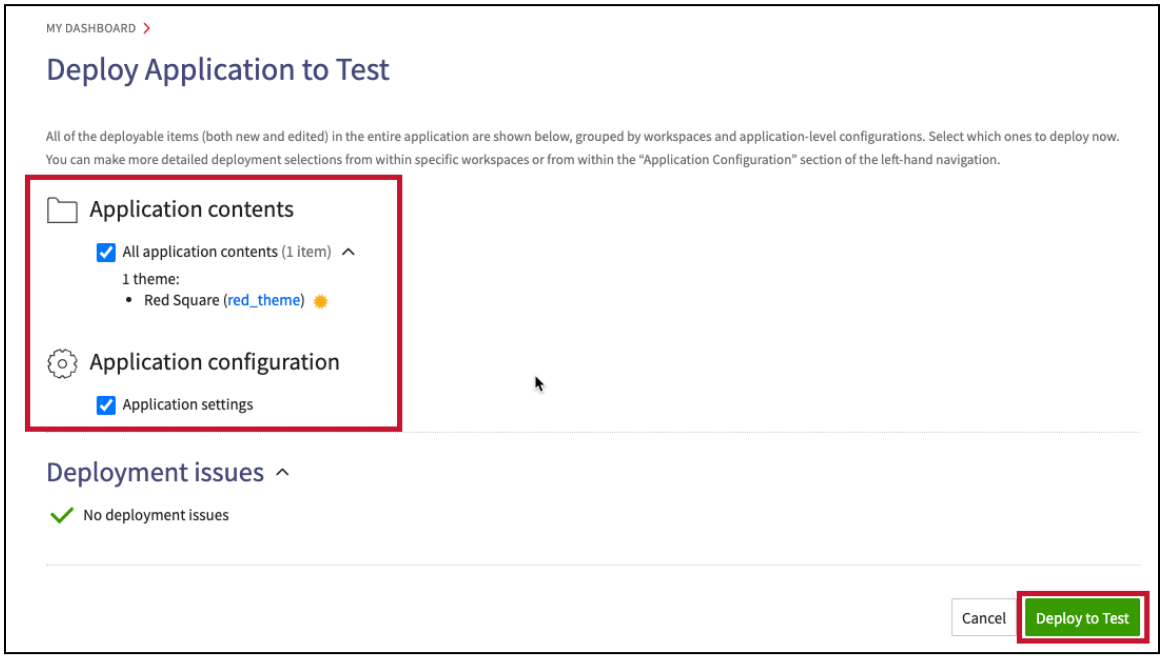
10. Under **Application Configuration**, click **Application settings** then select your theme for **Application default theme** and click **Save**.



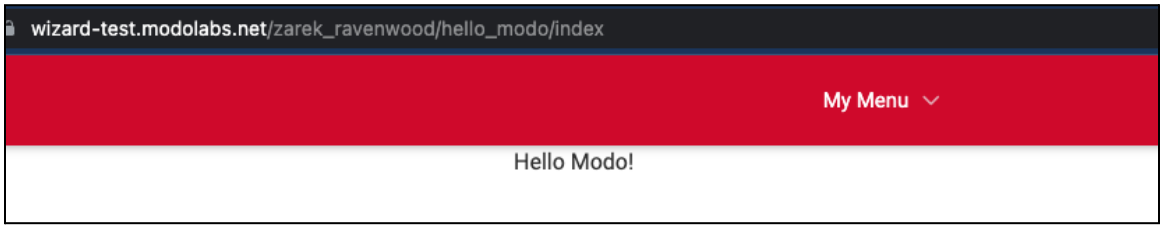
11. Click **My Dashboard** then click **Deploy** next to **Test**.



12. Verify that your theme is being deployed under **Application contents** then click **Deploy to Test**.



13. Refresh your tab that contains your Test view link for your **Hello Modo!** page with the new theme!



Congratulations! You have successfully completed all the tasks to view pages in an application and have added a theme to make your application look much better. All the pages you create from here automatically inherit the colors and fonts associated with your theme.

Lab 2 - Using XModule Resources and AWS

Estimated Time: 45 minutes

This lab provides additional basics for how to use XModule resources to find examples that you can start to tweak. We continue to improve XModule's capabilities and we update our documentation to provide as much flexibility in developing engaging experiences for your Modo app. This labs introduces you to some best practices for taking advantage of these resources.

Task 1: Using the XModule Sandbox to test elements

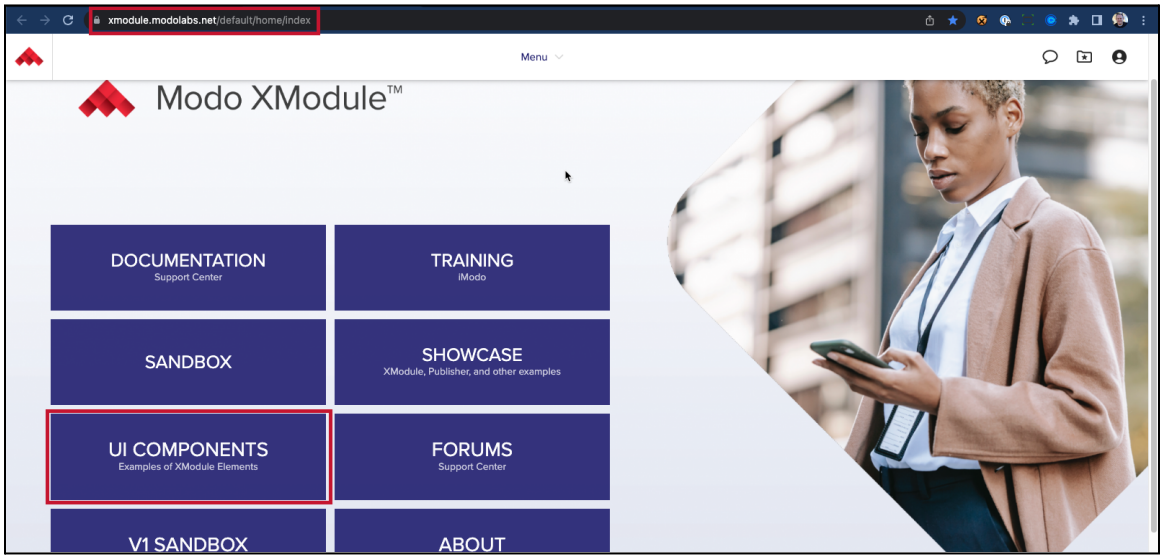
Context: The XModule Sandbox allows you to preview components and their properties before you add them to your project. This editor is much like W3Schools HTML and Javascript editors and other online coding preview platforms. The UI Elements pages are live implementations of the XModule elements in various formats to help you quickly test the code in the XModule Sandbox and then copy those elements into your project.

Task Outline:

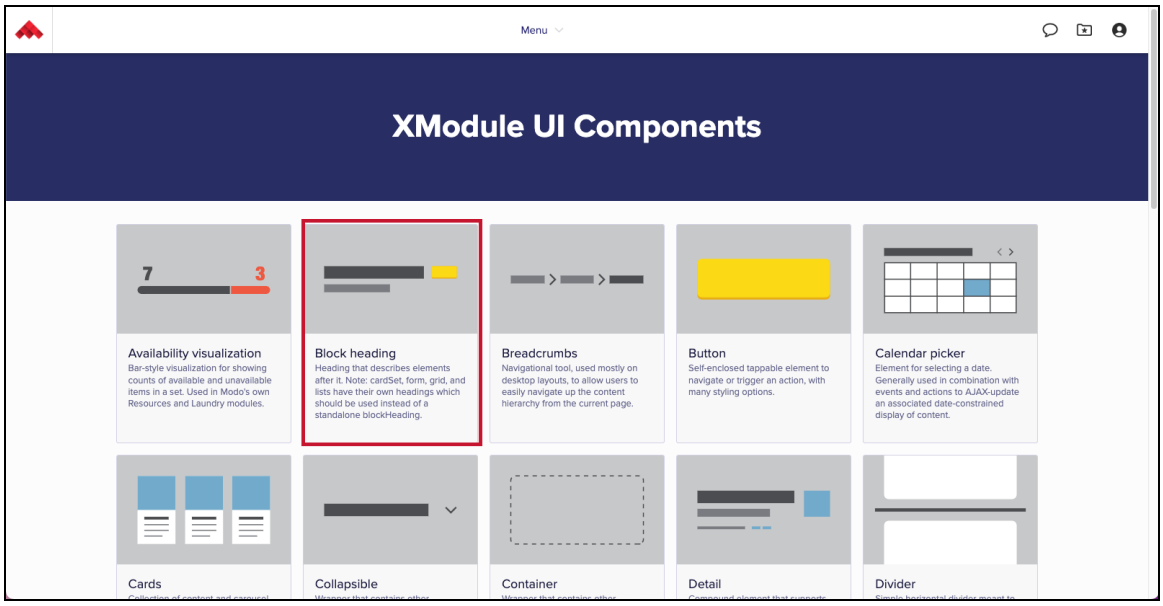
- Open the UI Component examples, select and isolate block heading example in the XModule Sandbox
- Extract a list item from the UI Component examples and add it to to the Sandbox with the block heading

Steps:

1. Click this link: https://xmodule.modolabs.net/ to open the XModule page then click UI Components.

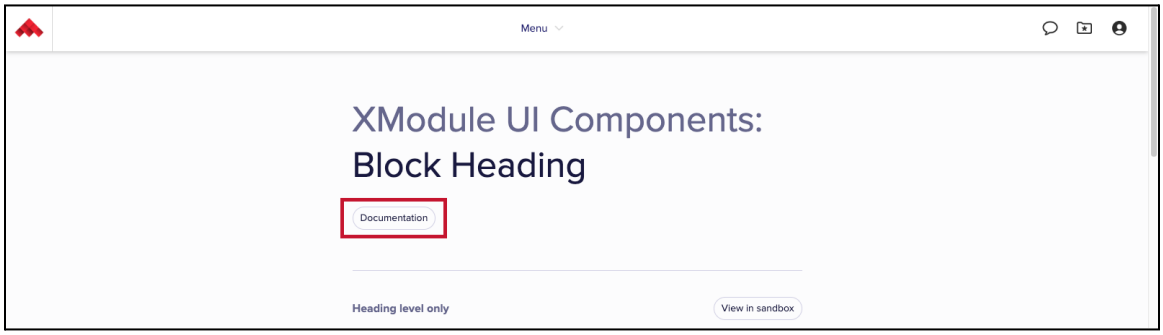


2. Click **Block heading** to load and view the Block heading examples.



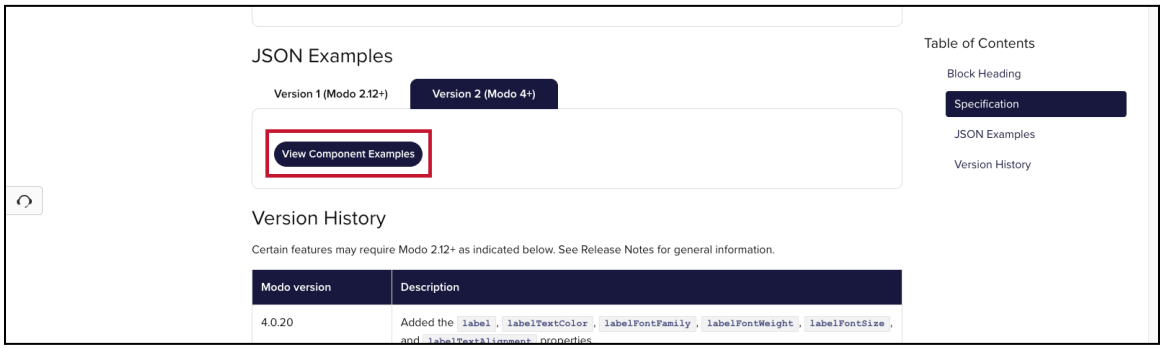
When you scroll the page, you can see different implementations of the block heading element.

3. Click the **Documentation** link at the top of the page to launch the support center document on the Block heading element.

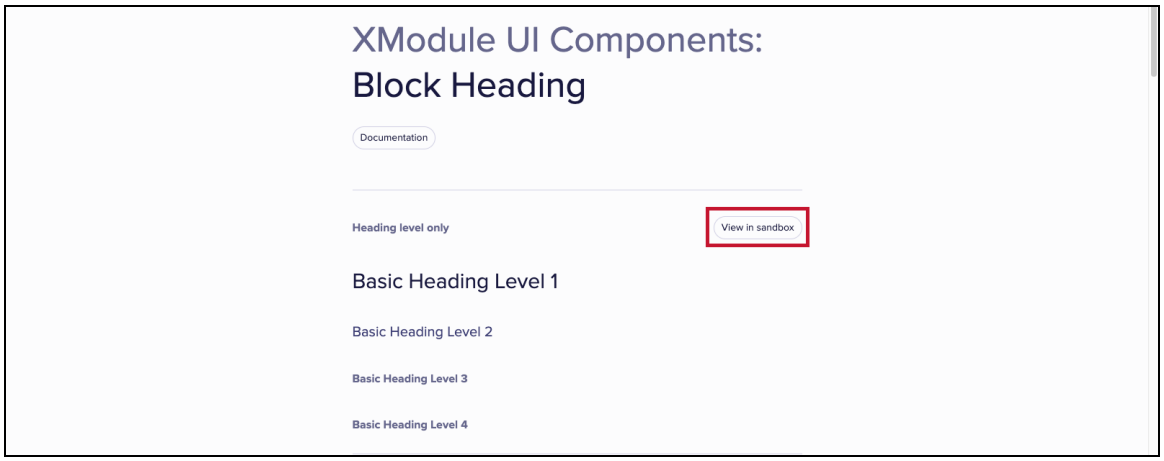


The documentation contains all the properties available for your element. While the examples show a lot of possibilities, we do not use every property.

4. Scroll down to the JSON Examples and notice that the Modo 4 examples takes you back to the Block heading component example.

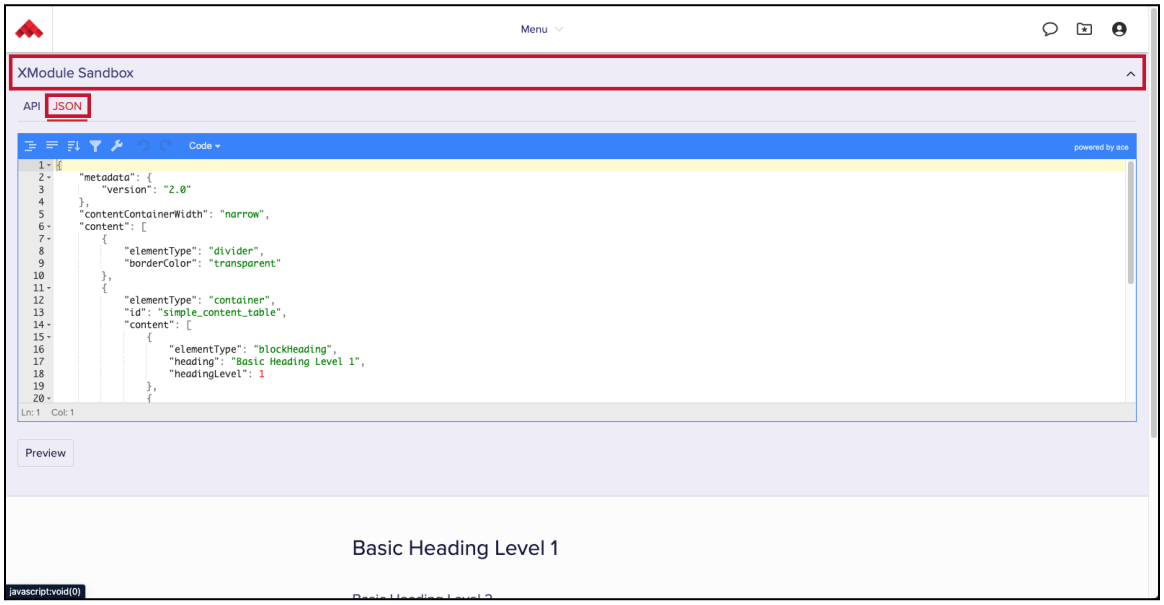


5. Return to the Block Heading UI Component examples page and click **View in sandbox** next to **Heading level only.**

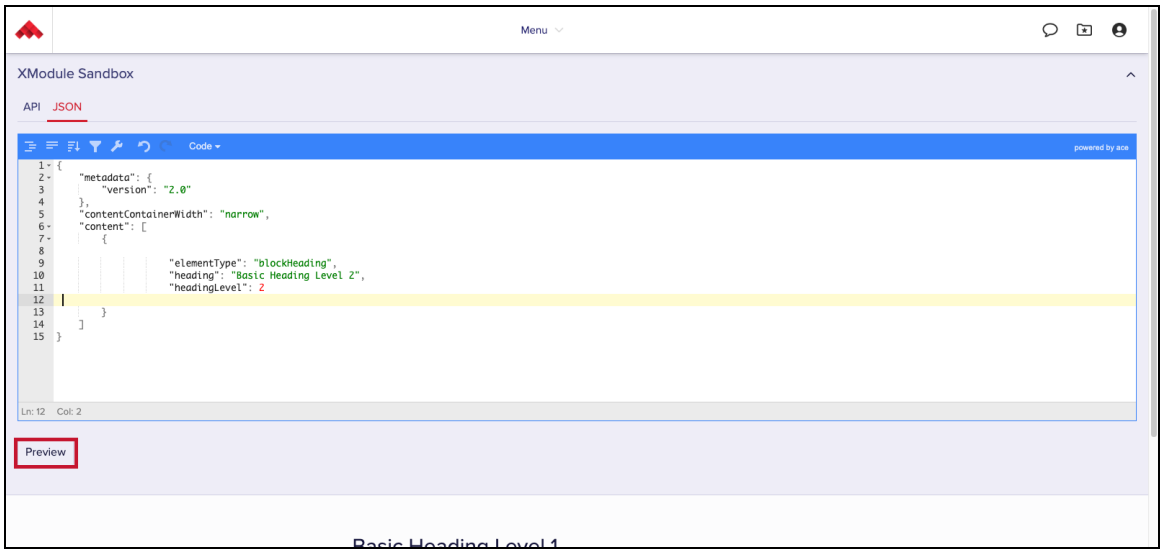


The Sandbox allows you to view the contents of JSON from either an API endpoint or JSON that you type into the JSON editor.

6. Click the XModule Sandbox collapsible section to open the JSON editor and click the JSON tab (if not already selected).

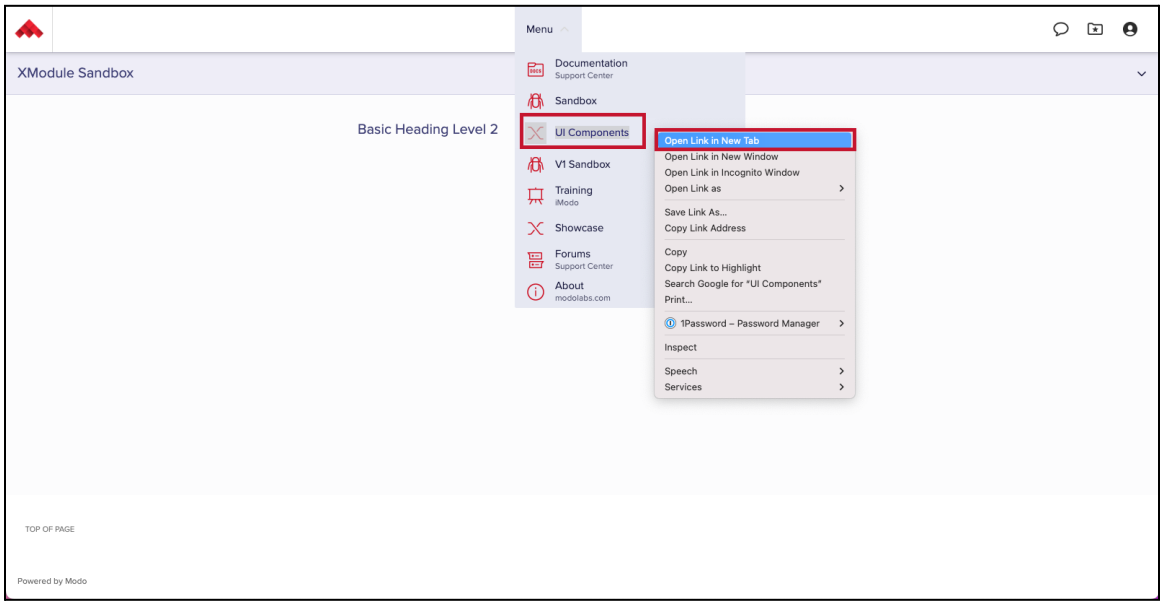


7. Isolate the XModule `blockHeading` element with the `heading` **Basic Heading Level 2** by selecting and deleting lines 8-20 and lines 12-23. Make sure to remove the comma on line 11 and then click **Preview**.

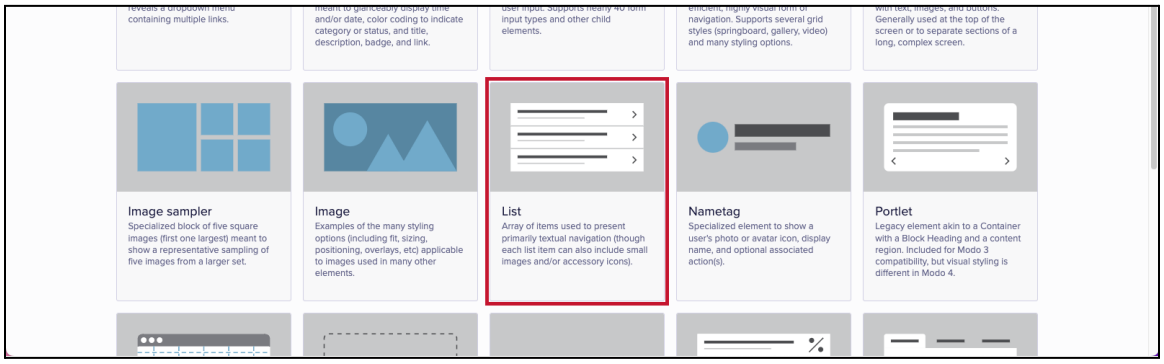


Note: Going forward, this process will be referred to as **Isolating the element.** It is the process of removing all other JSON from the UI element editor that is unneeded to build entirely valid XModule JSON that can be rendered properly. When asked to isolate the element in future tasks we will refer to some kind of unique identifier (string value specific to the element or element starting line number) to help you know which element you need to isolate.

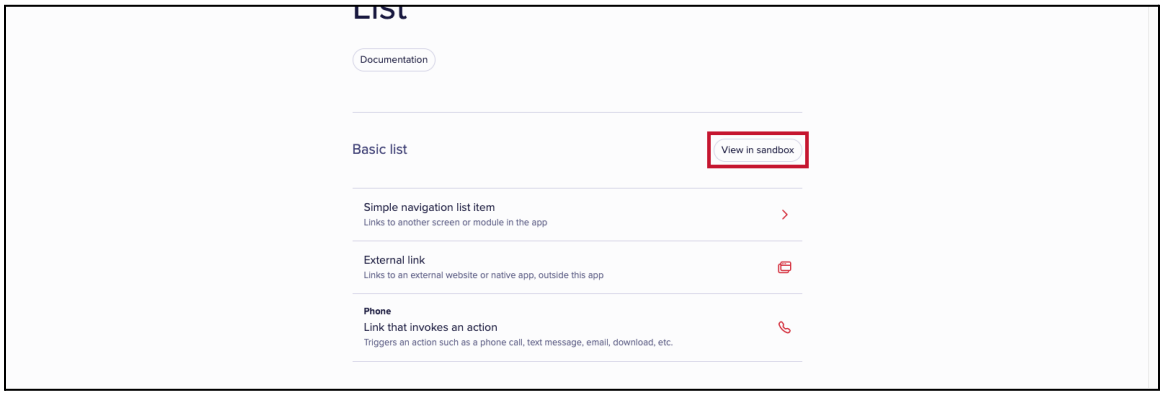
8. Click the **Menu** dropdown and open **UI Components** in a new tab or window.



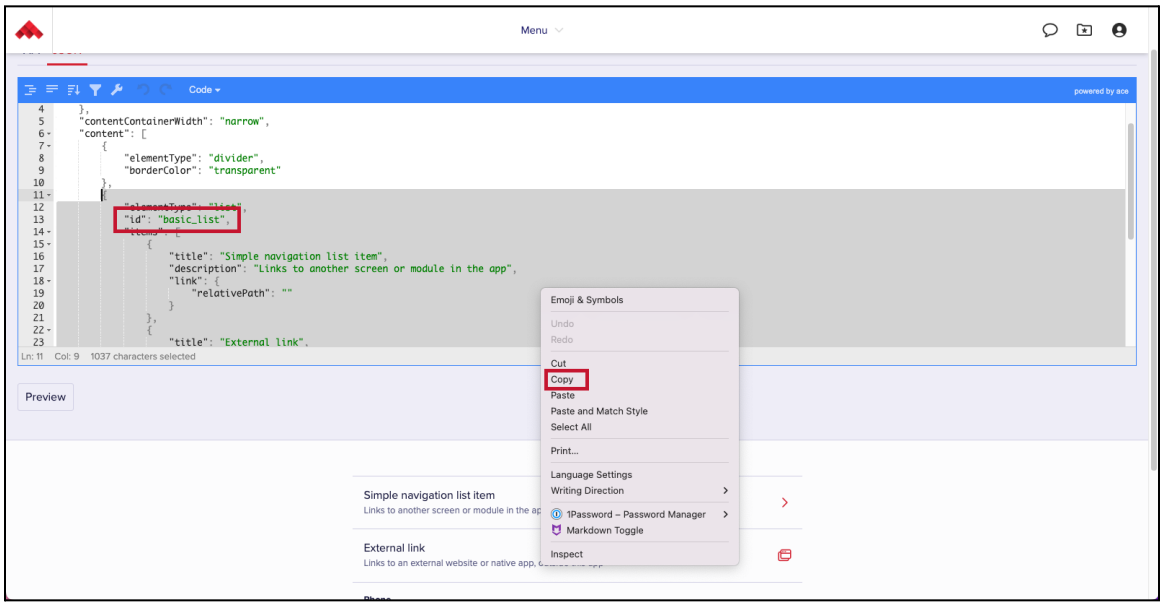
9. In the new window/tab, scroll down and click **List** from the UI elements.



10. After previewing the different styling options for the list items, click **View in sandbox** for the Basic list.



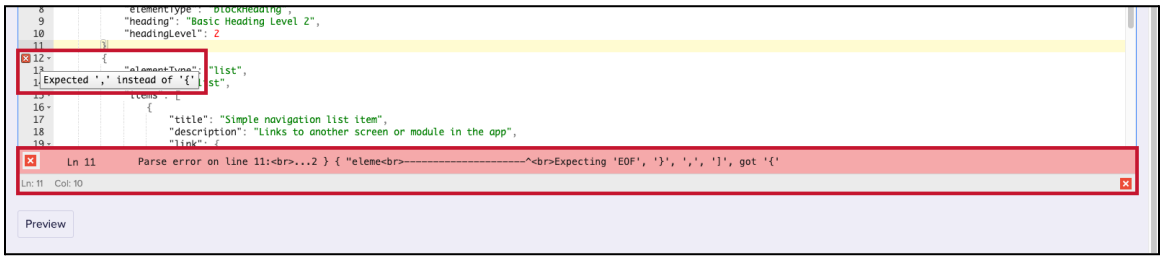
11. Click the XModule Sandbox collapsible section to open the JSON editor and click the JSON tab (if not already selected), extract (select and copy) the JSON list item with an id of `basic_list`.



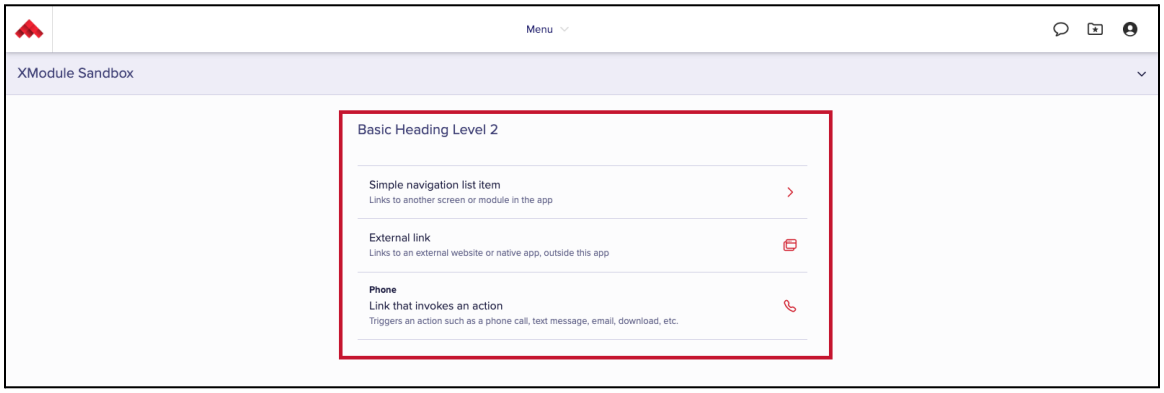
For the purposes of this training, **extracting an element** is the process of selecting one element from a JSON block and copying it to your clipboard. This can be done by clicking the disclosure widget for the desired element to collapse it, selecting the resulting line including the opening and closing braces and adding the item to your clipboard. If available, a distinguishing characteristic of the element will be used to identify the element or the element's starting line number will be used. Extracted elements must be placed inside a valid container item as part of an array and will not render properly in the XModule Sandbox without additional JSON.

12. Return to the Sandbox screen that contains your blockHeading element, expand the XModule Sandbox collapsible to view the JSON content, type a comma on line 11, paste the extracted element into the JSON editor, then click Preview.

Note: If you see an error icon while editing using the XModule you have an error in your JSON formatting. This must be corrected before you press Preview or your changes will be discarded and the last known functional JSON will remain in the JSON editor. If you click the error icon on the bottom of the page you will see a summary of the errors and if you hover over a line item with an error you will see more data about the specific error encountered.



13. View the resulting JSON and make sure it looks like the image below.



Congratulations! You have successfully used the UI Component examples to create your first XModule example with the XModule Sandbox. The XModule Sandbox gives you the opportunity to build wireframes to test out your XModule visualizations without needing to create an API endpoint and have your JSON hosted.

Task 2: Hosting XModule JSON in npoint.io

Context: The XModule Sandbox does not likely use the same theme that your customer application uses. If you want to know what your JSON looks like in your application with your theme, you need to host the JSON in

a URL-based location. There are many services available but my favorite is npoint.io. It is an open source and free to use service that supports both public and private JSON bins. This can be extremely useful for providing designers a tool for building wireframes quickly.

Goal: Host some XModule Code in and external JSON bin tool and create an XModule to show the JSON in your application.

Task Outline:

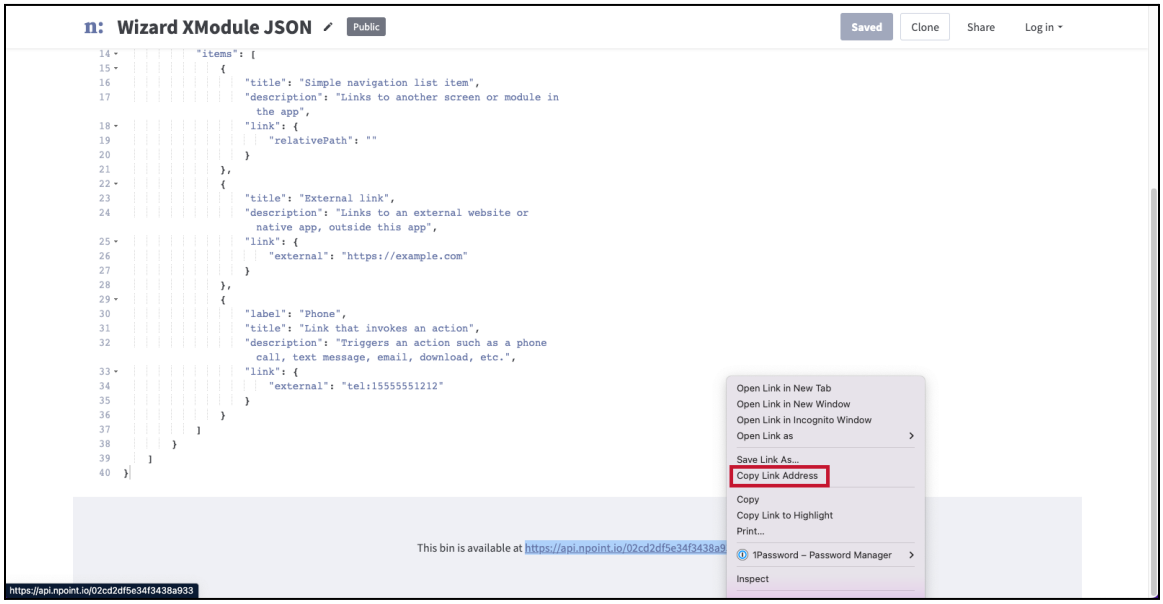
- Use npoint.io to host the JSON you just created in the previous task
- Create, deploy, and preview an XModule named **Lab 2: Task 2**
- Modify the header and preview it to make sure you are loading content from the npoint.io bin

Steps:

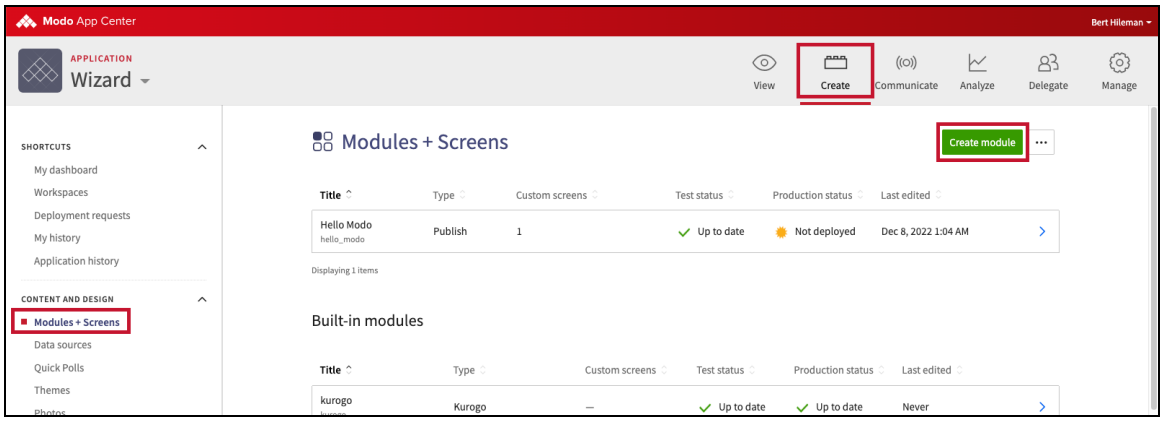
1. Navigate to https://npoint.io/ and click Create JSON Bin.

2. Copy and paste the contents from your XModule Sandbox (the `blockHeading` and `list` elements) into your npoint bin rename your JSON bin click **Save**.

3. Scroll to the bottom of the screen and copy the API url.



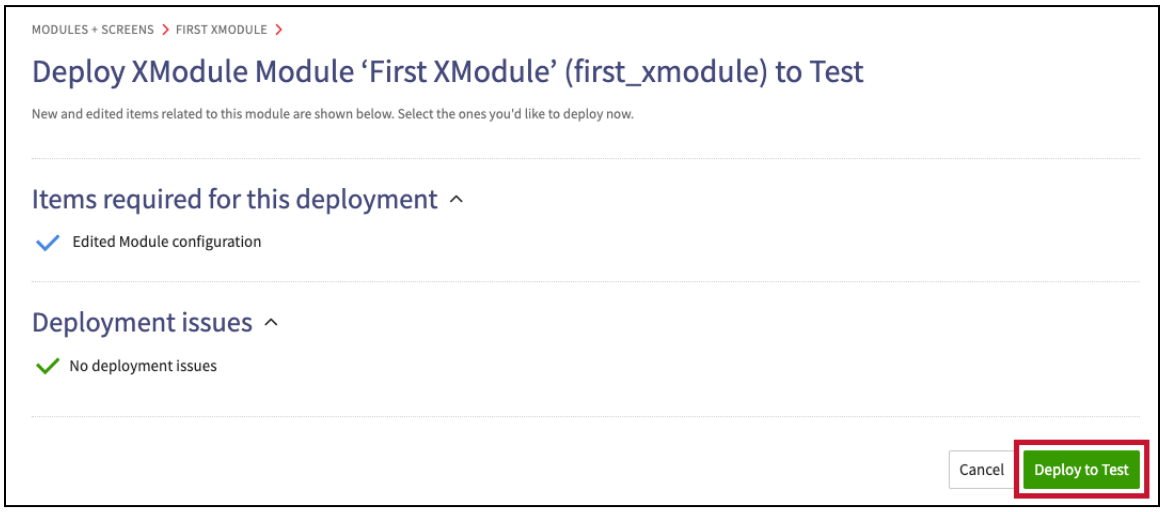
4. In your Modo App under Create, click Modules + Screens > Create module.



5. Complete the form with New Module form with the following information:
    a. Select **XModule** for **Module type**.
    b. Type **Lab 2: Task 2** for **Display name**.
    c. Verify that the path auto completed to **lab_2_task_2**.
    d. Paste the contents of your clipboard for **XModule API URL**.
    e. Click **Create module**.

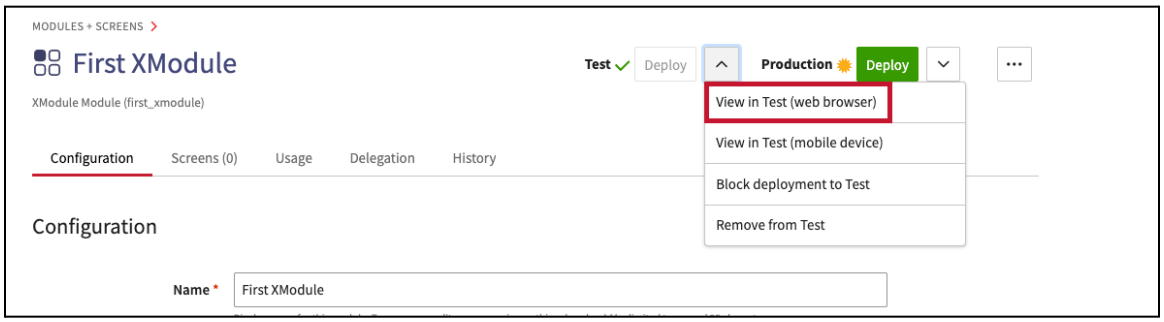
6. While on your **Lab 2: Task 2** module, click **Deploy** next to **Test.**

7. Click **Deploy to Test** on the confirmation page.



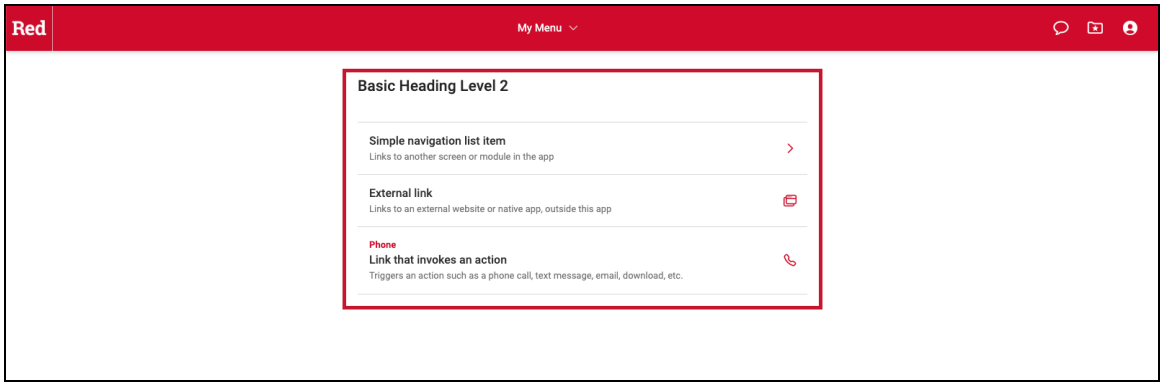
Note: You can deploy each content item as you edit it and then you can quickly test the item. When instructed to **Deploy to test** you will be expected to **Deploy** the items to test from either the Module, the configuration section, the workspace, or the **My dashboard** level without accompanying screenshots. When in doubt, deploying from the My dashboard level can deploy all pending changes to the environment selected.

8. Once the module is deployed (you can tell because the Deploy button is disabled on the module, click the dropdown next to the Test environment and click **View in Test (web browser)**.



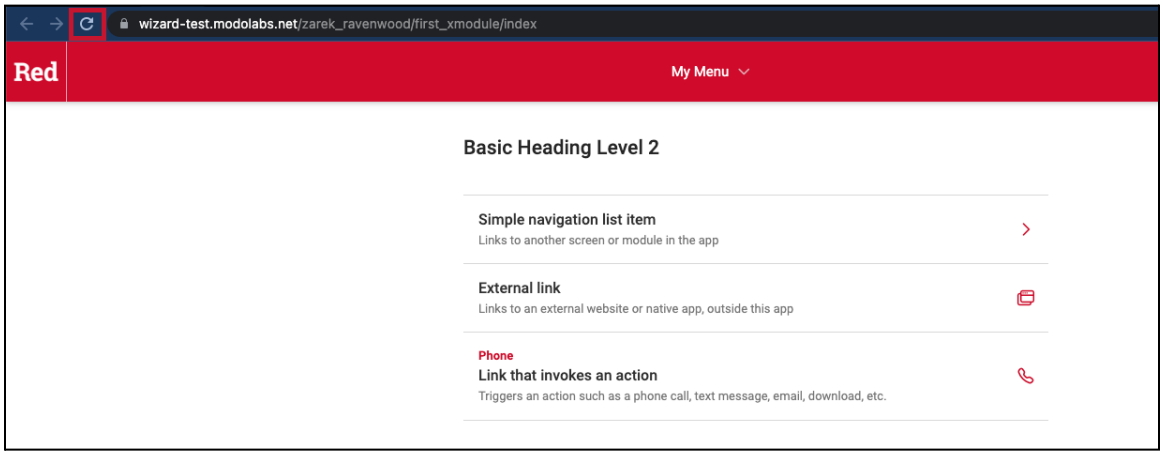
Note: Going forward you will be asked to Deploy and then Preview the item. This includes the **Deployment** process and the **View in {environment} (web browser)** process.

9. You should see the same content you had in the Sandbox but now it has your application's theme applied.

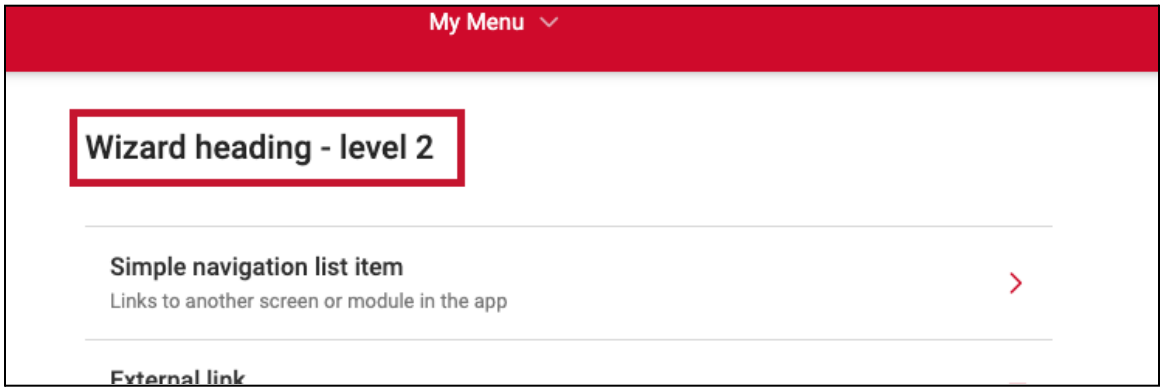


10. In your npoint.io bin, change the value for the `heading` property of `blockHeading` element then press **Save** (should be close to line 9).

11. Return to your Modo application's **Lab 2: Task 2** page and click refresh.



12. Verify the page has picked up the new title you specified in your npoint.io JSON bin.



Congratulations! You have successfully hosted some JSON and viewed the page in the Modo application. Any changes you make to the npoint.io bin are automatically available in your Modo application once the page

is reloaded. This occurs for users any time they come back to the requested page inside the application. Please remember that cache settings may affect the timeliness of the returned content.

Task 3: Accessory Icons, Accessory Buttons, and (we don't need no stinkin') Badges

Context: Now that you have a `listItem` element on our page we should explore some of the ways we can manipulate the items to take advantage of some advanced functionality. There are little things that can be done that enhance the user experience beyond what is typically available.

Assumptions: You still have your npoint.io bin open where you are editing your list items.

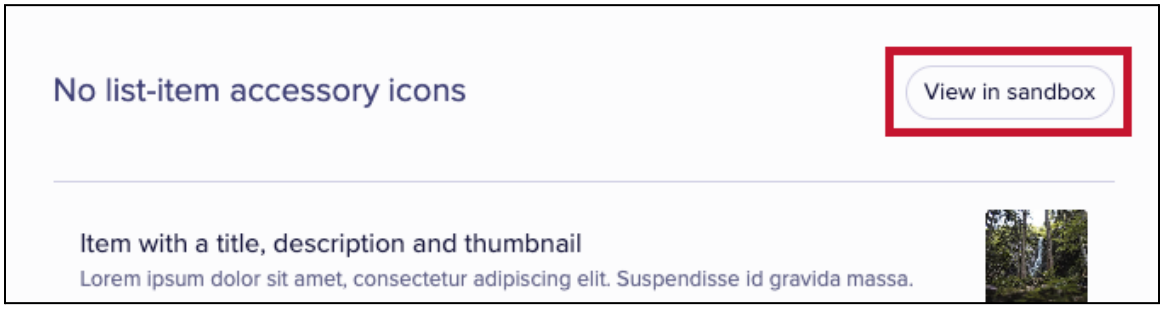
Goal: Add badging and accessory buttons to your list items.

Task Outline:

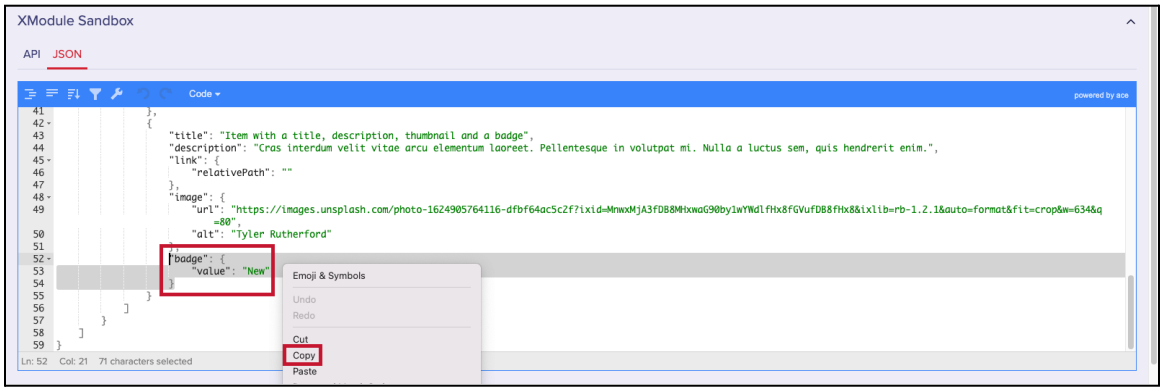
- Add a badge element to one list item in your npoint.io bin JSON
- Add the List's Accessory Button items to your existing list in npoint.io

Steps:

1. Click the following link to open the XModule UI Components to the List element. https://xmodule.modolabs.net/default/xmodule_ui_examples/page?relativePath=.%2Flist.json
2. Scroll to the No list-item accessory icons element and click View in Sandbox.



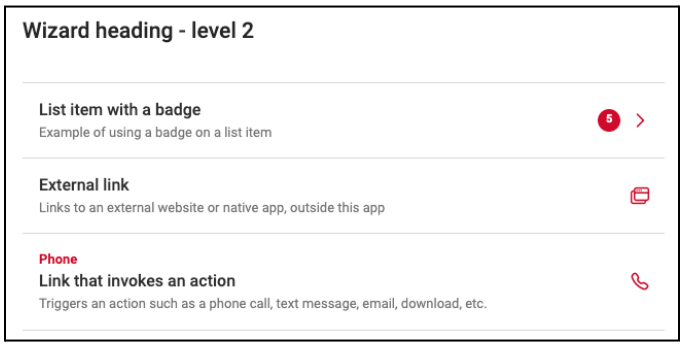
3. Expand the XModule Sandbox, click the JSON tab and extract the `badge` element on the page (lines 52-54).



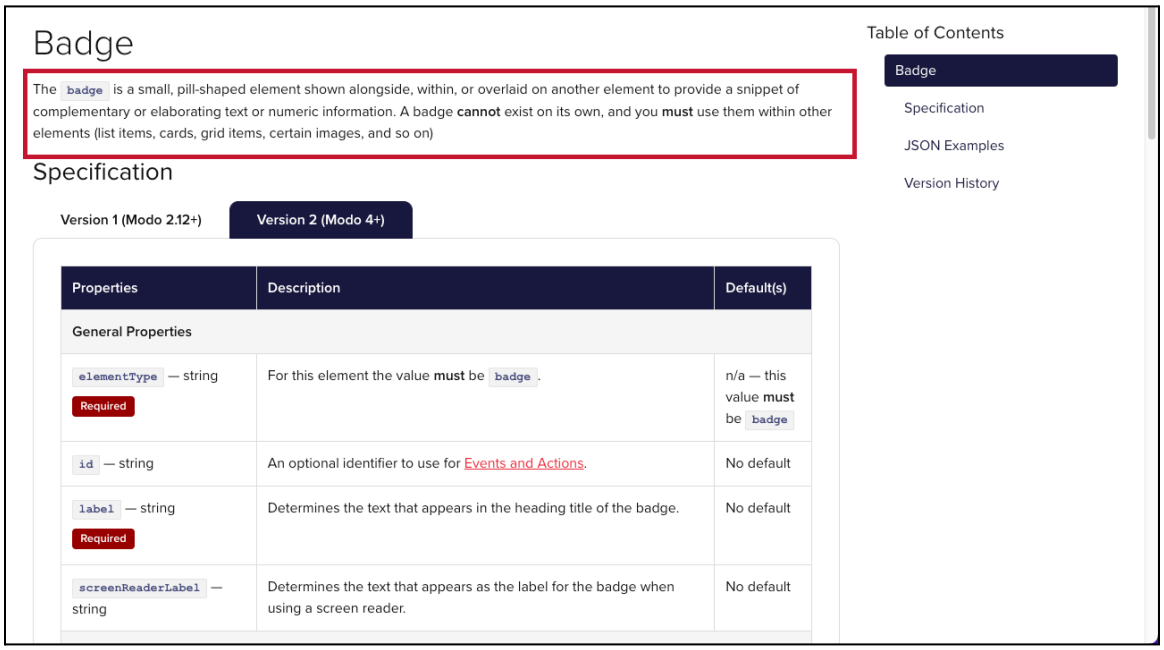
4. In your npoint.io JSON bin, change the first item to match the following:
    a. Type **List item with a badge** for **title**
    b. Type **Example of using a badge on a list item** for **description**
    c. After the link / URL object's closing brace, add a comma and paste in the `badge` element you extracted

d. Change the `badge` element's value to **5**, then press **Save**.

5. Refresh the **Lab 2: Task 2** page to view the updated badging information for your list item.

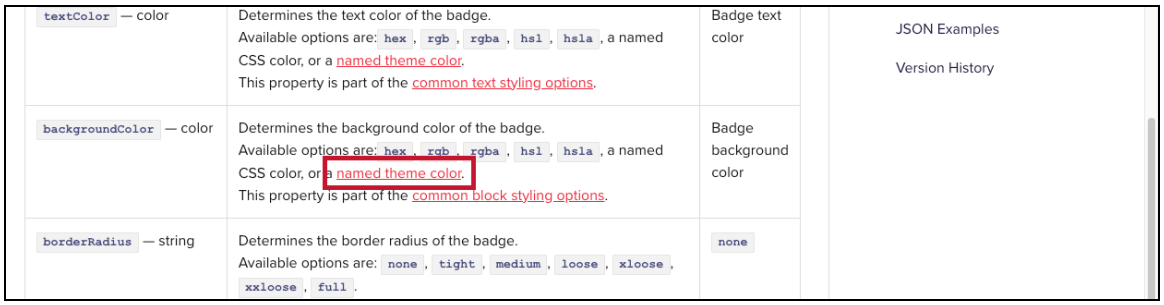


6. Click the following link to view the Badge documentation and read the description section: https://support.modolabs.com/support/solutions/articles/13000087932-badge



Note: The badge provides a way to draw users attention to actions.

7. Scroll down the documentation to read the `backgroundColor` property for the `badge` element then click the **named theme color** link in the description column.



8. Review the description on the **Supported Named Color Values** page then find and copy **secondary_text_color** to your clipboard.

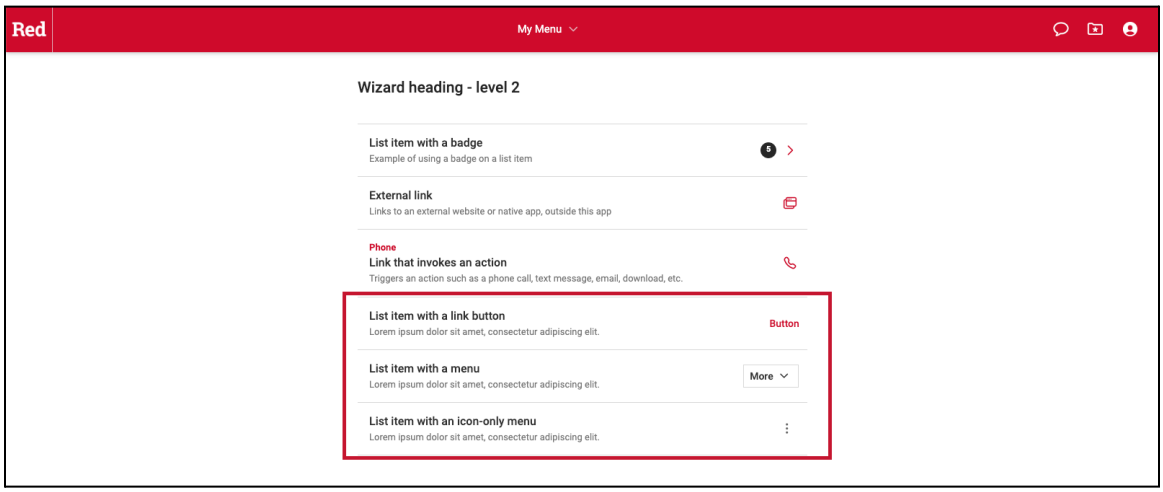
9. In your npoint.io bin add a `backgroundColor` property the value `theme:secondary_text_color` as the value the following line for your `badge` element, then press **Save**.

10. Refresh your page to review the updated `badge` element background color.

11. Return to the UI Components List examples, find **Accessory Buttons** from the list and click **View in sandbox**.

12. Extract all the `listItem` elements from the items array and add them to your list's items array in your npoint.io JSON bin and click **Save**.

13. Refresh your **Lab 2: Task 2** preview page to see the changes.



14. Return to your npoint.io JSON bin and use the disclosure widgets to collapse the first three items in your list's `items` array. Let's examine the difference between items 4-6 in the list. Edit the fourth item in the list to have the following format:

a. After the line `"elementType": "linkButton"` (approximately line 46), either type or copy and paste the following:

```
                "url": {
                    "external": "https://modolabs.com/"
                },
```

b. For the link object around line 52, change the link information to the following:

```
            "link": {
                "external": "https://google.com/"
             }
```

c. Ensure your final JSON looks like the snippet below.

d. Save your changes, refresh your **Lab 2: Task 2** XModule review page and click the list item you just edited and then click the Button. Adding the accessory button allows you to have multiple actions in the same object.

15. Review the 5th item in the list. The accessoryButton object can also use `dropdown` as the `elementType` so you can add more options to the list's actions.

16. The 6th item in our list object provides an alternative way to approach the dropdown and provides an `accessoryIcon` instead of a dropdown `title`. When omitting a title it is important to provide a `screenReaderTitle` to maintain your apps accessibility compliance.

Optional: Try adding list items with external link types of sms, tel, and mailto and see what happens with the Accessory Icons

Congratulations! Using tools like the XModule UI Component Examples, the XModule Sandbox, and npoint.io allow you to quickly and easily test XModule elements and their visual display properties. These items can be critical tools in your arsenal for rapid prototyping and development.

Lab 3 - JWTs, Authentication, and the Modo Communicate Directory

Estimated Time: 2 hours

Modo uses JWTs to send data from the Modo application server to your web service as a secure means of communication. The JWT payload contains information about your Modo Application as well as the authenticated user. These values are critical in creating hyper-personalized experiences.

Additionally, any information provided from your IdP can be automatically stored in your Modo Communicate Directory. Since we are going to be configuring and editing an IdP, it makes sense to include that discussion in this

Task 1: Configuring an AWS Lambda Function and your Modo App to show HTTP Request header values

Context: While npoint.io is a quick and easy way to host JSON bins it does not provide the ability to process the HTTP request or look at the HTTP method. Modo has an AWS environment created specifically for use in training customers on XModule concepts. This shared environment requires us to follow very specific guidelines and be careful with our naming conventions so we do not impact anyone else's functionality. Please take extra care to follow the written instructions and do NOT use the names provided in the screenshots.

Goal: Create a Lambda function with a function URL and view the header values in the first tab of a three tabbed layout.

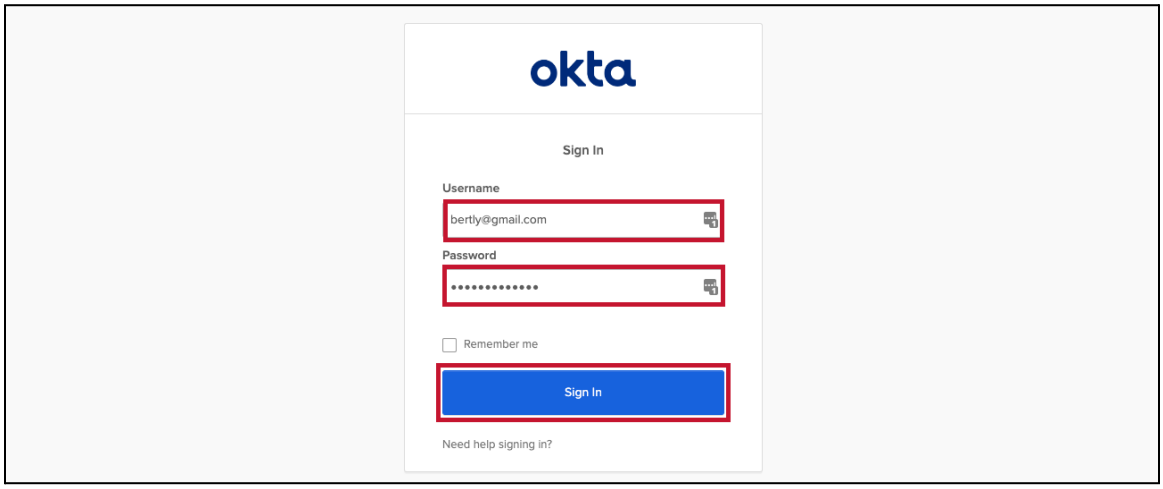
Task Outline:

- Create a Lambda function named **{xmod-prefix}-lab3-task1** and uses a function URL

- Log the HTTP Request information (event) and return XModule JSON from the function that contains 3 tabs with no content
- Create an XModule named **Lab 3: Task 1** that uses your function URL
- Check CloudWatch for the HTTP headers that are part of the event
- Change the **title** of the first tab to HTTP Headers and add all of the HTTP headers as XModule list items and add the list to the first tab
- Change your **Lab 3: Task 1** XModule to use **Application Key** for **JWT Key** and deploy those changes and preview the results.

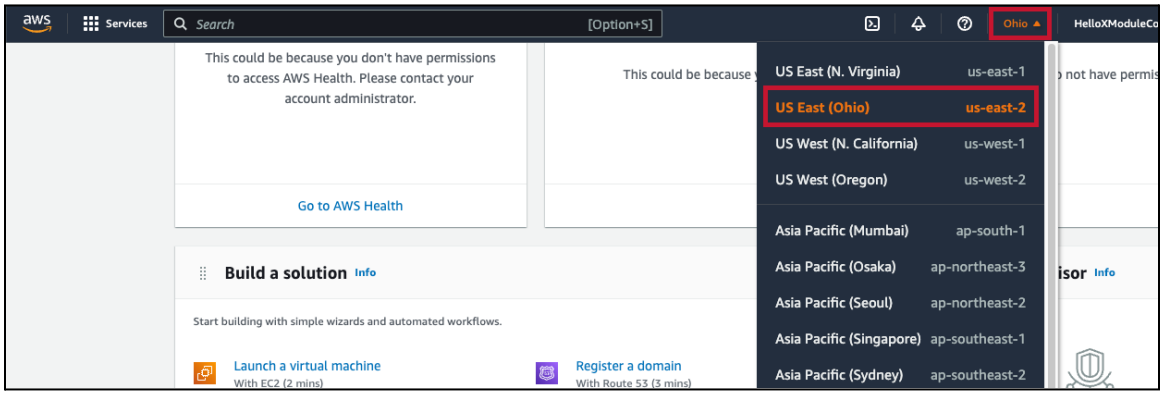
Steps:

1. Click this link: https://dev-68580071.okta.com/ to launch the Okta login for the AWS development environment, type your **Username** and **Password** then click **Sign In**.

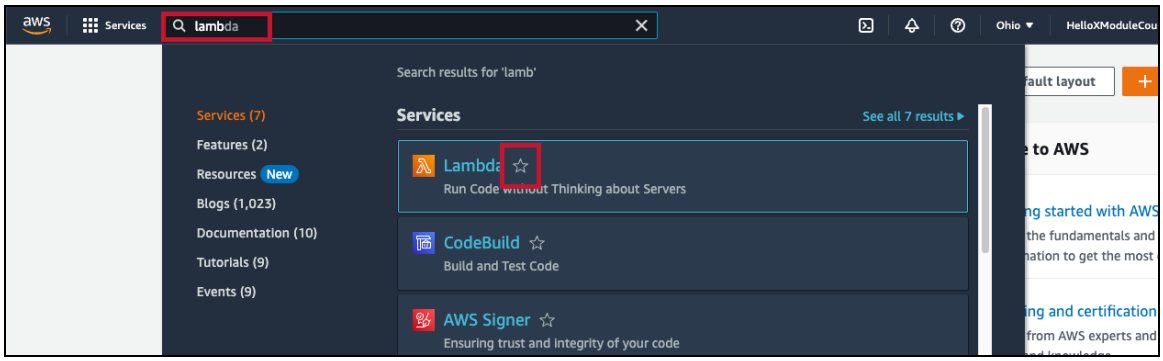


2. In **My Apps** click **AWS Account Federation** to sign into AWS.

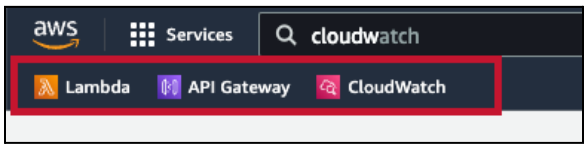
3. Once you successfully log into AWS, we need to make sure you are the correct region and we need to set favorites. Look at the AWS region dropdown and make sure **Ohio** is selected (select **US East (Ohio)** if not already selected).



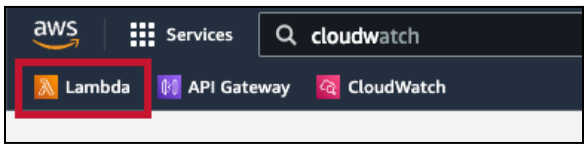
4. Click the Search box and start typing **Lambda** until you see Lambda appear on your screen then click the Favorite button (star) to add it to your favorites.



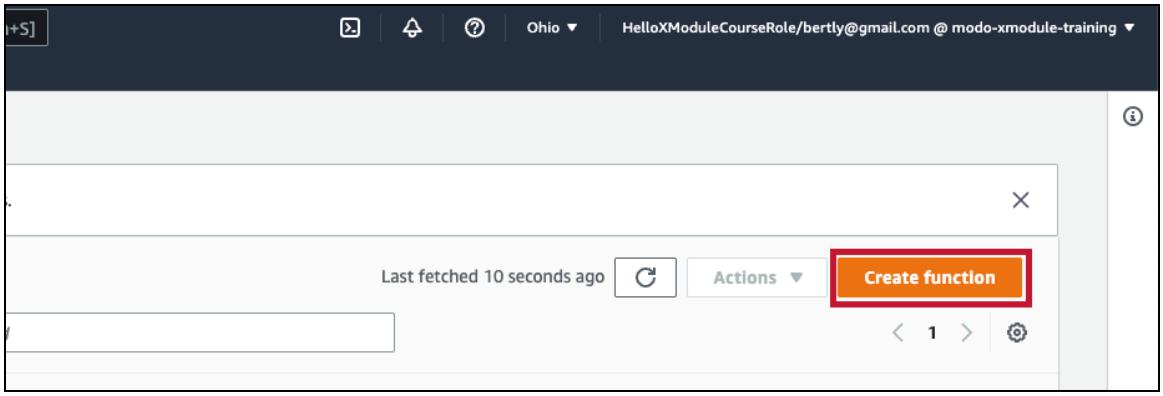
5. Repeat the process with **API Gateway** and **CloudWatch** so we have easy access to each of those services. When you are done, those three services should appear as quick launch options.



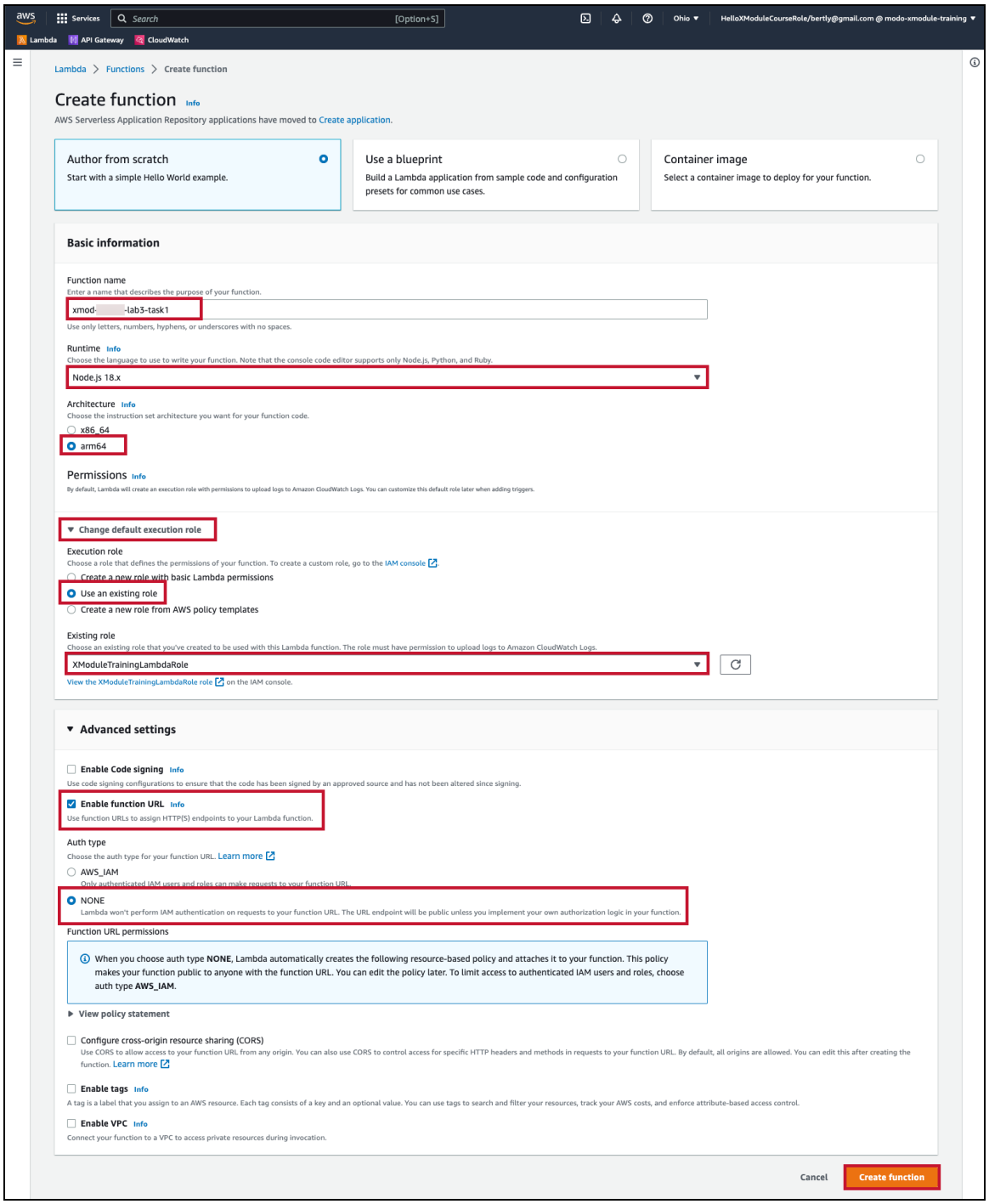
6. Click the Lambda quick launch to open AWS Lambda functions.



7. Click **Create function** to create a new Lambda function.

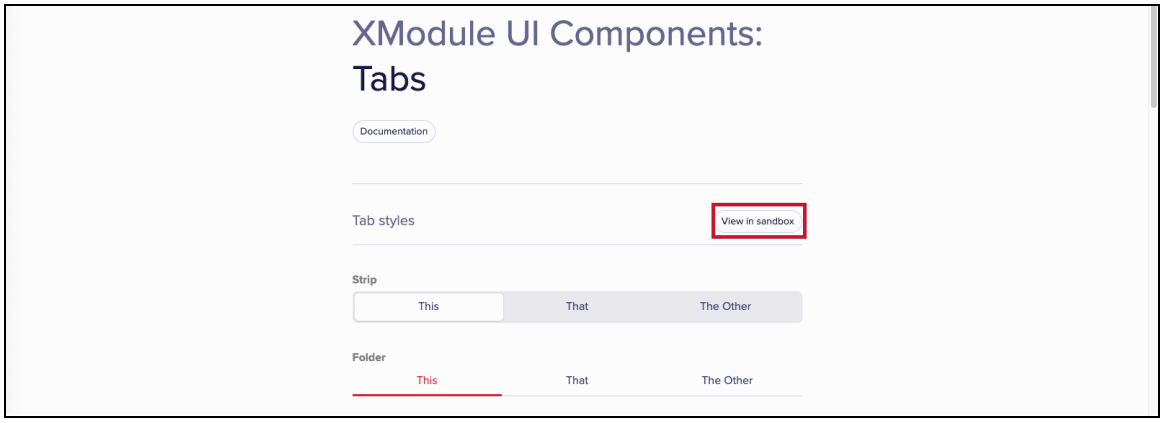


8. Complete the Create function form with the following information:
    a. Select Author from scratch for the function type.
    b. **{xmod-prefix}-lab3-task1** for Function name. The **{xmod-prefix}** comes from your student information and starts with **xmod-** which is a security requirement for students when creating any Lambda function or API gateway. The part after **xmod-** is your assigned application identifier. Make sure you are using your **{xmod-prefix}** to help us organize our Lambda functions and API Gateways.
    c. Select **Node.js 18.x** for **Runtime**.
    d. Click **arm64** for **Architecture**.
    e. Expand **Change default execution role**
    f. Click **Use an existing role** for **Execution role**
    g. Select **XModuleTrainingLambdaRole** for **Existing Role**
    h. Click to expand Advanced settings
    i. Click to check **Enable function URL**
    j. Click **None** for **Auth type**
    k. Click **Create function**

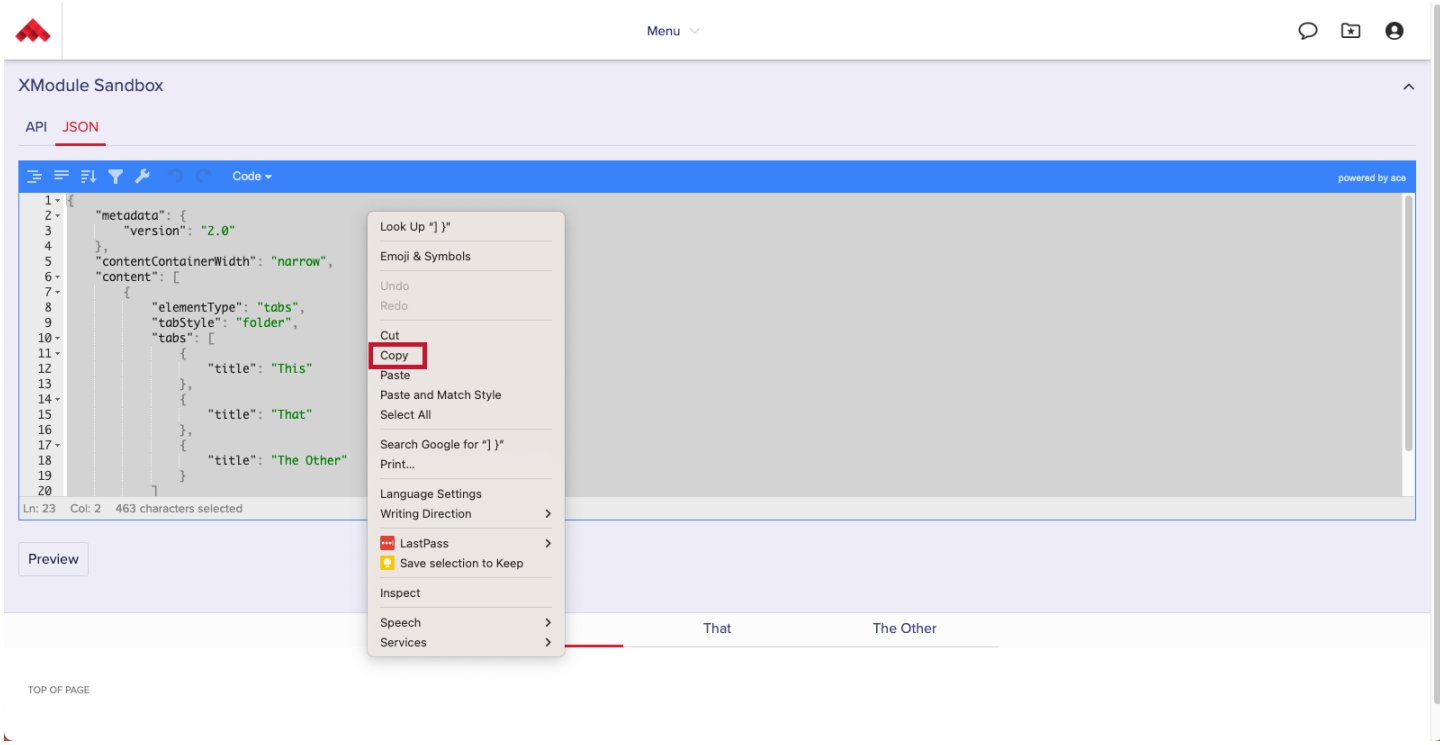


This brings up the Lambda console which allows you to write NodeJs code directly in AWS. The Lambda console supports both Javascript and Python editing directly from the web-based console. We will be providing code for and using Javascript throughout this course.

9. Click the following link: https://xmodule.modolabs.net/default/xmodule_ui_examples/page?relativePath=.%2Ftabs.json and select a series of tabs to view in the XModule sandbox.



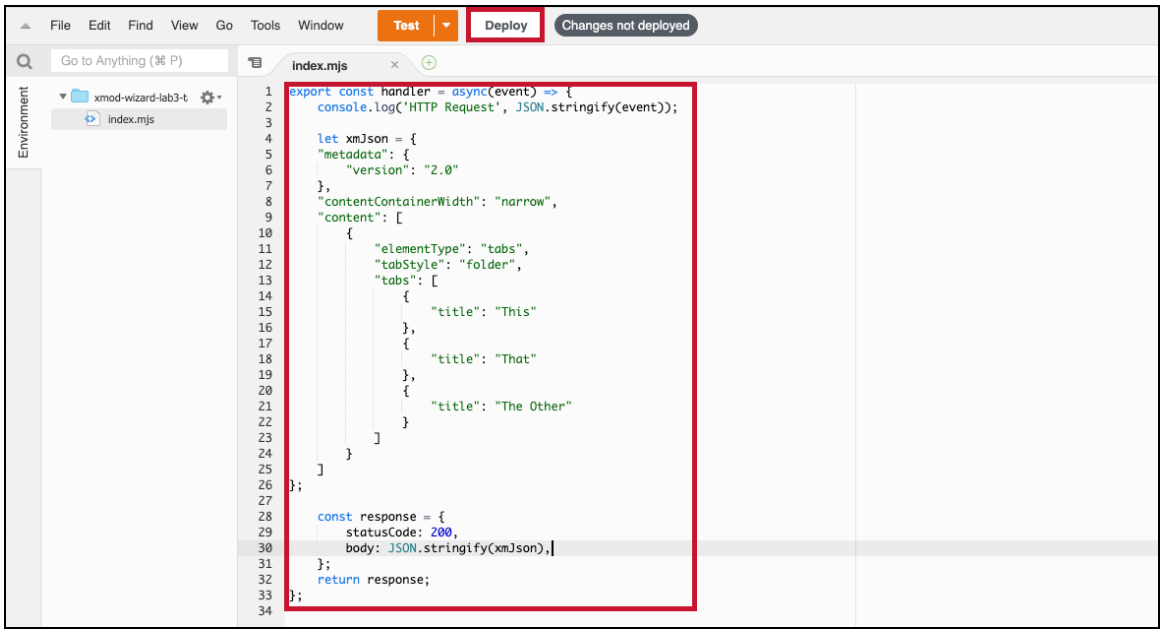
10. Isolate the JSON for the tab style you want to use and copy the JSON to your clipboard.



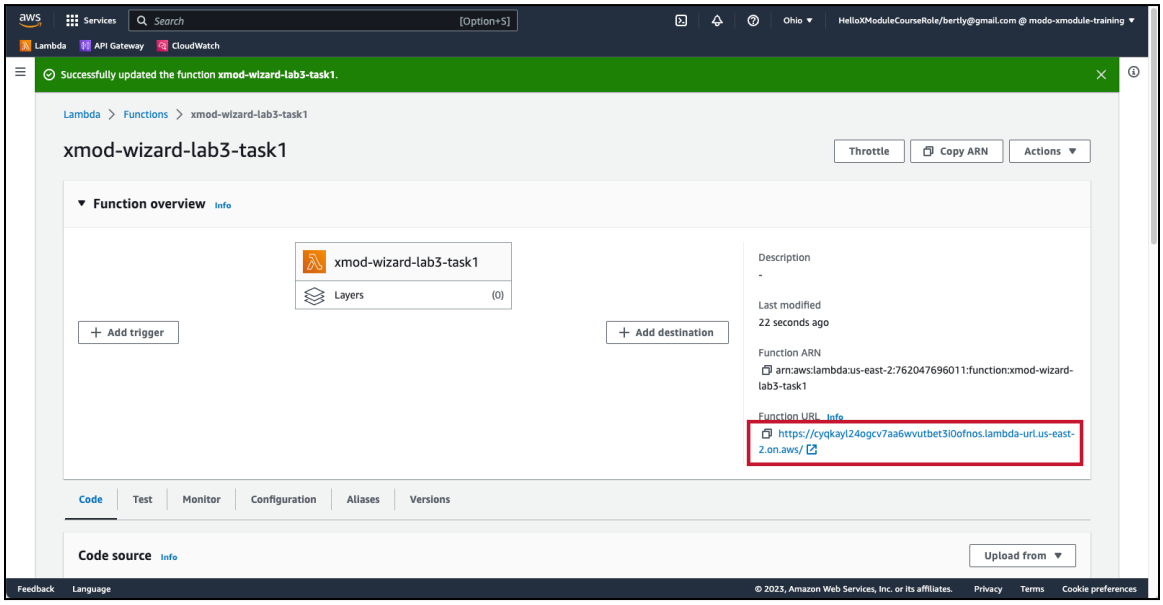
11. In the AWS Lambda console click the index.mjs file to show it in the editor and complete the following:
    a. Delete the `//TODO implement code` from line 2.
    b. Type the following code starting with line 2:

```
console.log('HTTP Request', JSON.stringify(event));

let xmJson =
```

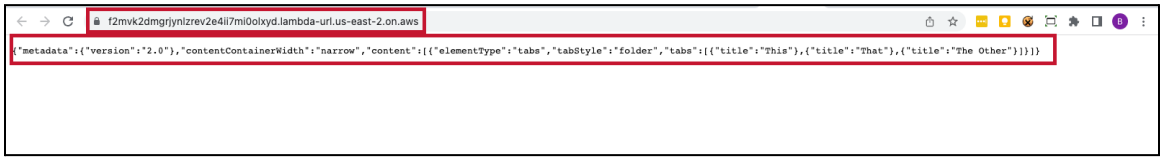
    c. Copy and paste all the JSON from the XModule sandbox on the right of the = sign in the above code (don't worry about formatting).
    d. In the body of the response, change `'Hello from Lambda'` to the `xmJson` variable you just created.
    e. Click **Deploy** to save the changes.



12. Once the **Deploy** button is disabled, click the function URL link.

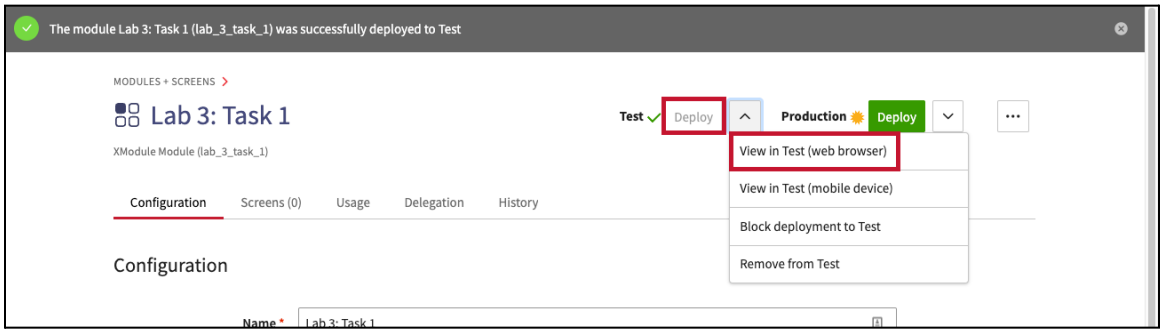


13. Verify the launched page shows your XModule JSON.

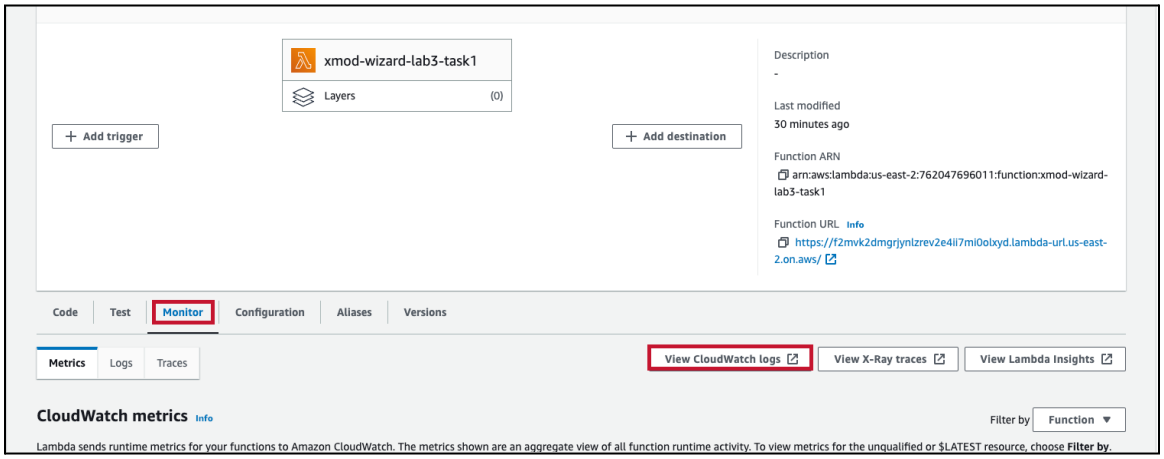


14. Create an XModule in your assigned Modo application using the following settings:
    a. Select **XModule** for **Module type**
    b. Type **Lab 3: Task 1** for **Display name**
    c. Make sure **lab_3_task_1** autocompletes for path
    d. Paste your function URL for **XModule API URL**.
    e. Click **Create module**.

15. Deploy the module to **Test** and preview the module in the web broswer using the dropdown next to the **Test Deploy** button.

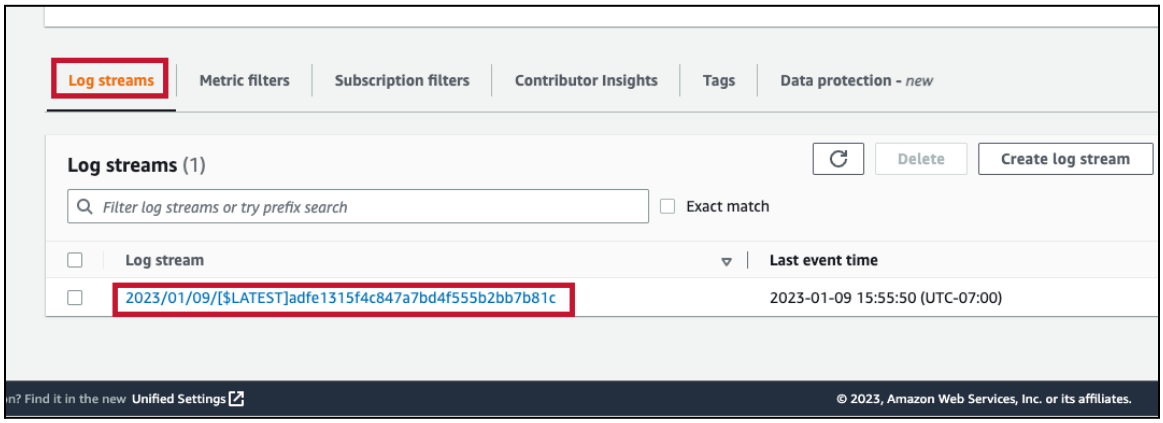


16. Return to the AWS console for your Lambda function, click **Monitor** then click **View CloudWatch logs**.



CloudWatch is the logging service for your AWS functions and any time you add `console.log()` to your code, the items contained within the parentheses is logged for that function in CloudWatch. This can be super useful for debugging your code.

17. In CloudWatch, click the top most line in **Log streams** to view the latest logs.



18. Find and expand the line that contains **HTTP Request** and find the **headers** sent to that Lambda function.

19. Return to your Lambda console tab, click Code then add the following code between lines 2 and 4:

```
let xmList = {
    "elementType": "list",
    "listStyle": "grouped",
    "items": []
};

if ('headers' in event) {
    for (const key in event.headers) {
        xmList.items.push({
            "label": key,
            "title": event.headers[key]
        });
    }
}
```

20. In your tabs array, change the first tab element to the following:

```
            {
                "title": "HTTP Headers",
                "content": [
                    xmList
                ]
```

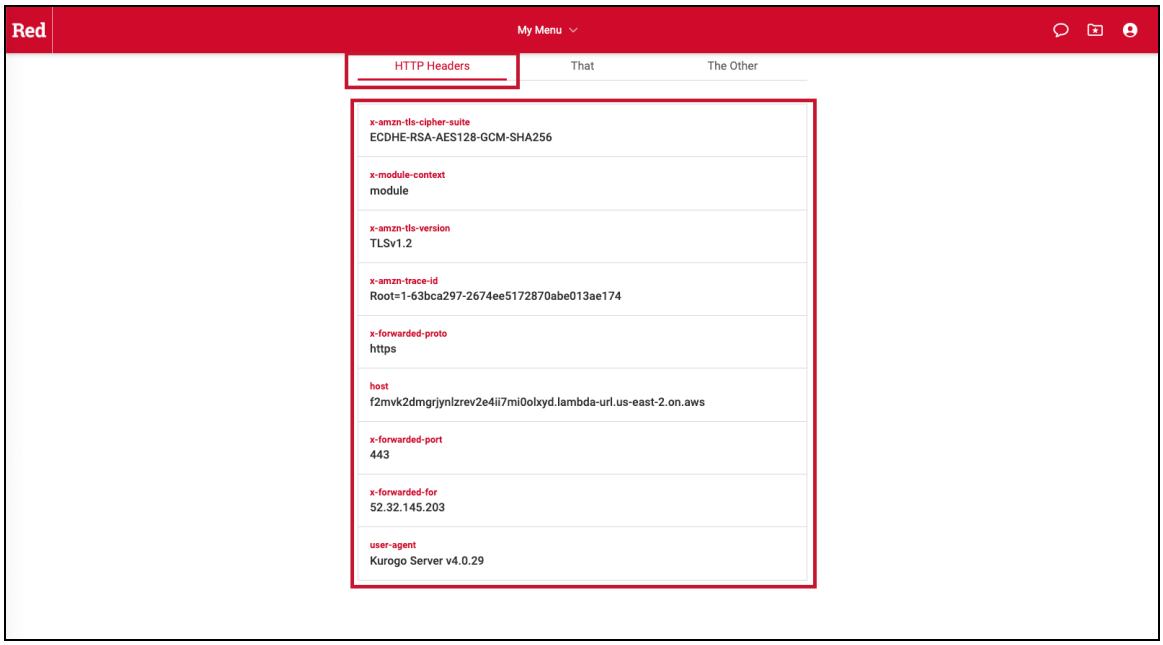
```
        },
```

21. Verify that your JSON is similar to the following then **Deploy** your changes.

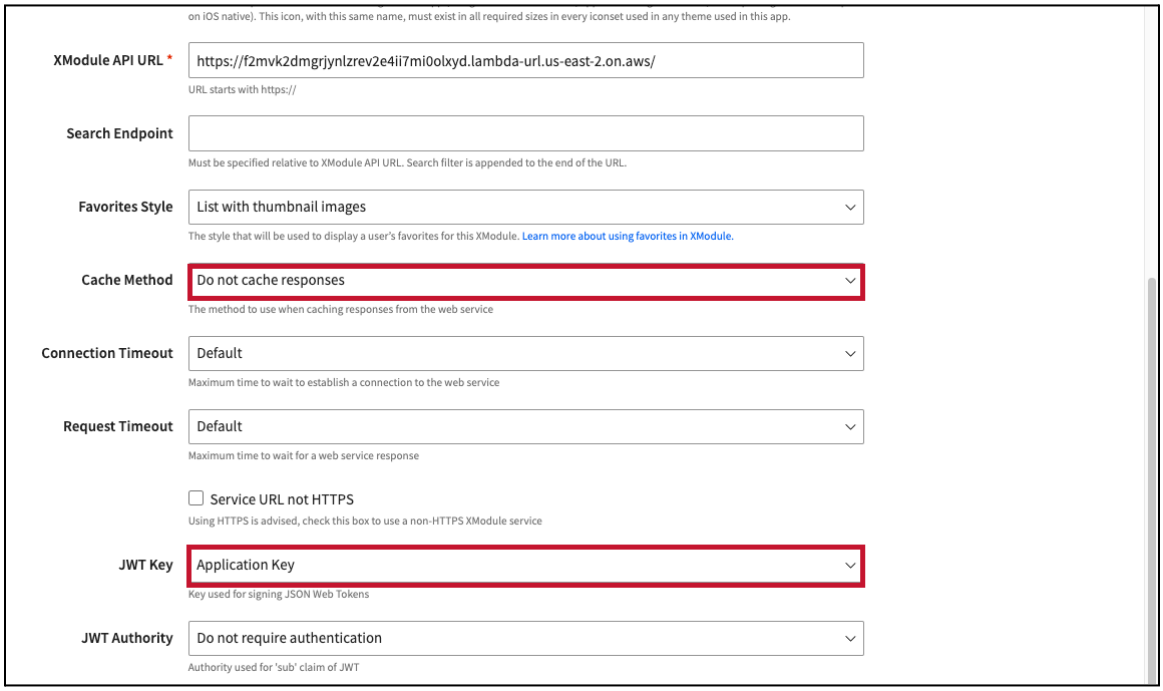

```
export const handler = async(event) => {
    console.log('HTTP Request', JSON.stringify(event));

    let xmList = {
        "elementType": "list",
        "listStyle": "grouped",
        "items": []
    };

    if ('headers' in event) {
        for (const key in event.headers) {
            xmList.items.push({
                "label": key,
                "title": event.headers[key]
            });
        }
    }

    let xmJson = {
    "metadata": {
        "version": "2.0"
    },
    "contentContainerWidth": "narrow",
    "content": [
        {
            "elementType": "tabs",
            "tabStyle": "folder",
            "tabs": [
                {
                    "title": "HTTP Headers",
                    "content": [
                        xmList
                    ]
                },
                {
                    "title": "That"
                },
                {
                    "title": "The Other"
                }
            ]
        }
    ]
};

    const response = {
        statusCode: 200,
        body: JSON.stringify(xmJson),
    };
    return response;
};
```

What did we just do? We created an XModule list that didn't have any items. We checked for headers in the HTTP Request and looped through each key on the JSON event.header object and added the key as the label and the value as the title. Then we changed the title of the first tab to HTTP Headers and added the list to the tab's content array.

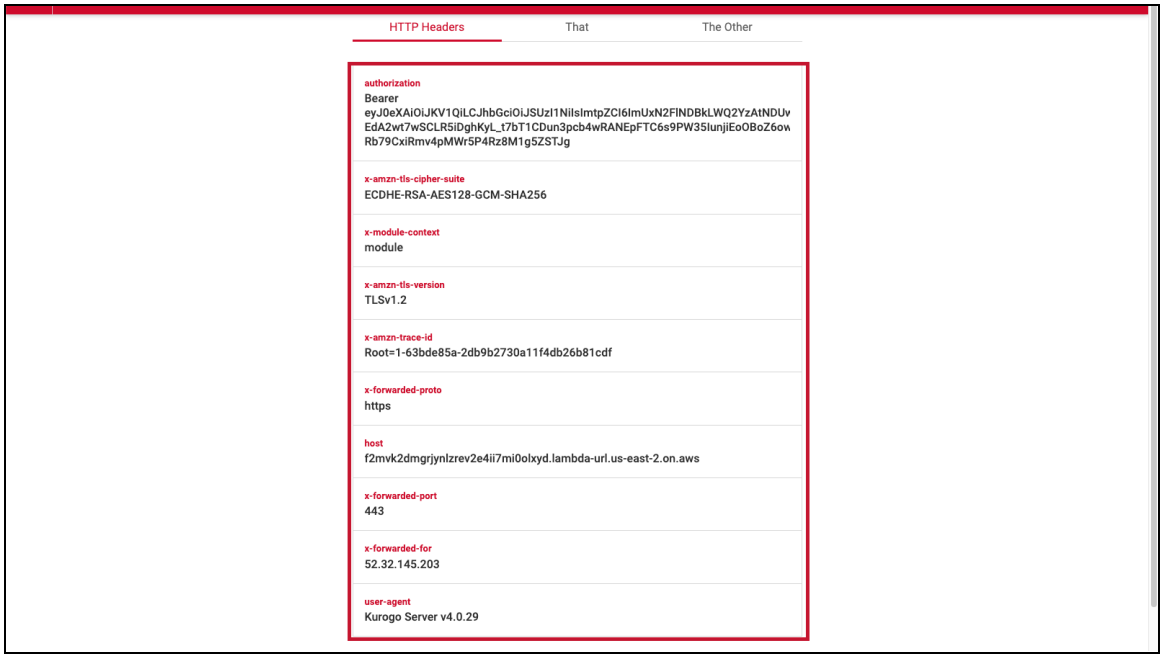
22. Return to your Lab 3: Task 1 XModule preview screen and refresh it to view the HTTP headers in the first tab to verify that your XModule function is working properly.



23. Go back to your XModule and configure the following settings:
    a. Select **Do not cache responses** for **Cache Method**
    b. Select **Application Key** for **JWT key**
    c. Click **Save**



24. Deploy your changes to Test then refresh the preview page that shows your headers to see the authorization header value.



Congratulations! You have successfully processed the HTTP Headers. An XModule like this is a powerful tool for verifying the content of the HTTP Request. It may be useful to have one of these available as you start developing content with a new customer or if you want to know for sure what is being sent over from the Modo application.

Task 2: Adding header values and changing the Modo JWT header

Context: Sometimes an existing web service requires the authentication token provided by the IdP as the Authorization header while other web services may require a separate API key and API token as part of the HTTP request header. This can be accomplished using the Additional feed configurations on your XModule.

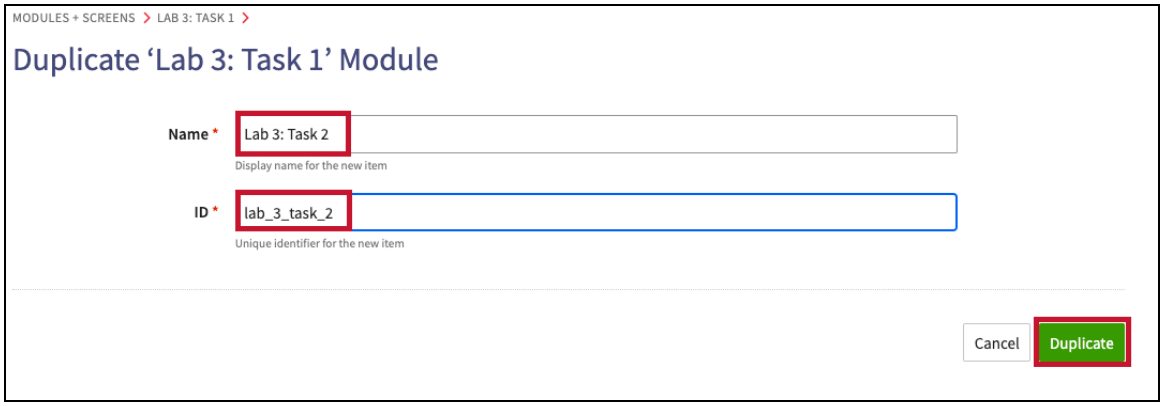
Goal: Edit the Additional feed configurations to change the Modo JWT header and pass additional values in the HTTP headers request.

Task Outline:

- Create a new XModule named **Lab 3: Task 2** (can be done by duplicating the module)
- Add **“jwtAuthorizationHeader”: “modoJWT”** to the **Additional feed configurations** then deploy and preview those changes
- Add **“SomeHeaderKey”: “SomeHeaderValue”** and **“AnotherHeaderKey”: “AnotherHeaderValue”** to the HTTP request using **Additional feed configurations** then deploy and test those changes

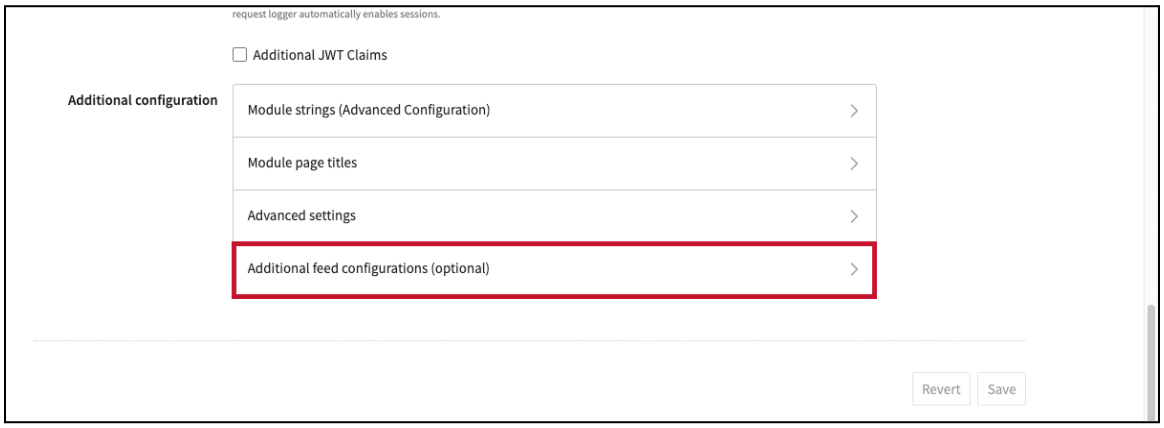
Steps:

1. Click the **…** (Additional Operations) button for your Lab 3: Task 1 XModule then click **Duplicate**.
2. Complete the duplicate module form by doing the following:
    a. Change Name to Lab 3: Task 2
    b. Change ID to lab_3_task_2
    c. Click **Duplicate**



Note: From here on out, this will be a step called **Duplicating the Module** and only the new name will be provided. It is expected that you will match the name and ID appropriately as shown in this example.

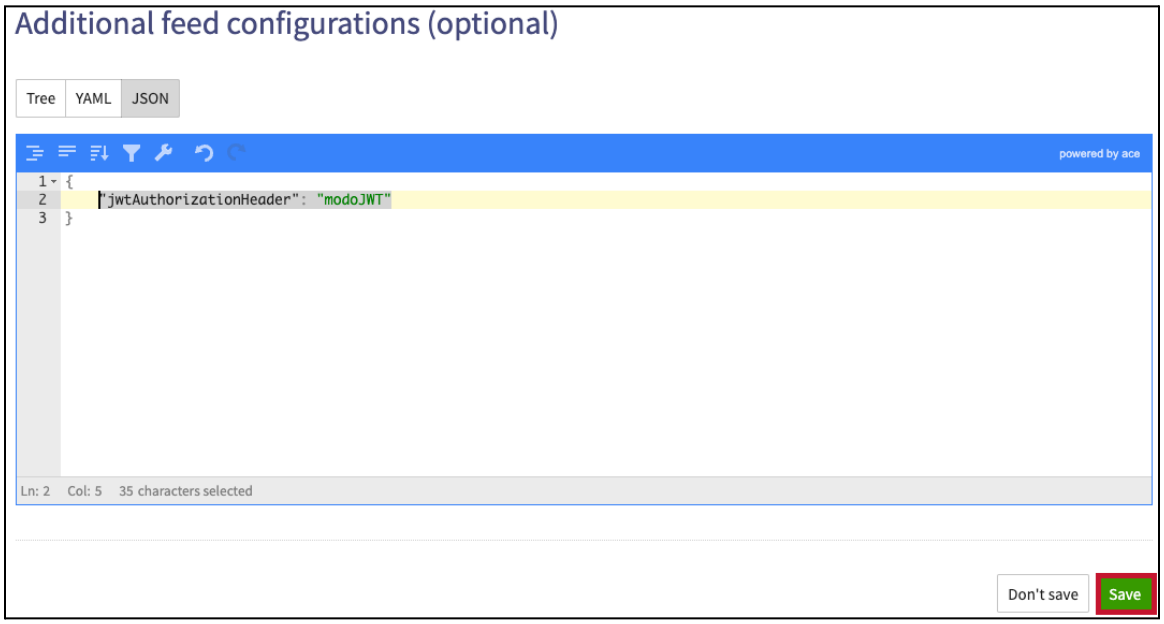
3. Scroll to the bottom of the module **Configuration** page and click **Additional feed configurations (optional)**



4. Click **JSON** for the JSON editor then copy and paste the following code between the brackets.

```
"jwtAuthorizationHeader": "modoJWT"
```

5. Verify that the content looks like the image below then click **Save**.

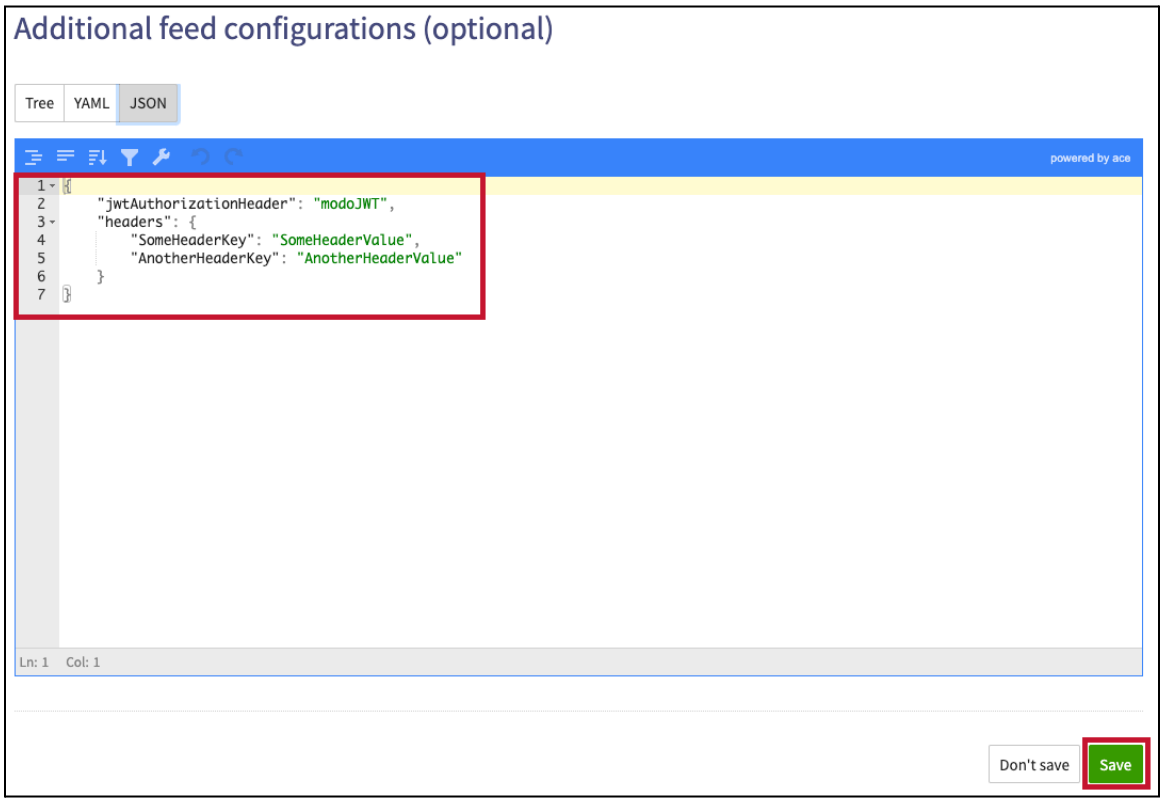


6. Deploy **Lab 3: Task 2** to test then preview the Module to verify the new header value for the Modo JWT.

7. Return to the Lab 3: Task 2 module Configuration page, scroll to the bottom and click Additional feed configurations (optional), click the JSON button to bring up the JSON editor, adjust the JSON to contain the following.

```
{
    "jwtAuthorizationHeader": "modoJWT",
    "headers": {
        "SomeHeaderKey": "SomeHeaderValue",
        "AnotherHeaderKey": "AnotherHeaderValue"
    }
}
```

8. Verify that it looks like the image below then click **Save**.



9. Deploy the changes to test and preview the page to view the new values and ensure it looks like the image below (it is okay if the values do not show up in the same order).

Congratulations! You have successfully changed the Modo JWT and added additional header parameters. This can be super useful when your application requires the authorization header for the Authentication Token or when you need to authorize your web service by additional header values.

Task 3: Decoding the JWT and verifying the signature

Context: As seen in our last example, the JWT is passed in as a header value which we changed from the Authorization header to whatever you have selected. The JWT payload is base64 encoded so you can use your backend server to decode the JWT pretty easily. Additionally, JWT.io provides libraries in a multitude of languages which perform JWT signing validation. The method that typically verifies the signature can also decrypt the payload. Modo provides an Application Key to sign the data which can be validated using a provided JWK. There are many libraries that help users to complete this process based on the programming language you are using. Since we are using NodeJS the libraries we will be using are:

- jwk-to-pem - converts a JWK into a PEM certificate that can be used to verify the signature
- jsonwebtoken - checks the JWT provided to see if it is signed appropriately

The libraries are added to the project through Lambda layers. Lambda layers allow us to package and upload a language specific library that can then be referenced in our code. We have taken care to add all the Lambda layers we believe you will need for this project.

We will be using the Lab 3: Task 2 XModule since it contains the modified Modo JWT header value. Most of this work will be done inside the AWS Lambda function we created in Task 1. You should close the Task 1 preview window if it is still open.

Goal: Verify the Modo JWT signature and decode the payload using the Lab 3: Task 2 XModule by importing the libraries, validating the signature, and decoding the payload and adding those values to the middle tab.

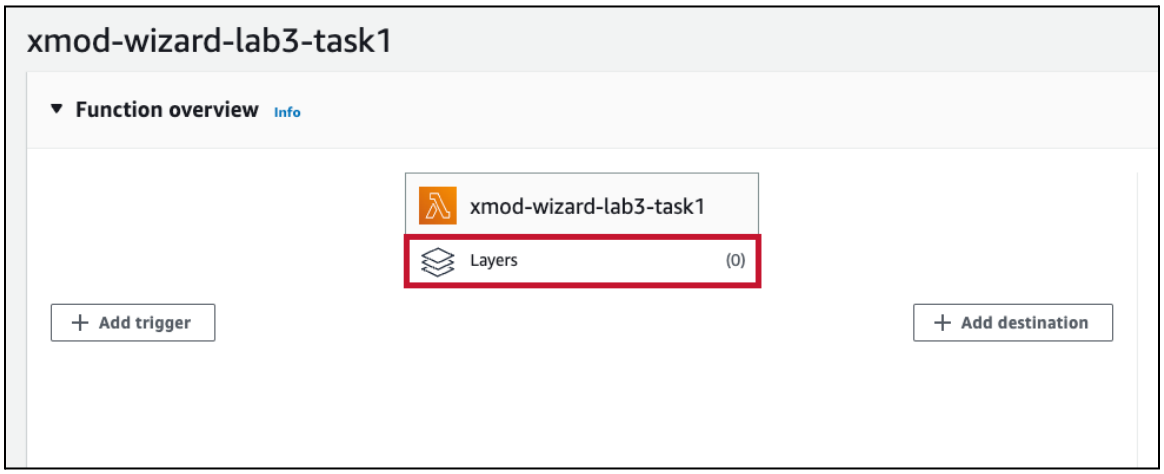
Task Outline:

- In the **{xmod-prefix}-lab3-task1** Lambda function, add the **jwk-to-pem** and **jsonwebtoken** Lambda layers to your project.
- Add variables to add jwk-to-pem and jsonwebtoken to your library for the well-known JSON Web Key from your application (see student guide for this link)

- Decode the payload using the values specified and create a card set to hold all of the decoded payload values then display the cardset in your second tab which should be titled **Payload Values**

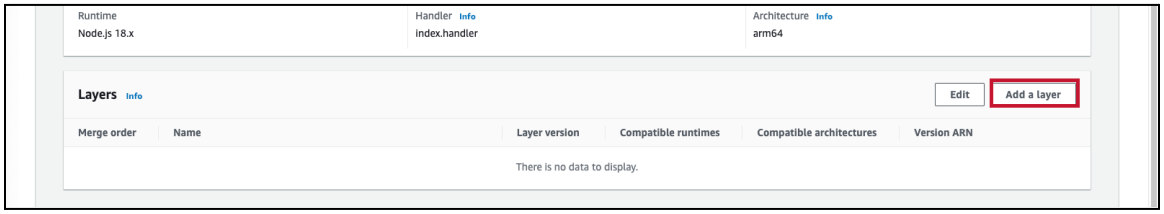
Steps:

1. In the AWS console for **{xmod-prefix}-lab3-task1**, click **Layers** in the **Function overview** section.

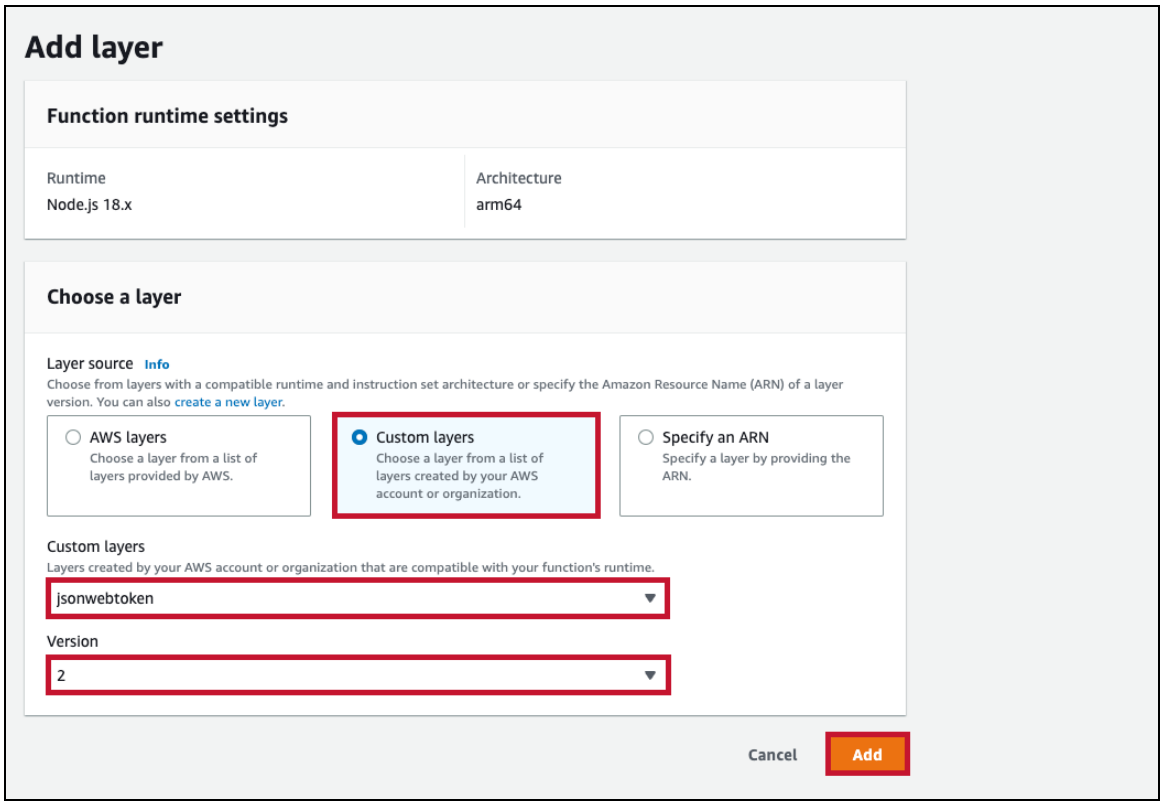


Note: Layers are found at the very bottom of the page. You can alternatively scroll to the bottom of the page to view the Layers section.

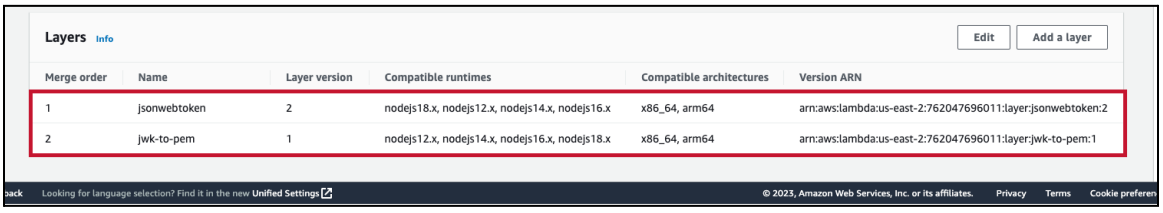
2. Click **Add a layer**.



3. Click **Custom layers**, select **jsonwebtoken** from the **Custom layers** dropdown, select **2** for **Version** then click **Add**.



4. Follow steps 1-3 to add **jwk-to-pem** version **1** as an additional Lambda layer then verify that you have both **jsonwebtoken** and **jwk-to-pem** as Layers in your Lambda function.



5. Return to the Code source and add a couple of new rows above line 1. In those rows add the following code below to lines 1 and 2:

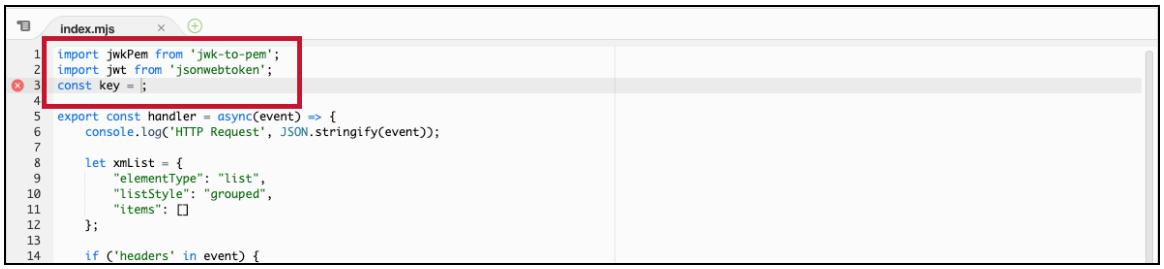
```
import jwkPem from 'jwk-to-pem';
import jwt from 'jsonwebtoken';
```

6. Next we need to create a variable to hold the well-known JWK. To discover the JWK for your application click the JWKS Location from your Student guide and copy the values inside the `keys` array to your clipboard.

7. Add the following code to line 3 of your Lambda function.

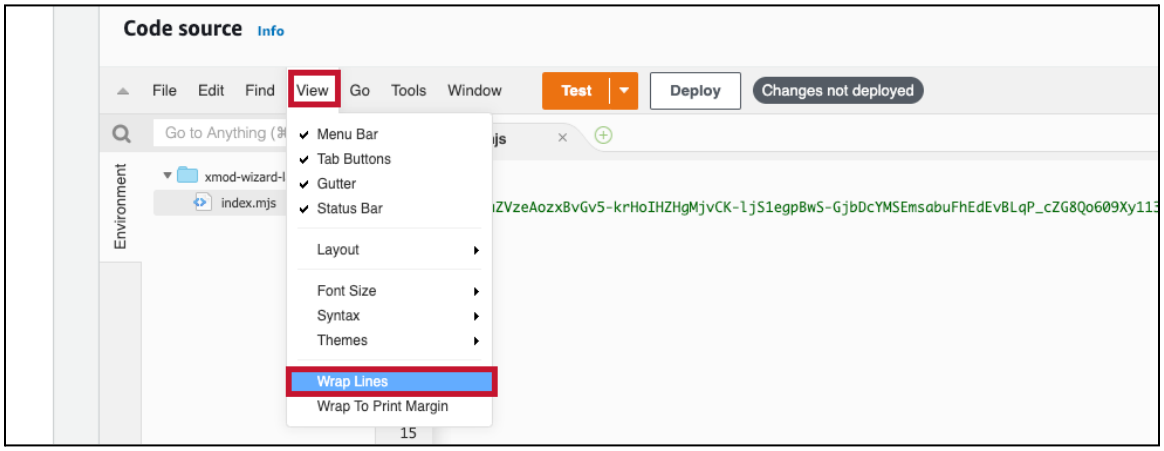
```
const key = ;
```

8. The resulting Lambda should look similar to the image below.

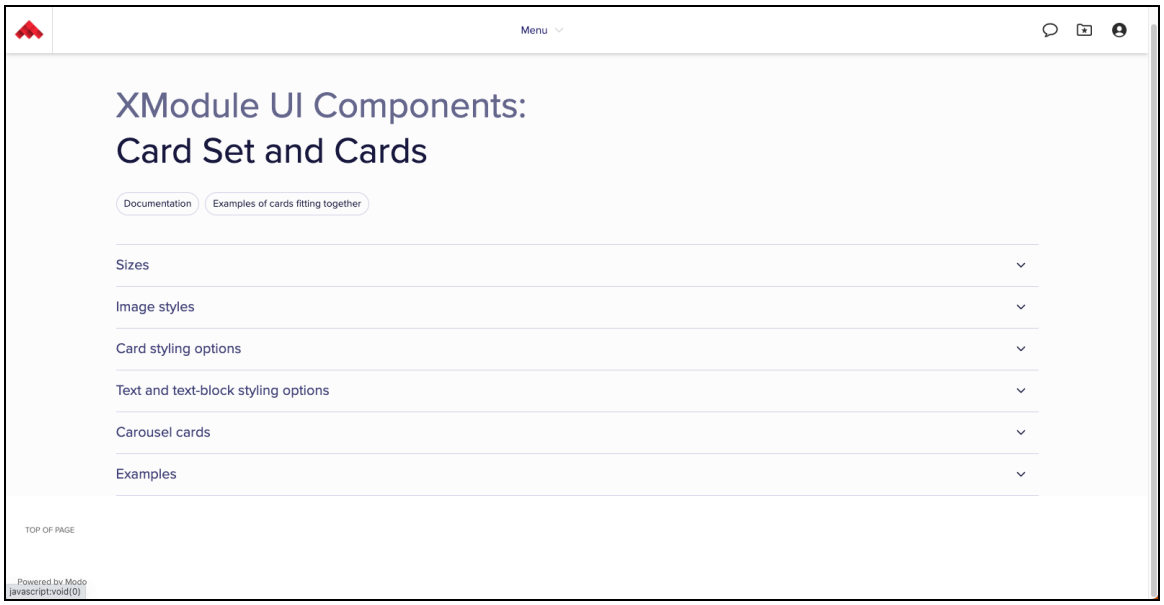


9. Move your cursor in front of the semicolon on line 3 and paste the contents of your clipboard using `ctrl+v / cmd+v` to set the `key` variable to the key you copied from your JWKS. The results may look like the screenshot below.

10. You can apply Word Wrap to the console by clicking **View** > **Wrap Lines.**

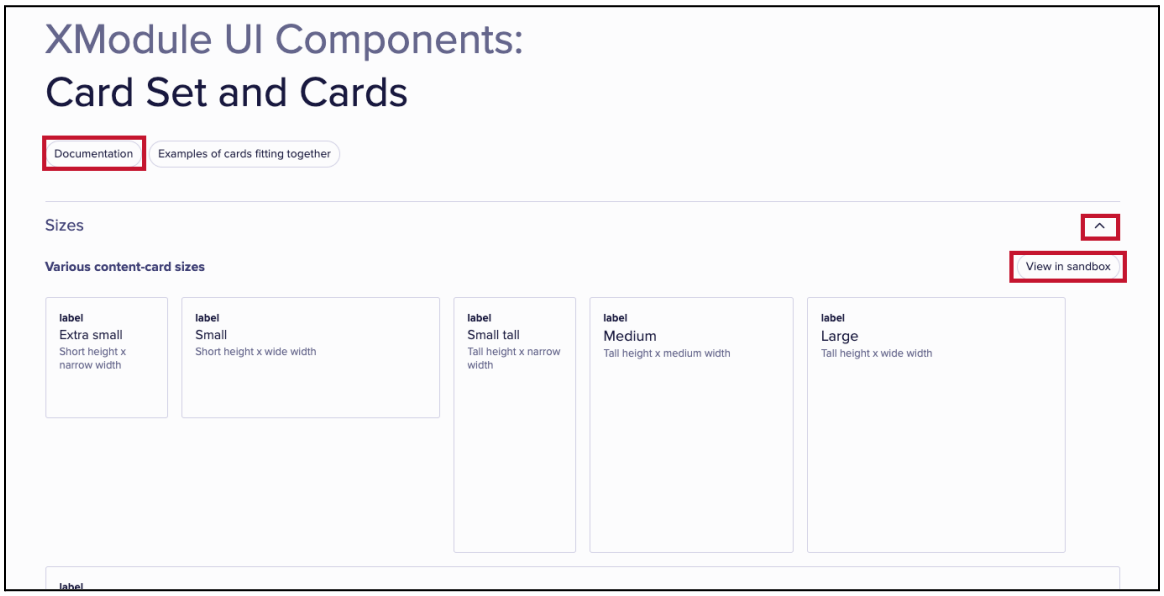


11. Now we need to do the following steps:
    a. Create a variable to hold the `payload` that is set to `null`
    b. Check for `headers` in the `event` object and look for `authorization` in the `headers` part of `event`
    c. Use the jwtPem and jsonwebtoken libraries to validate the signature and decode the payload
    d. Set **Payload Values** for the 2nd tab's title
    e. Create a `cardSet` object to contain all the values for the payload and add it to the 2nd tab's content array.
    f. You can try this on your own or follow the steps below to complete this process.
12. Go to the Card Set examples by clicking the following link: https://xmodule.modolabs.net/default/xmodule_ui_examples/page?relativePath=.%2Fcards.json

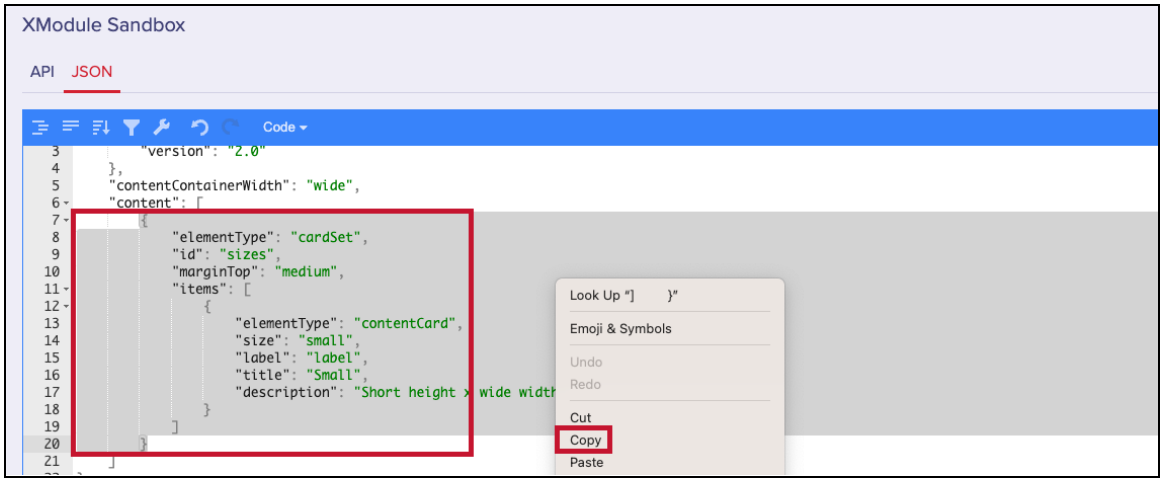


Spend a couple of minutes looking through all the Card Set examples. You are welcome to choose whichever style you like best but these instructions will use an example from the Sizes collapsable.

13. Click the **Documentation** button to open Card Set documentation in a new tab (we will look at that later), then expand the **Sizes** collapsable, and click **View in sandbox**.



14. Isolate and extract the `cardSet` element and keeping only the `"size": "small"` card in the cardSet's items array.



15. Around line 23 add a couple of empty lines to hold the following code:

```
let xmCardSet = ;
```

16. Place your cursor before the ; and paste the extracted element to set the variable to that element.

17. Add the following lines of code around line 38.

```
let payload = null; //create a variable to hold the decoded payload values

//check that headers exist in event and that authorization is in event.headers
if ('headers' in event && 'modojwt' in event.headers) {

}
```

18. Inside our if statement (around line 42) add the following two lines of code:

```
    payload = jwt.verify(event.headers.modojwt.split(" ")[1], jwkPem(key));
    console.log("Payload", payload);
```

The jsonwebtoken library provides a verify function that takes the JWT (all values of the Modo JWT header except the string `Bearer`) and the PEM or shared secret string. The jwk-to-pem library converts the JWK to a PEM and has only one method that is called by using the variable name and putting the key in parentheses. I like to log the results so I can view the payload values but we will be outputting this instead so this is really not critical.

19. Make sure we have some empty space after the above code and around line 39 add the following code:

```
//if payload is not empty
if (payload != null) {
    for (let key in payload) {
```

```
        }
    }
```

20. Return to the cardSet element you added and add a new property `"size": "small"`, after the line containing the id property, and cut the contents from the items array and put it on your clipboard.

21. Inside the payload for loop type **xmCardSet.items.push(** then paste the contents of your clipboard and finally type **);** to complete the statement.

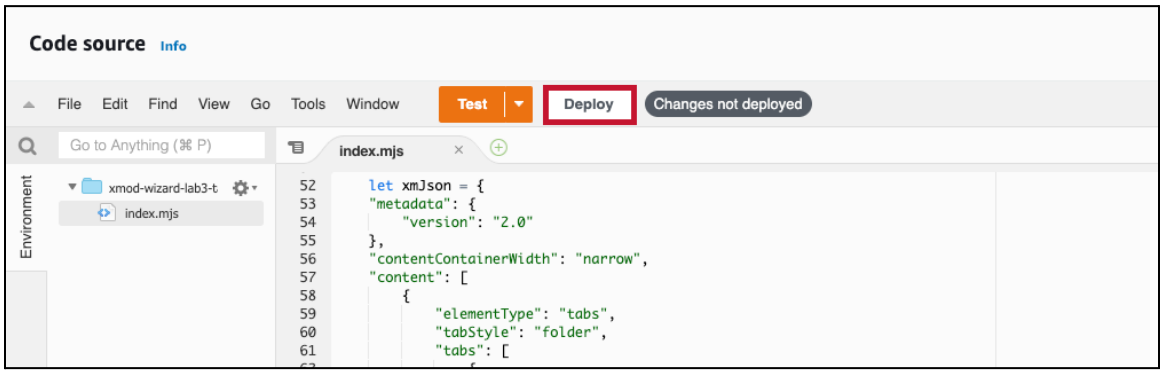
22. Inside the xmCardSet.items.push modify the contents by doing the following:
    a. Delete the `"size": "small"` property declaration (it was added to the cardSet element so all items will be rendered small unless overridden)
    b. Delete the `"label": "label"` line.
    c. Change the title line to `"title": key,`
    d. Change the description line to `"description": payload[key].toString()`
    e. Verify the new code looks like the screenshot below.

23. Find the second tab in the tabs array and modify it as outlined below:

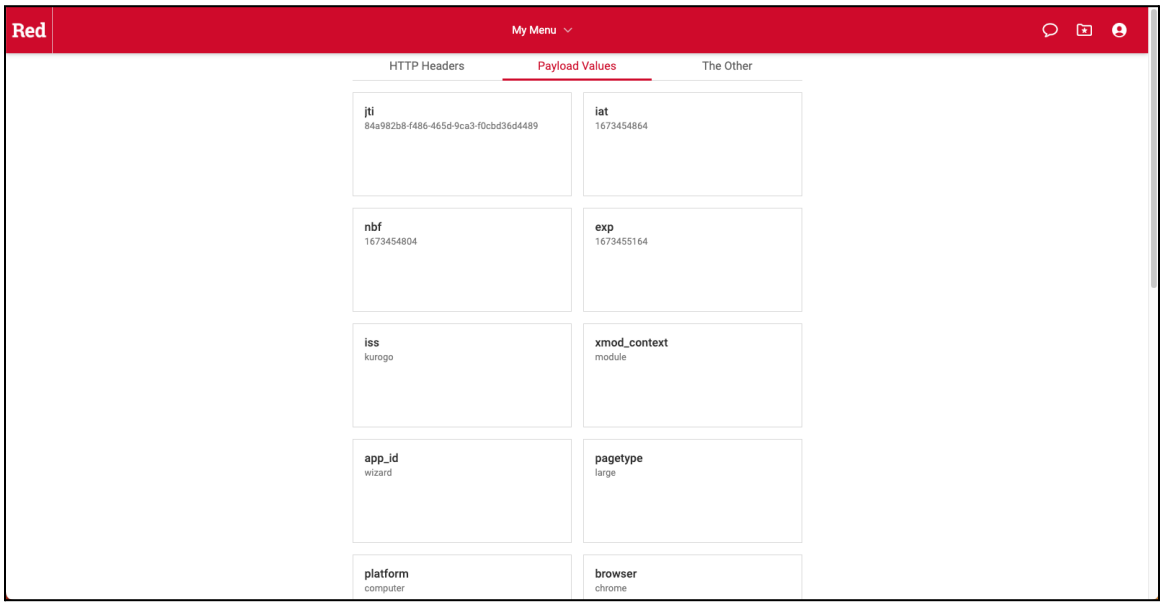
a. Change the title to **Payload Values**
b. Add the `content` property and the `xmCardSet` to the content array.
c. Verify that the content looks like the image below.

```
let xmJson = {
"metadata": {
    "version": "2.0"
},
"contentContainerWidth": "narrow",
"content": [
    {
        "elementType": "tabs",
        "tabStyle": "folder",
        "tabs": [
            {
                "title": "HTTP Headers",
                "content": [
                    xmList
                ]
            },
            {
                "title": "Payload Values",
                "content": [
                    xmCardSet
                ]
            },
            {
                "title": "The Other"
            }
        ]
```

24. Click **Deploy** to commit your changes to the server.



25. Return to your Lab 3: Task 2 Xmodule preview page and refresh it to view the second tab data.



Congratulations! You have successfully decoded the payload using the Modo provided JWK. Using libraries to complete these tasks accelerates the process and ensures signature validation. Each library usage is different so make sure you read the documentation.

Task 4: Passing and configuring authentication properties for the JWT

Context: Any value passed from the authentication authority can be used in the JWT but we must configure the authentication authority to send those values to your web service in the HTTP request. This is done by looking at the authority and mapping attributes in the authority to names you expect in the web service. This mapping process allows you to have different authorities with different properties and still use one web service.

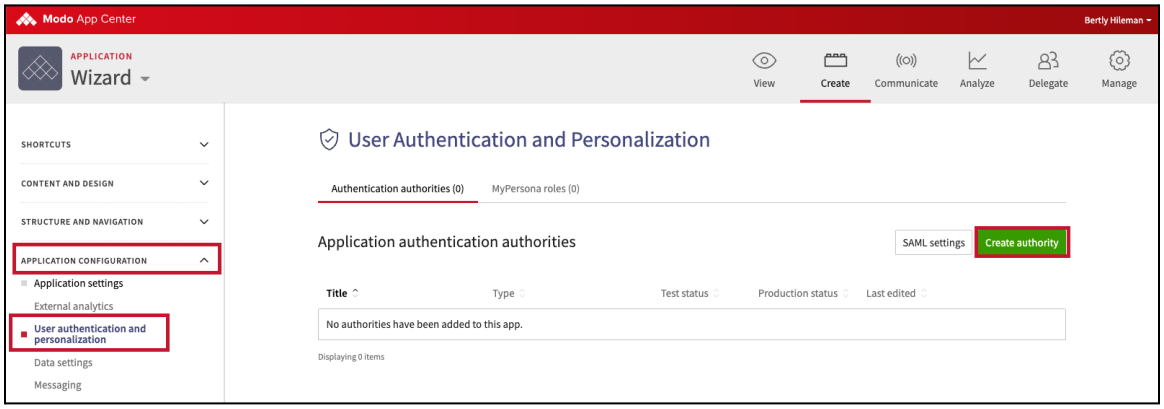
Goal: Build a nametag using data available from the IdP.

Task Outline:

- Create a new YAML authority named Character Login using the data from the following URL: https://static.modolabs.com/xmodule_certification/yaml_auth/xmod_cert_auth.yml then log in using that character information
- View the User information available from your IdP
- Add JWT standard claims mapping as well as attribute mapping found in step 11 of this procedure
- Change your XModule to use the new Character Login JWT Authority and deploy and test all of your changes
- Create a Name Tag element to hold character information and display it in the third tab with

Steps:

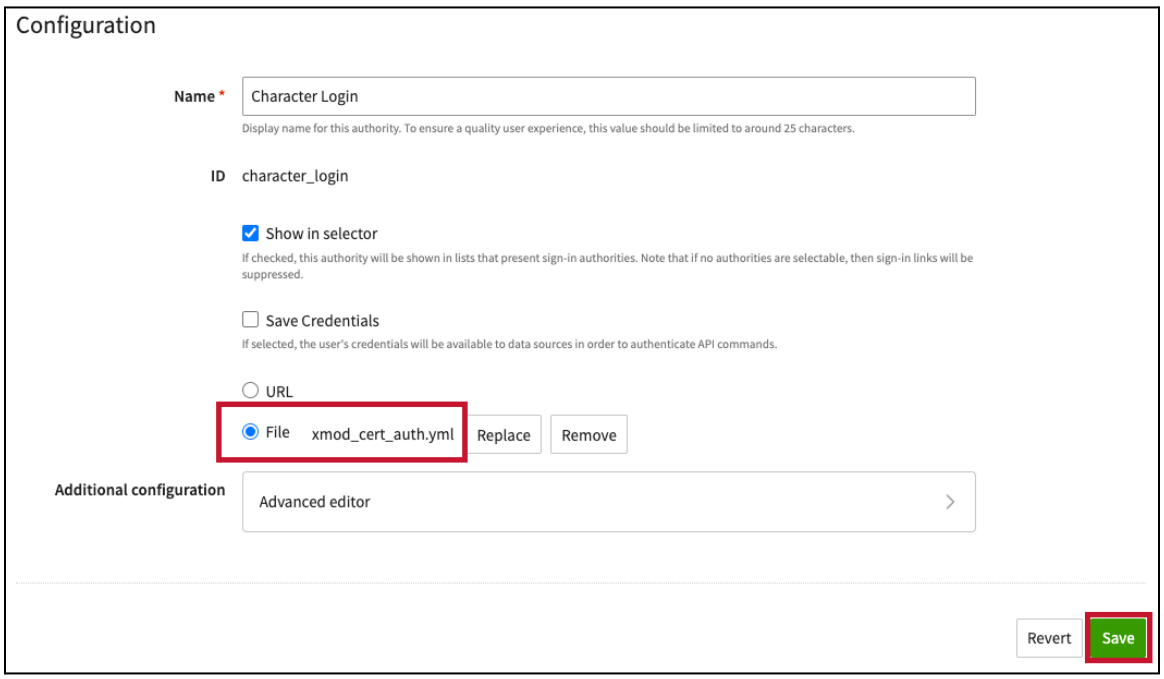
1. In Modo App Center under Create, click to expand **Application Configuration**, click **User authentication and personalization**, and under the **Authentication authorities** tab click **Create authority**.



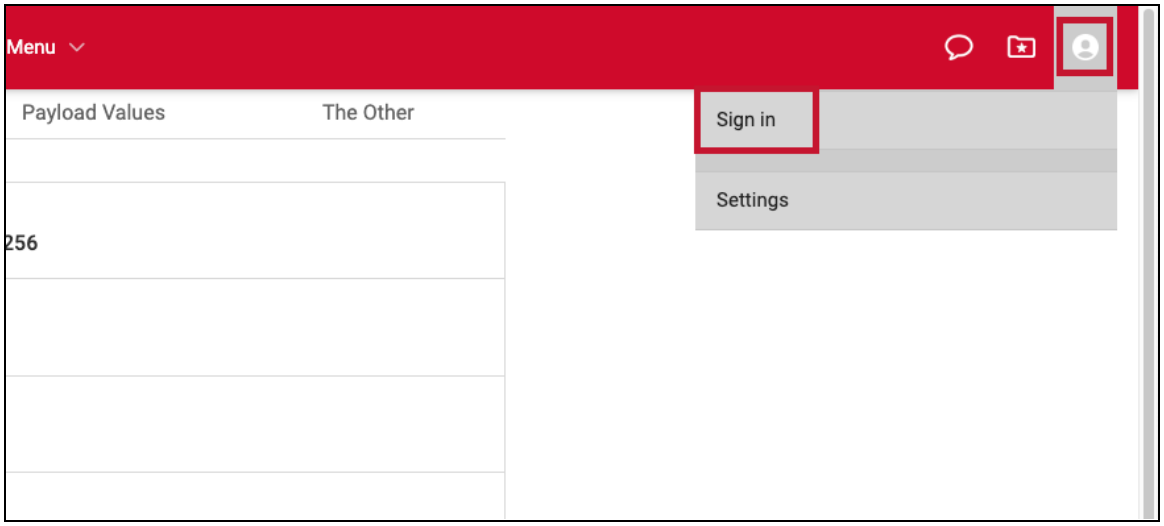
Note: In this screenshot I have collapsed the Shortcuts, Content and Design and Structure and Navigation sections of the left navigation.

2. Configure the New Authority using the following settings:
    a. Select YAML file authentication for Authority
    b. Type Character Login for Name
    c. Make sure character_login is set for ID
    d. Click Create authority.

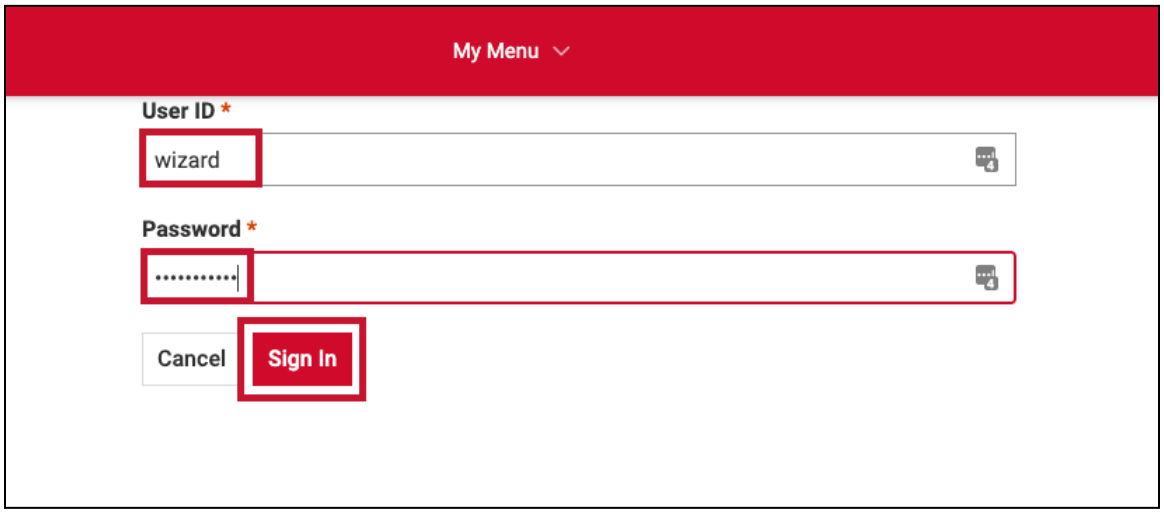
3. Save the file from [this link] to your computer, then click **File**, **Upload**, and select the YAML file provided button, and click **Save.**
   https://static.modolabs.com/xmodule_certification/yaml_auth/xmod_cert_auth.yml in the URL field then click **Save**.



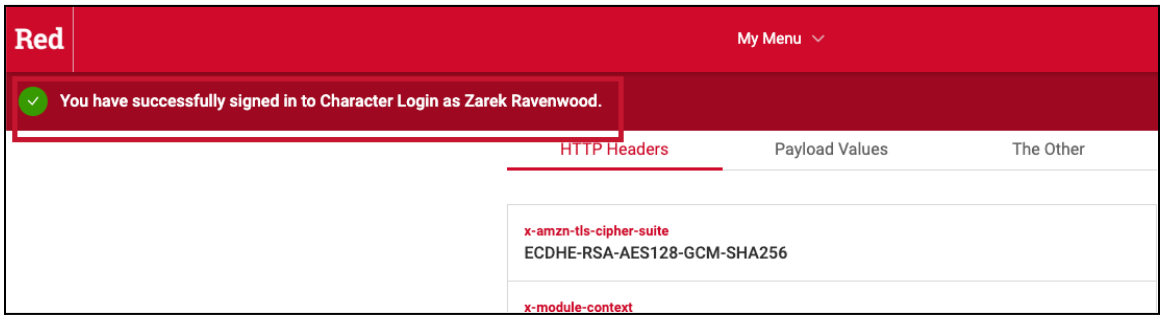
4. Deploy the file to Test and then return to your Lab 3: Task 2 preview, refresh the page, and click the Account menu, then click **Sign In**.



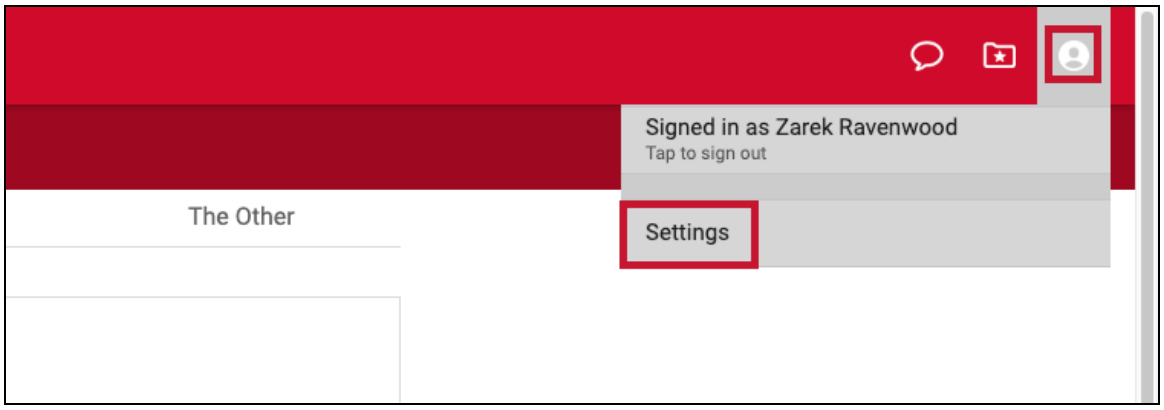
5. Type your assigned application identifier for **User ID** and **Password123** for Password then click **Sign In**.



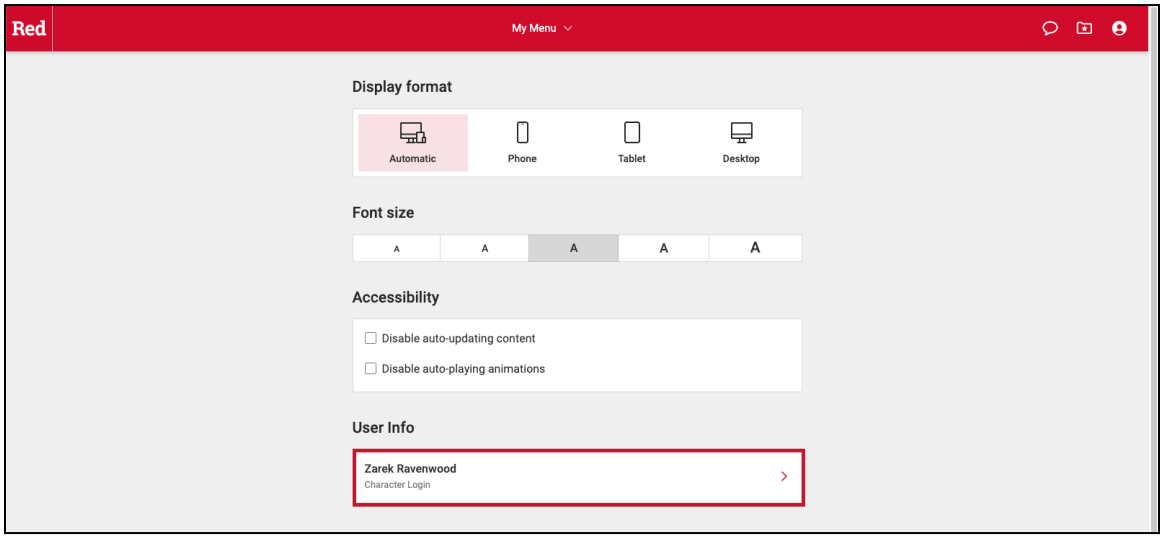
6. You should receive a banner message stating you have successfully logged in as your character.



7. Click the Account menu then click **Settings**.

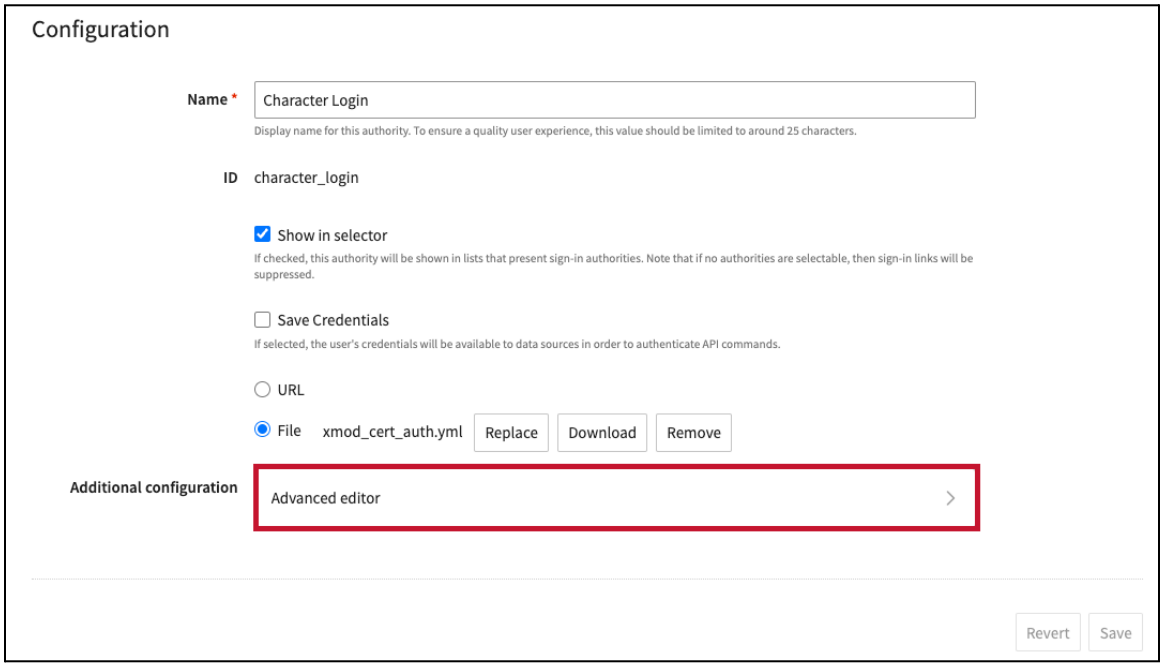


8. Click **User Info** to see the information provided from your new IdP.



9. These values can be mapped and passed to the JWT using the steps found in this document: https://support.modolabs.com/support/solutions/articles/13000091958-configuring-an-authority-to-send-attributes-with-jwt

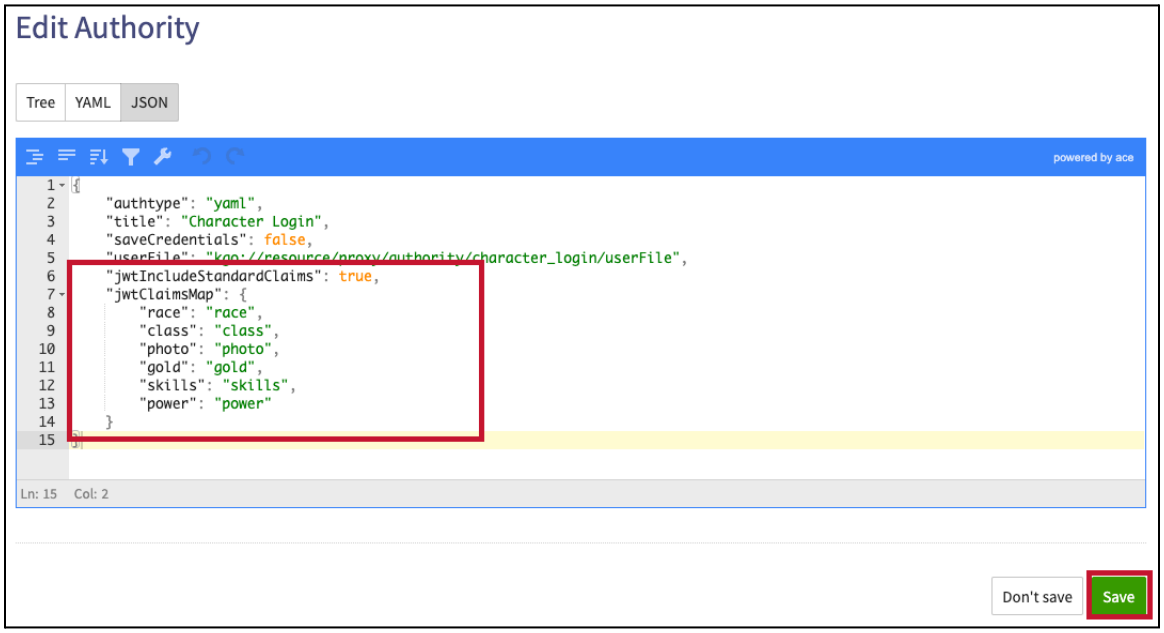
10. Return to your Modo App Character Login authority and click **Advanced editor**.



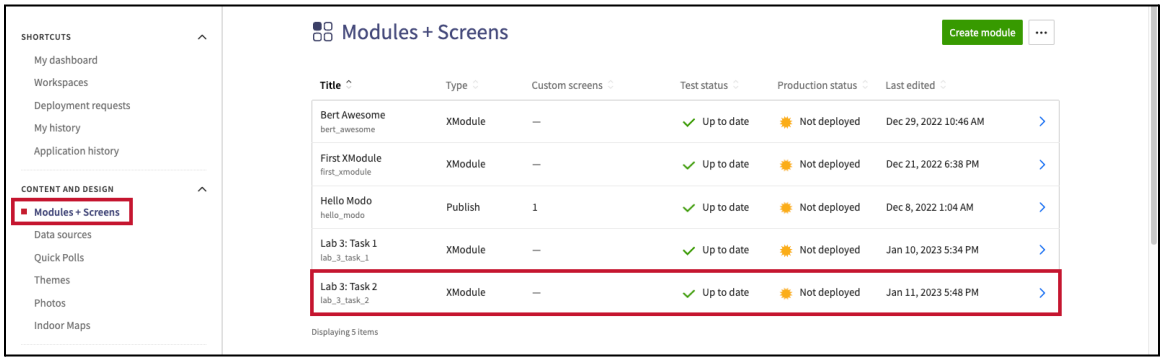
11. Click the JSON button, type a , at the end of line 5 then create a new line before the closing brace. Paste the code snippet below to enable jwtStandardClaims and map new values to your JWT.

```
"jwtIncludeStandardClaims": true,
"jwtClaimsMap": {
    "race": "race",
    "class": "class",
    "photo": "photo",
    "gold": "gold",
    "skills": "skills",
    "power": "power"
}
```

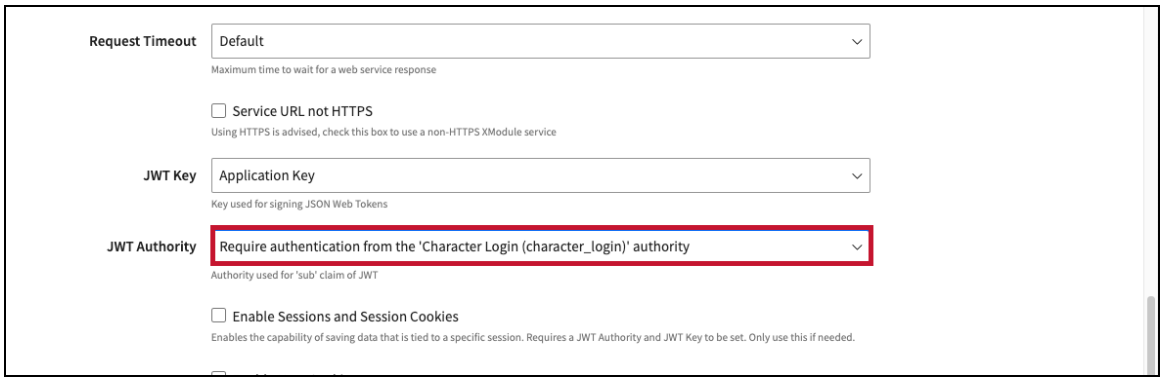
12. Verify that your Advanced Editor settings look like the one below then click **Save**.



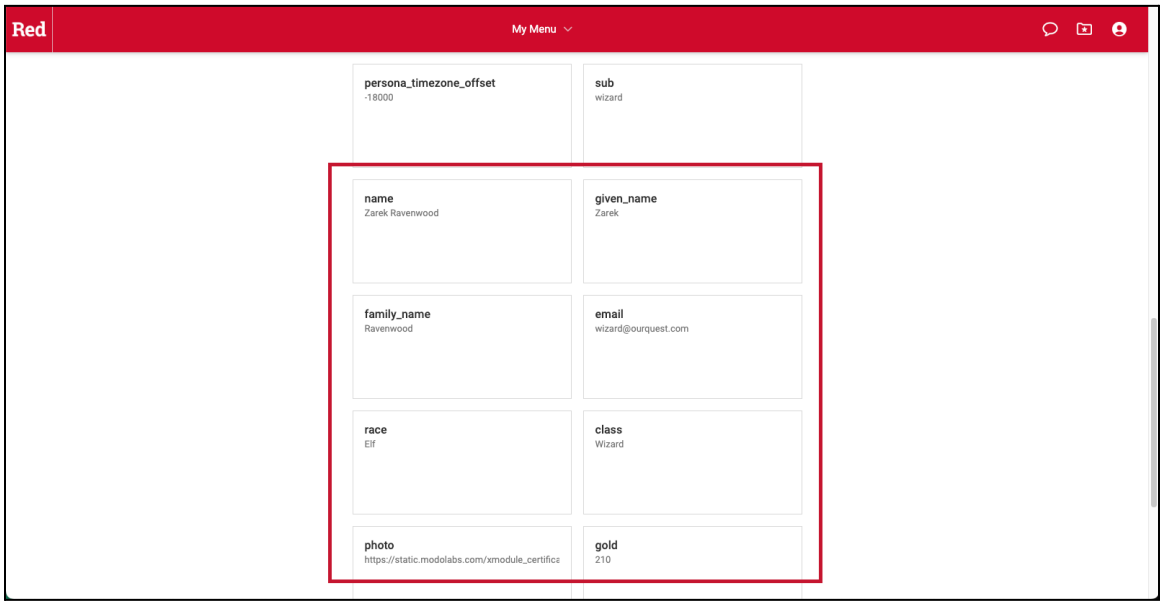
13. Deploy your changes to Test then click **Module + Screens**, **Lab 3: Task 2**.



14. Scroll down the Configuration page and select **Require authentication from the 'Character Login (character_login)' authority** for **JWT Authority** and click **Save**.



15. Deploy your changes to test then refresh the Lab 3: Task 2 preview page and view the 2nd tab to verify that new values have been added to our JWT payload information.



16. A great way to display the currently logged in user information is by using the Nametag element. Click the following link to view the different nametag visualizations: https://xmodule.modolabs.net/default/xmodule_ui_examples/page?relativePath=.%2Fnametag.json. Select one visualization, click **View in sandbox,** and extract the select nametag element.

17. Return to your `lab3-task1` Lambda function and create a new variable named `xmNametag` below the `xmCardSet` variable and set the value equal to the contents of your clipboard.

```
    let xmNametag = {
            "elementType": "nameTag",
            "id": "custom_style",
            "nameTagStyle": "horizontal",
            "name": "Ada Imani",
            "nameFontSize": "large",
            "nameFontWeight": "bold",
            "description": "B2B Manager",
            "imageHeight": "large",
            "imageWidth": "large",
            "imageBorderRadius": "medium",
            "image": {
                "url": "https://source.unsplash.com/i2hoD-C2RUA",
                "alt": "Photo of Ada Imani"
            },
            "accessoryButton": {
                "actionStyle": "emphasized",
                "title": " Contact ",
                "borderRadius": "full"
            }
        };
```

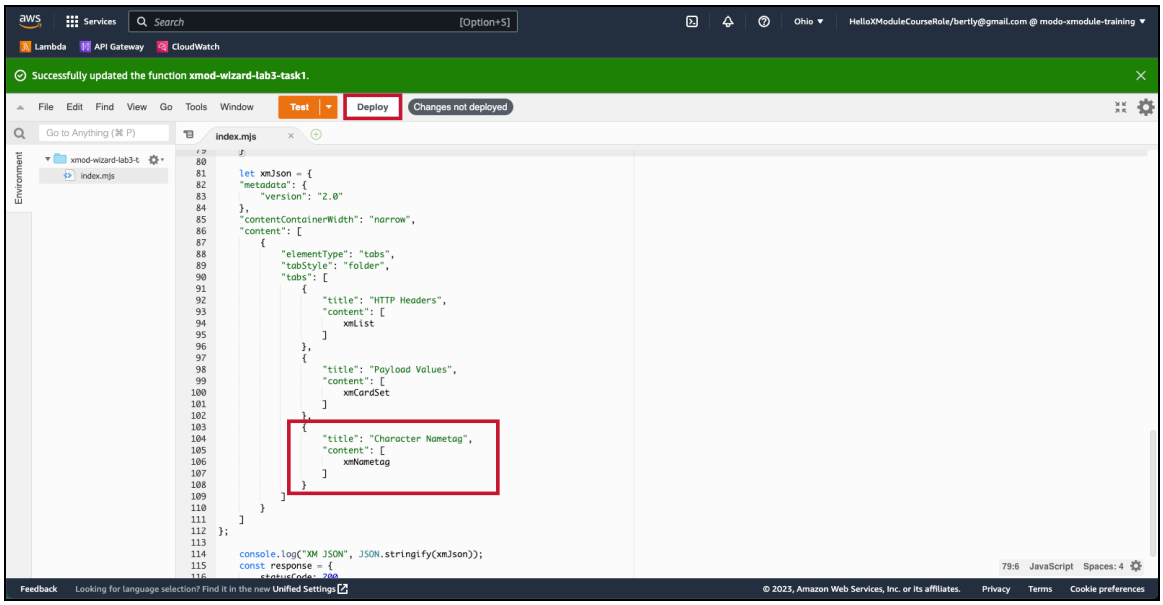
Note: This initialized the object to have values so if we are logged out we will still display properly without causing issues.

18. Add the following code before the closing brace for your payload checking if statement:

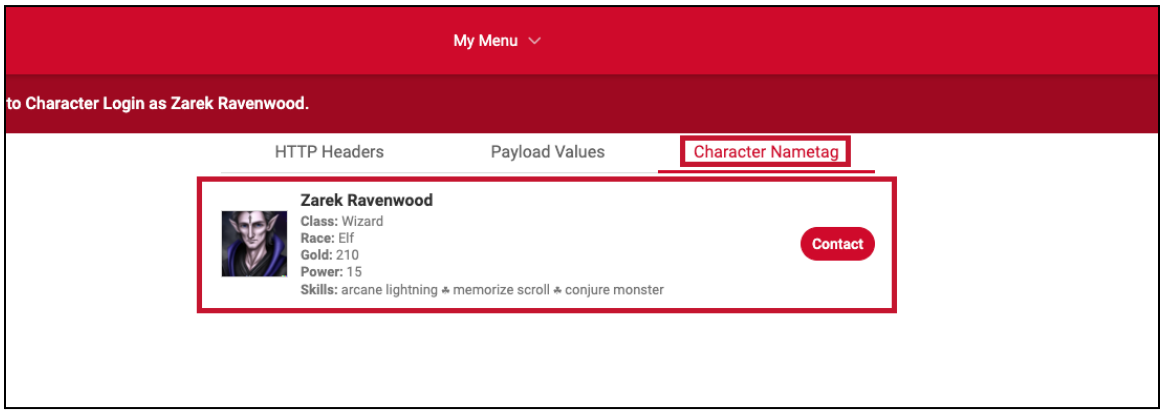
```
        //make sure they are logged in and have a name attribute before continuing
        if ('name' in payload) {
            xmNametag.name = payload.name;
            xmNametag.description = `<b>Class:</b> ${payload.class}<br><b>Race:</b>
${payload.race}<br><b>Gold:</b> ${payload.gold}<br><b>Power:</b>
${payload.power}<br><b>Skills:</b> ${payload.skills.join(" &#9752; ")}`;
            xmNametag.image = { "url": payload.photo, "alt": `Photo of ${payload.name}`
};
        }
```

19. Make sure the contents look like the image below then scroll down to the 3rd tab.

20. Change the title of tab 3 to Character Nametag and add the xmNametag variable to the content array for the 3rd tab then click Deploy.



21. Refresh your Lab 3: Task 2 preview and click the Character Nametag tab to view the nametag.



22. If you log out and back in a couple of times as different characters you can see the nametag update. The passwords are all the same and the application identifiers include (cleric, druid, fighter, monk, paladin, ranger, rogue, sorcerer, warlock, and wizard).

Congratulations! You have successfully set up a test authentication authority and configured your XModule to consume that information. The YAML authority is a great tool for testing. You can use it to match the production authentication authority's properties and test various user experiences.

Task 5: Authentication and the Modo Communicate Directory

Context: The Modo Communicate directory allows you to send personalized messages through the Modo application directly to specific recipients based on their logged in attributes. The Modo Communicate Directory is used to store user's My Circle / Buddy List information and can be used as a lightweight, user-specific database in your Modo application.

Goal: Configure the IdP to auto populate the Modo Communicate directory.

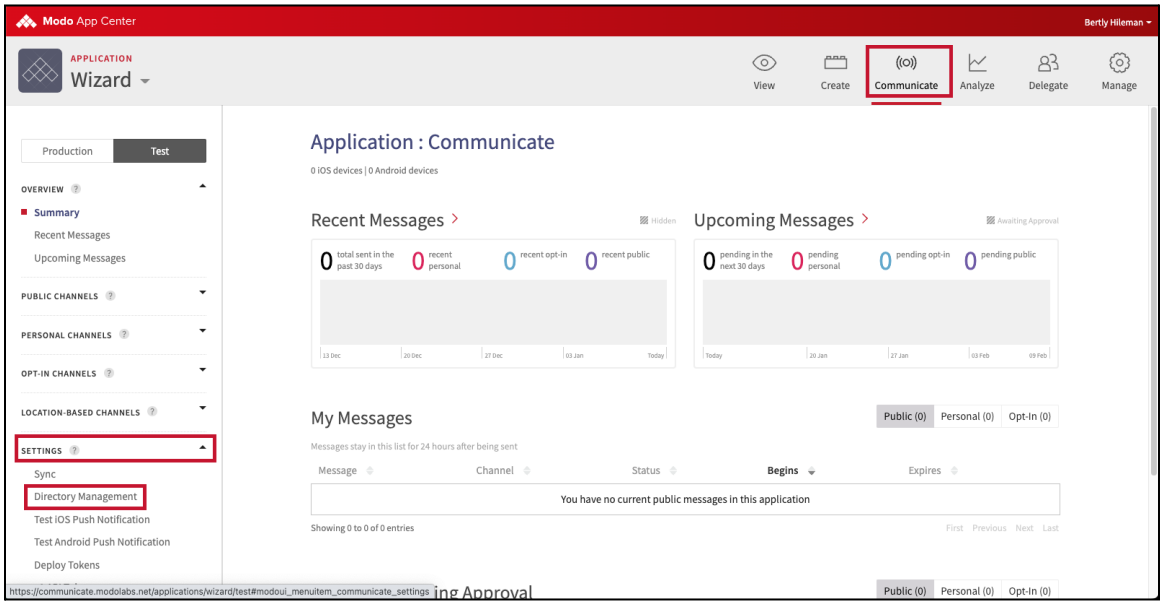
Task Outline:

- Configure the Communicate directory in test to hold string attributes for Gold, Class, Race, and Power and an array attribute for Skills
- Connect Messaging to the Character Login authority

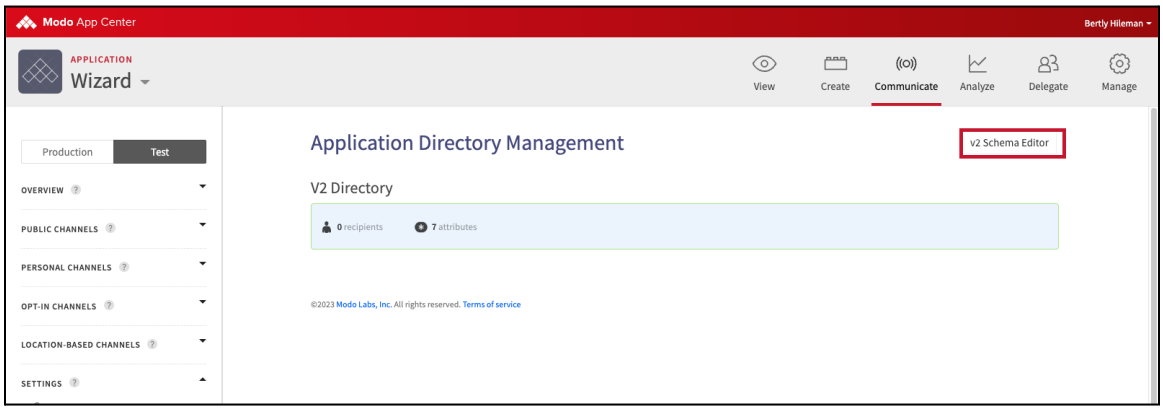
- Configure the authentication authority to automatically populate that information when the customer logs in

Steps:

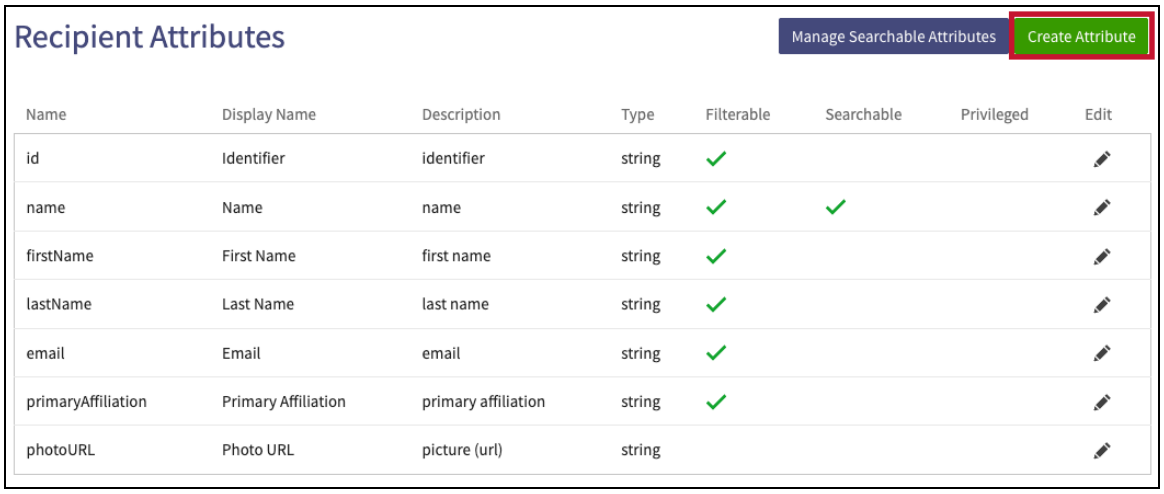
1. Click **Communicate** in Modo App center, click **Test**, click to expand **Settings** then click **Directory Management**.



2. Click **v2 Schema Editor**.

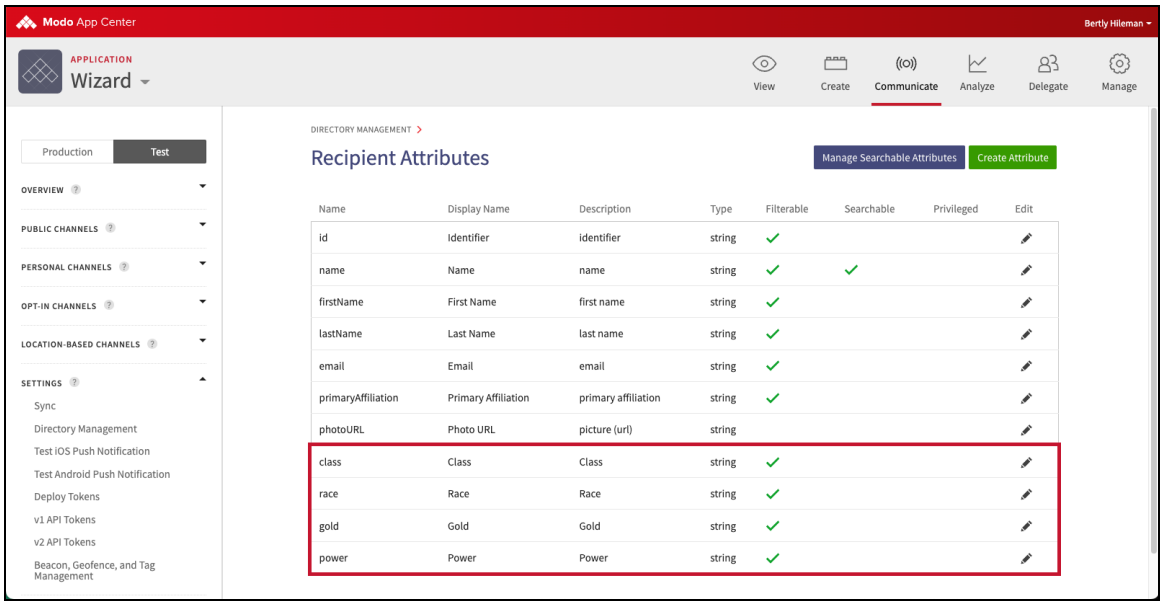


3. For these next steps we need to edit the Schema to have the attributes we want to include in the Communicate Directory. Some attributes are automatically added when we link authentication to the Directory but others require mapping and the Directory Schema needs to be updated to allow for those attributes. We are going to include the following attributes:
    a. Class
    b. Race
    c. Gold
    d. Power
    e. Skills
4. To create the class attribute do the following:
    a. Click **Create Attribute**

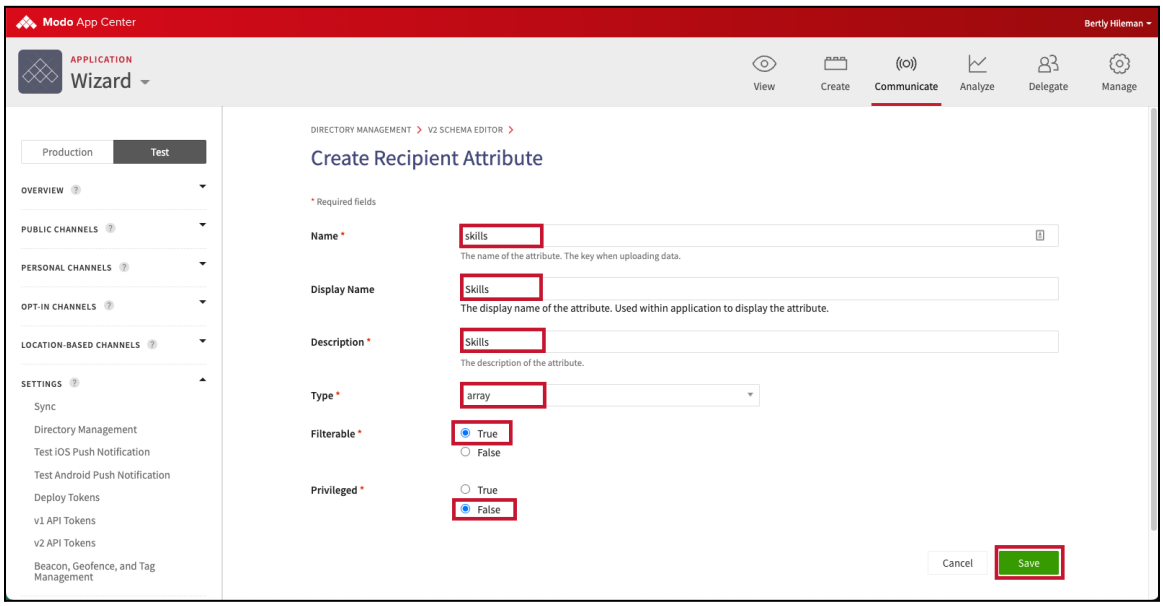


b. Type **class** for **Name**
c. Type **Class** for **Display Name**
d. Type **Class** for **Description**
e. Select **string** for **Type**
f. Select **True** for **Filterable**
g. Select **False** for **Privileged**
h. Click **Save**

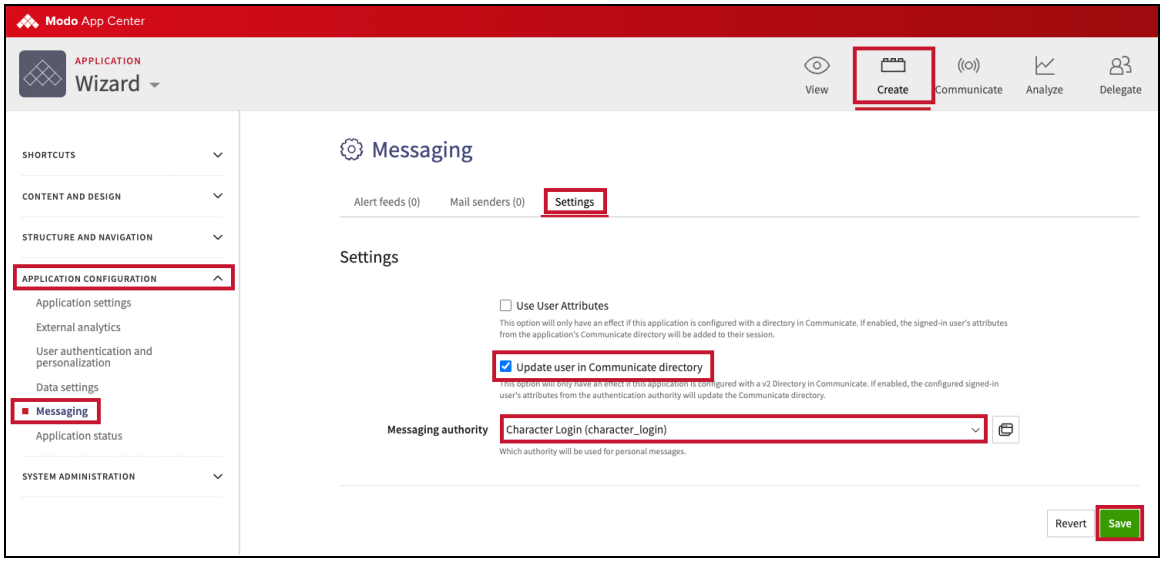
5. Repeat this process for **Race**, **Gold**, and **Power**.
6. Verify that all the attributes are as shown in the image below:



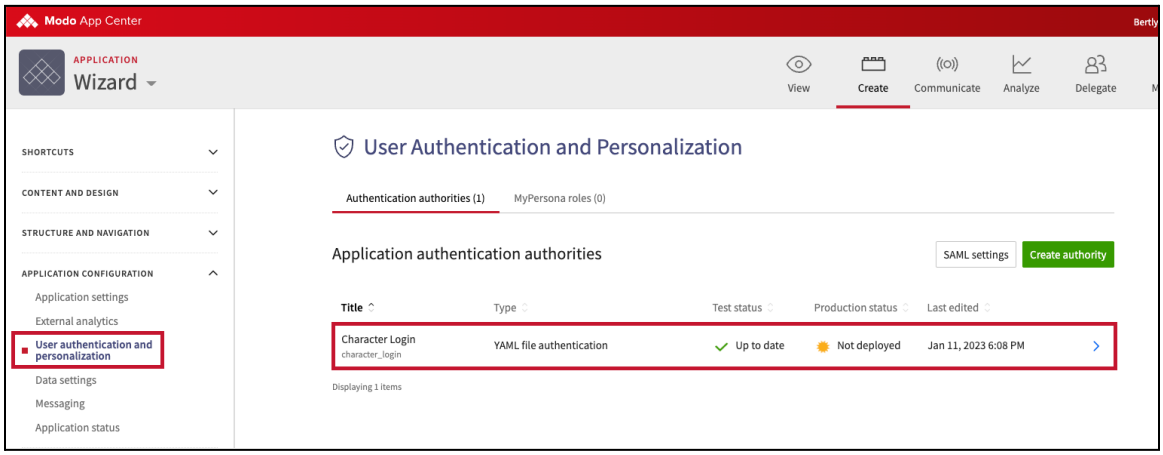
7. Click **Create Attribute** to start creating the skills attribute then complete the attribute by doing the following:
    a. Type **skills** for **Name**
    b. Type **Skills** for **Display Name** and **Description**
    c. Select **array** for **Type**
    d. Select **True** for **Filterable**
    e. Select **False** for **Privileged**



8. With the schema set, we are ready to link the Communicate Directory to the IdP. Click **Create**, click to expand **Application Configuration**, click **Messaging**, click **Settings**, click to check **Update user in Communicate directory**, select **Character Login** for **Messaging authority** then click **Save**.



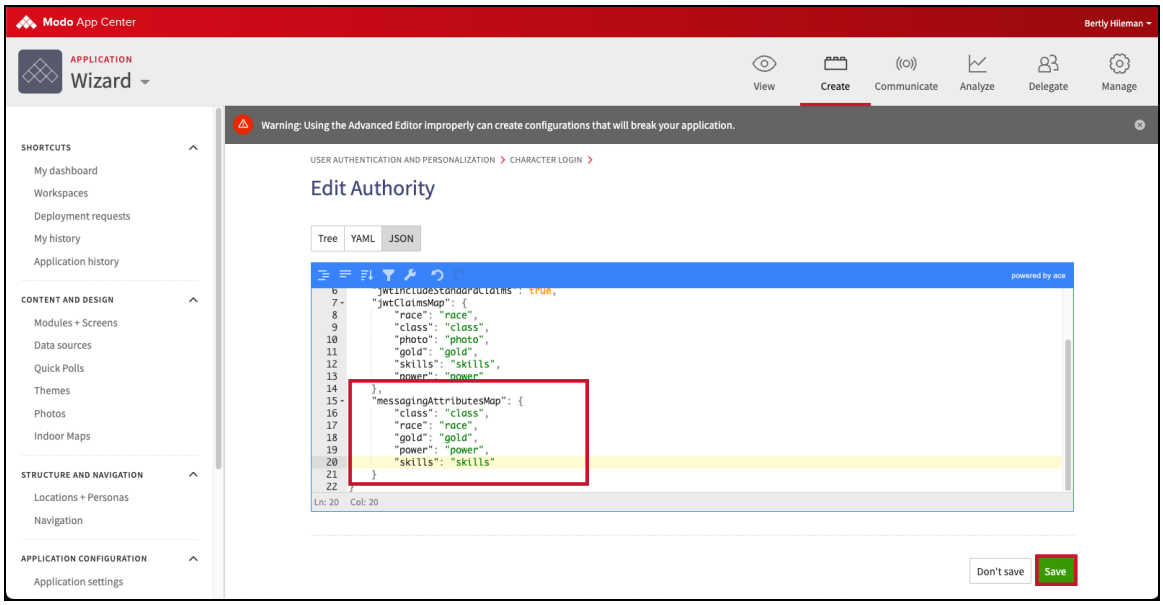
9. Under **Application Configuration**, click **User authentication and personalization**, then click the **Character Login** authority.



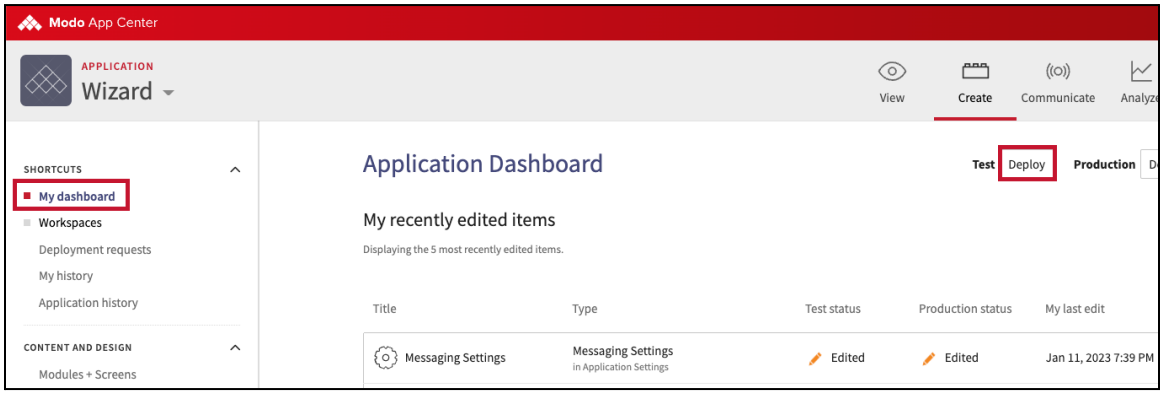
10. Click the Advanced Editor, then click JSON, and add the following entries below the `jwtClaimsMap` object:

```
  "messagingAttributesMap": {
      "photoURL": "photo",
      "class": "class",
      "race": "race",
      "gold": "gold",
      "power": "power",
      "skills": "skills"
  }
```

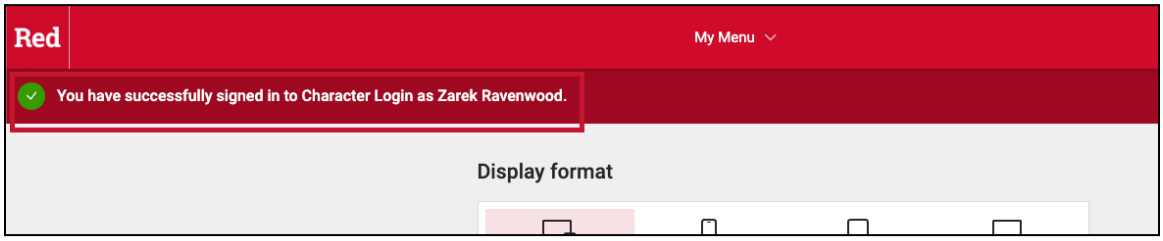
11. Make sure the **Edit Authority** screen looks similar to the one below then click **Save**.



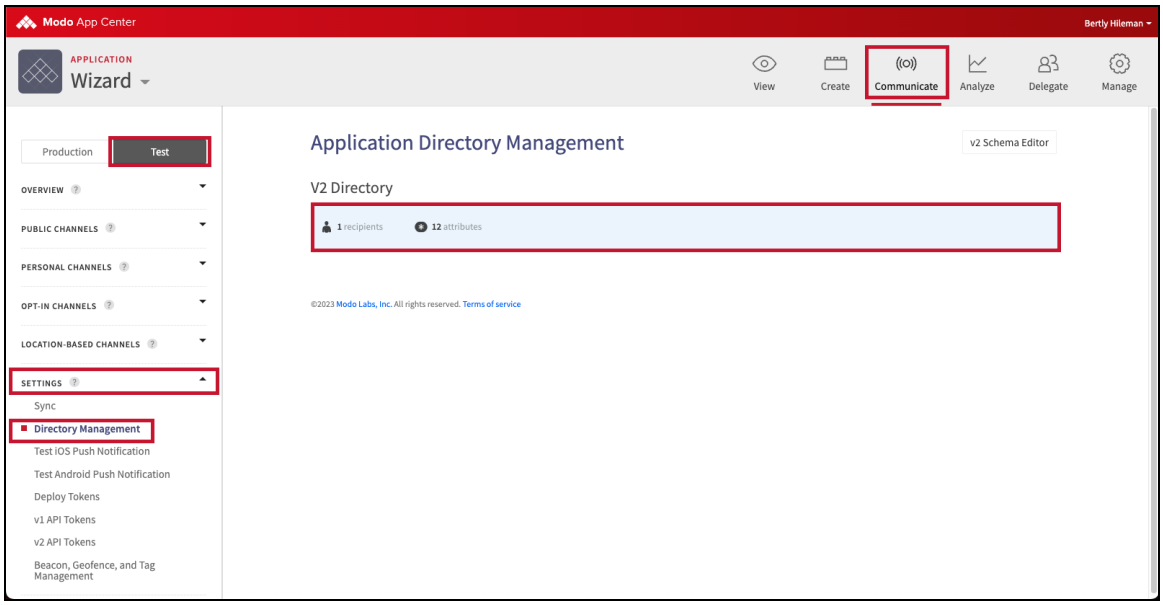
12. Under **Shortcuts**, click **My Dashboard**, then click to **Deploy** all outstanding changes to **Test**.



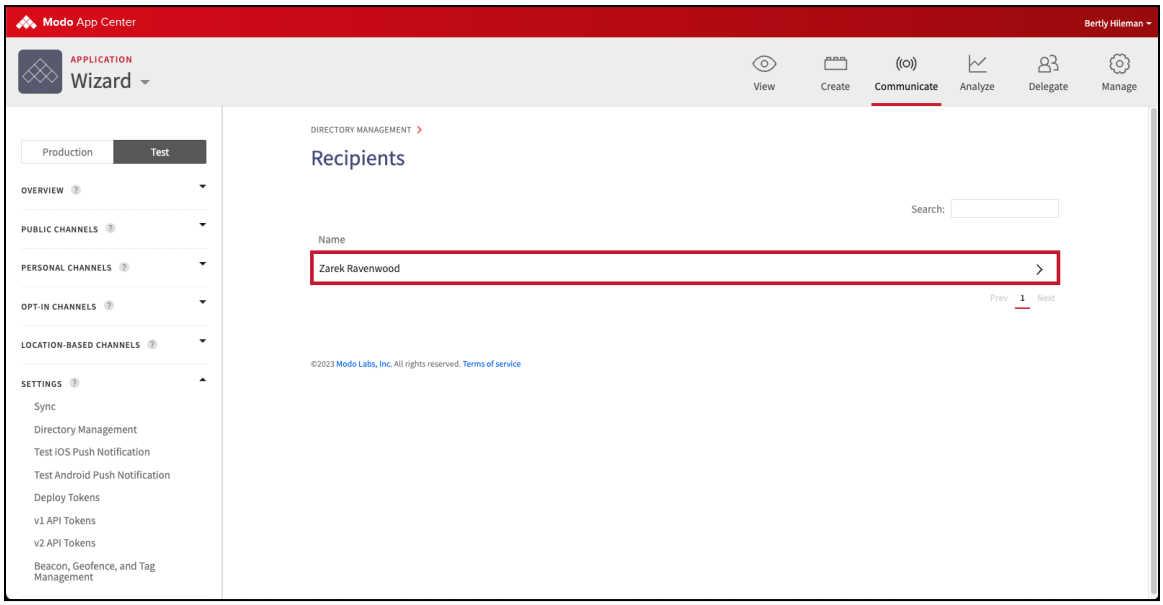
13. Return to the Lab 3: Task 2 preview screen, sign out and then sign back in.



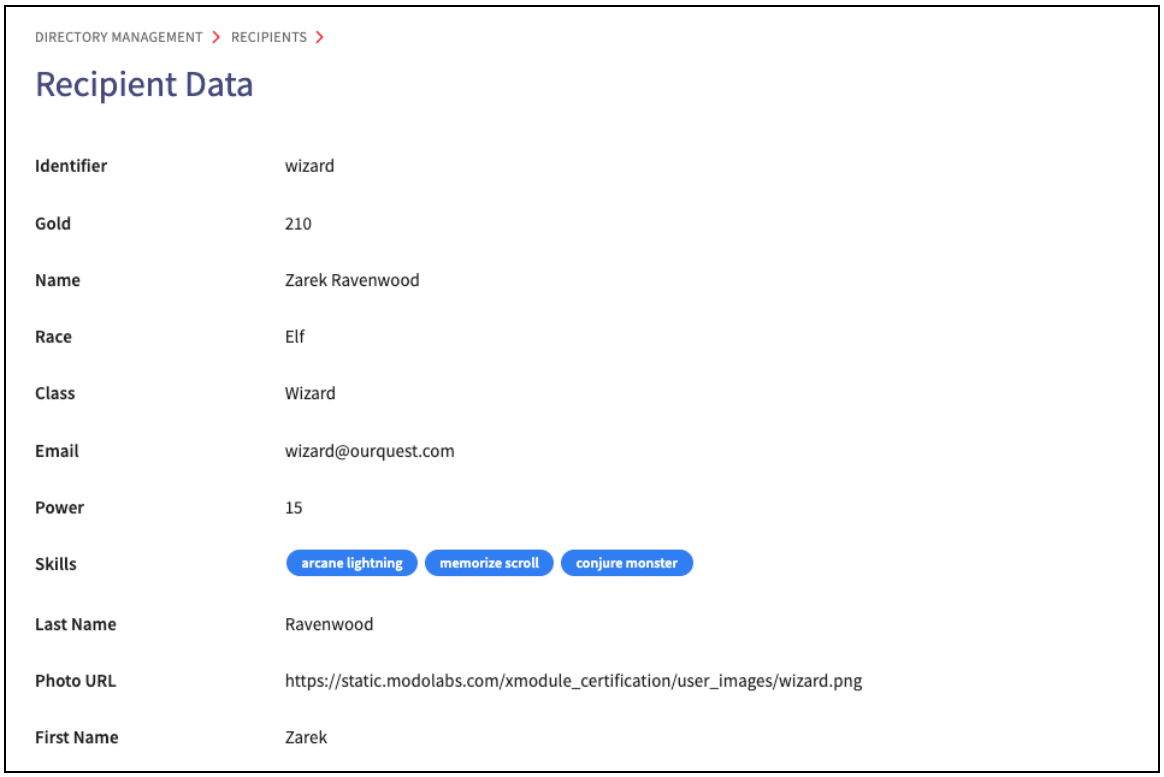
14. In Modo App Center, click Communicate, click Test, click to expand Settings, then click Directory Management. You should see one recipient in the directory, click the Directory to view the recipients.



15. Click the recipient to view their information.



16. Verify that all your same attributes are showing. You can also log in as other characters to help populate the Communicate Directory.



Conclusion: These tasks have helped you follow best practices to decode the JWT and have helped you make connections between the Authentication Authority and the Modo application. You can now pass your user data in the JWT and automatically populate the communicate directory with the same source.

Lab 4 - RelativePath - AJAX Content and other link types

Estimated Time: 4 hours

Throughout XModule you will see a `relativePath` declaration. The relativePath is used in XModules to determine the destination for additional web service calls. These paths must be in relation to the initial URL provided to create the XModule. The minimum relationship is that they are in the same domain name. This ensures that the person creating the calls are making those calls to the same web service. If you want to use URLs from different domains to build the same page, you either need to request the data programmatically or you need to use an XComponent that allows you to create multiple XComponent blocks with their own URL schemes (more on that later).

Task 1: Create a multipage XModule

Context: A single XModule element may have multiple pages / screens as part of the experience you are building. Each of these additional pages must be part of the same web service, or at least have the same domain as the base for all the pages. The links between these screens is denoted by the relativePath declaration which is in relation to the initial page called. The relativePath could refer to the same URL but include query string parameters to determine the JSON that is rendered or it could point to a different URL location.

Consider the following URLS:

- https://docs-training-api.modolabs.net/statuspage

- https://docs-training-api.modolabs.net/statuspage/detail?id=fqj3nh4zcb4y

If the first URL was used to create the XModule, the relativePath to the 2nd page would be `detail?id=fqj3nh4zcb4y`.

While we will use just one Lambda function, we are going to use two different API Gateway methods to call the same Lambda function. We will look for a path parameter to determine which JSON we are going to render.

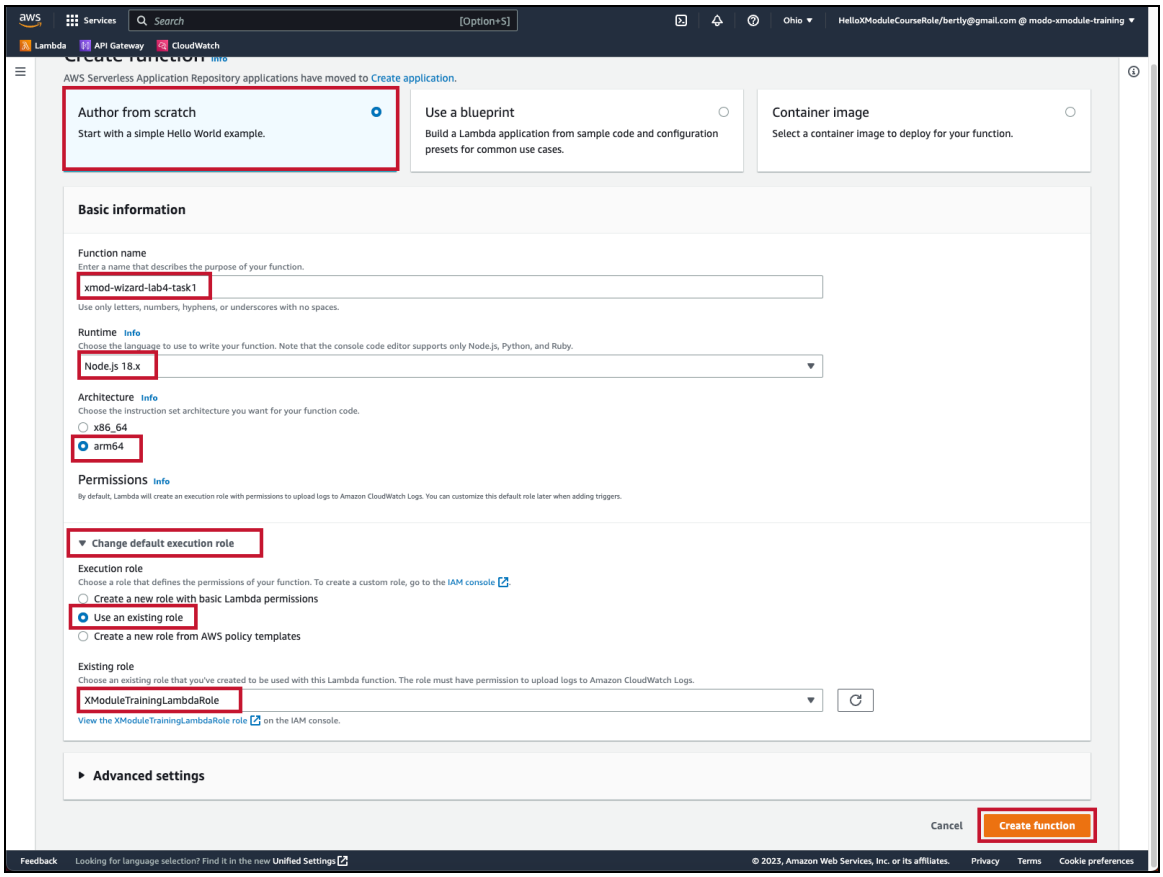
Goal: Make a quest status page with a main page and a details page to see whether a quest is available for us to start.

Task Outline:

- Create a Lambda function named **{xmod-prefix}-lab4-task1** that does NOT use a function URL that logs the HTTP Request
- Create a public REST API named **{application_identfier} API** with **/quest-status** as one resource and **/quest-status/details/{id}** as the other resource.
- Create GET methods for both of the resources above that point to the **{xmod-prefix}-lab4-task1** Lambda function with **Lambda proxy integration** checked.
- Deploy the API to a new stage named **test** and invoke the **/quest-status/details/{id}** GET method to view the contents of the page
- Add fully qualified XModule Status List JSON to your function and return the JSON from your function.
- Create a new XModule using the **/quest-status** endpoint
- Add the contents from this link as a new file named quests.mjs in your Lambda function and configure your index.mjs file to load the contents of the quests.mjs file as a variable named **data** (steps 33 - 38)
- Extract the status list as its own variable and build Status List items from the data retrieved from quests.mjs with the Status List Item's **relativePath** pointing to **"/details/" + quest.id**
- Configure your Lambda function to look for the ID path parameter to return a detail object's JSON instead of the status list
- Create a find function that gets the quest by the specified id and fills out the details using the information found in the quest

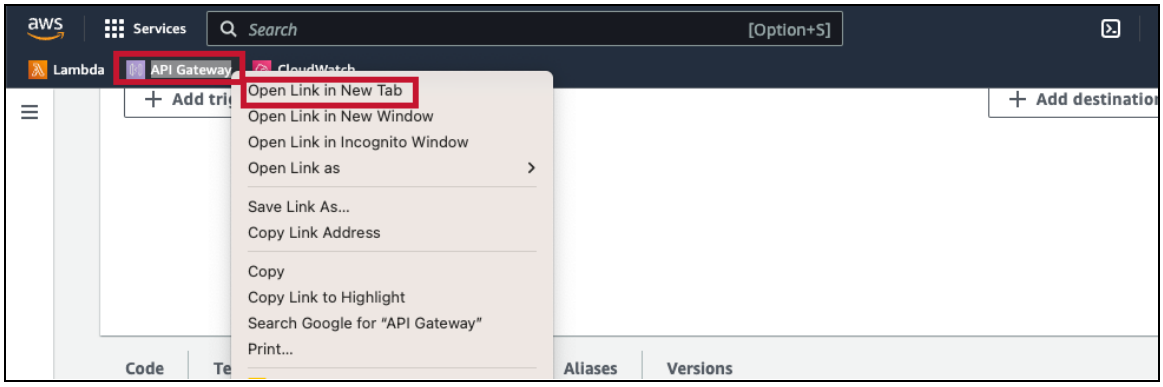
Steps:

1. Create a new Lambda function in your AWS console named **{xmod-prefix}-lab4-task1** using the settings mentioned in Task 3: Lab 1.

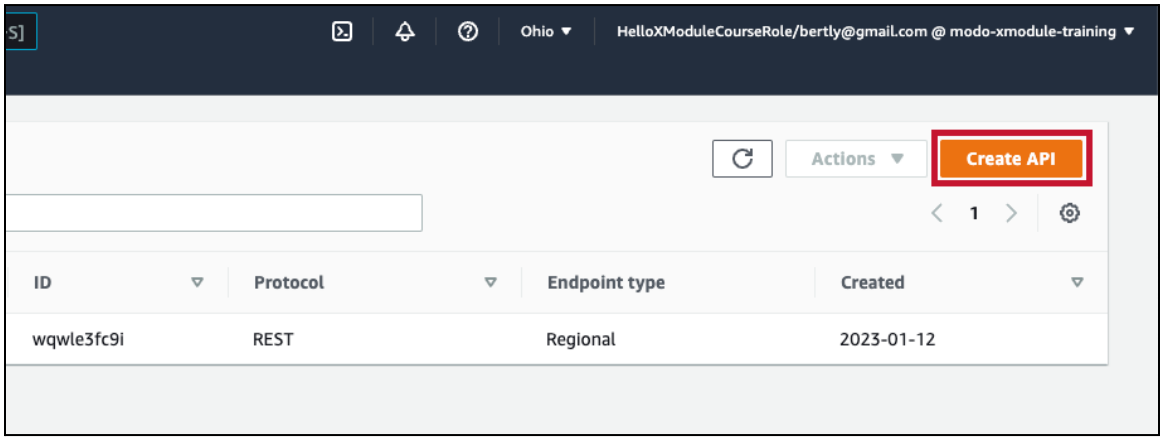


2. Replace line 2 of the Lambda function with `console.log("HTTP Request", JSON.stringify(event));` and Deploy your changes to commit them to the server.

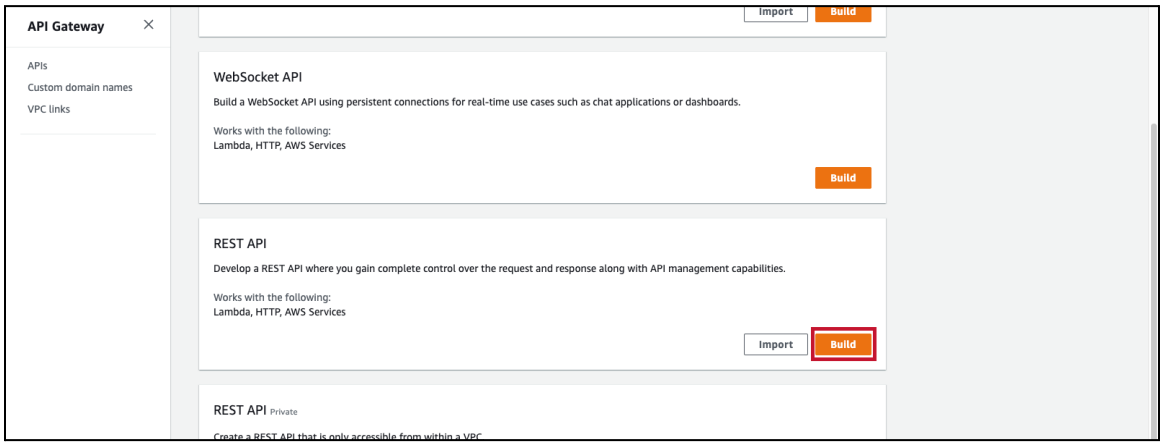
3. In order to build something that has multiple screens and has different URLs for the screen, we need to use the API Gateway to create those requests. Use the API Gateway favorite to open the API Gateway in a new tab.



4. Click **Create API**.

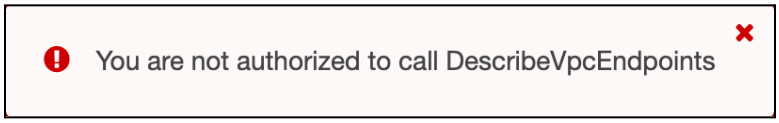


5. Scroll down to **REST API** and click **Build**.



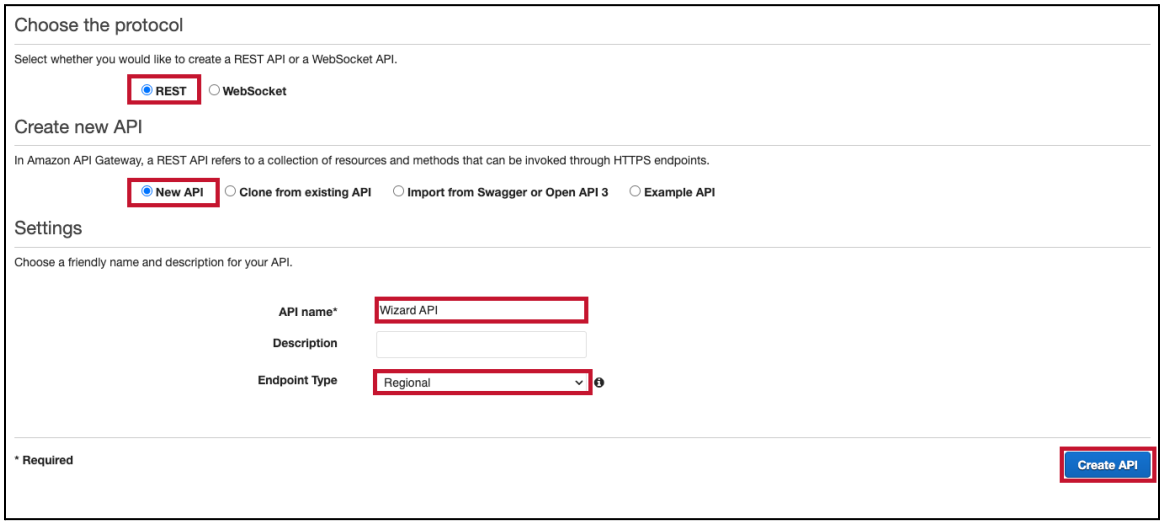
Note: Please make sure you are not using the **REST API Private** version. Those APIs are only accessible within the AWS environment.

6. If you see the following error message you can click the **x** to close the error. We are not using **VpcEndpoints**.

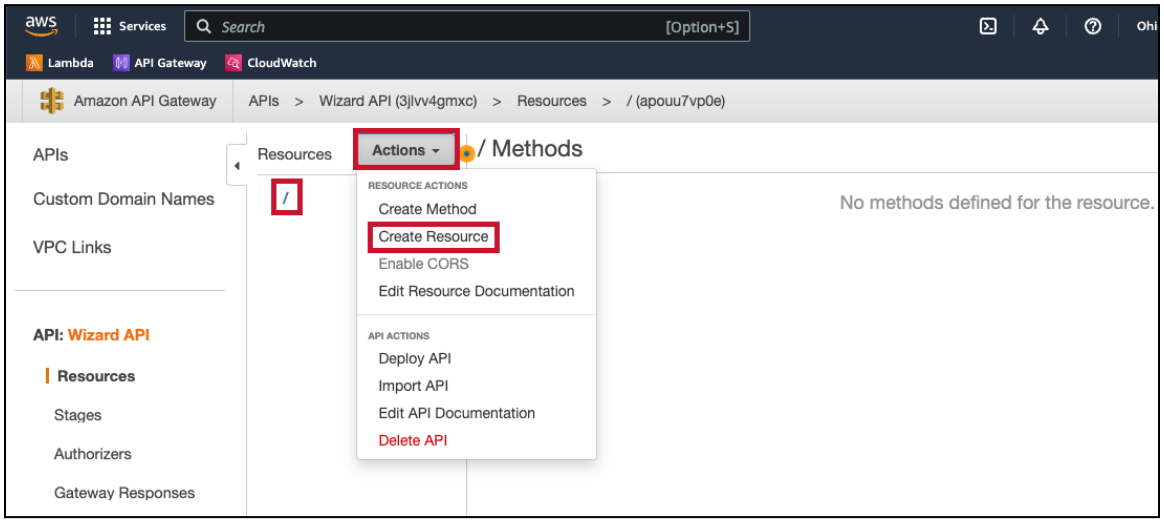


7. Build the API using the following settings:
    a. Select **REST** for **Choose the protocol**.
    b. Select **New API** for **Create new API**.

c. Type your **{application_identifier} API** where {application_identifier} is your assigned application identifier for **API name**.
d. Select **Regional** for **Endpoint Type** (the default).
e. Click **Create API**.

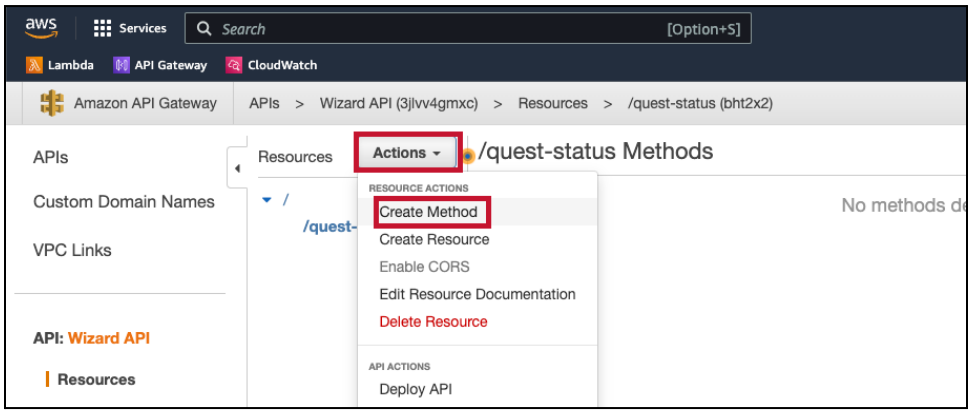


8. For the API part of this task we are going to do the following:
   a. Create a resource called **quest status** and point it to the {xmod-prefix}-lab4-task1 function we just created
   b. Create a resource under **quest status** named **details**
   c. Create a resource under **details** that accepts a path parameter named **id** that also points to the **{xmod-prefix}-lab4-task1** function.
   d. If you need instructions on this process please complete the following steps. This task will be the only task that contains the complete steps for this process. Any further API gateway tasks will assume you can repeat the following steps. Otherwise, skip to step #21.
9. The default or root node (/) is selected and we are going to create a resource on that node. Click **Action**, **Create Resource**.

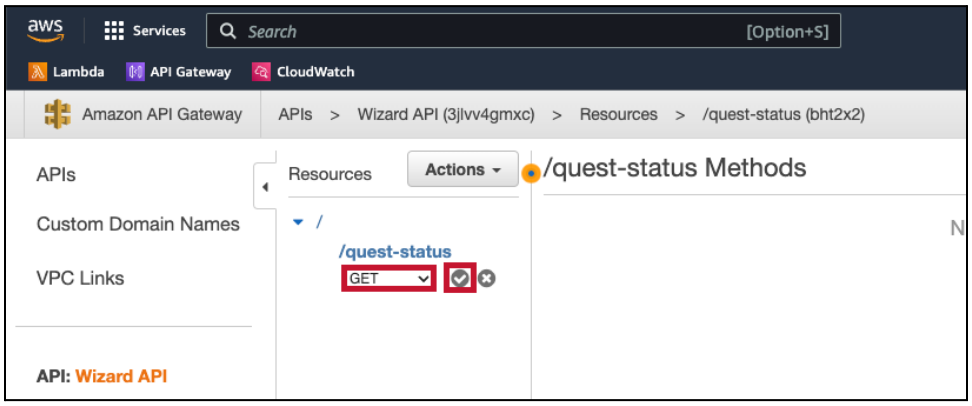


10. Type **quest status** for **Resource Name** (the Resource Path should fill automatically) then click **Create Resource**.

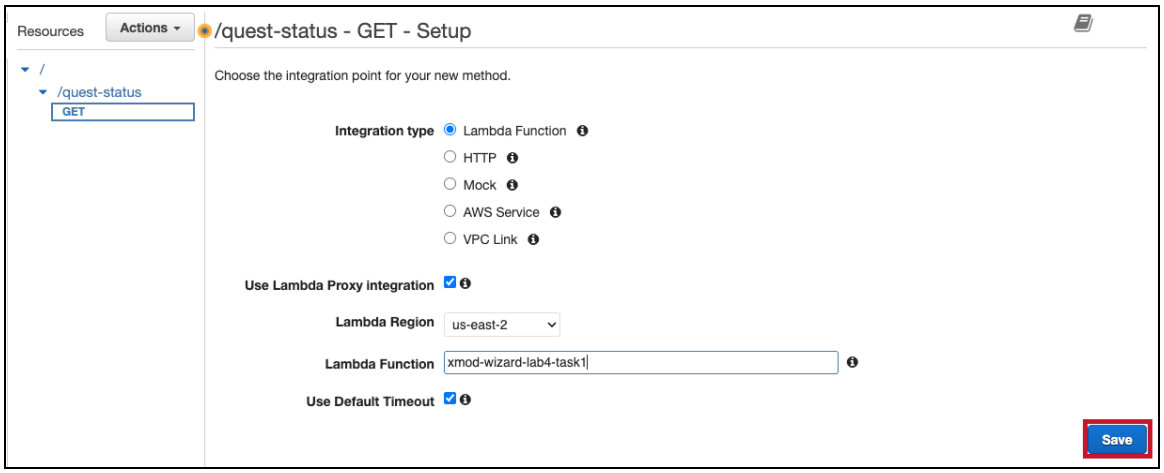
11. The new **/quest-status** resource is automatically selected so click **Actions**, **Create Method**.



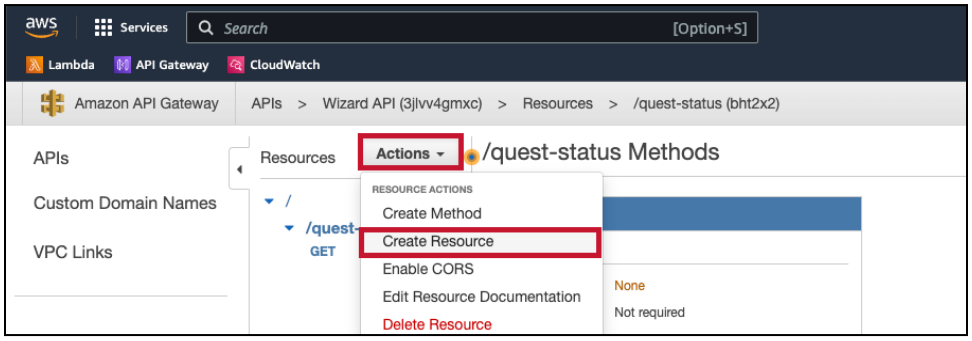
12. Select **GET** from the Method dropdown and click the check to add it.



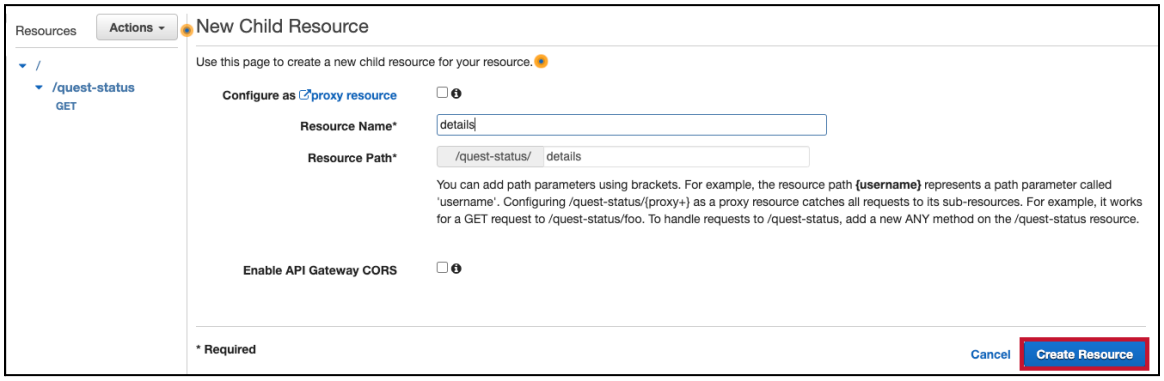
13. Complete the GET - Setup screen by doing the following:
    a. Select **Lambda Function** for **Integration type**
    b. Click to check **Use Lambda Proxy integration**
    c. Select **us-east-2** for **Lambda Region** (default selection)
    d. Type **{xmod-prefix}-lab4-task1** for **Lambda function** (it should start to autocomplete as you type)
    e. Click to check **Use Default Timeout** (default selection)
    f. Click **Save**.



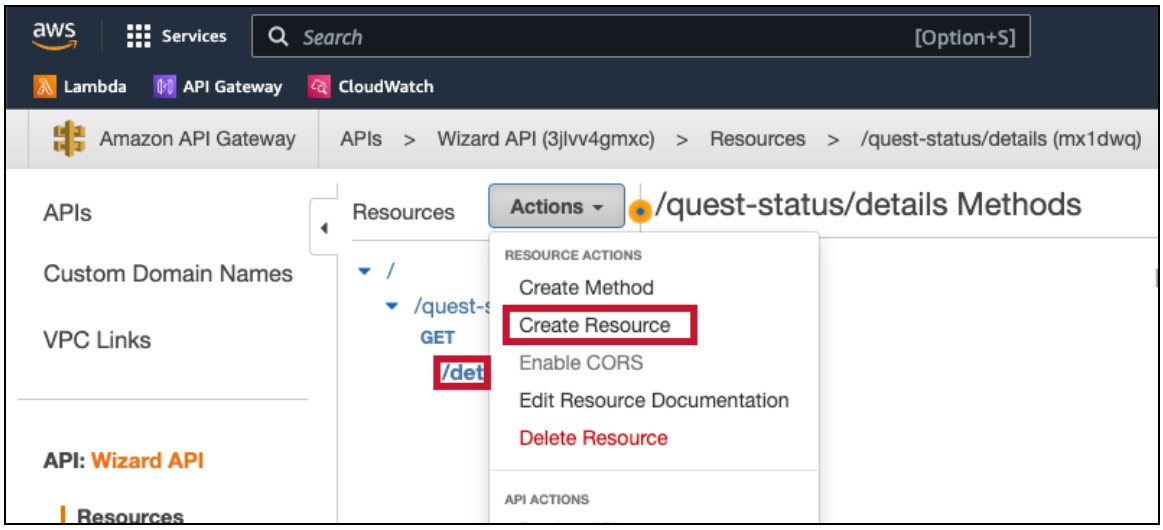
14. Click **/quest-status** in the **Resources** tree then click **Actions**, **Create Resource**.



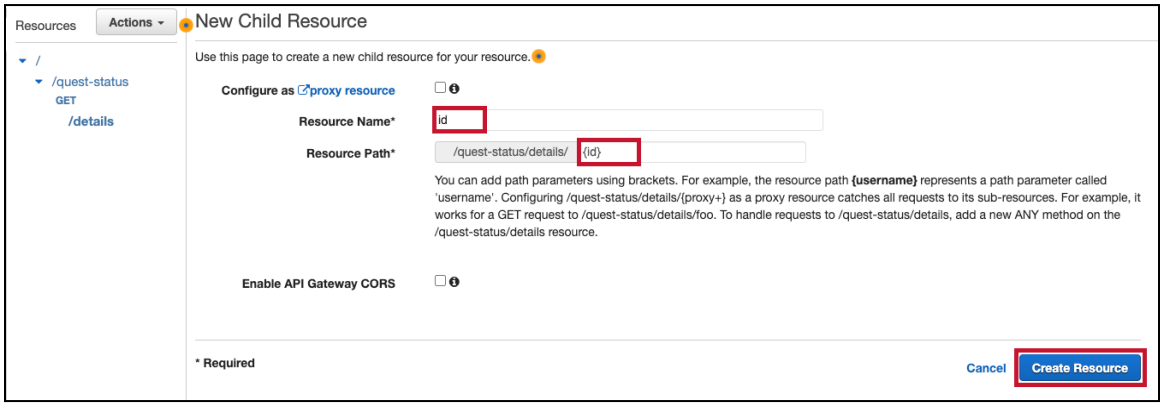
15. Type **details** for **Resource Name** then click **Create Resource**.



16. The **/details** node will be selected. Click **Actions**, **Create Resource** to make a child resource for the /details path element.

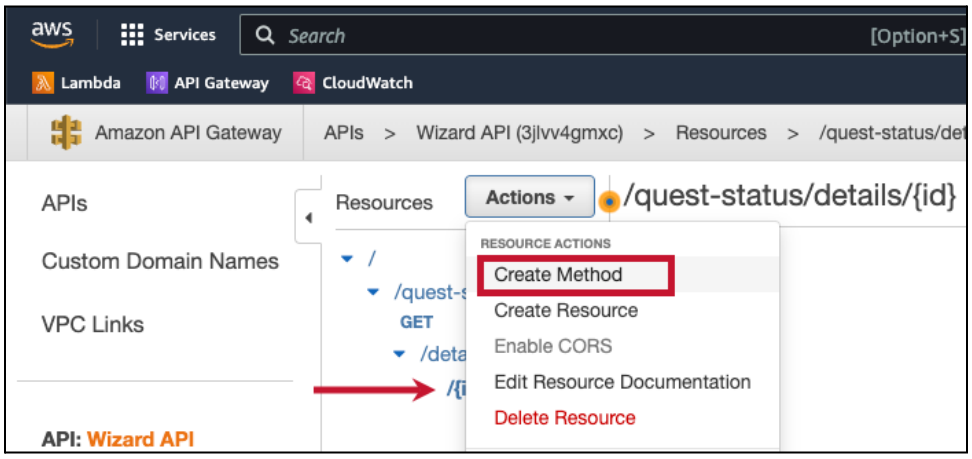


17. Type **id** for **Resource Name** then type **{id}** for **Resource Path**.



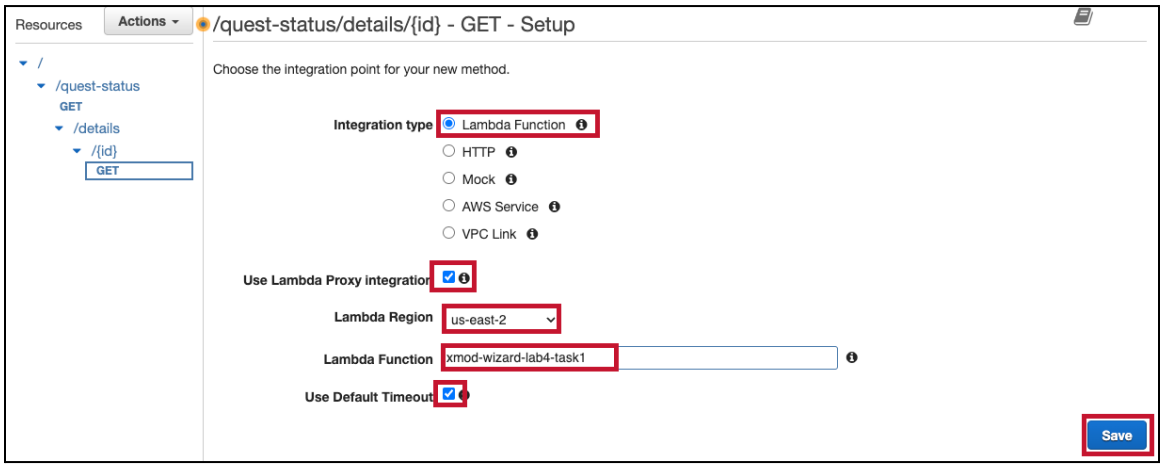
Note: Using **{id}** for the Resource Path causes the API Gateway to pass the last part of the path as a path parameter.

18. The **/{id}** path should be selected in the **Resource** tree so click **Actions**, **Create Method**.

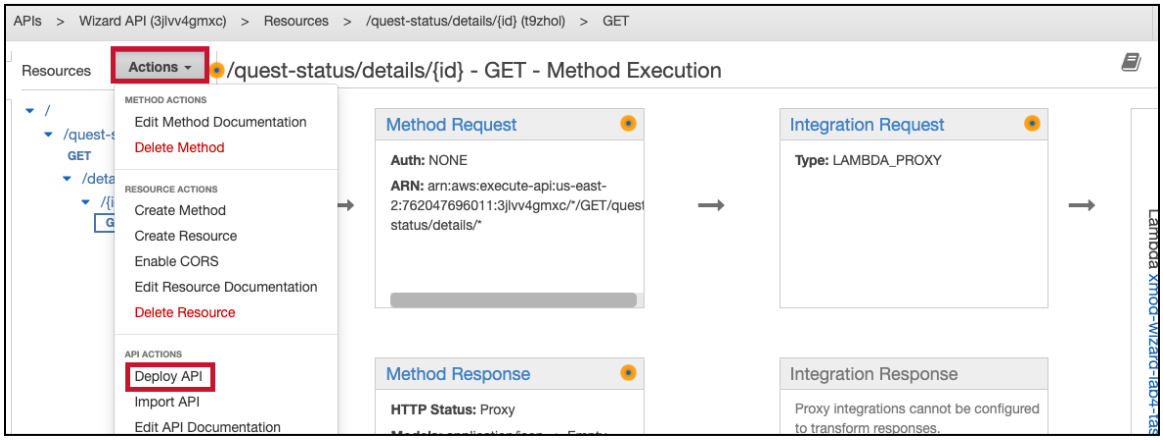


19. Select **GET** from the method dropdown then click the checkmark.

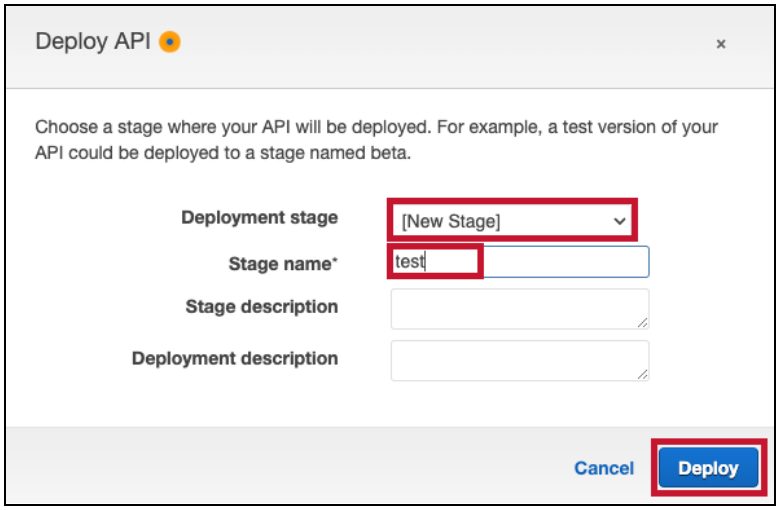
20. Complete the GET - Setup form using the same settings from step 12.



21. Now that our API gateway is set up to talk to our Lambda function, we need to deploy the API gateway. Click **Actions**, **Deploy API**.



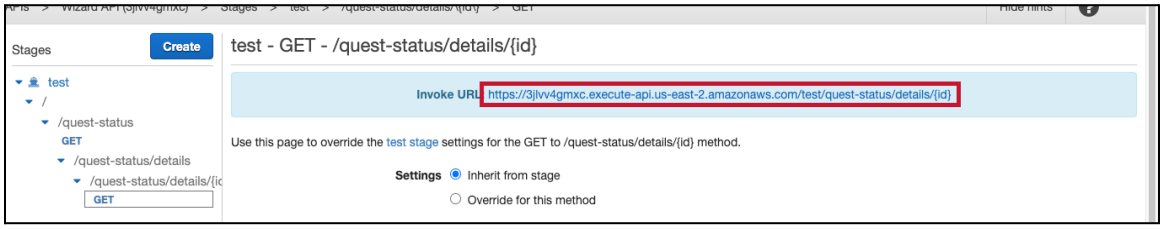
22. In the **Deploy API** window, select **[New Stage]** for **Deployment stage**, type **test** for **Stage name** then click **Deploy**.



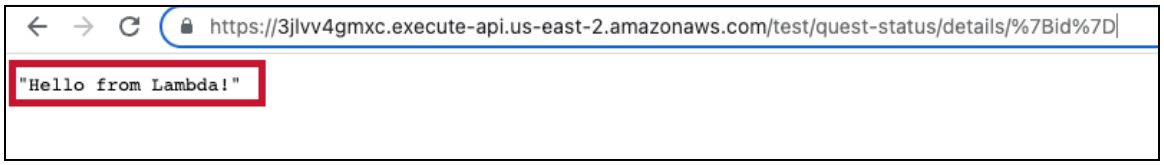
23. If you get the following error you can ignore it or click the **x** to close it.

24. In the Stages tree, click **GET** below **/quest-status/details/{id}**

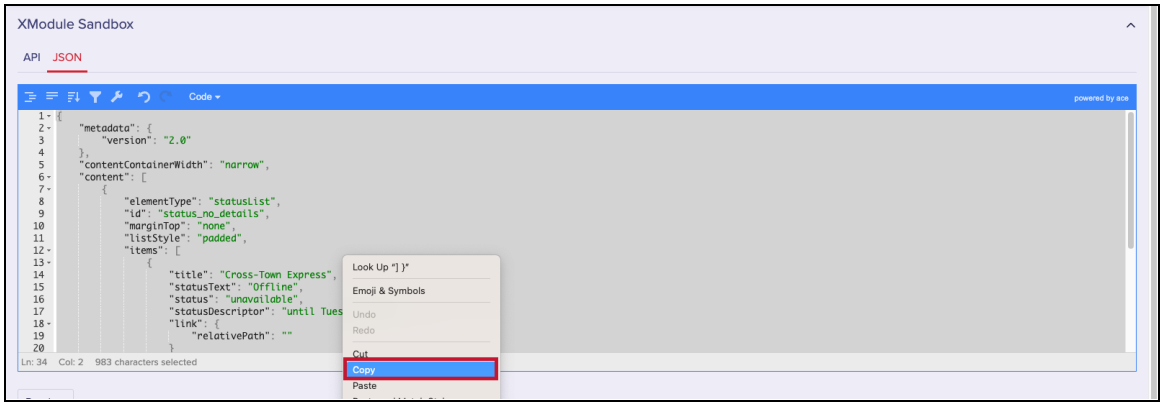
25. Click the **Invoke URL** to view the page in your browser window.



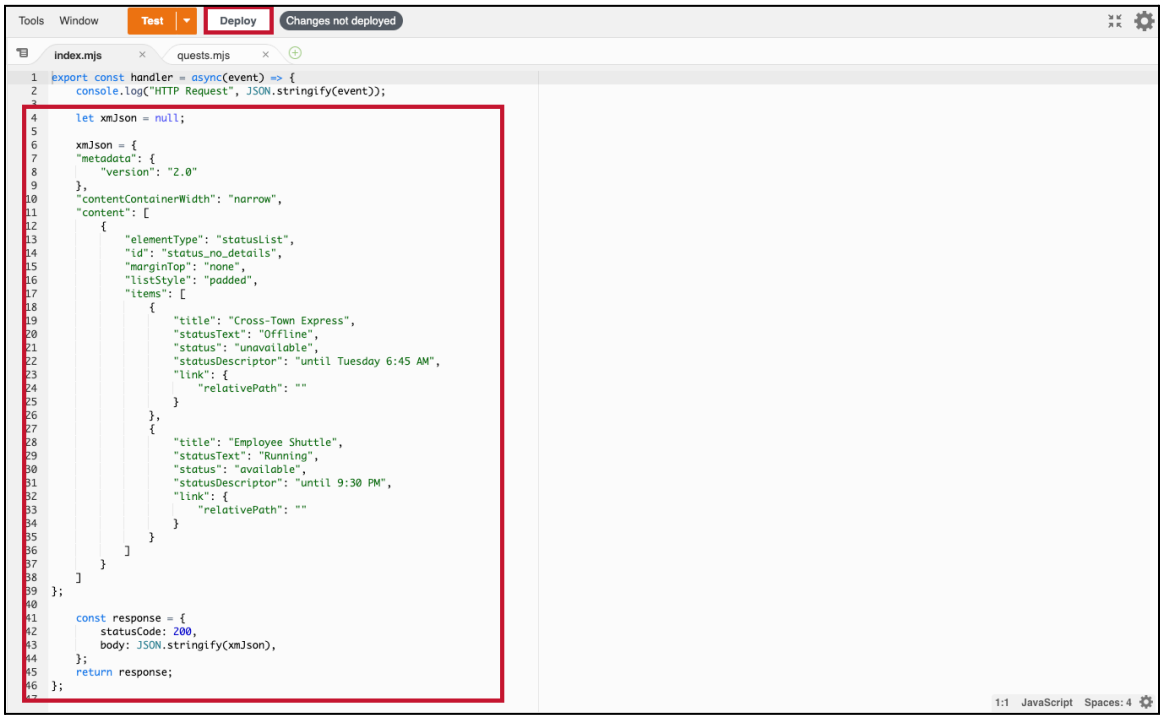
26. Verify that you see "`Hello from Lambda!`" on the screen.



27. We are done with our API Gateway for now. We need some kind of JSON to show on the screen and this is a Quest Status screen so we should use the Status List item. Click the following link to open the Status List examples: https://xmodule.modolabs.net/default/xmodule_ui_examples/page?relativePath=.%2Fstatus_list.json
28. Review the potential items, click **View in sandbox**, isolate a status list style and copy it to your clipboard.

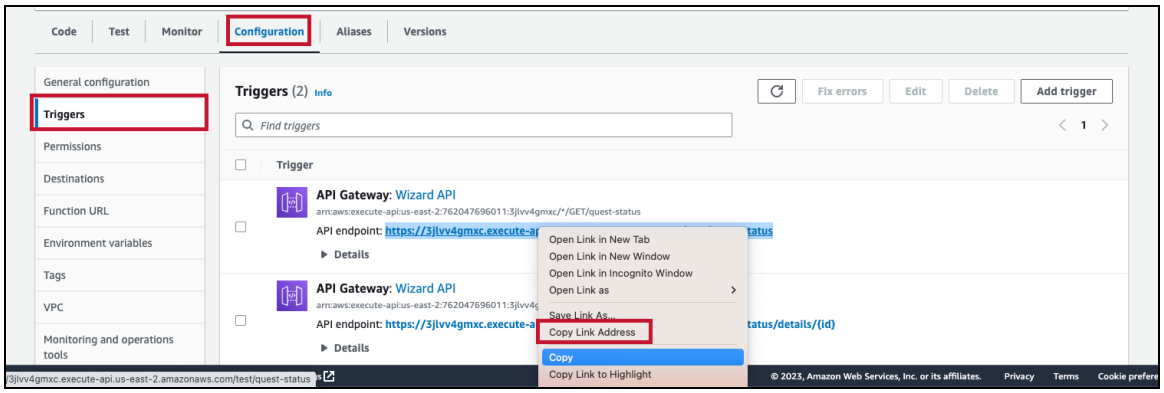


29. Return to your lab4-task1 Lambda function in the AWS Lambda console editor and create an `xmJson` variable to hold your status list JSON and set it equal to `null`, assign the `xmJson` variable the contents of your clipboard, configure the Lambda to return the Status List XM JSON element, and finally **Deploy** your changes.

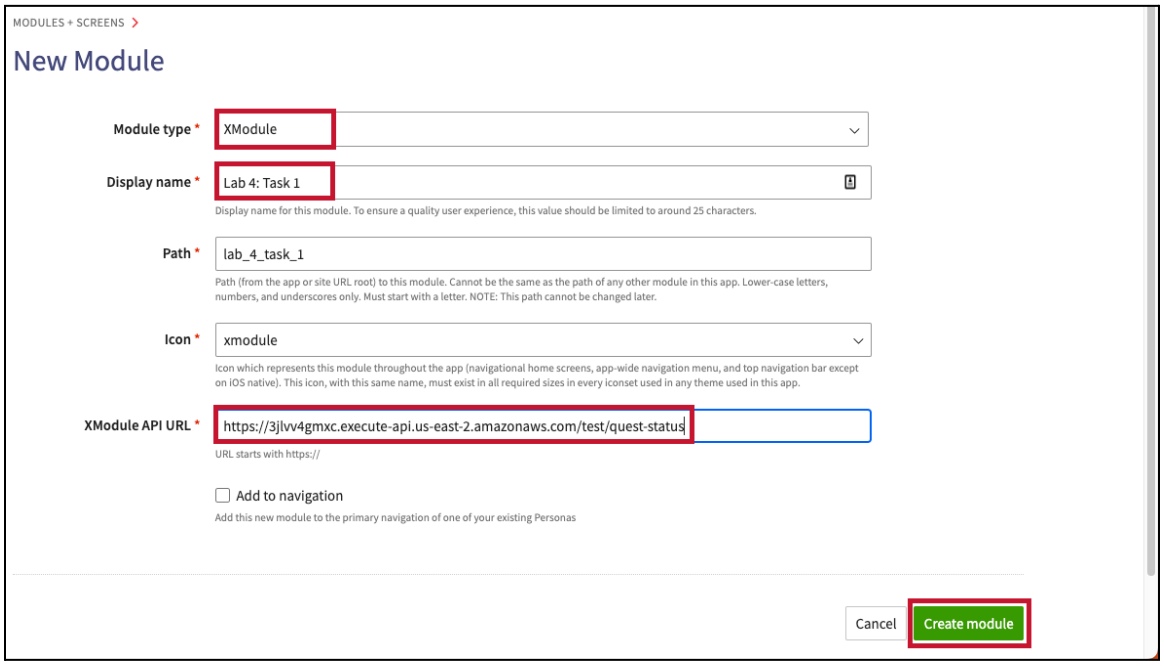


Note: In this case we are separating the variable declaration and assignment into separate steps. This may be helpful in other labs where you need to return a different JSON structure based on the type of response you are making.

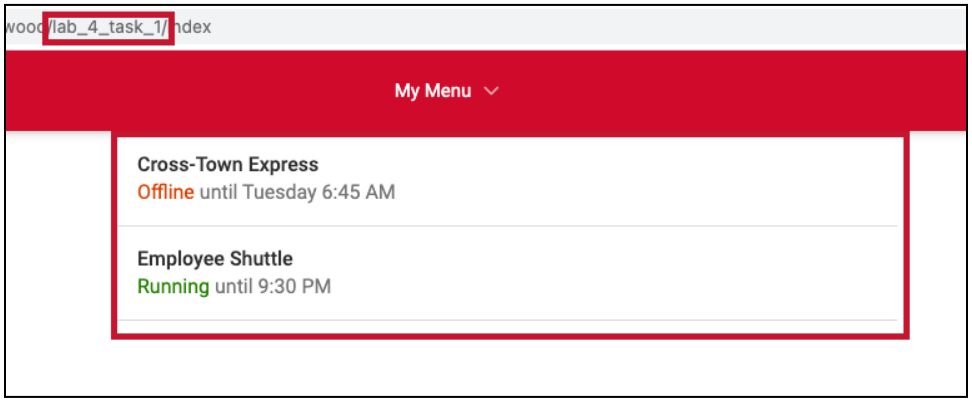
30. On the **Configuration** tab of your Lambda function you can find the API Gateway URLs in the **Triggers** section. Copy the link ending in quest-status to your clipboard.



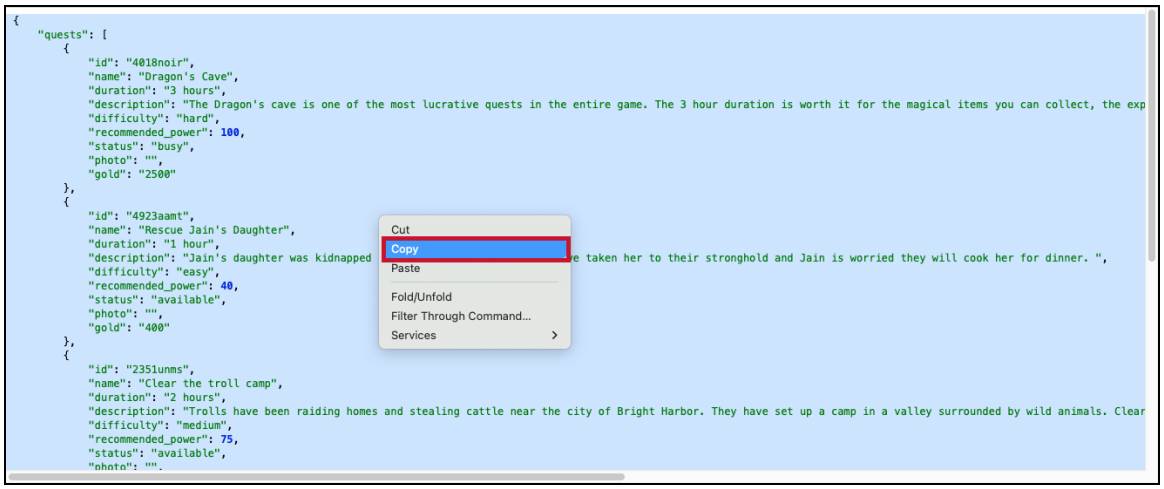
31. Create a new XModule named Lab 4: Task 1 and paste the url you copied to your clipboard for XModule API URL then click Create module.



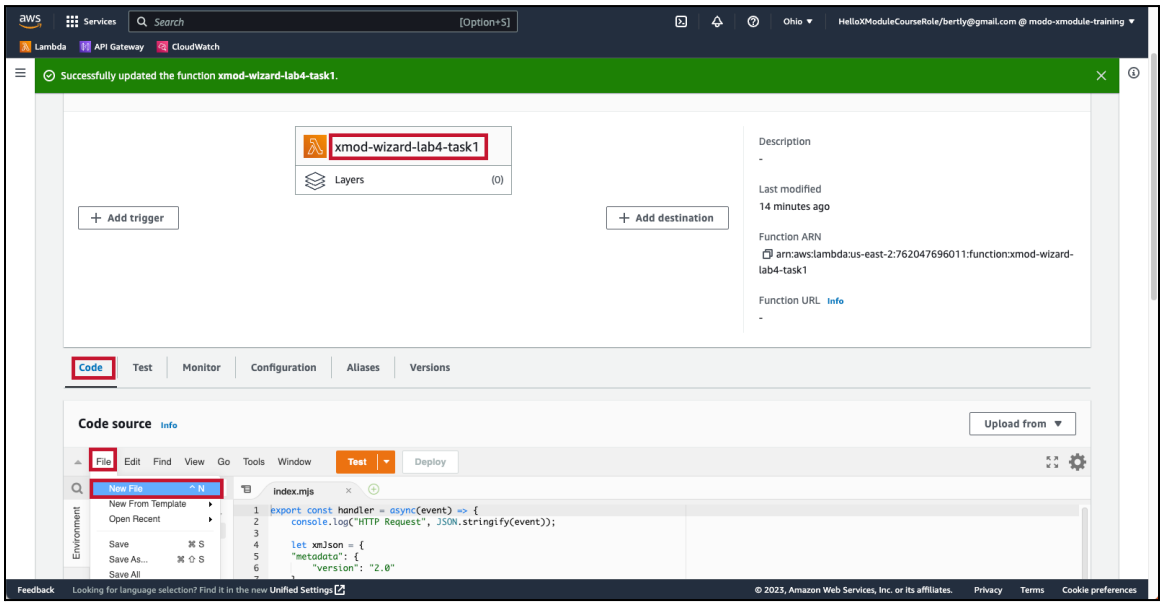
32. Deploy the Lab 4: Task 1 XModule to **Test** and preview the Module to see the status list.



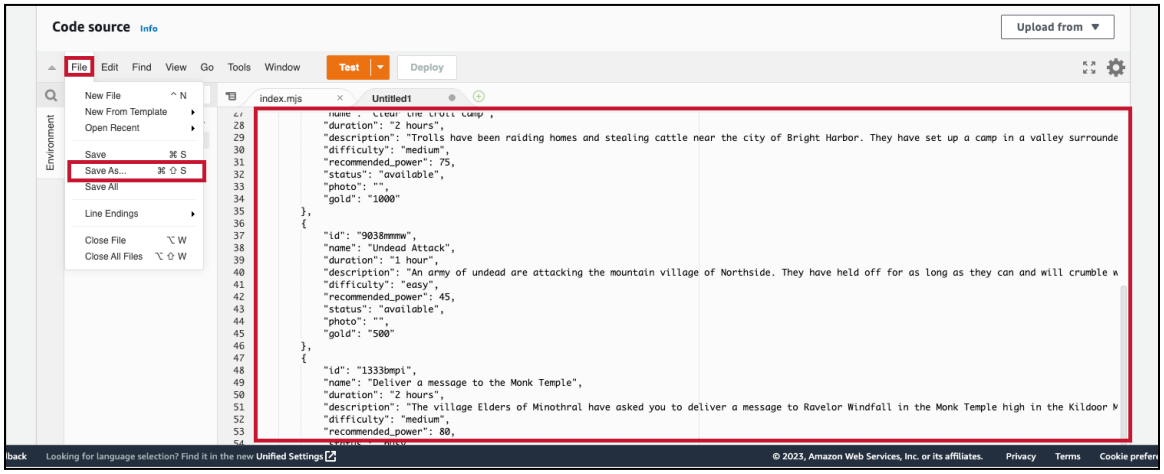
33. Now we should parse some real data and add it to our status list. Click the following link to download the xmodule_certification_quests.json file: https://static.modolabs.com/xmodule_certification/data_files/xmodule_certification_quests.json
34. Open and copy the contents of your JSON file to your clipboard.



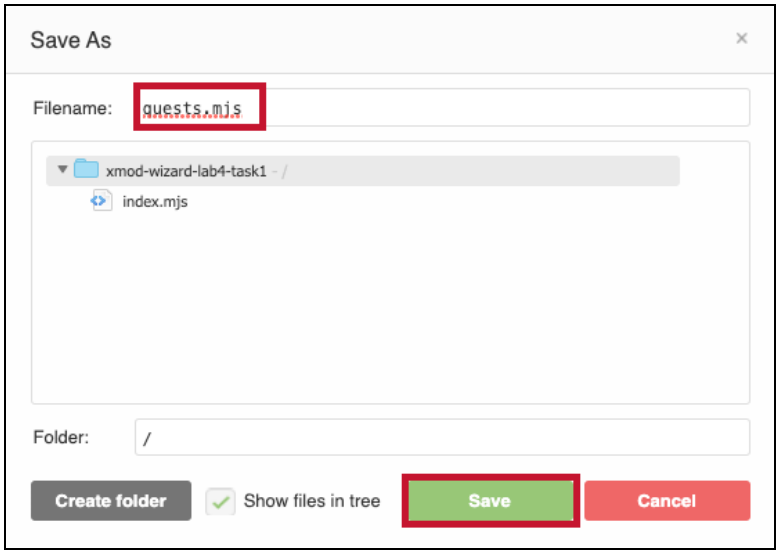
35. In your **lab4-task1** AWS Lambda console **Code** tab click **File > New File**.



36. Type **export default** on line 1 of the untitled tab then paste the contents of your clipboard starting on line 2, then click **File**, **Save As…**



37. Type **quests.mjs** for **Filename** then click **Save**.

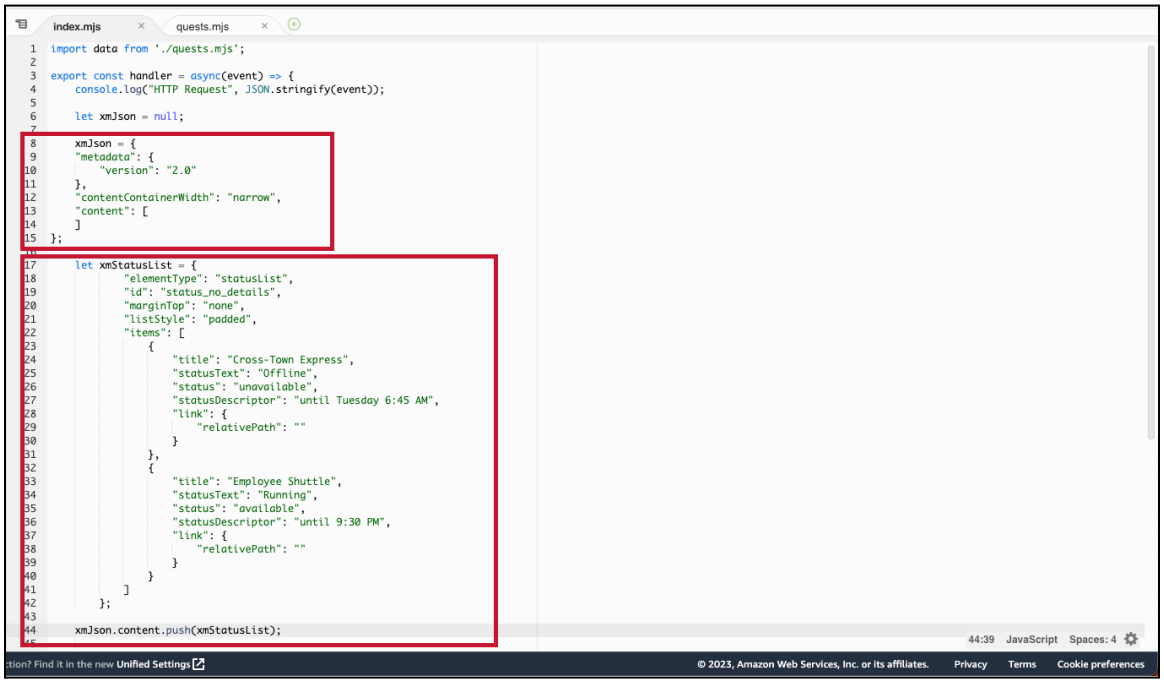


This JSON data is what we will parse as if we had either retrieved the information from the database or if we have received it from another API web service.

38. Click the index.mjs tab to bring it to focus and reference the quests.json file to make it available to your index.mjs file by adding the following code to line 1.

```
import data from './quests.mjs';
```

39. Extract the statusList element from the content array and configure it as a JSON object with the variable named `xmStatusList` then push the `xmStatusList` variable to the `xmJson` content array.



40. Now that we have an `xmStatusList` element, we need to iterate through the data and add those items to the content array. Extract one item from the `statusList` elements `items` array and put it into a for loop after validating that data is not null. Once it is in the for loop, replace the title, statuses and links with the appropriate data from the `quests` JSON information. Once you have built the item

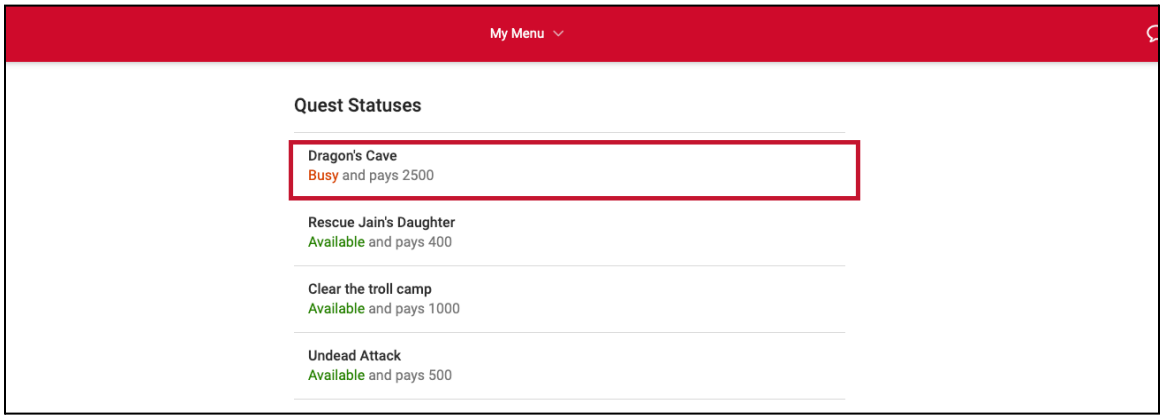
appropriately, delete the items from the `xmStatusList` items array and add `"heading": "Quest Statuses"` as an attribute to the `xmStatusList` element.

Note: The status list items properties should be replaced with the following information:

a. Set **title** to the quest name
b. Set the **statusText** to the quest status
c. Set the **status** to `"available"` if the status is available otherwise set it to `"unavailable"`
d. Set the **statusDescriptor** to `` `and pays {quest gold} ``
e. Set the link's **relativePath** to `"/details/{quests_id}"`
f. Deploy your changes.

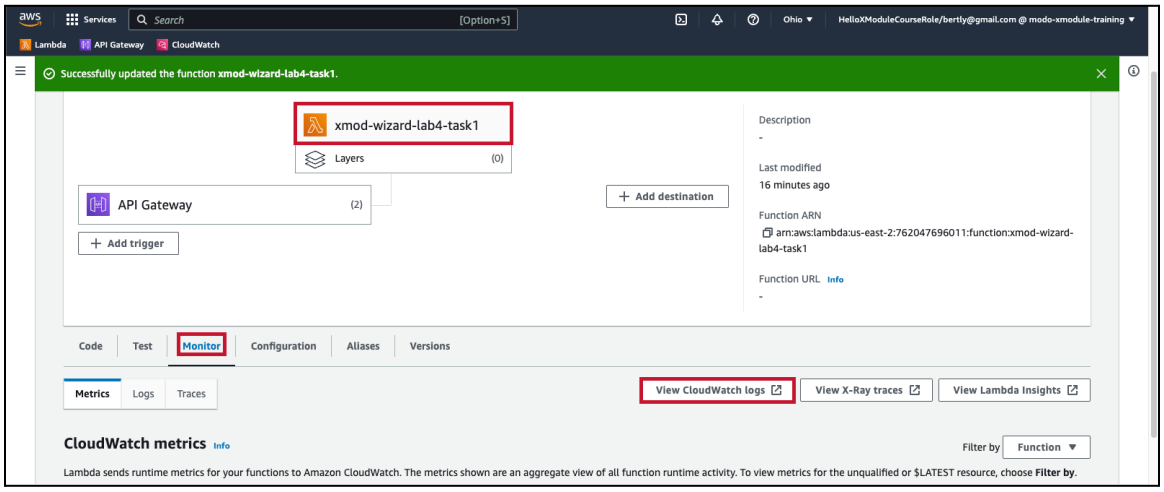
Note: Your code may look differently from this but as long as it generates the appropriate JSON I don't care how you approach it.

41. Refresh your Lab 4: Task 1 module to view the changes you've made and click one of the items in the list



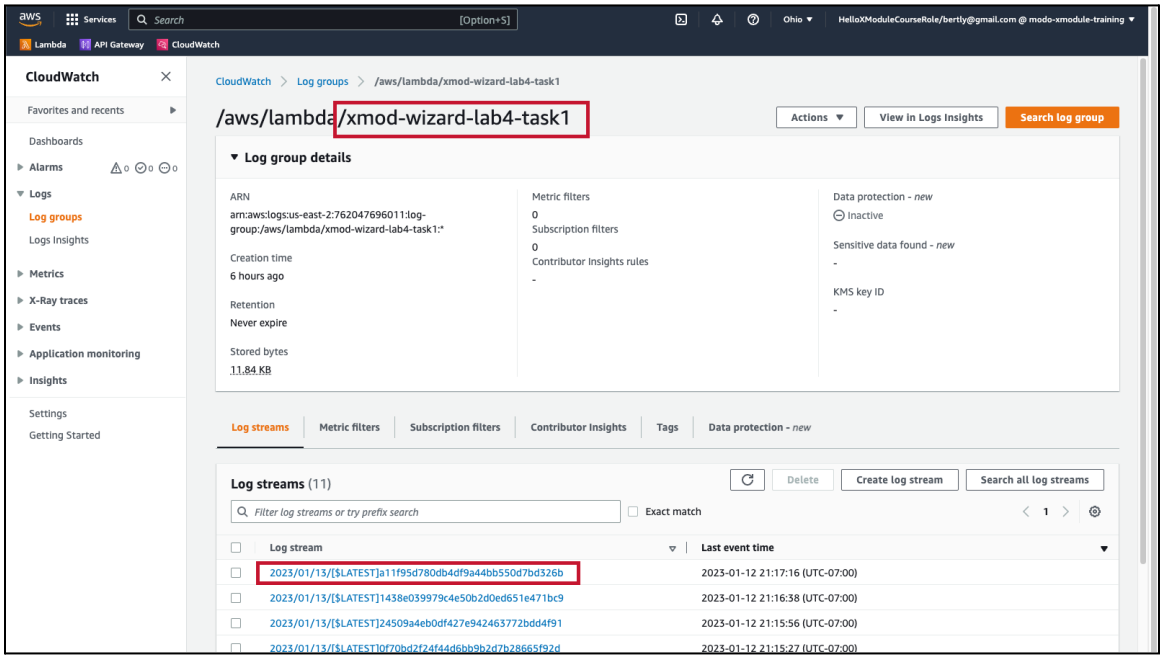
Note: The page refreshes but it does not show the details. We need to set some conditions on what to show based on whether the path parameter id is found.

42. Return to your Lambda function, click the Monitor tab, then click View CloudWatch logs.

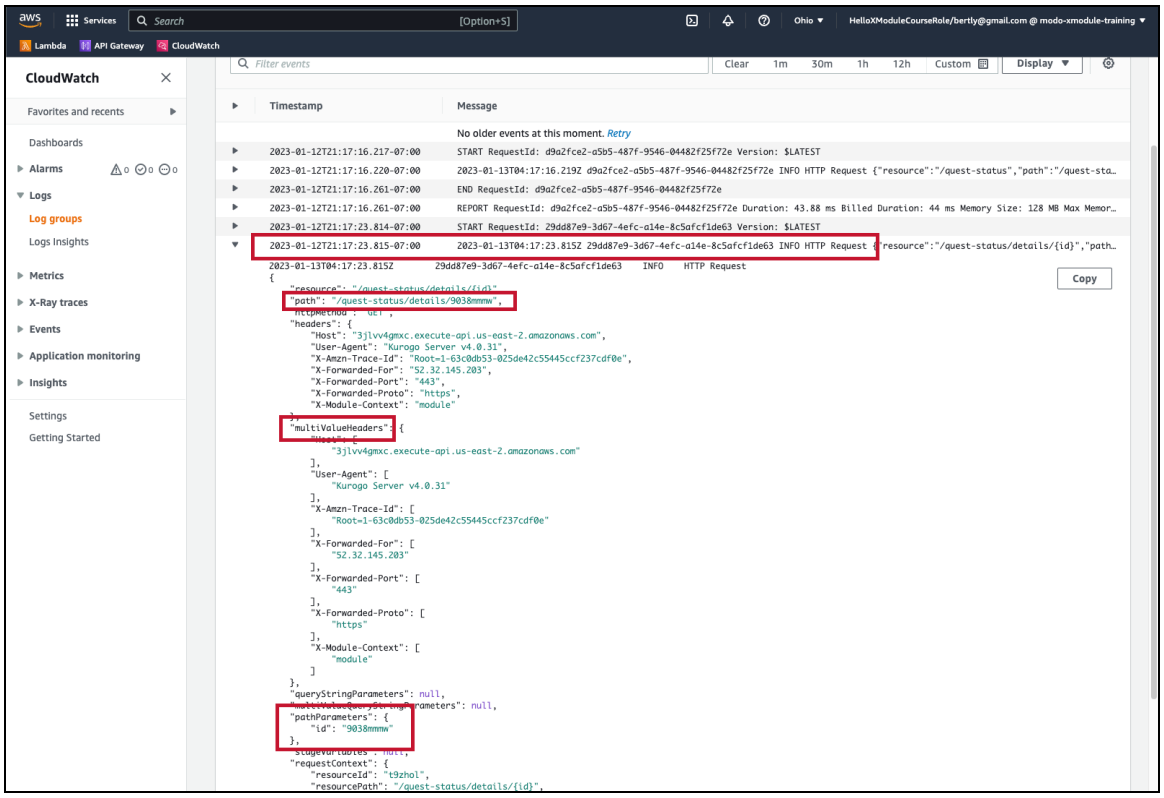


Note: Logs are being kept since that was the first thing we configured when we created the function.

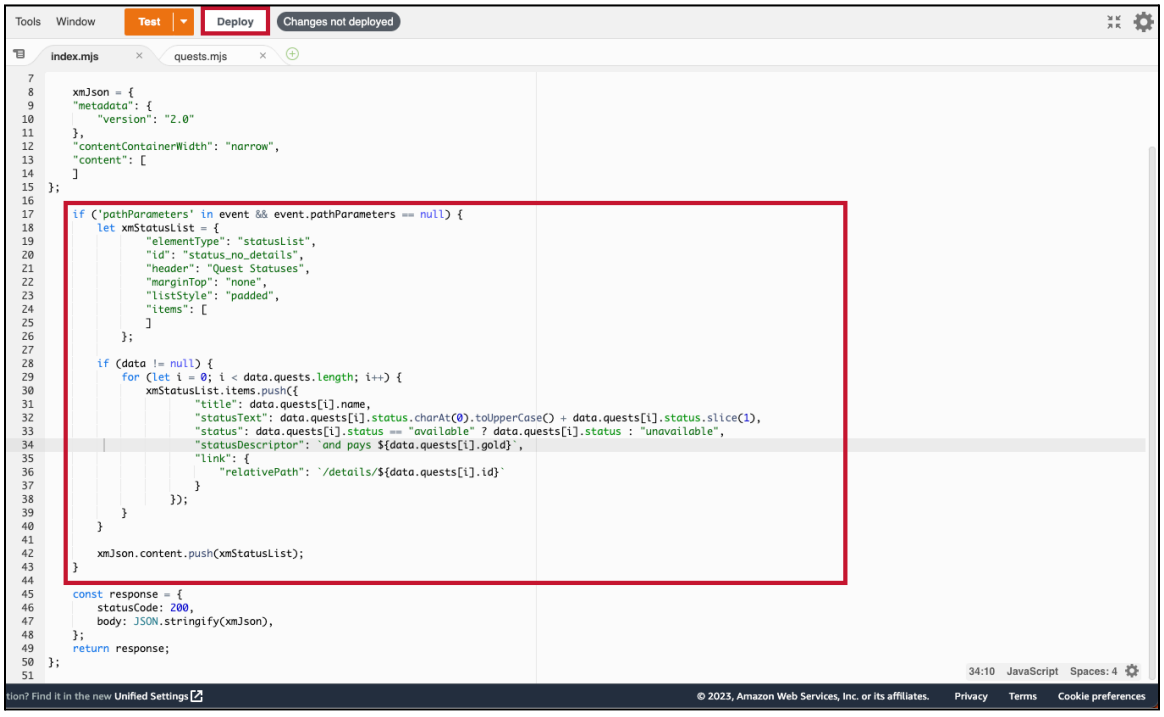
43. Click the topmost Log stream to view the log.



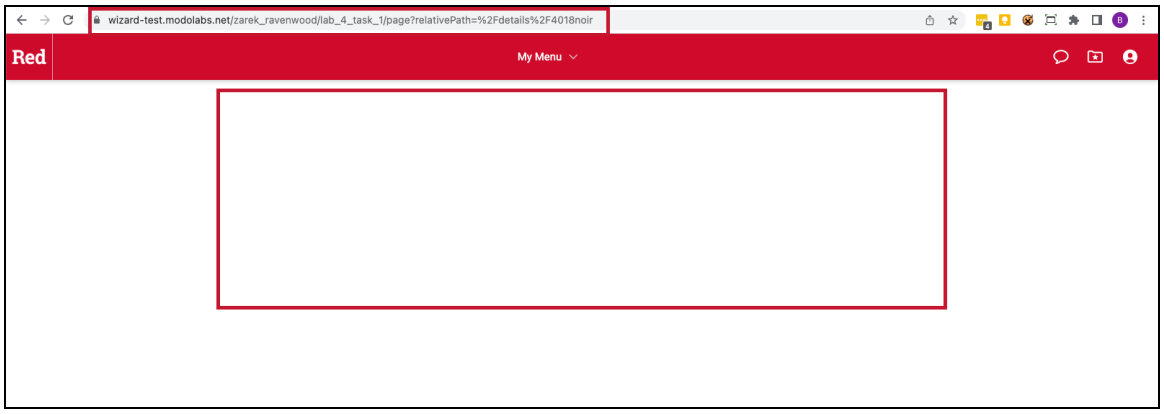
44. Find the HTTP Request with the latest time and click to expand that item and you will notice a couple of things:
    a. The id is part of the path string value.
    b. Values are no longer all lowercase like they were when using Function URLs
    c. The pathParameters are an object which is null if nothing is sent but contains a value if set



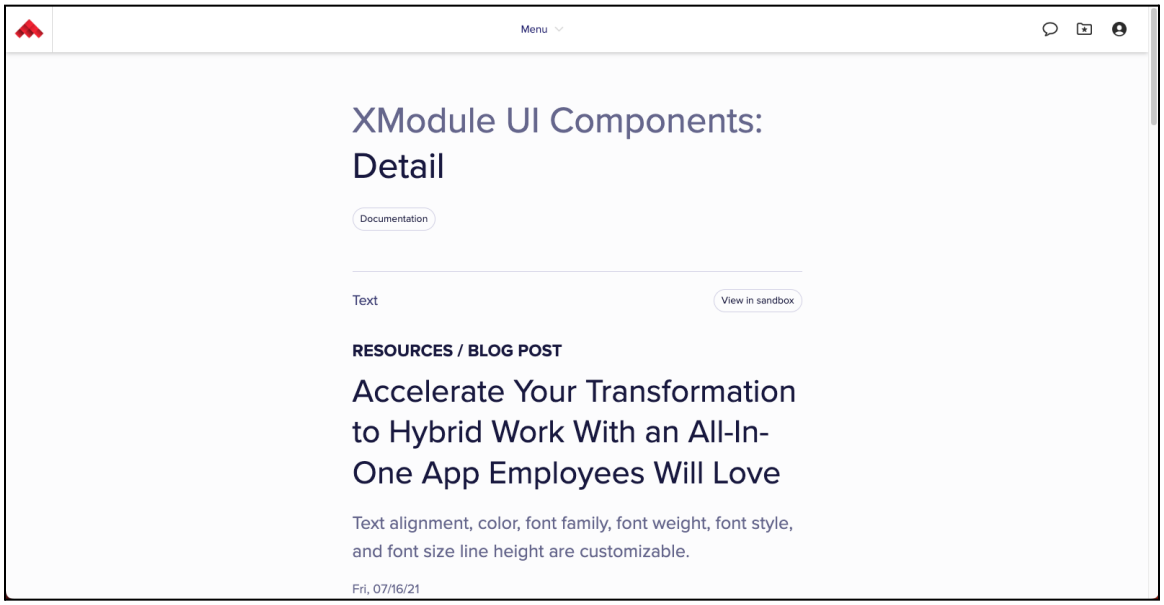
45. Return to your Code and input a condition to only generate the Status List if pathParameters are null and id does not exist in pathParameters and wrap all of the status list related code inside that condition. Deploy that update.



46. Preview the page, click an event and the page should be blank!



47. Since we have Quest Details let's use the Detail element. Click the following link to launch the Detail examples: https://xmodule.modolabs.net/default/xmodule_ui_examples/page?relativePath=.%2Fdetail.json

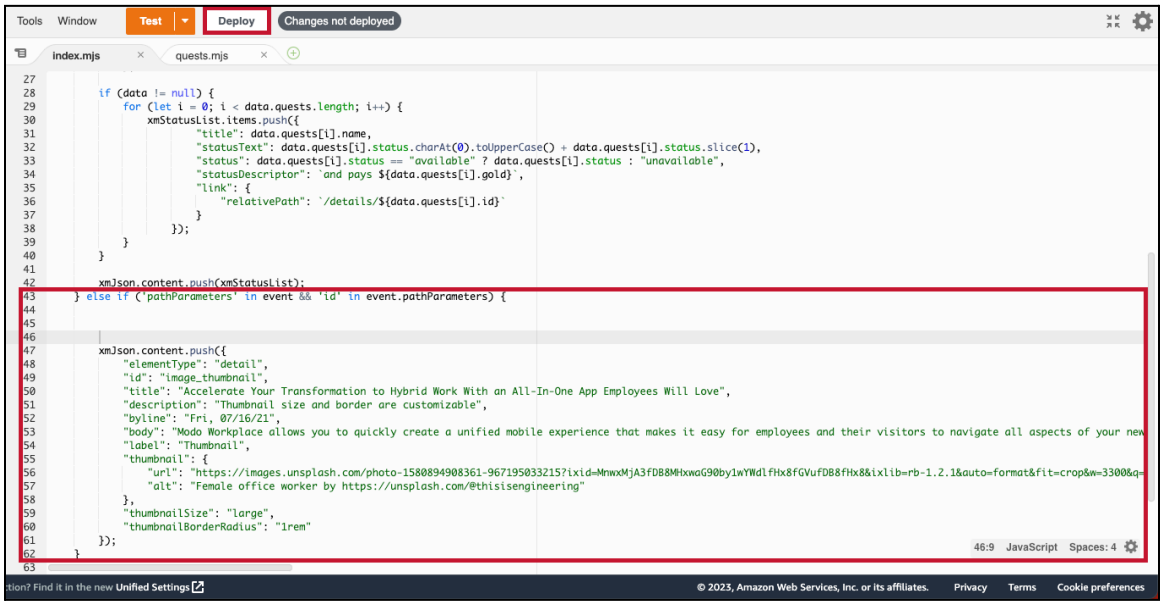


48. Click **View in sandbox** for the Thumbnail Detail element and copy the element to your clipboard.

49. Add an else if condition to your `pathParameters` condition to execute different code if `id` exists in the `pathParameters` object.

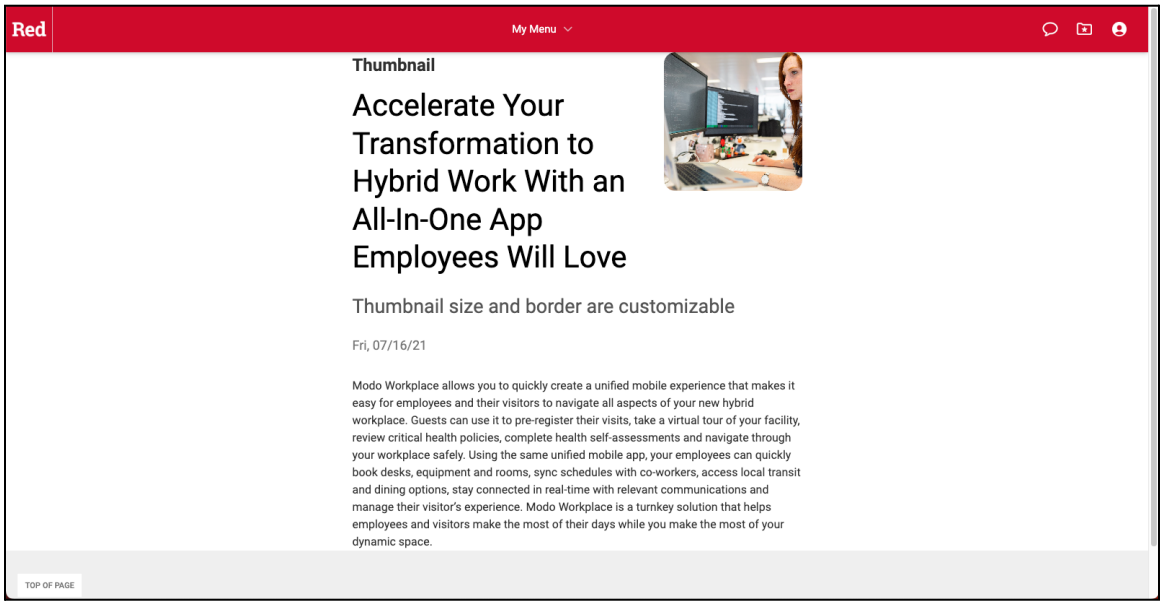
```
    if ('pathParameters' in event && event.pathParameters == null) {
        let xmStatusList = {
                "elementType": "statusList",
                "id": "status_no_details",
                "heading": "Quest Statuses",
                "marginTop": "none",
                "listStyle": "padded",
                "items": [
                ]
            };

        if (data != null) {
            for (let i = 0; i < data.quests.length; i++) {
                xmStatusList.items.push({
                        "title": data.quests[i].name,
                        "statusText": data.quests[i].status.charAt(0).toUpperCase() + data.quests[i].status.slice(1),
                        "status": data.quests[i].status == "available" ? data.quests[i].status : "unavailable",
                        "statusDescriptor": `and pays ${data.quests[i].gold}`,
                        "link": {
                            "relativePath": `/details/${data.quests[i].id}`
                        }
                    });
            }
        }

        xmJson.content.push(xmStatusList);
    } else if ('pathParameters' in event && 'id' in event.pathParameters) {

    }

    const response = {
```

50. Put a couple of extra spaces inside the new condition then type xmJson.content.push(); then put the cursor in the middle of the parentheses and paste your clipboard contents then deploy your changes.



```
Tools  Window  Test  Deploy  Changes not deployed

index.mjs  quests.mjs

        if (data != null) {
            for (let i = 0; i < data.quests.length; i++) {
                xmStatusList.items.push({
                        "title": data.quests[i].name,
                        "statusText": data.quests[i].status.charAt(0).toUpperCase() + data.quests[i].status.slice(1),
                        "status": data.quests[i].status == "available" ? data.quests[i].status : "unavailable",
                        "statusDescriptor": `and pays ${data.quests[i].gold}`,
                        "link": {
                            "relativePath": `/details/${data.quests[i].id}`
                        }
                    });
            }
        }

        xmJson.content.push(xmStatusList);
    } else if ('pathParameters' in event && 'id' in event.pathParameters) {


        xmJson.content.push({
            "elementType": "detail",
            "id": "image_thumbnail",
            "title": "Accelerate Your Transformation to Hybrid Work With an All-In-One App Employees Will Love",
            "description": "Thumbnail size and border are customizable",
            "byline": "Fri, 07/16/21",
            "body": "Modo Workplace allows you to quickly create a unified mobile experience that makes it easy for employees and their visitors to navigate all aspects of your new
            "label": "Thumbnail",
            "thumbnail": {
                "url": "https://images.unsplash.com/photo-1580894908361-967195033215?ixid=MnwxMjA3fDB8MHxwaG90by1wYWdlfHx8fGVufDB8fHx8&ixlib=rb-1.2.1&auto=format&fit=crop&w=3300&q=
                "alt": "Female office worker by https://unsplash.com/@thisisengineering"
            },
            "thumbnailSize": "large",
            "thumbnailBorderRadius": "1rem"
        });
    }

46:9  JavaScript  Spaces: 4

tion? Find it in the new Unified Settings

© 2023, Amazon Web Services, Inc. or its affiliates.  Privacy  Terms  Cookie preferences
```

51. If you refresh your Lab 4: Task 1 preview screen it should show the Details contents since the URL contains the relativePath that tells it to look at a specific item.

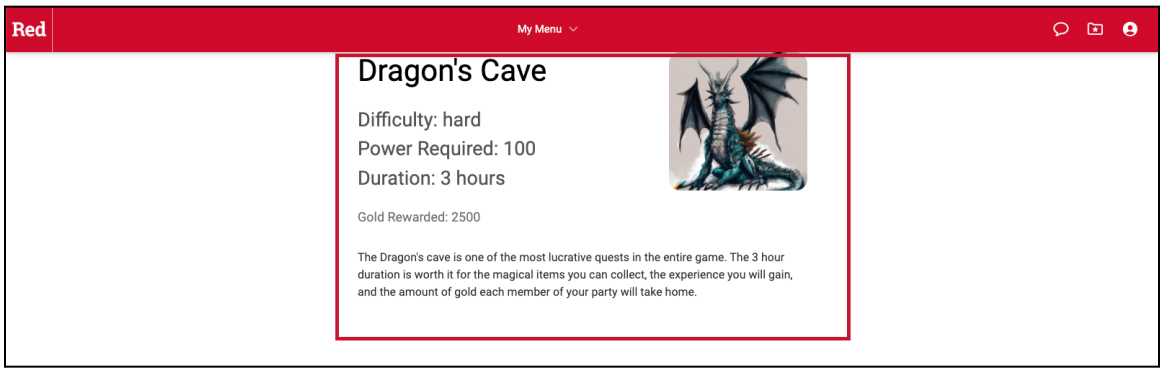


52. In the AWS Code editor copy and paste the following find function.

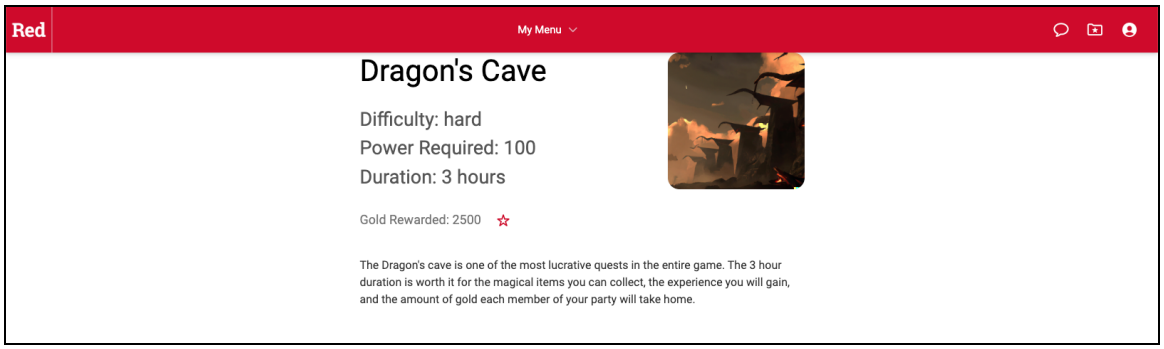
```
const quest = data.quests.find( ({ id }) => id === event.pathParameters.id );
```

53. Configure the Detail element with the following:
    a. The quest name for title
    b. The quest difficulty, recommended power, and duration for description
    c. The quest gold for byline
    d. The quest description for body
    e. For thumbnail
        i. The quest photo for url
        ii. The quest name for alt
    f. Deploy your changes

54. Preview your changes and tweak the styling until the content looks good.



55. Click the back button to return to the quests list and select a couple of other quests to see their details.
56. We can quickly add Bookmarking / Favoriting to our items with very little code. Click the following link to view the Bookmark element documentation: https://support.modolabs.com/support/solutions/articles/13000097196-bookmark
57. Scroll down to the code snippet and copy it to your clipboard.
58. Find the detail element you are building for each quest and add a button array property after the thumbnail property and paste the content into the buttons array.
59. Change the bookmark element's data with the following:
    a. Set **{quest description}** for **description**
    b. Set **{quest photo}** for **URL**
    c. Set **{quest name}** for **title**
    d. Set "**/details/{quest id}**" for the URL **relativePath**
    e. **Deploy** the changes and refresh and preview the page.



Note: The bookmark element also supports a refreshRelativePath link that allows you to specify a URL that can be used for updating the bookmark data (e.g. showing the status of bookmarked items that are dynamic).

60. Bookmark a couple of items then check the Favorites folder to see how they appear.

Congratulations! You have successfully created a multiple page layout with different URLs and used both the status list element and the details element.

Task 2: Loading Pages Via AJAX

Context: The `relativePath` link type is used for loading AJAX content into your page. This is especially useful when loading content from independent data sources that may have different response times. The initial page can load while the request is being sent off for additional data. In this example we will use the following items to show how this works:

- A `sideBySide` element that contains a `blockHeading` in each side and grouped `list` with one item with the server's date time as the title
- The left side should load with the initial page load
- The right side should load only the list via AJAX and should have enough latency that we can see a different time on each side of the `sideBySide` element.
- Use a query string parameter in the `relativePath` to determine whether to render a `content` array or a `regionContent` array.

The items being added through AJAX must be inside a regionContent array instead of a traditional content array.

Goal: Build a page that loads a segment of content via AJAX.

Note: Now that you have seen some of our programming approaches we will leave most of the programming up to you from here on out and will only provide our recommendations on the steps you should follow and the resources you should use to complete the steps. If you get stuck, please ask for assistance.

Task Outline:

- Create a Lambda function with function URL named {xmod-prefix}-lab4-task2 and log the HTTP request
- Create an XModule JSON variable that contains only the "metadata" and "version" properties
- Create a simple side-by-side element using the UI Components
- Add a blockHeader to the left with heading **Loaded in initial call** and to the right with heading **Loaded by AJAX**

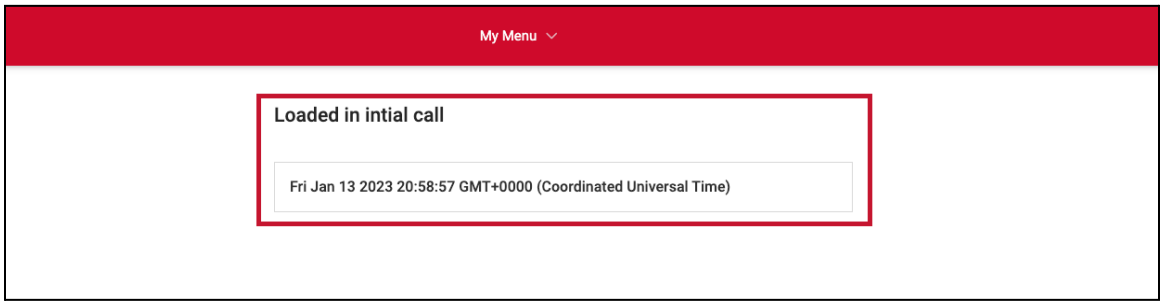
- In the left side, display the current date time as a string
- In the right, create a container that loads the same page via AJAX using some kind of **relativePath** query string parameter
- Return the JSON above if no query string parameters are available as an item in the content array of your XModule JSON object's variable
- Check for query string parameters specified in the AJAX call and respond with **regionContent** that displays the current date time as a string

Steps:

1. Create a new Lambda function titled **{xmod-prefix}-lab4-task2** with a corresponding Function URL and log the HTTP Request information (see Lab 4: Task 1, steps 1 & 2 for examples).
2. Create an `xmJson` variable that sets the object's metadata object. It has the following code only:

```
let xmJson = {
    "metadata": {
        "version": "2.0"
    }
};
```

   Note: Sometimes it is valuable to create XModule object that only has the required metadata section. This allows you to add additional properties as you determine what is needed and where it should go.
3. Create a new XModule named **Lab 4: Task 2** that points to your function URL. (We do not need to worry about any additional configuration settings for this module.)
4. Go to the Sandbox / UI Component examples and find and extract a sideBySide element's JSON. This is going to be the main element in the content array.
5. Focus on building the left side content and do the following:
   a. Put a `blockHeader` element as the first item in the content array with **Loaded in initial call** for heading.
   b. Add a list item using `"listStyle": "grouped"` that has one item in the list containing a title with the current server time
   c. Deploy your changes and preview your content.



   Note: The element spans both columns since there was no content specified for the right side.

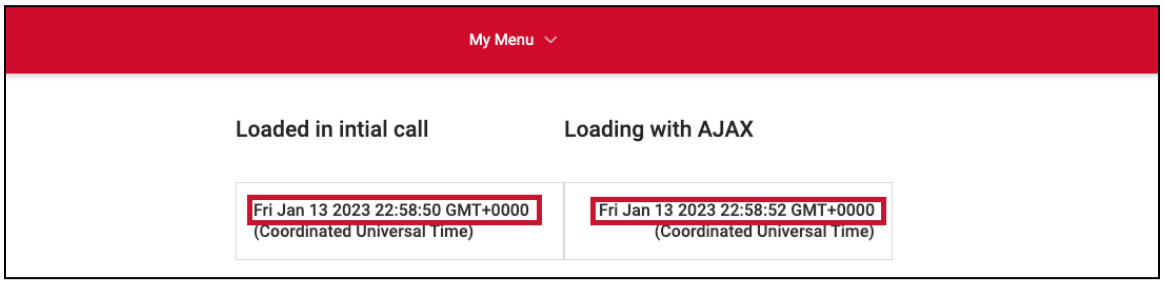
6. Focus on building the right side by doing the following:
   a. Copy the blockHeading element to the right content array and type **Loaded with AJAX** for heading.

b. Add a container element with the content set to `{ "ajaxRelativePath": "?ajax=true" }` or to whatever query string data for which you want to check. More information on this can be found here: https://support.modolabs.com/support/solutions/articles/13000088951-ajax-content
c. Configure your Lambda function to return the `xmJson` with a `content` block if the query string parameter does not exist or is null and to return `regionContent` containing a list with server status if the ajax query string parameter is found.

Hint: I decided to use a function to generate the grouped list that sets the title to current server time. Here is the function I used:

```
    //Creates a grouped list with one item that has the current date as the title of
the lone item
    function getListWithDateTime() {
        return {
            "elementType": "list",
            "listStyle": "grouped",
            "id": "basic_list",
            "items": [
                {
                    "title": new Date().toString()
                }
            ]
        };
    }
```

7. Validate the output in your **Lab 4: Task 2** module.



Note: You can see that the right side loaded 2 seconds after the left side.

Congratulations! You have successfully introduced AJAX during the page load process. Remember that the relativePath can either be pointing back to the same URL or it can point to a `relativePath` that is in another directory.

Task 3: Load page with Geolocation

Context: Geolocation requires an additional request to the device to enable location services as soon as the page loads. This additional AJAX request is triggered by settings configured with your XModule. The request adds a `geolocation` property to the JWT with the `lon` and `lat` properties populated.
In this example we will use the following items to show how this works:

- A list element with items that are loaded via AJAX which show the longitude and latitude from the JWT when using location services.

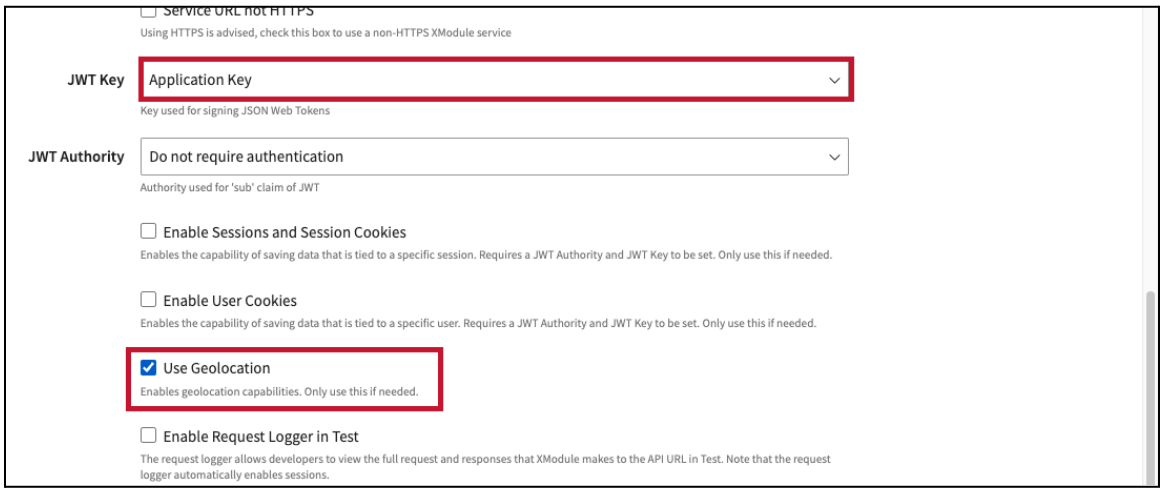
Goal: Build a page that shows the current geolocation information.

Task Outline:

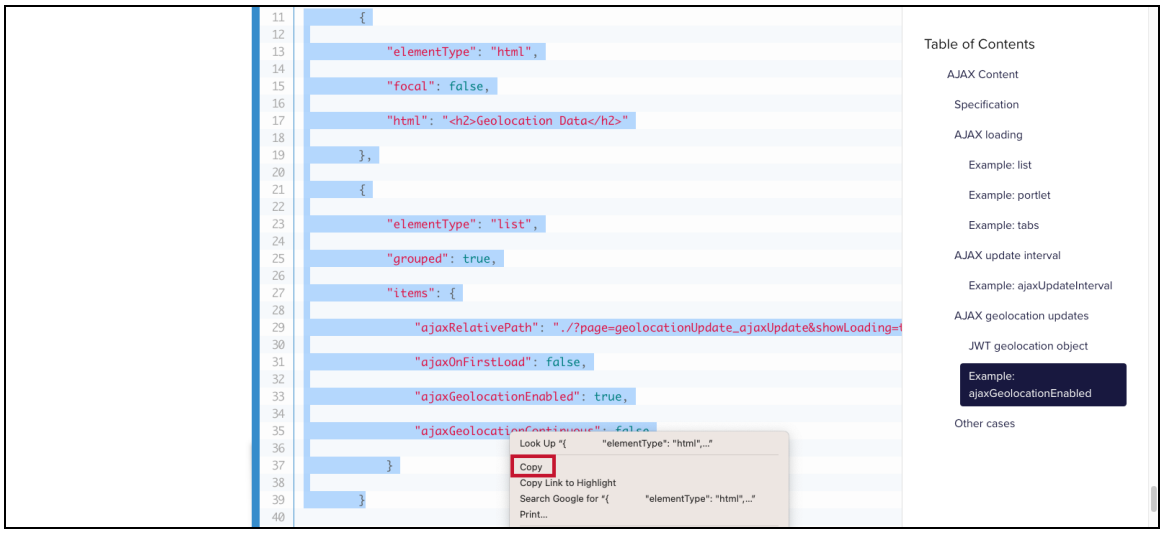
- Create a Lambda function with function URL named {xmod-prefix}-lab4-task2 and log the HTTP request
- Create an XModule JSON variable that contains only the “metadata” and “version” properties
- Copy the AJAX Geolocation content from the support docs and paste it into the sandbox and correct issues to make it valid Modo 4 JSON
- Verify that you can return valid JSON using the list information from the Sandbox (add it to your content array)
- Add the **jsonwebtoken** and **jwk-to-pem** layers to your Lambda function and use it to decode the Authorization header and get values from the JWT
- Replace the data in the list with information from the JWT and return it using **regionContent** instead of the standard **content** section.

Steps:

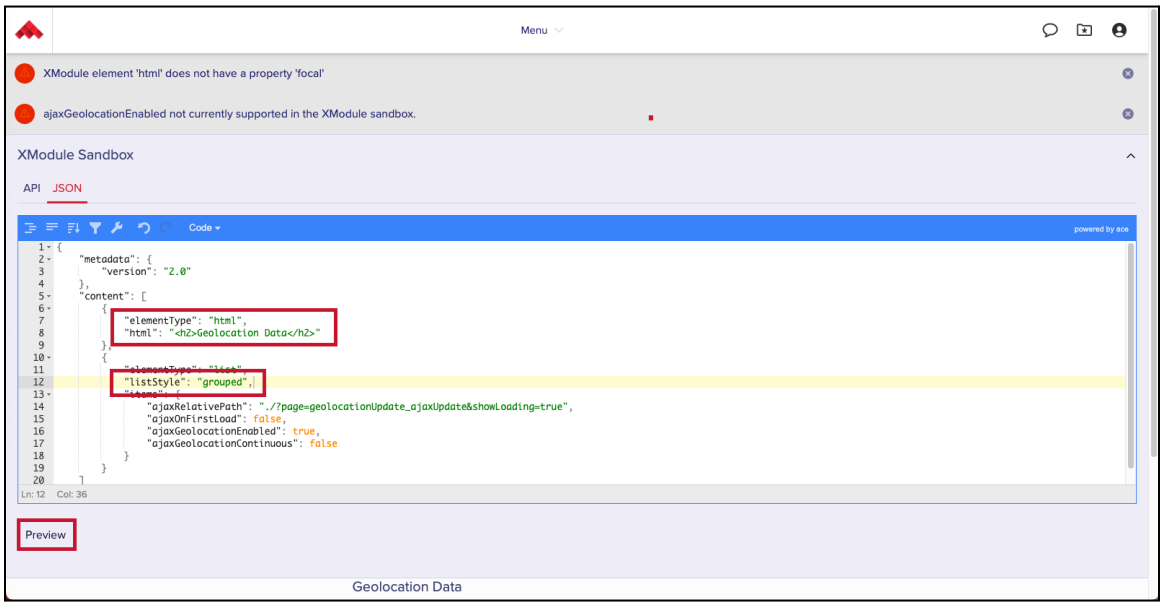
1. Create a new Lambda function titled **{xmod-prefix}-lab4-task3** with a Function URL and log the HTTP Request information (see Lab 4: Task 1, steps 1 & 2 for examples).
2. Create an `xmJson` variable that sets the object’s metadata object.
3. Create a new XModule named **Lab 4: Task 3** that points to the Function URL.
4. Once the Module is created, make the following additional configurations:
    a. Select **Application Key** for **JWT Key**
    b. Click to check **Use Geolocation**
    c. Click **Save**.



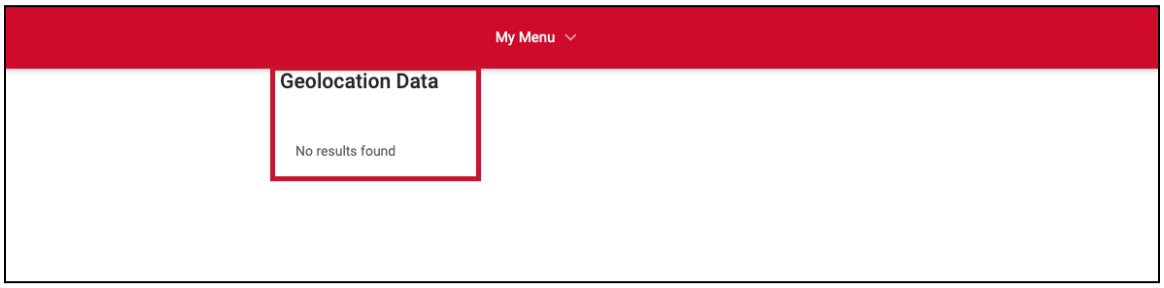
5. **Deploy** the changes to **Test**.
6. Review the Geolocation data found in this article: https://support.modolabs.com/support/solutions/articles/13000088951-ajax-content#ajax-geolocation-updates-0-8
7. Scroll down to **Example: ajaxGeolocationEnabled** JSON and copy the contents of lines 11 - 39 and paste them into the content array of the XModule Sandbox (https://xmodule.modolabs.net/default/sandbox/index) and click Preview.



8. The XModule Sandbox will show you errors because in Modo 4 the html element does not support the `focal` property on html elements and you cannot have geolocation on Sandbox items. Correct the html element issue by deleting the line that says: `"focal": false`, change the `"grouped": true`, line in the list to `"listStyle": "grouped"`, and click Preview.

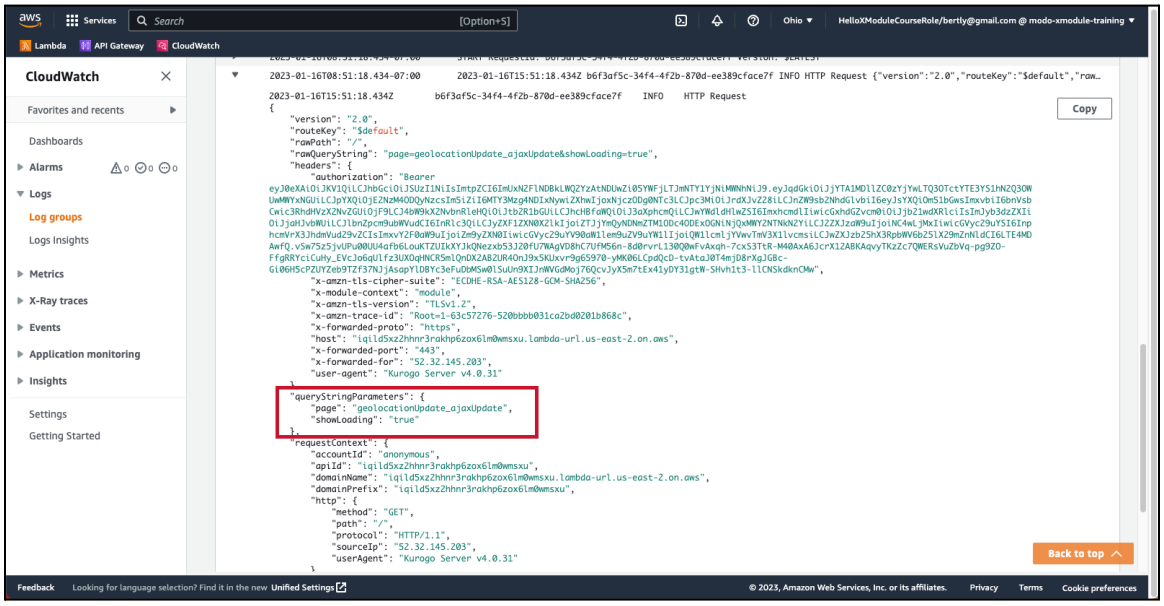


9. Copy the resulting content section JSON to your Lambda function and add it to your xmJson variable then return that variable as the body of your JSON response. **Deploy** those changes and then preview your Lab 4: Task 3 XModule.



10. If you check your CloudWatch logs for this function you can see that it is actually being called twice each time you load the page. The initial request has no query string parameters for the HTTP Event but

the 2nd one has two different parameters. We need to key off the existence of those parameters to trigger our Geolocation AJAX functionality.

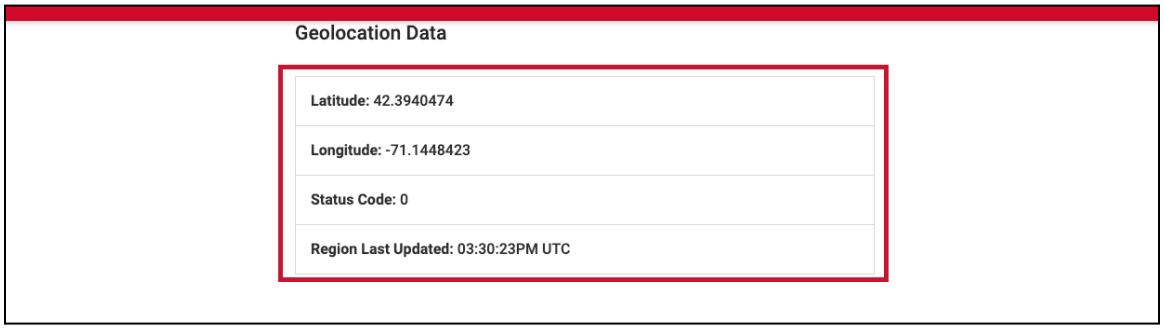


11. Set a condition to check for the page query string parameter to make sure it is set to “” and change the xmJson variable to return the AJAX Geolocation regionContent information. Return to the AJAX support documentation and copy the regionContent from the next block and add it to your xmJson object then Deploy your changes. (see code snippet below)

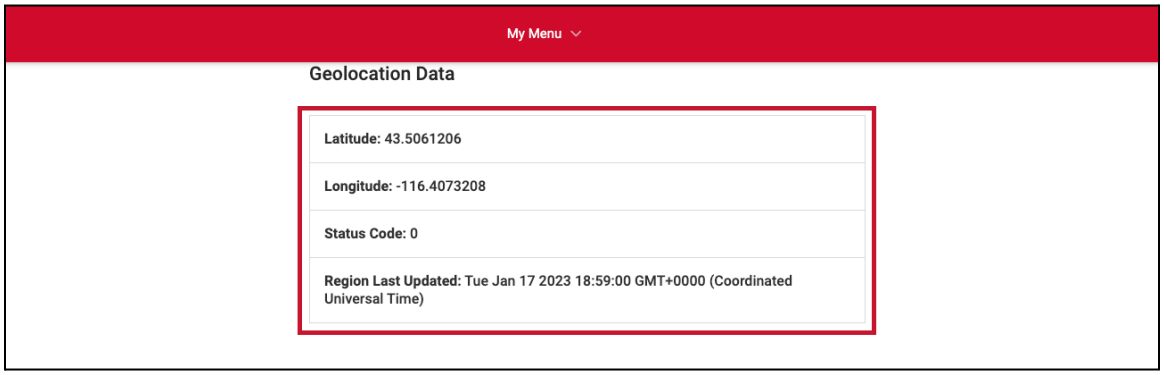
```
    if ('queryStringParameters' in event && event.queryStringParameters != null &&
'page' in event.queryStringParameters && event.queryStringParameters.page ==
'geolocationUpdate_ajaxUpdate') {
        xmJson.regionContent = [
            {

                "title": "<b>Latitude:</b> 42.3940474"

            },

            {

                "title": "<b>Longitude:</b> -71.1448423"

            },

            {

                "title": "<b>Status Code:</b> 0"

            },

            {

                "title": "<b>Region Last Updated:</b> 03:30:23PM UTC"
```

```
        }
    ];
}
```

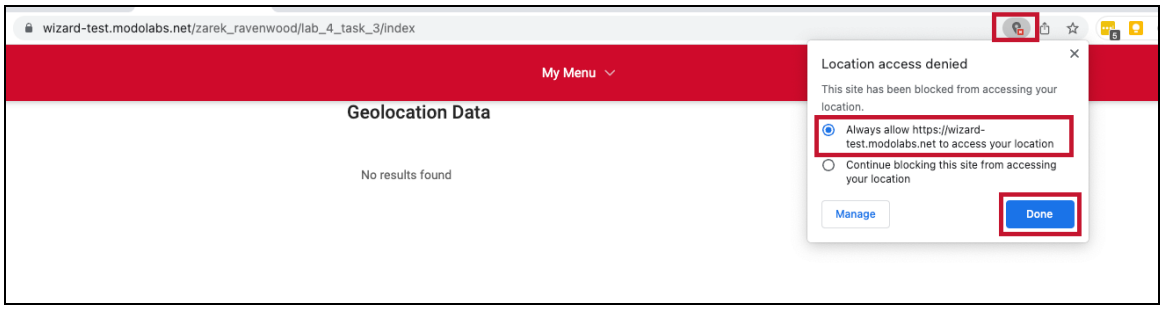
12. Preview your Lab 4: Task 3 page to view the updated version.



13. This is just using the hardcoded values found in the list item and needs to be updated to accept the geolocation settings that are passed in the JWT. You should inspect the JWT information on the response that contains the region content to see how that information is passed. You will likely need to allow your location through your browser to actually pass longitude and latitude data.
14. Add the `jsonwebtoken` and `jwk-to-pem` Lambda layers to this Lambda function (this is a repeat steps 1-4 of Lab 3: Task 3)
15. Copy and paste the code that initializes the use of those libraries and uses them to decode the payload from Lab 3: Task 3 to this function, changing `modojwt` to `authorization` where you check the headers.
16. Create some variables to hold the data you are adding to the list items, initialize them to the default values of the list then check to see if the payload has the values and change them to the payload values.
17. Change the list's items to return the variable information, **Deploy** the changes and **Preview** the page.



Note: You may not see any change to the Geolocation information. This could be because your location services are not allowed. You may have to go into your system settings or browser settings to allow location services.



Congratulations! You have successfully enabled geolocation in your XModule content. This can be paired with forms (our next group of labs) to help you capture the user's geolocation when they submit the form. Alternatively you can use this to validate that a user is within a specific geofence when they perform an action (such as scanning a QR code).

Lab 5 - Forms

Estimated Time: 2 hours

User input allows deeper customization in your Modo app. Most forms are sent as a POST to the server at the `relativePath` you specify and the form data is included in the body of the HTTP request with `Content-Type: application/x-www-form-urlencoded`. There are libraries that automatically convert form data into `key:value` paired data. You can alternatively change the method for a form post to a GET method that sends the data as query string parameters.

Uploading data as part of the form submission further changes the type to `Content-type: multipart/form-data`. You must handle/save the file on upload and change it to the appropriate format. However, if an AWS S3 bucket is available, you can upload the files directly to the AWS bucket by creating a bucket that authorizes Modo's AWS account as a trusted user with your S3 bucket. This changes the HTTP Request to `Content-Type: application/x-www-form-urlencoded` format and provides the original filename as well as the uploaded file's unique identifier value as part of the form submission.

Task 1: Standard Form submission with Progressive Disclosure

Context: Most forms use standard form elements like Text, Text Area, Select, Radio Buttons, etc. Some of these elements allow for Progressive Disclosure. This is a process of showing additional information based on a previous selection. This lab will introduce you to some of the standard form elements and allow you see how those responses are shared through both POST and GET methods.

Goal: Handle standard form submission and implement static progressive disclosure

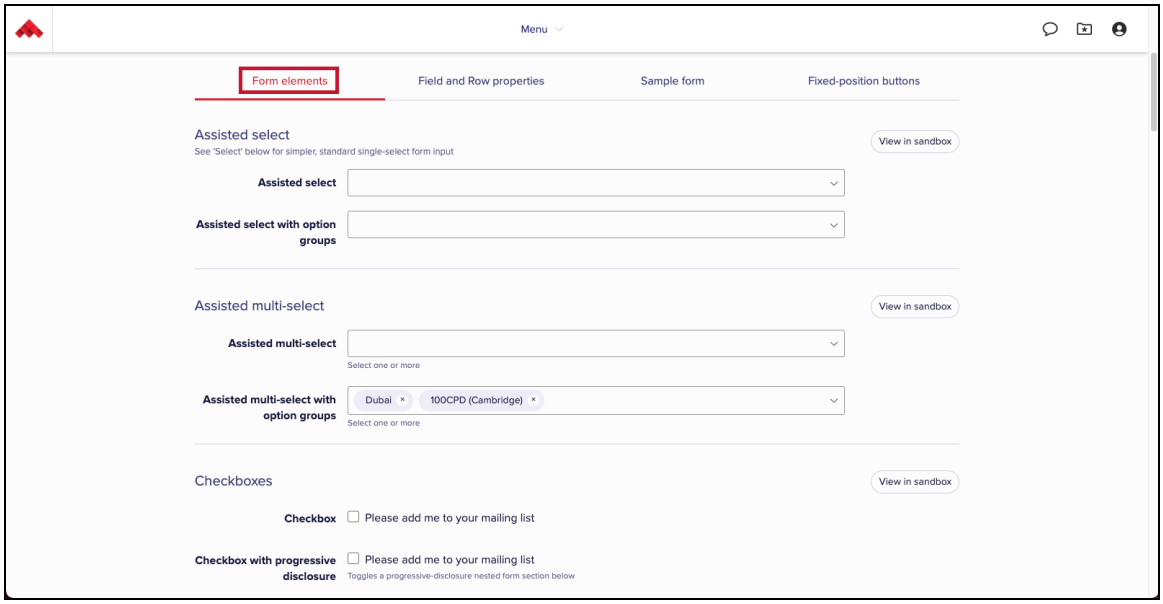
Task outline:

- Select a sample form from UI Components and edit it in the Sandbox
- Make the **Company** field required
- Change **Favorite animal** to **Animal companion** and add **Dragon**, **Emu**, **Kangaroo**, and **Unicorn** to the list

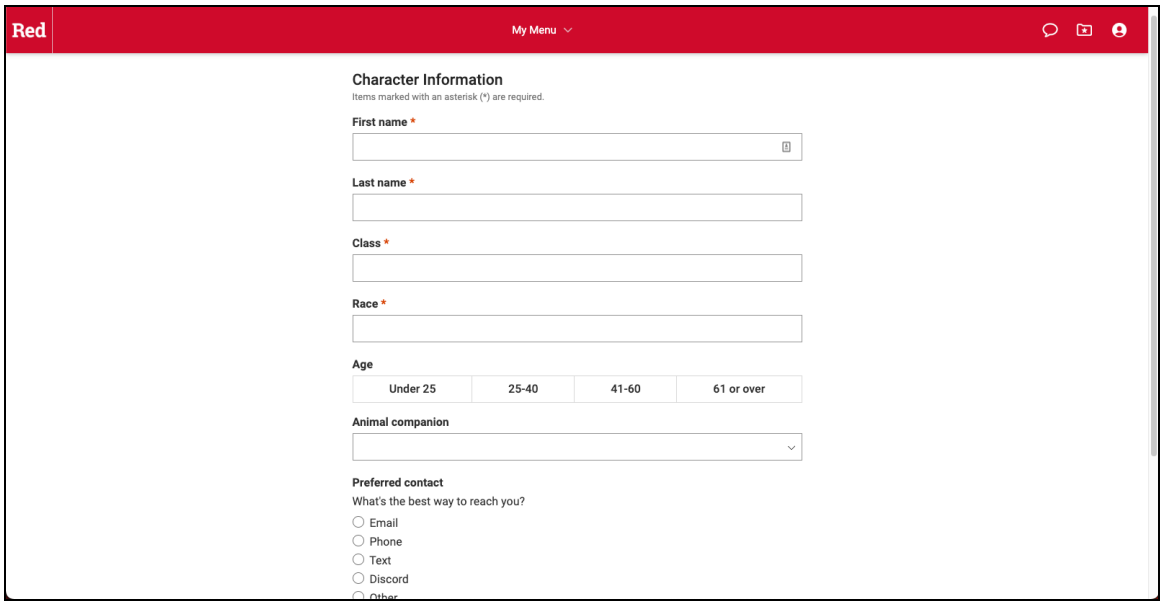
- Add an additional option for **Preferred contact** that accepts a text area element that has a label stating **Please describe other communication methods** and adjust the height of the text area so you can see more than 2 rows of text.
- Add the form to a Lambda function and create an XModule that points to the URL
- Have the form **POST** on submit and view the results, then change it to a **GET** request and compare the way results are handled
- Add a banner message to the page when the form is submitted letting them know the form was submitted successfully

Steps:

1. Click the following link and view the Form elements tab and test out the interaction with most of the form elements:
   https://xmodule.modolabs.net/default/xmodule_ui_examples/page?relativePath=.%2Fform.json



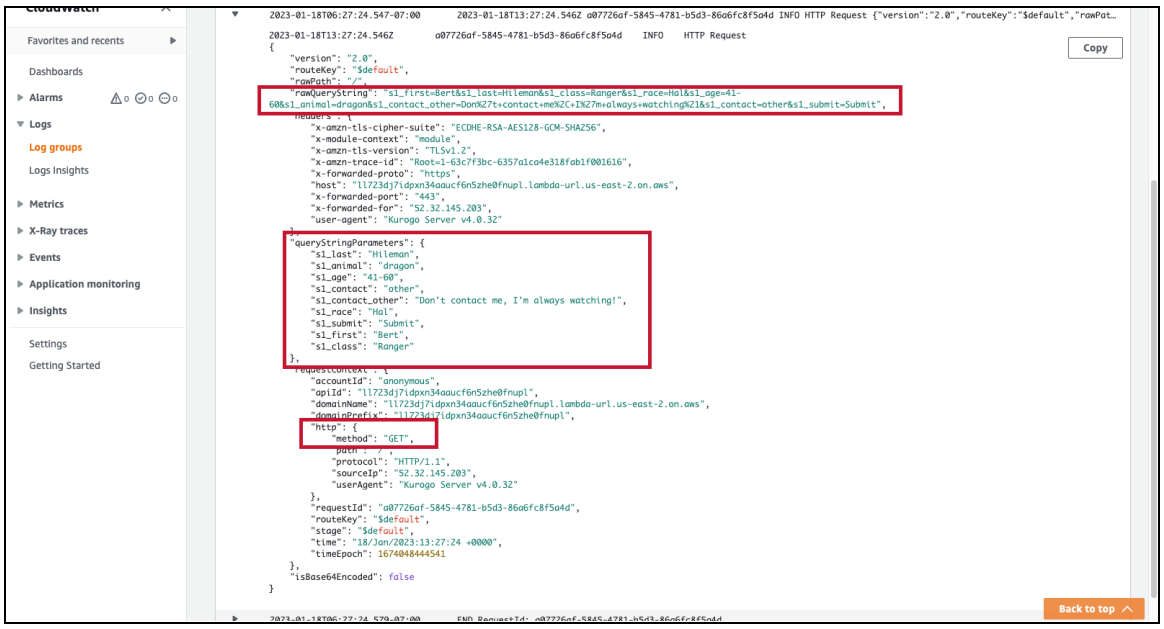
2. Select either the **Sample form** tab or the **Fixed-postion buttons** tab and click **View in sandbox**.
3. Change / add the following elements to your form in the Sandbox:
   a. Change the forms heading to **Character information** and delete the `buttons` array..
   b. Change **Company** to **Class** and make the field required
   c. Change **Title** to **Race** and make the field required
   d. Change **Favorite animal** to **Animal companion** and add **Dragon**, **Emu**, **Kangaroo**, and **Unicorn** to the list
   e. Add an additional option for **Preferred contact** that accepts a text area element that has a label stating **Please describe other communication methods** and adjust the height of the text area so you can see more than 2 rows of text.
   f. Verify that your JSON renders correctly when you click **Preview**.
4. Create a new Lambda function named **{xmod-prefix}-lab5-task1**, log the HTTP Request information, create a variable named `xmJson` to contain and return the form's JSON from your Sandbox.
5. Create a Function URL and then create an XModule using the URL, Deploy to test, and preview the module to make sure it looks like the image below.



6. Fill out the form, submit it, and look for a POST event in your CloudWatch logs.

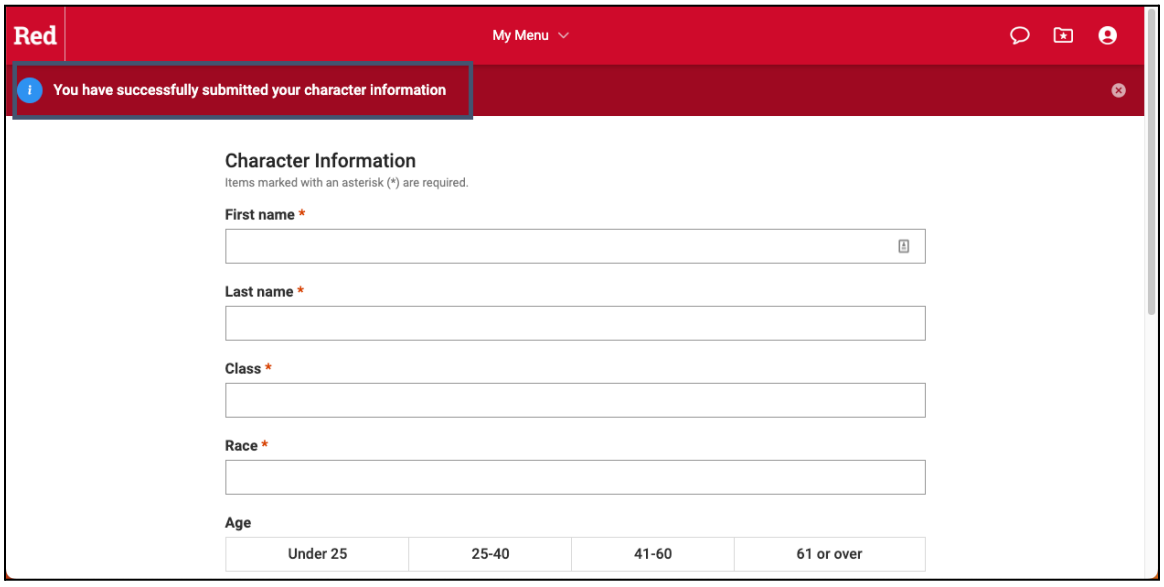
Note: Using Function URLs results in base64 encoded results. You can use a base64 decoder to view the way the response is actually sent. When using the API Gateway with your AWS Lambda functions the results are typically not base64 encoded. If you are using NodeJS libraries such as express and serverless form data is received as a JSON object so you can access it as key: value pairs.

7. Add `"requestMethod": "GET"` to your form element in your Lambda function to change from a POST method to a GET method, Deploy your changes, refresh the Lab 5: Task 1 preview page, fill out and submit the form, then check CloudWatch for how the form data has been submitted to the backend system.



8. The form refreshing to an empty state is the only visual cue we receive to let us know the form submitted. It would be nice to employ some kind of message to let us know the form has been submitted and the data recorded if applicable. For this feature we can use banner messages. Click the following link to view the Banner Messages information: https://support.modolabs.com/support/solutions/articles/13000088957-banner-messages
9. Add the following to your code:
    a. Check for a POST httpMethod or a GET httpMethod with query string parameters with at least one form element in the object
    b. If the above condition exists, create a Banner message (documentation found here: https://support.modolabs.com/support/solutions/articles/13000088957-banner-messages)
        i. Type **You have successfully submitted your character information** for **message**
        ii. Type **info** for **type**
    c. Deploy your changes and test

    Note: Since we changed the httpMethod to GET the URL may have query string parameters as part of the URL. If you just refresh the page, it will act as a form resubmission so make sure the module id is the last part of your URL before you reload the page.



Congratulations! You have successfully added your first form and tested out both POST and GET methods and have added a banner message. The next step would be processing that form to do something but that is beyond the scope of this class!

Task 2: Progressive Disclosure with AJAX

Context: There may be instances where there are too many progressiveDisclosure options to include in the initial page creation. This is often the case when building a custom facilities reporting form where there are multiple countries, multiple cities per country, multiple buildings per city, multiple floors per building, and multiple options on those floors. In these instances you may choose to add the main list countries and then load the cities, buildings, floors, and options via an AJAX call.

Goal: Implement Progressive Disclosure with AJAX.

Task outline:

- Create an XModule form in the Sandbox that has a select element and uses progressive disclosure to show a City, Building, and floors once the location has been selected
- Use the following JSON as a data source in side the app to serve as a readonly database: https://static.modolabs.com/xmodule_certification/data_files/locations.json
- Create a form titled Select your location that has a dropdown to select the top level location
- Set up progressive disclosure to load from an ajaxRelativePath and load the city names and floors as progressive ajax content.

JSON for a form with the correct number of nested fields:

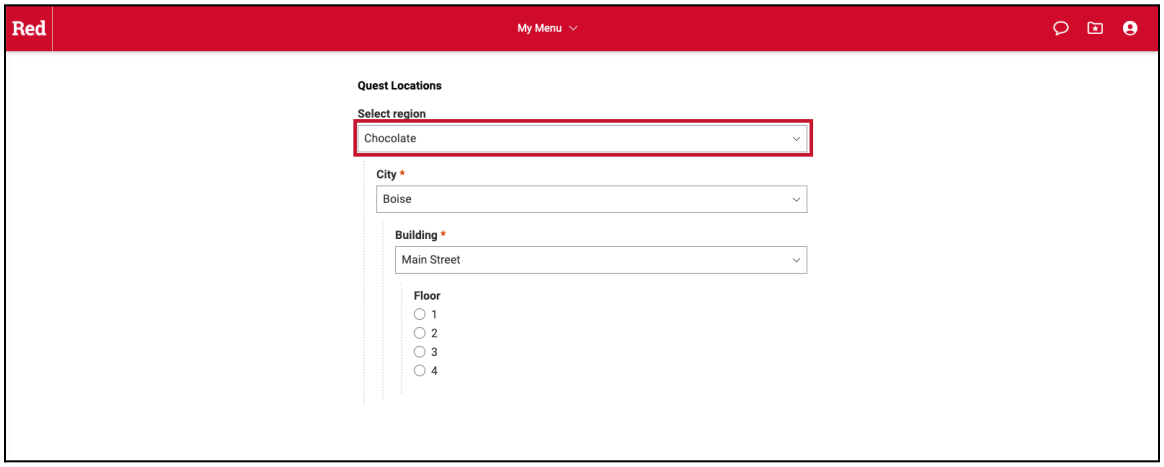
```
{
    "metadata": {
        "version": "2.0"
    },
    "contentContainerWidth": "narrow",
    "content": [
        {
            "elementType": "form",
            "heading": "Quest Locations",
            "id": "select",
            "items": [
                {
                    "elementType": "formInputSelect",
                    "name": "select",
                    "nested": true,
                    "label": "Select region",
                    "options": {
                        "caramel": "Caramel",
                        "chocolate": "Chocolate",
                        "jackfruit": "Jackfruit",
                        "rockyroad": "Rocky Road",
                        "strawberry": "Strawberry",
```

```
                    "vanilla": "Vanilla"
                },
                "progressiveDisclosureItems": {
                    "chocolate": [
                        {
                            "elementType": "formInputSelect",
                            "name": "city",
                            "label": "City",
                            "nested": true,
                            "required": true,
                            "options": {
                                "Boise": "Boise"
                            },
                            "progressiveDisclosureItems": {
                                "Boise": [
                                    {
                                        "elementType": "formInputSelect",
                                        "name": "building",
                                        "label": "Building",
                                        "nested": true,
                                        "required": true,
                                        "options": {
                                            "main_street": "Main Street"
                                        },
                                        "progressiveDisclosureItems": {
                                            "main_street": [
                                                {
                                                    "elementType":
"formInputRadio",
                                                    "name": "floor",
                                                    "label": "Floor",
                                                    "options": {
                                                        "1": "1",
                                                        "2": "2",
                                                        "3": "3",
                                                        "4": "4"
                                                    }
                                                }
                                            ]
                                        }
                                    }
                                ]
                            }
                        }
                    ]
                }
```

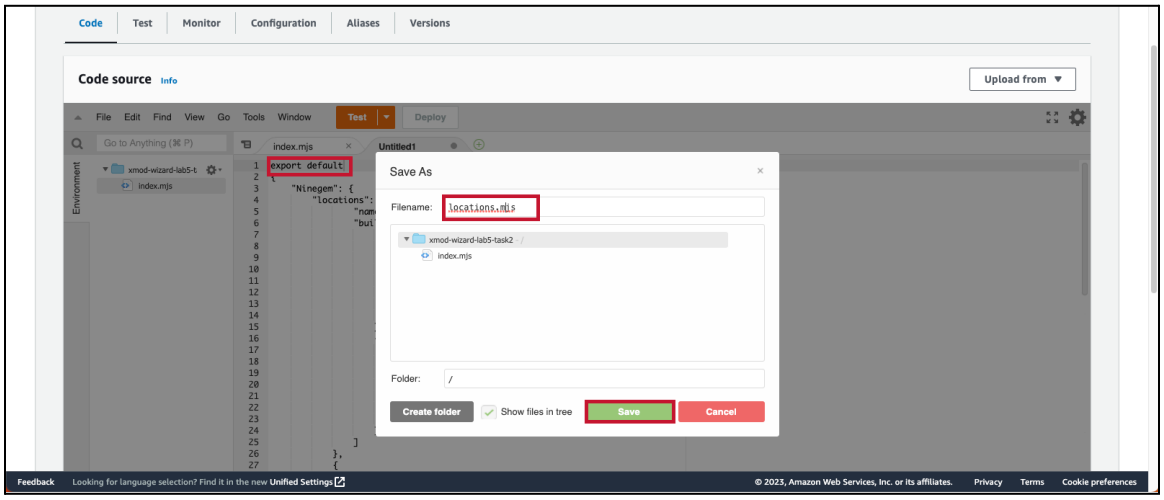
```
                }
            ]
        }
    ]
}
```

Steps:

1. Create a Lambda function named **{xmod-prefix}-lab5-task2**, log the HTTP Request information, and use the JSON above as your xmJson variable, return the JSON in the response body and Deploy the changes.
2. Create a Function URL and use that URL to create a new Lab 5: Task 2 XModule with the default settings and Preview your Lab 5: Task 2 XModule. Select Chocolate in the Select Region textbox to load the City, Building, and Floor progressive disclosure content.



3. Click the following URL and copy the JSON from that page to your clipboard: https://static.modolabs.com/xmodule_certification/data_files/locations.json
4. Create a new file in your Lambda function, paste the contents of your clipboard into the new page, type **export default** at the top of the page and save it as **locations.mjs**.



5. Add `import data from './locations.mjs';` to the top of your index.mjs file to import all the location information as data.

Note: Our next major step is to loop through the data and build out options and progressiveDisclosureItems that point to AJAX endpoints to return the contents.

6. Find the line that says `"chocolate": [` and click the disclosure widget on the left side to collapse all the contents and copy those contents to your clipboard.

7. Create a function that accepts a `region` variable and returns the chocolate progressiveDisclosure items and make a reference to the function immediately following the chocolate declaration. Deploy the content and preview the page to make sure that you still see the options when you select chocolate.
8. Move the first select element into its own variable and add the variable to the `items` array and make sure the page still renders properly.
9. Delete the contents of **options** and **progressiveDisclosureItems** and set them to an empty JSON object and add those items with a loop.

```
function buildPgItems(region) {

    let pgItems = [{
        "elementType": "formInputSelect",
        "name": "city",
        "label": "City",
        "nested": true,
        "required": true,
        "options": {
            "Boise": "Boise"
        },
        "progressiveDisclosureItems": {
            "Boise": [{
                "elementType": "formInputSelect",
                "name": "building",
                "label": "Building",
                "nested": true,
                "required": true,
                "options": {
                    "main_street": "Main Street"
                },
                "progressiveDisclosureItems": {
                    "main_street": [{
                        "elementType": "formInputRadio",
                        "name": "floor",
```

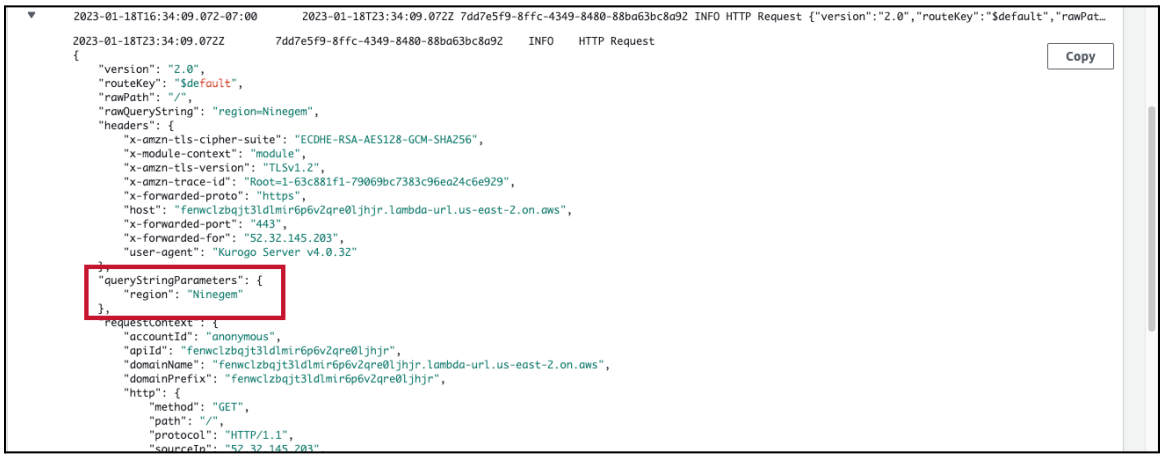
```
                        "label": "Floor",
                        "options": {
                            "1": "1",
                            "2": "2",
                            "3": "3",
                            "4": "4"
                        }
                    }]
                }
            }]
        }
    }];

    return pgItems;
}

let mainSelect = {
            "elementType": "formInputSelect",
            "name": "select",
            "nested": true,
            "label": "Select region",
            "options": {},
            "progressiveDisclosureItems": {}
};

//build main list and set disclosure items
for (var key in data) {
    mainSelect.options[key] = key;
    mainSelect.progressiveDisclosureItems[key] = buildPgItems();
}
```

10. If you Deploy and preview the Lambda you should see the three regions and all selections should return the same data.
11. Add the following code to the line that sets the progressiveDisclosureItems (see code below).

```
for (var key in data) {
    mainSelect.options[key] = key;
    mainSelect.progressiveDisclosureItems[key] = { "ajaxRelativePath": "?region=" +
key  };
}
```

Note: If you preview the page now, there will be no progressive disclosure items to show but you will see a loading indicator each time you switch selections. We are going to adjust the function we created to build it if we specific content returned for each region based on the options available. It may be a good idea to start logging the xmJson you are building so you can see the code for yourself.

12. Open the CloudWatch logs for your function and look at the HTTP Request for the most recent call to your Lambda function. You are looking for queryStringParameters which indicate it tried to load additional data via AJAX.

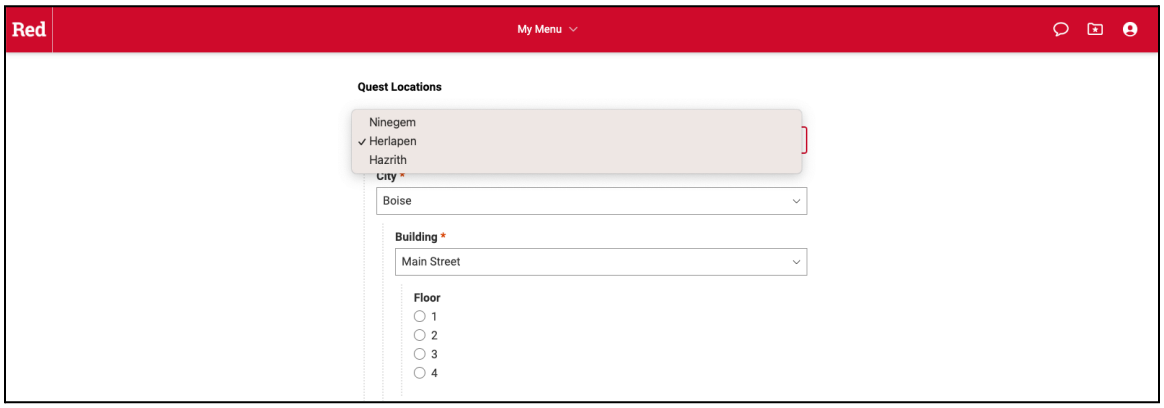


13. Add a condition to add regionContent to your xmJson if the region query string parameter exists and add the contents of your progressive disclosure function to the regionContent while passing the region to the function.

```
    //if we are looking for the region, we can add regionContent to our JSON and build out the progressive disclosure items.
    if ('queryStringParameters' in event && 'region' in event.queryStringParameters) {
        xmJson.regionContent = buildPgItems(event.queryStringParameters.region);
    }
```

Note: This must be added once the xmJson variable already exists.

14. When you preview the content now, you can see that the city options load each time you switch locations.



15. Modify the function that creates our progressiveDisclosureItems to create all the elements based on the items included in the locations.mjs file. Some strategies for this include:
    a. Create a function that makes a city element
    b. Create a function that creates a building element
    c. Create a function that builds the floors element
    d. Dynamically create options and progressiveDisclosureItems for each element
16. This is a pretty difficult project so looking in the Appendix for the code for Lab 5: Task 2 is an acceptable solution if you end up spending too much time on making this code.

Task 3: Assisted Select / Assisted Multiselect with and without AJAX

Context: The assistedSelect and assistedMultiselect elements allow list filtering as you type in the provided box. This is a great way to help users quickly find the options they are searching for in larger data sets. The assistedSelect element supports progressiveDisclosure or items loaded via an AJAX call but not both at the same time. If you want to load a form with items that have already been selected and you want to use AJAX you must provide the selected values to the options array when the form is generated or it will not be respected.

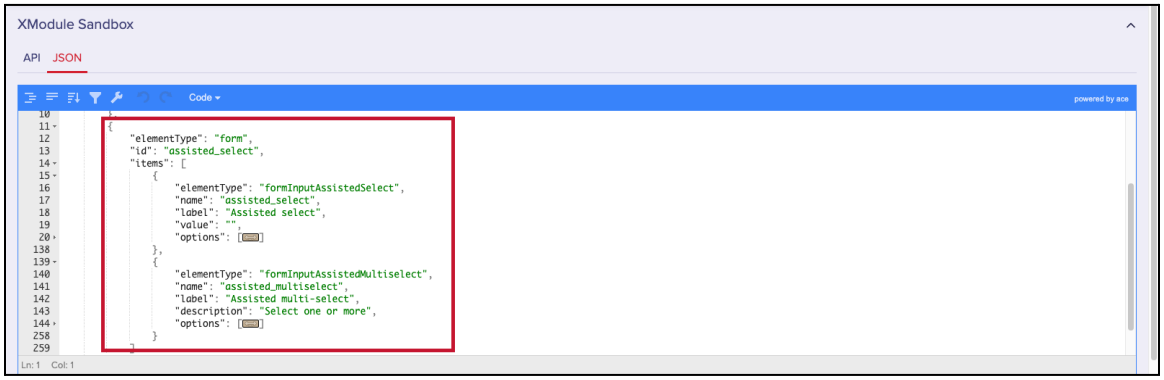
Goal: Implement the assistedSelect and assistedMultiselect elements with and without AJAX.

Task Outline:

- Get the Assisted Select and Assisted Multiselect elements from the UI Components form element create a form element with only those two fields
- Create a Lambda function for this task and save the timezones.json file from the following location as our timezones.mjs file: https://static.modolabs.com/xmodule_certification/data_files/timezones.json
- Configure the Assisted Select and Assisted Multiselect elements to load the the items via AJAX
- Configure the Lambda to filter the timezone information based on the search parameters and then return results using elementFields and specifying the data in the options array.
- Change the form to have some default selected values for each Assisted select type

Steps:

1. Go to the UI Component Examples Form elements and view the Assisted select element in the Sandbox, reduce the content to isolate the first `formInputAssistedSelect` element.
2. Extract `formInputAssistedMultiselect` element from the UI Component examples and add it to the Sandbox that contains the formInputAssistedSelect element.



3. Create a new Lambda function named **{xmod-prefix}-lab5-task3** with Function URL, log the HTTP Request information, and create an `xmJson` variable and paste the contents of the Sandbox with the two assisted select elements to the page and Deploy your changes.
4. Create an XModule named **Lab 5: Task 3** using the Function URL as the API URL and save it using the default settings. **Deploy** to **Test** and preview the page.
5. Now we are going to make those form elements use AJAX to load the options. To do this we need to complete the actions on both elements:
    a. Delete the options and its accompanying array of options
    b. Add the following 4 properties and values:

```
"useAjaxSearching": true,
```

```
                "ajaxSearchingRelativePath": "",
                "searchFilterParameter": "search",
                "minimumInputLength": 2
```

Note: Each of these values should be set. For more information on exactly what these settings do, please refer to the Assisted Select XModule documentation.

6. Deploy and preview the page and type **mou** in the search field to trigger the AJAX search for values.
7. Open CloudWatch for your Lambda function and find the most recent HTTP Request logged information.

Note: You can see that mou was passed as a query string parameter with the key search since that is what we set for `searchFilterParameter`.

8. Create a new file named timezones.mjs in your Lambda function with **ƒ** as the first line of the and the JSON from the following link as the data: https://static.modolabs.com/xmodule_certification/data_files/timezones.json
9. Add **`import data from "./timezones.mjs"`** as the first line of your `index.mjs` file then add a condition to look for the search query string parameter and filter the timezones list based on whether the search criteria exists and add elementFields to your `xmJson` object where it sets the options to the list you specified.

```
    if ('queryStringParameters' in event && 'search' in event.queryStringParameters) {
        let tzList = {}, searchString =
event.queryStringParameters.search.toLowerCase();
        for (let [key, value] of Object.entries(data.items)) {
            if (key.toLowerCase().includes(searchString) ||
value.toLowerCase().includes(searchString)) {
                tzList[key] = value;
            }
        }

        xmJson.elementFields = {
            "options": tzList
        };
```

```
    }
```

Optional: If you finish early, you can practice adding a default value selected. This means you need to have the selected item(s) set for value and those options must exist in the options array at runtime.

Congratulations! You have successfully added items to your Assisted Select and Assisted Multiselect items using AJAX. This is an excellent way to load large datasets into Assisted select type elements.

(Optional) Task 4: Save selected favorite foods to communicate database

Context: Modo's Communicate API allows you to leverage the Modo Communicate Directory as a lightweight database to store personal information. This can allow for rapid development of hyper-personalized experiences that engage and add extra value to your users.

Goal: Use the Communicate API as a lightweight database to save the current user's favorite food choices.

Task Outline:

- Create a new Lambda function with function URL, log the HTTP Request and create an xmJson object using the provided JSON to start out
- Create a new Lab 5: Task 4 XModule using the function URL.
- Add an attribute to the Communicate Schema with the following information:
    - **Name**: favoriteFoods
    - **Display Name**: Favorite Foods
    - **Description**: A selection of favorite foods
    - **Type**: array
    - **Filterable**: True
    - **Privileged**: False
- Use the provided JSON to build the form to select favorite foods
- Create a Communicate v2 API token that has directory access
- Connect to the Modo Communicate API
- Download and test the Communicate Postman API collection
- Add logic to write data to the logged in person's directory entry
- Read the directory when the page is created and populate the form with the correct participant data

Starting JSON:

```
{
    "metadata": {
        "version": "2.0"
    },
    "contentContainerWidth": "narrow",
    "content": [
        {
            "elementType": "blockHeading",
            "heading": "Favorite Food Selection for {YOURNAME}",
            "description": "When questing gets tiresome, we like to settle down and eat
our favorite foods! Select your favorite foods from the list below."
        },
        {
```

```
        "elementType": "form",
        "id": "checkbox",
        "items": [
            {
                "elementType": "formInputCheckboxGroup",
                "name": "devices",
                "label": "Favorite Foods",
                "description": "Check all that apply",
                "options": {
                    "american": "American",
                    "chinese": "Chinese",
                    "french": "French",
                    "indian": "Indian",
                    "italian": "Italian",
                    "japanese": "Japanese",
                    "korean": "Korean",
                    "mediterranean": "Mediterranean",
                    "mexican": "Mexican"
                }
            }
        ],
        "buttons": [
            {
                "elementType": "formButton",
                "name": "reset",
                "title": "Reset",
                "buttonType": "reset",
                "actionStyle": "destructiveQuiet",
                "minWidth": "8rem"
            },
            {
                "elementType": "formButton",
                "name": "submit",
                "title": "Submit",
                "buttonType": "submit",
                "actionStyle": "constructive",
                "minWidth": "8rem"
            }
        ],
        "trackDirtyStateButtonNames": [
            "submit",
            "reset"
        ],
        "buttonsHorizontalAlignment": "center",
        "buttonsFixedPosition": true
    }
```

```
    ]
}
```

1. Create a new Lambda function named **{xmod-prefix}-lab5-task4** with Function URL, log the HTTP Request information, and create an `xmJson` variable and paste the contents of the JSON snippet above to the page and Deploy your changes.
2. Create an XModule named **Lab 5: Task 4** using the Function URL as the API URL and save it using the default settings. **Deploy** to **Test** and preview the page.
3. Click Modo Communicate, switch to the test environment, expand Settings, and click V2s
4. Asdf
5. Asdf
6. As
7. Dfas
8. fd
9. As
10. Dfa
11. Sdf
12. Asdf
13. Asdf
14. As
15. df

Lab 6 - Events and Actions

Estimated Time: 1.5 hours

The support page describes events and actions as follows:

> **Events** are triggers you can put on specific elements which initiate Actions. **Actions** are the available responses an element can have for an Event. Many elements support Actions and can have an Event perform an action on them, but only certain elements can trigger an Event.

Modo's 4.0.19 release introduced a dramatic shift in the way events are handled. The following tasks help reinforce Modo's suggested programming best practices when implementing and responding to events.

Task 1: Directory Search

Context: One of the most common enterprise desires is creating a more robust directory search form than is available in Modo's out-of-the-box directory functionality. You can create an XModule that searches names, email addresses, titles, departments, etc all from one search box. Since these results can be quite lengthy we want to limit the search to start after typing a minimum number of characters.

Goal: Create a directory search XModule

Task Outline:

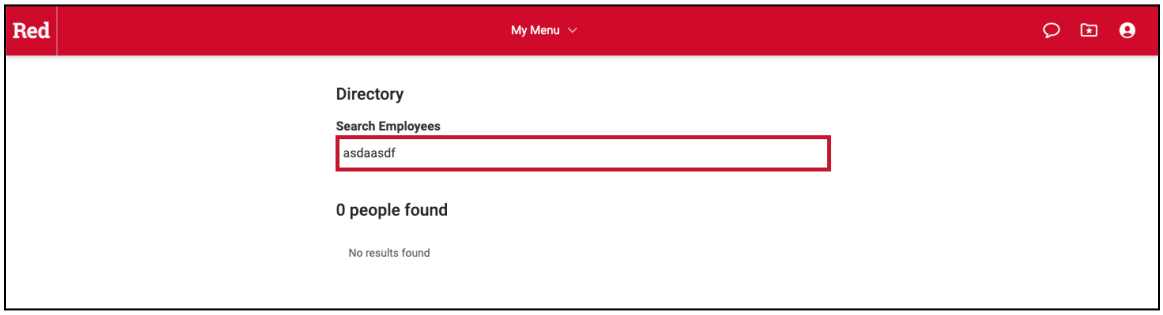
- Create a new Lambda function, log the HTTP Request and create an xmJson object using the provided JSON to start out
- Create a Function URL for your Lambda function and use that URL to create a new Lab 6: Task 1 XModule.
- Add a change event to your form that propagates the form arguments and performs an ajaxUpdate against the employee_list element.
- Add the directory.json from the following link as directory.mjs in your project: https://static.modolabs.com/xmodule_certification/data_files/directory.json
- Import the data into your index.mjs file do the following:
    - Add logic to check for the search arguments as part of the HTTP request
    - Filter the results based on those arguments / loop through the items in the list and see if the criteria matches
    - Build out list items showing the person's name and occupation as a minimum
- Test pressing enter when typing and see what happen
- Add a submit event to your form events to override the default form behavior

Initial JSON

```
let xmJson = {
    "metadata": {
        "version": "2"
    },
    "content": [
        {
            "elementType": "blockHeading",
            "heading": "Directory"
        },
        {
            "elementType": "form",
            "relativePath": "",
            "items": [
                {
                    "elementType": "formInputText",
                    "label": "Search Employees",
                    "name": "search_info",
                    "placeholder": "Search for name, occupation, or phone"
                }
            ]
        },
        {
            "elementType": "list",
            "heading": "Type to show list",
            "id": "employee_list"
        }
    ]
};
```

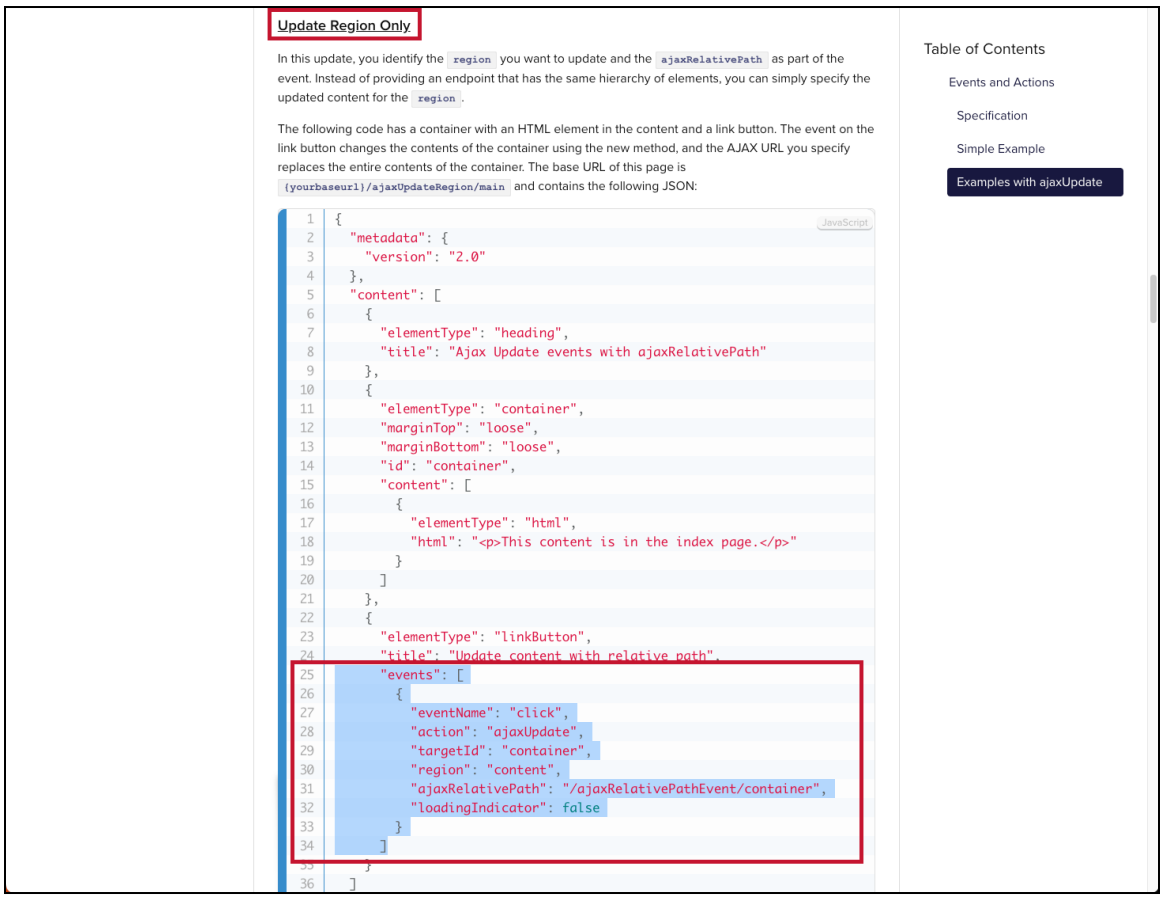
Steps:

1. Create a new Lambda function named **{xmod-prefix}-lab6-task1**, log the HTTP Request, add the code snippet above to set a `xmJson` variable, return the `xmJson` variable as the body of the response, and **Deploy** your changes.
2. Create a Function URL for your Lambda function and create a new XModule named **Lab 6: Task 1** using the Function URL, leaving all other settings at their defaults, then Deploy to test and preview your module.



Note: Typing does not do anything because you are using a standard text input that only submits when you click the Submit button. We are going to wire this up to submit via an AJAX change event.

3. Click the following link to open the XModule AJAX documentation and click the Version 4.0.19 and Later tab: https://support.modolabs.com/support/solutions/articles/13000089974-events-and-actions#examples-with-ajaxupdate-0-3
4. In the **Update Region Only** section, select and copy the event information from lines 25 - 34 to your clipboard.

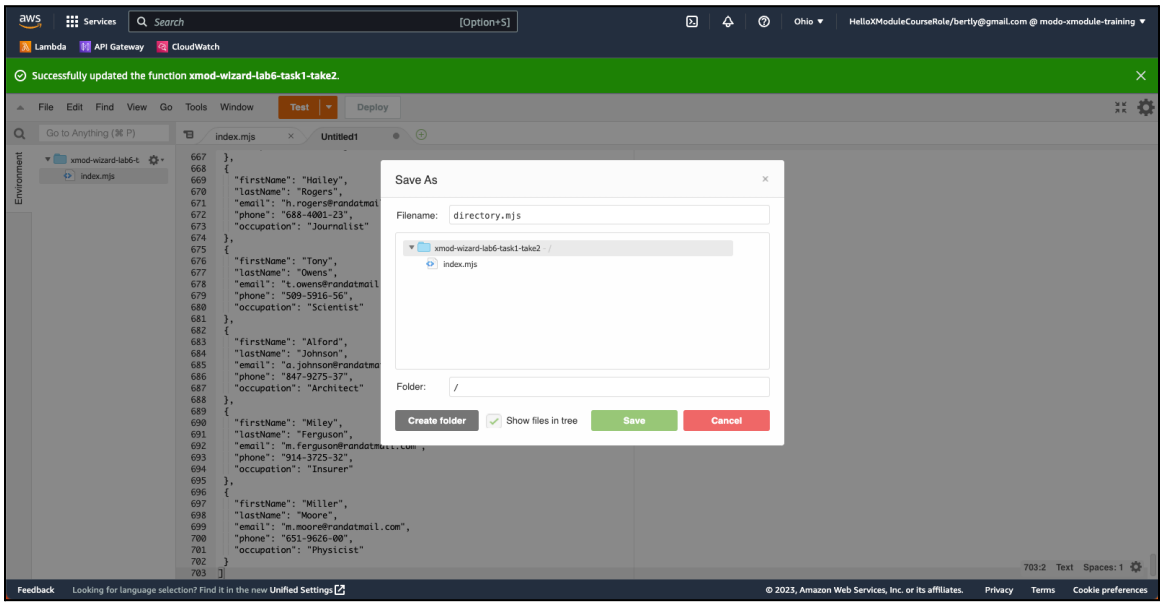


5. Paste it after the items array of your form element then make the following changes:

a. Type **change** for **eventName**
b. Type **employee_list** for **targetId**
c. Delete the line with `"region": "content",`
d. Set the **ajaxRelativePath** to an empty string `""`
e. Delete the `loadingIndicator` line and add `"propagateArgs": true`
f. Verify that it looks like the image below then Deploy your changes.

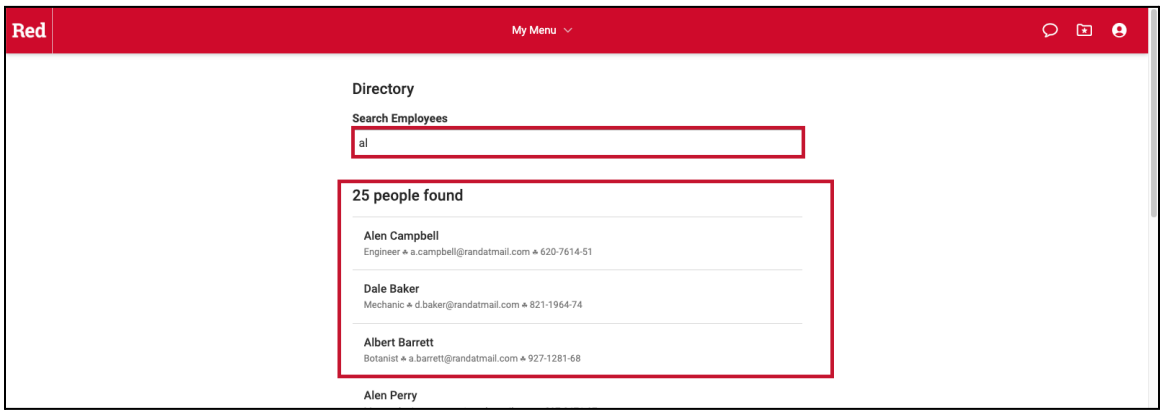
6. Type **al** for **Search Employees** then open the CloudWatch logs for lab6-task1 and look at the most recent HTTP Request information. You should have query string parameters for the form element we have specified.

7. Below the xmJson declaration in your Lambda function, set some conditions to execute some code if the **search_info** query string parameter exists.
8. Create a new file, type **export default** press enter and paste the contents of the following file for the rest of the content and save the file as directory.mjs: https://static.modolabs.com/xmodule_certification/data_files/directory.json



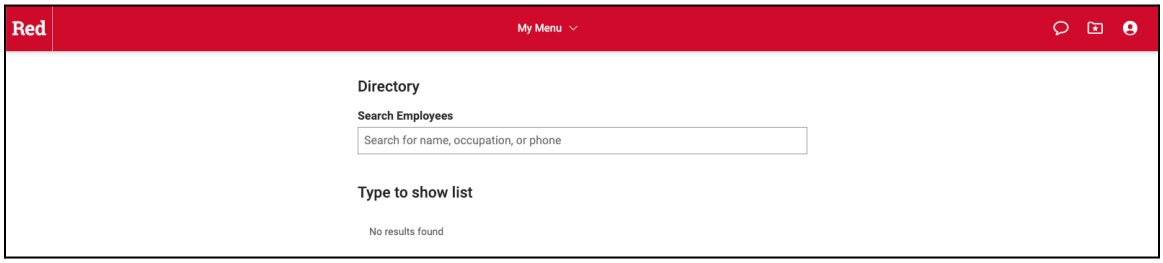
9. On the first line of your index.mjs file, type `import data from "./directory.mjs";`
10. Add code to search each entry in the directory data to see if it matches the search criteria Inside your query string parameter checking logic. The search code should do the following:
    a. Declare an array object to hold the list items you want to show
    b. Check whether the search string is found in the name or occupation as a minimum
    c. Add found items to the array for the list items
    d. Add an elementFields property to the xmJson variable that sets the items to the directory list and the heading to the number of people found for the search criteria.
    e. It could look something like the example below but when you feel it is ready, click Deploy.

11. Test it by typing al and see if you get 25 people from the list.



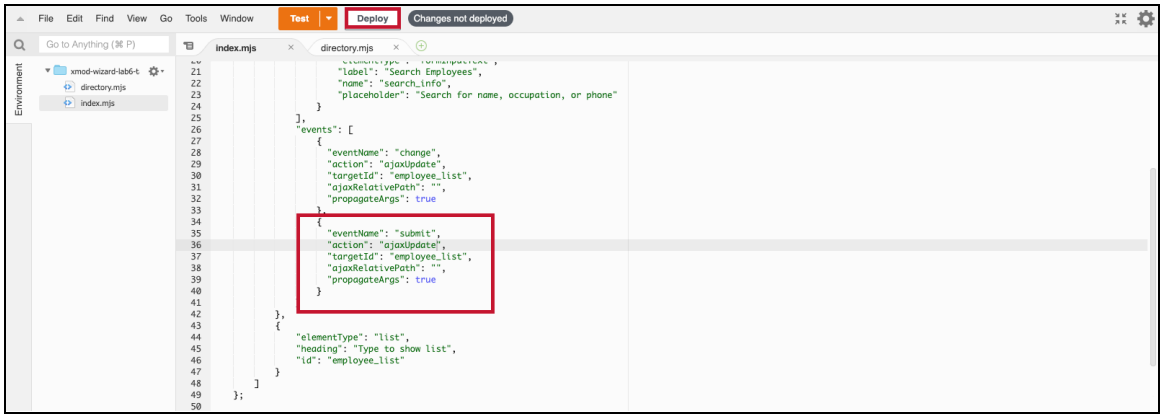
Note: If you don't get this many results you may be looking for case sensitivity in your search terms.

12. Press enter / return to see what happens to the form.



Note: If you look at CloudWatch at the most recent log group and the most recent HTTP Request you should see the http.method is set to POST. This is an undesirable result which can be fixed by overriding the form's submit event.

13. Copy the **change** event and paste it as the second item in the events array and replace the word **change** with **submit** and deploy your changes.



Congratulations! You have successfully developed a form that uses AJAX to return a result set.

Task 2: Filtering results using a dropdown

Context: Filtering is a desirable option where there is a lot of data presented at the same time. For example, finding the correct example in [Modo's showcase] is optimized by applying filters. These filters may be applied as query string parameters and result in a full page postback, as seen in the Modo Showcase, or you can handle the rendering using AJAX.

Goal: Use a dropdown list to filter content in a grid / cardset.

Task Outline:

- Select either a cardset or a grid to display a list of the quests that are available (must have an image and text options, cardset is preferred and what we will use)
- Create a Lab 6: Task 2 Lambda function with Function URL and the card set element you selected and make the corresponding XModule
- Add the xmodule_certification_quests.json file to your project as quests.mjs and import it into your index.mjs Lambda function.
- Replace the static card set data with data from the quests JSON information
- Build a form with options to filter the quest data that uses AJAX to update the list of shown elements

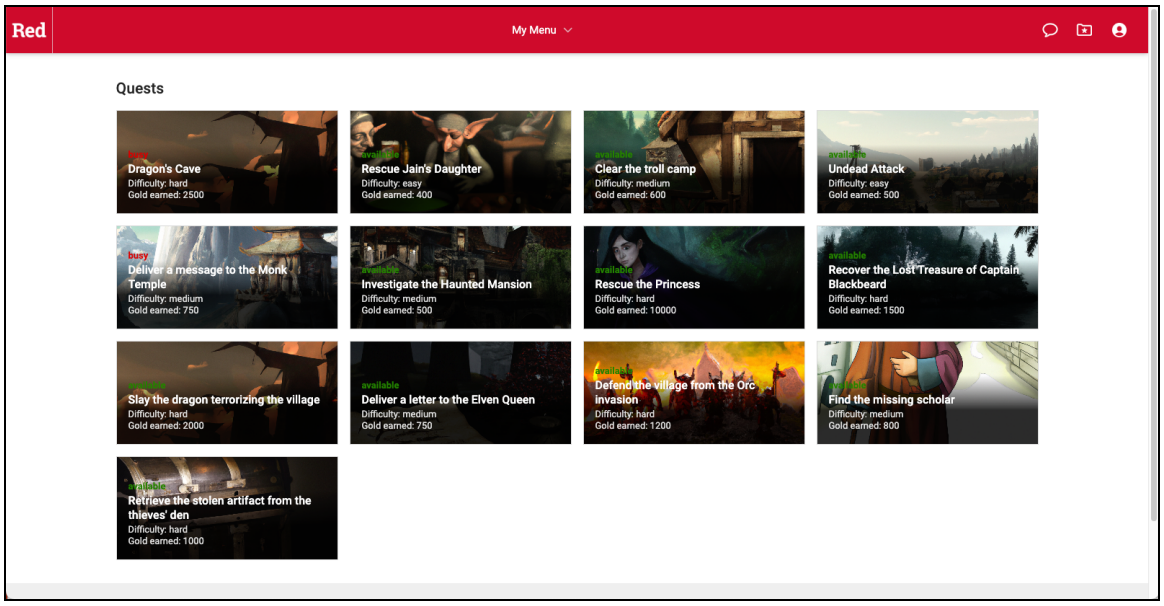
Steps:

1. Create a new Lambda function named **{xmod-prefix}-lab6-task2**, log the HTTP Request, add the code below to create an xmJson variable, then return that xmJson as the body of the request.

```
let xmJson = {
    "metadata": {
        "version": "2.0"
    },
    "contentContainerWidth": "wide",
    "content": [],
    "elementFields": {}
};
```

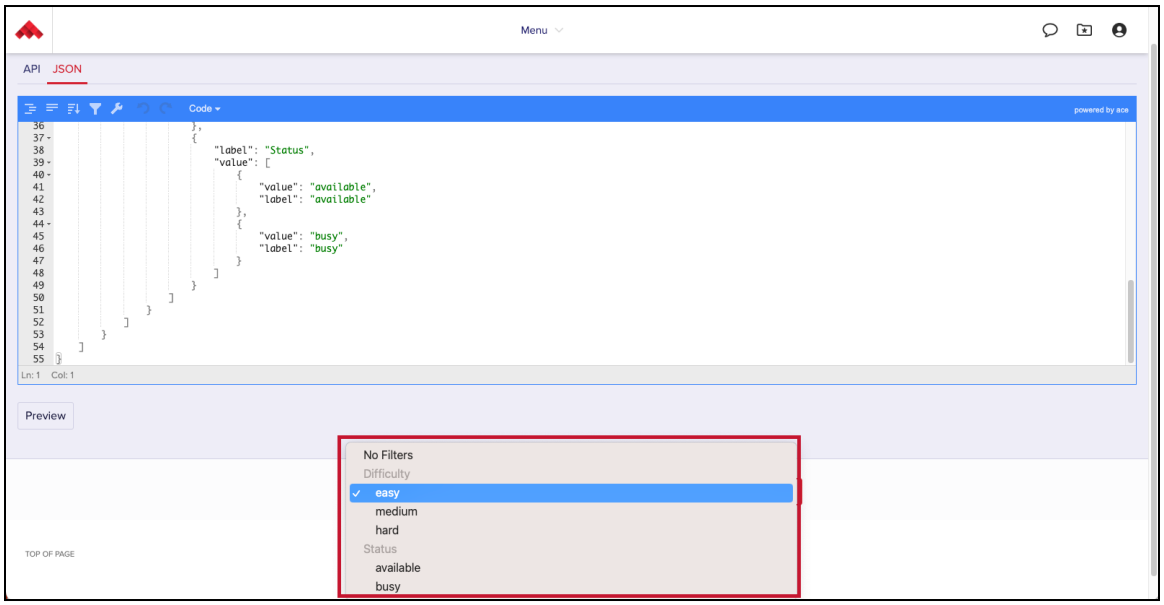
2. Click the following link and select a cardSet style that has both image and text and extract it from your Sandbox and add it to a xmCardSet variable in your Lambda function: https://xmodule.modolabs.net/default/xmodule_ui_examples/page?relativePath=.%2Fcards.json

3. Push the xmCardSet variable into your xmJson content array, deploy your Lambda function, then create a Function URL.
4. Create an XModule named **Lab 6: Task 2** with the Function URL as the API URL, deploy it to Test, and Preview your new XModule.
5. Create a new file in your Lambda function and type **export default** in the first line, press enter / return then copy and paste the JSON from the following link into the file and save it as **quests.mjs:** https://static.modolabs.com/xmodule_certification/data_files/xmodule_certification_quests.json
6. Import the quest data into your index.mjs file (must be the first line of the file).
7. Extract the cardSet items element into a function named **getCards** that takes a parameter named **filter** and have it do the following:
    a. Create an array named xmCards.
    b. Add cardSet items to the xmCards array by looping through all of the quests setting the following:
        i. Set **{quest name}** for **title**
        ii. Set **{quest status}** for **label**
        iii. Set **labelTextColor** to the **available_focal_text_color** if the status is available else set the color to **alarm_focal_text_color**
        iv. Set **Difficulty: {quest difficulty}
Gold earned: {quest gold}** for **description**
        v. For image do the following:
            1. Set **{quest photo}** for **url**
            2. Set **{quest name}** for **alt**
        vi. Return the xmCards array
8. Add **Quests** as the heading to your cardSet and set the cardSet items to the getCards function you just created.
9. Add the `cardset` element to the `xmJson` content array, **Deploy** your changes and **Preview** the changes.



Now that you have the quests visible, we need to start filtering. We should filter on status or difficulty. We can do this using a form select with option groups.

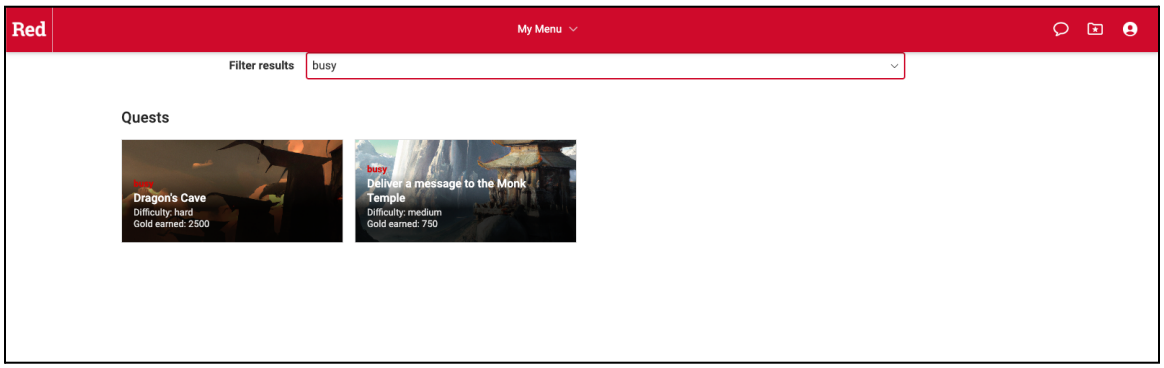
10. Go to the Form UI Components and send the **Select with option groups** example to the Sandbox.
11. Remove the other select element then edit the options to include No Filter as well as option groups for Difficulty and Status with the appropriate options below each group heading.



12. Extract the form element containing the Select with option groups object and add as a variable named xmFilter it as the first item in your xmJson content array.
13. Copy the event code from the previous exercise and this time we'll add it as a property to the formInputSelect element. Rename the targetId in the copied event to **quest_cards** and change the ID on the cardSet element to **quest_cards**.
14. Deploy the changes, Preview the page, then check the latest CloudWatch logs to see what happens when you select something from the filter options.

Proper use of AJAX only returns JSON specific to the action that is being performed. When the page loads, we build out all the items because there is no filter applied. When a filter is applied we only return the JSON that updates the desired elementFields. This is why we have each of those fields specified in the initial JSON object but we should only fill the appropriate sections based on the request.

15. Adjust the getCards function to check the filter parameter. Items should be added if any of the following conditions are true:
    a. No filter has been specified
    b. A filter of **none** has been specified
    c. If the filter value matches the status
    d. If the filter value matches the difficulty
16. Check for the query string parameters and do the following:
    a. If they exist, populate the elementField items by setting it equal to the getCards function and pass in the filter query string parameter
    b. If they do not exist, populate the content array with the xmFilter and xmCardSet elements with the cardSets items array set to the getCards function with no filter parameter set.
17. Deploy and Preview your changes to make sure your filter functions properly.

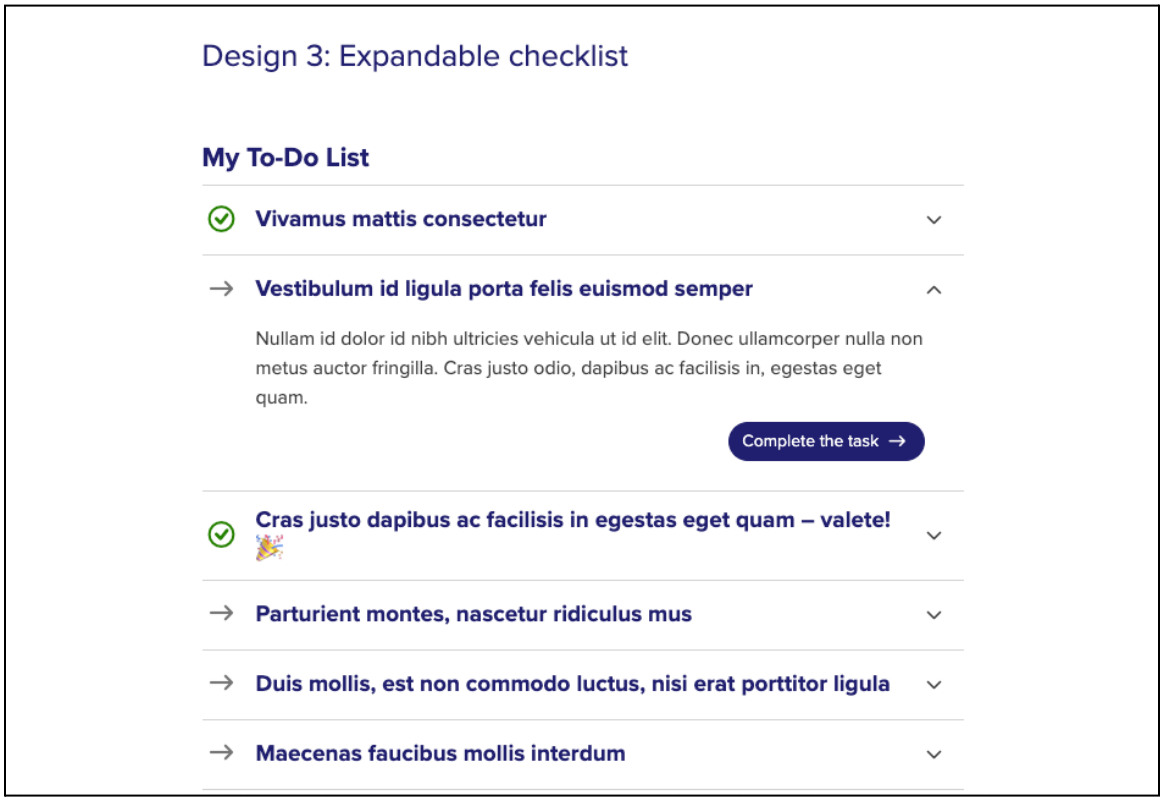


Congratulations! You have successfully created filtered content for your users. AJAX is fun!

Task 3: Adding multiple events to each click

Context: Sometimes the AJAX actions are as simple as showing or hiding content on the page based on completing a previous action. While you are not asynchronously loading content, the action of showing or hiding content on the page is still considered AJAX. Sometimes AJAX is a combination of showing / hiding content as well as AJAX loading content to the page. This requires raising multiple events on one item's action.

Goal: Modify the Expandable Checklist Showcase example to allow you to mark off items in the list and automatically select the next uncompleted task.



Task Outline:

- Get the JSON for the Expandable checklist (todo-list-3.json) from the showcase zip file found here: https://static.modolabs.com/showcase/xmodule/todo_list_examples.zip
- Add IDs to all of the collapsible elements and make all items in the JSON file collapsed by default except for the 2nd item
- Create a Lambda function named **{xmod-prefix}-lab6-task3** with an associated Lambda function and log the HTTP request
- Create a variable to hold the standard XModule JSON boilerplate information
- Add the content to the request in a different step
- Return the XModule JSON variable from the function and deploy your changes
- Create a new XModule named Lab 6: Task 3 that points to your function URL and preview the module
- For the 2nd collapsible in the list, override the click event to collapse the current element and show the next unchecked element and test the changes
- Add an ajaxUpdate event to the button click that changes the 2nd collapsible's list icon as well as the other events
- Duplicate the functionality to the rest of the uncompleted list items and test

Steps:

1. Click the following link then scroll to the bottom to preview the interaction for the Design 3: Expandable checklist showcase example: https://showcase.modolabs.com/showcase_2022/xcomponent_examples/_/to_do_list_examples

2. Download the zip files from the following link and open **todo-list-3.json** in your favorite text editor or copy and paste it into the XModule Sandbox for editing: https://static.modolabs.com/showcase/xmodule/todo_list_examples.zip
3. Find each of the `collapsible` elements and add add an id property to each with the named **taskX** where **X** is the number of the collapsible in the list (e.g. the element starting on like 21 should be have: `"id": "task1"`, added to the element.
4. Set the collapsible property to false for the element with `"id": "task1"`
5. Set all other collapsible elements `collapsed` property to **true**.
6. Create a new Lambda function named **{xmod-prefix}-lab6-task3**, log the HTTP Request and create a variable to hold the XModule JSON boilerplate JSON.
7. Extract the JSON elements from the content section of your todo-list-3.json file and add it programmatically to the XModule variable's content section.
8. Deploy your changes and click the Function URL link to view the exported JSON.


```
{"metadata":{"version":"2.0"},"content":[{"elementType":"divider","marginTop":"none","marginBottom":"none"},{"elementType":"collapsible","title":"Vivamus mattis
consectetur","id":"task1","titleFontSize":"1.1875rem","image":
{"icon":"confirm","iconColor":"theme:confirmation_focal_text_color","alt":"Complete"},"imageWidth":"2rem","imageHeight":"2rem","collapsed":true,"borderTopStyle":"none","borderBottomStyle":"none","heading
PaddingTop":"tight","headingPaddingBottom":"tight","marginTop":"none","marginBottom":"none","content":
[{"elementType":"container","paddingLeft":"2.75rem","paddingRight":"2rem","paddingBottom":"xtight","content":[{"elementType":"html","html":"Praesent commodo cursus magna, vel scelerisque nisl consectetur
et"}]}]},{"elementType":"divider","marginTop":"none","marginBottom":"none"},{"elementType":"collapsible","id":"task2","title":"Vestibulum id ligula porta felis euismod
semper","titleFontSize":"1.1875rem","image":
{"icon":"next","iconColor":"theme:tertiary_text_color","alt":"Incomplete"},"imageWidth":"2rem","imageHeight":"2rem","collapsed":false,"borderTopStyle":"none","borderBottomStyle":"none","headingPaddingTop
":"tight","headingPaddingBottom":"tight","marginTop":"none","marginBottom":"none","content":[{"elementType":"container","paddingLeft":"2.75rem","paddingRight":"2rem","paddingBottom":"xtight","content":
[{"elementType":"html","html":"Nullam id dolor id nibh ultricies vehicula ut id elit. Donec ullamcorper nulla non metus auctor fringilla. Cras justo odio, dapibus ac facilisis in, egestas eget quam."},
{"elementType":"buttonContainer","marginTop":"xtight","horizontalAlignment":"right","buttons":[{"elementType":"linkButton","size":"small","borderRadius":"full","actionType":"emphasized","title":"Complete
the task","icon":"next","iconPosition":"right","link":{"external":"https://www.example.com/path/to/deep/link"}}]}]}]},{"elementType":"divider","marginTop":"none","marginBottom":"none"},
{"elementType":"collapsible","id":"task3","title":"Cras justo dapibus ac facilisis in egestas eget quam – valete! 🎉","titleFontSize":"1.1875rem","image":
{"icon":"confirm","iconColor":"theme:confirmation_focal_text_color","alt":"Complete"},"imageWidth":"2rem","imageHeight":"2rem","collapsed":true,"borderTopStyle":"none","borderBottomStyle":"none","heading
PaddingTop":"tight","headingPaddingBottom":"tight","marginTop":"none","marginBottom":"none","content":
[{"elementType":"container","paddingLeft":"2.75rem","paddingRight":"2rem","paddingBottom":"xtight","content":[{"elementType":"html","html":"Nullam id dolor id nibh ultricies vehicula ut id elit."}]}]},
{"elementType":"divider","marginTop":"none","marginBottom":"none"},{"elementType":"collapsible","id":"task4","title":"Parturient montes, nascetur ridiculus mus","titleFontSize":"1.1875rem","image":
{"icon":"next","iconColor":"theme:tertiary_text_color","alt":"Incomplete"},"imageWidth":"2rem","imageHeight":"2rem","collapsed":true,"borderTopStyle":"none","borderBottomStyle":"none","headingPaddingTop"
:"tight","headingPaddingBottom":"tight","marginTop":"none","marginBottom":"none","content":[{"elementType":"container","paddingLeft":"2.75rem","paddingRight":"2rem","paddingBottom":"xtight","content":
[{"elementType":"html","html":"<p>Nulla vitae elit libero, a pharetra augue. <a href='https://example.com'>Maecenas sed diam</a> eget risus varius blandit sit amet non magna. Parturient montes, nascetur
ridiculus mus. Lorem ipsum dolor sit amet, consectetur adipiscing elit.</p><ul><li>Integer posuere erat a ante venenatis dapibus</li><li>Posuere velit aliquet</li><li>Donec sed odio dui</li><li>Fusce
dapibus, tellus ac cursus commodo</li></ul><p>Donec sed odio dui. Vivamus sagittis lacus vel augue laoreet rutrum faucibus dolor auctor. Donec ullamcorper nulla non metus.</p>"},
{"elementType":"buttonContainer","marginTop":"xtight","horizontalAlignment":"right","buttons":[{"elementType":"linkButton","size":"small","borderRadius":"full","actionType":"emphasized","title":"Do the
thing","icon":"next","iconPosition":"right","link":{"external":"https://www.example.com/path/to/deep/link"}}]}]}]},{"elementType":"divider","marginTop":"none","marginBottom":"none"},
{"elementType":"collapsible","id":"task5","title":"Duis mollis, est non commodo luctus, nisi erat porttitor ligula","titleFontSize":"1.1875rem","image":
{"icon":"next","iconColor":"theme:tertiary_text_color","alt":"Incomplete"},"imageWidth":"2rem","imageHeight":"2rem","collapsed":true,"borderTopStyle":"none","borderBottomStyle":"none","headingPaddingTop"
:"tight","headingPaddingBottom":"tight","marginTop":"none","marginBottom":"none","content":[{"elementType":"container","paddingLeft":"2.75rem","paddingRight":"2rem","paddingBottom":"xtight","content":
[{"elementType":"html","html":"Nulla vitae elit libero, a pharetra augue. Donec sed odio dui. Vestibulum id ligula porta felis euismod semper. Etiam porta sem malesuada magna mollis euismod. Fusce
dapibus, tellus ac cursus commodo, tortor mauris condimentum nibh, ut fermentum massa justo sit amet risus."},
{"elementType":"buttonContainer","marginTop":"xtight","horizontalAlignment":"right","buttons":[{"elementType":"linkButton","size":"small","borderRadius":"full","actionType":"emphasized","title":"Ipsa
scientia potestas est","icon":"next","iconPosition":"right","link":{"external":"https://www.example.com/path/to/deep/link"}}]}]}]},{"elementType":"divider","marginTop":"none","marginBottom":"none"},
{"elementType":"collapsible","id":"task6","title":"Maecenas faucibus mollis interdum","titleFontSize":"1.1875rem","image":
{"icon":"next","iconColor":"theme:tertiary_text_color","alt":"Incomplete"},"imageWidth":"2rem","imageHeight":"2rem","collapsed":true,"borderTopStyle":"none","borderBottomStyle":"none","headingPaddingTop"
:"tight","headingPaddingBottom":"tight","marginTop":"none","marginBottom":"none","content":[{"elementType":"container","paddingLeft":"2.75rem","paddingRight":"2rem","paddingBottom":"xtight","content":
[{"elementType":"html","html":"<p>Curabitur blandit tempus porttitor. Donec sed odio dui. Nullam quis risus eget urna mollis ornare vel eu leo. Lorem ipsum dolor sit amet, consectetur adipiscing elit.
</p><p>Nulla vitae elit libero, a pharetra augue. Maecenas sed diam eget risus varius blandit sit amet non magna.</p>"},
{"elementType":"buttonContainer","marginTop":"xtight","horizontalAlignment":"right","buttons":[{"elementType":"linkButton","size":"small","borderRadius":"full","actionType":"emphasized","title":"Go do
the thing","icon":"next","iconPosition":"right","link":{"external":"https://www.example.com/path/to/deep/link"}}]}]}]},{"elementType":"divider","marginTop":"none","marginBottom":"none"}]}
```


9. Create an XModule named **Lab 6: Task 3** that uses the Function URL for XModule API URL, deploy it to test and preview the module.

10. Complete the following steps to add an event to the **Complete the task ->** button in the second collapsible element:
    a. Delete the "link": element
    b. Add the code below to the button:

```
"events": [
    {
        "eventName": "click",
        "action": "close",
```

```
                "targetId": "task2"
            },
            {
                "eventName": "click",
                "action": "open",
                "targetId": "task4"
            }
        ]
```

c. Deploy your change and preview the output by clicking the **Complete the task ->** button.

As long as the current element you are on collapses correctly and it shows the next “To Do” item in the list, you are ready to change the list’s image and color to the completed text version. If you are still not opening and closing the element’s correctly, please contact an instructor for assistance.

11. Add the following event to the button’s event’s array, deploy the function, and click the button to preview it in test:

```
            {
                "eventName": "click",
                "action": "ajaxUpdate",
                "targetId": "task2",
                "ajaxRelativePath": "",
                "postData": {
                    "complete": true
                }
            }
```

12. Open CloudWatch to view the difference between the page load HTTP Request and the button press HTTP Request.
13. Create a condition in your code to evaluate the difference between the requests and return the `content` element if it is the initial request
14. Add code to return elementFields to fulfill the condition where the button was pressed that returns elementFields that sets the image element.

```
xmJson.elementFields = {
    image: {
        "icon": "confirm",
        "iconColor": "theme:confirmation_focal_text_color",
        "alt": "Complete"
    }
}
```

15. Try the code again to see how the interaction responds.

Note: The hidden element should be shown again. This is because elements affected by AJAX Update revert to their original state with the exception of the items that have changed from the update. Since the original state of the “task2” element is open, the element reverts to an open state. To fix this, set collapsed: true as a field for the element in the return JSON.

16. If this works correctly, apply the same events to each button while updating the IDs appropriately then Deploy to Test and Preview your changes.
17. sdf

Lab 7: Full Page Layout

Estimated Time: 30 minutes

Many customers want a desktop portal to compliment their mobile application. Your Modo application can support both platforms by leveraging responsive design principles. There are certain properties you can apply to your JSON that target desktop compatible layout instead of a traditional mobile first design. The following tasks help you discover how those settings function inside Modo.

Task 1: Column types, contentContainerWidth, and headers

Context: Many desktop portal centric designs bring content to the forefront using columns. Modo supports two major column types: simple columns and responsive columns. While these elements do not need to be the topmost element in your `content` array, they do need to be top level elements and cannot be nested inside any other element (e.g. they cannot be in the content array of a container element). To maximize the viewable space for your columns, you should set your `contentContainerWidth` to an appropriate viewport size. You can also set a `header` for the page that is rendered as a header element and sits outside the normal page viewport / container.

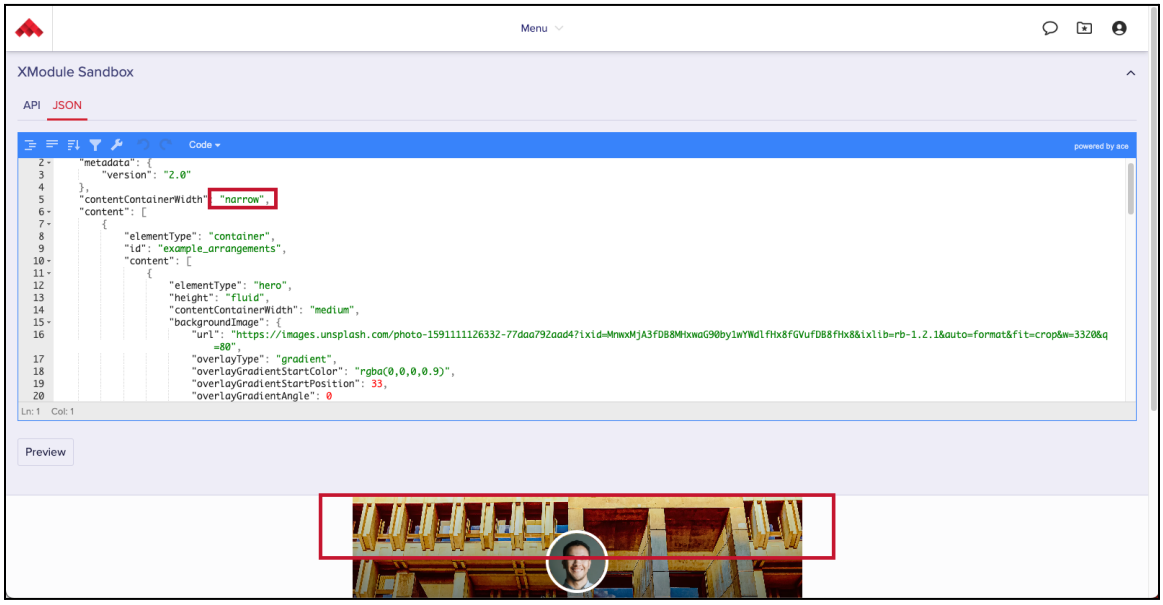
Goal: Test display properties on the different column types and experiment with how the `contentContainerWidth` property affects page layout and see how the header

Task Outline:

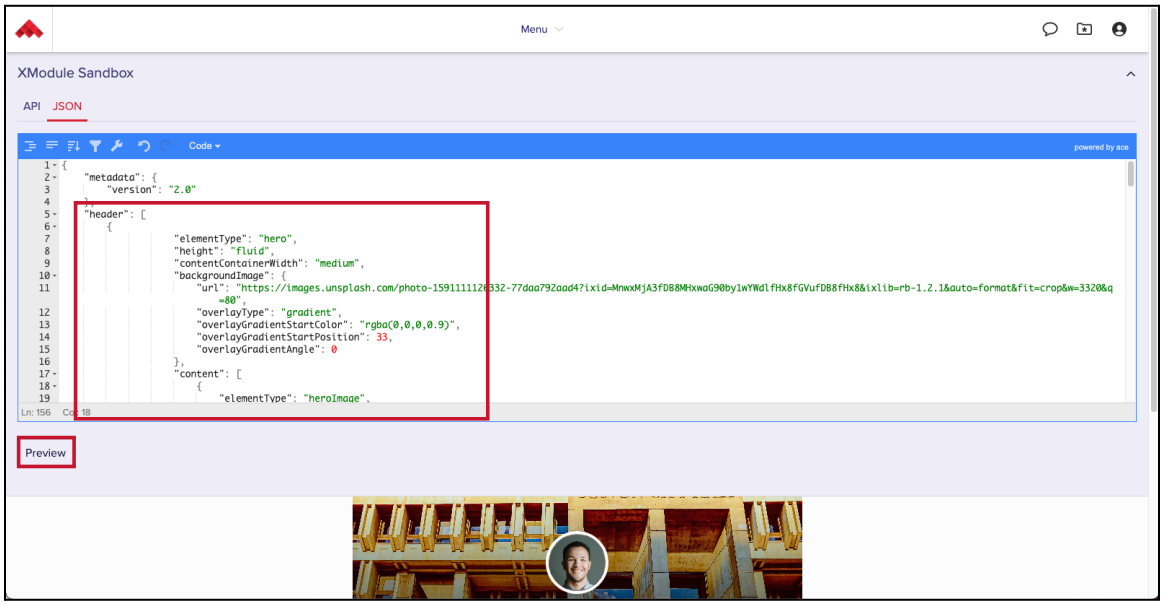
- Find the hero element to extract and view in the XModule Sandbox
- Change the contentContainerWidth property to narrow, medium, wide and full then go back to narrow when you preview how each looks
- Copy the hero block to the header section
- Add a responsive three column element to the page using JSON from step 5 and preview the output
- Copy and paste the responsive three column JSON, add a divider below the existing columns and paste a new column group below the divider then change the sideMargins and gutters to their non-default settings to see what happens
- Change the second responsive three column element to a simple three column element and change the columns and content headings appropriately and preview the changes
- Use the Developer toolbar to preview how the pages respond on different screens

Steps:

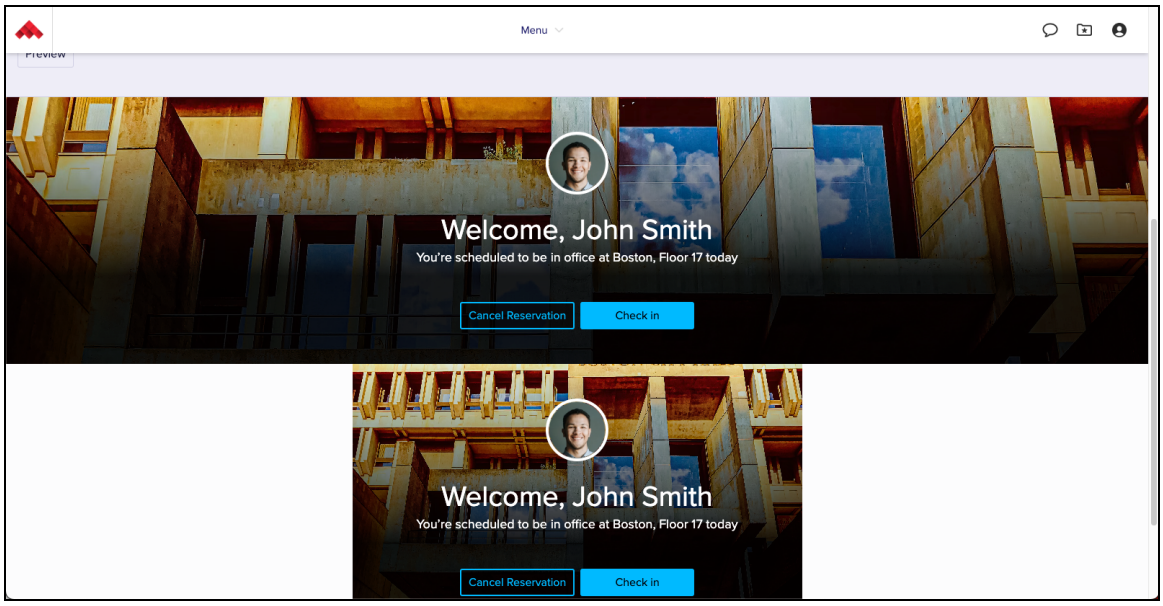
1. Open the UI Component Hero examples by clicking the following link: https://xmodule.modolabs.net/default/xmodule_ui_examples/page?relativePath=.%2Fhero.json
2. Select an example and isolate it in the XModule Sandbox.
3. Locate the contentContainerWidth property on line 5 in the Sandbox. This property is explained in the Screen Level Properties support document and has allowable values of `narrow`, `medium`, `wide`, and `full`. Change the property to try each of these values, clicking **Preview** between each one to view the changes, finally set the property to **narrow** and click **Preview**.



4. Add a new property above contentContainerWidth name **header** which is an array then copy the hero block into the header array and click **Preview**.



Note: The result has the items in the `header` spanning the entire width of the page while the `hero` element inside the `content` array is constrained to the narrow viewport.



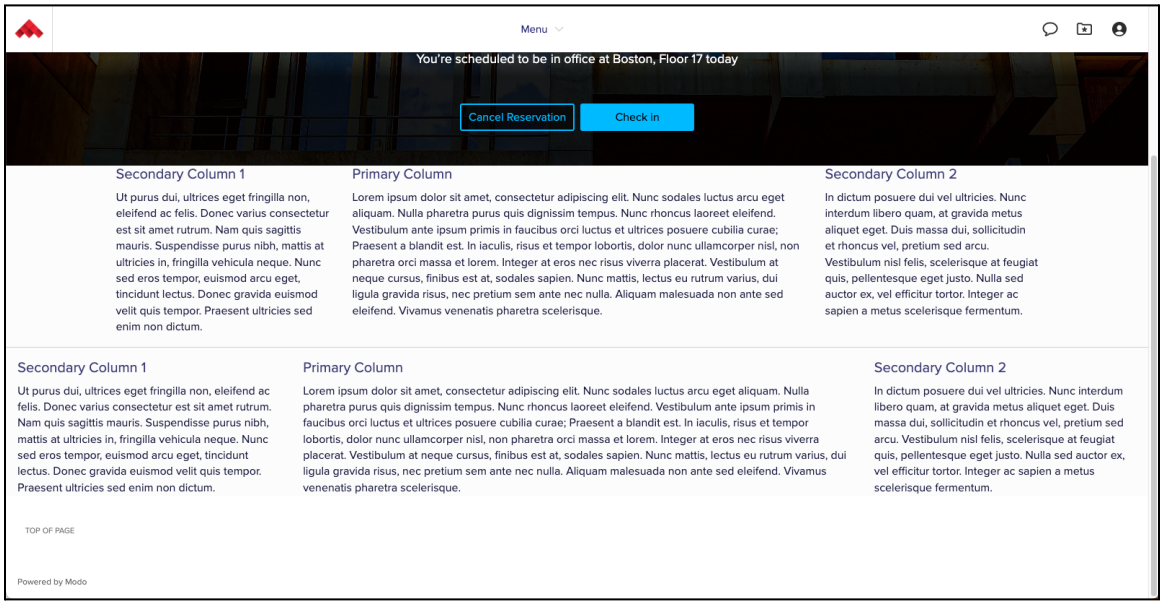
5. Remove the hero from the content array and add the JSON below to the content array.

```
        {
            "elementType": "responsiveThreeColumn",
            "primaryColumn": {
                "content": [
                    {
                        "elementType": "html",
                        "html": "<h2>Primary Column</h2><p>Lorem ipsum dolor sit amet,
consectetur adipiscing elit. Nunc sodales luctus arcu eget aliquam. Nulla pharetra
purus quis dignissim tempus. Nunc rhoncus laoreet eleifend. Vestibulum ante ipsum
primis in faucibus orci luctus et ultrices posuere cubilia curae; Praesent a blandit
est. In iaculis, risus et tempor lobortis, dolor nunc ullamcorper nisl, non pharetra
orci massa et lorem. Integer at eros nec risus viverra placerat. Vestibulum at neque
cursus, finibus est at, sodales sapien. Nunc mattis, lectus eu rutrum varius, dui
ligula gravida risus, nec pretium sem ante nec nulla. Aliquam malesuada non ante sed
eleifend. Vivamus venenatis pharetra scelerisque.</p>"
                    }
                ]
            },
            "secondaryColumn1": {
                "content": [
                    {
                        "elementType": "html",
                        "html": "<h2>Secondary Column 1</h2><p>Ut purus dui, ultrices
eget fringilla non, eleifend ac felis. Donec varius consectetur est sit amet rutrum.
Nam quis sagittis mauris. Suspendisse purus nibh, mattis at ultricies in, fringilla
vehicula neque. Nunc sed eros tempor, euismod arcu eget, tincidunt lectus. Donec
gravida euismod velit quis tempor. Praesent ultricies sed enim non dictum.</p>"
                    }
                ]
            },
```

```
            "secondaryColumn2": {
                "content": [
                    {
                        "elementType": "html",
                        "html": "<h2>Secondary Column 2</h2><p>In dictum posuere dui
vel ultricies. Nunc interdum libero quam, at gravida metus aliquet eget. Duis massa
dui, sollicitudin et rhoncus vel, pretium sed arcu. Vestibulum nisl felis, scelerisque
at feugiat quis, pellentesque eget justo. Nulla sed auctor ex, vel efficitur tortor.
Integer ac sapien a metus scelerisque fermentum.</p>"
                    }
                ]
            }
        }
```

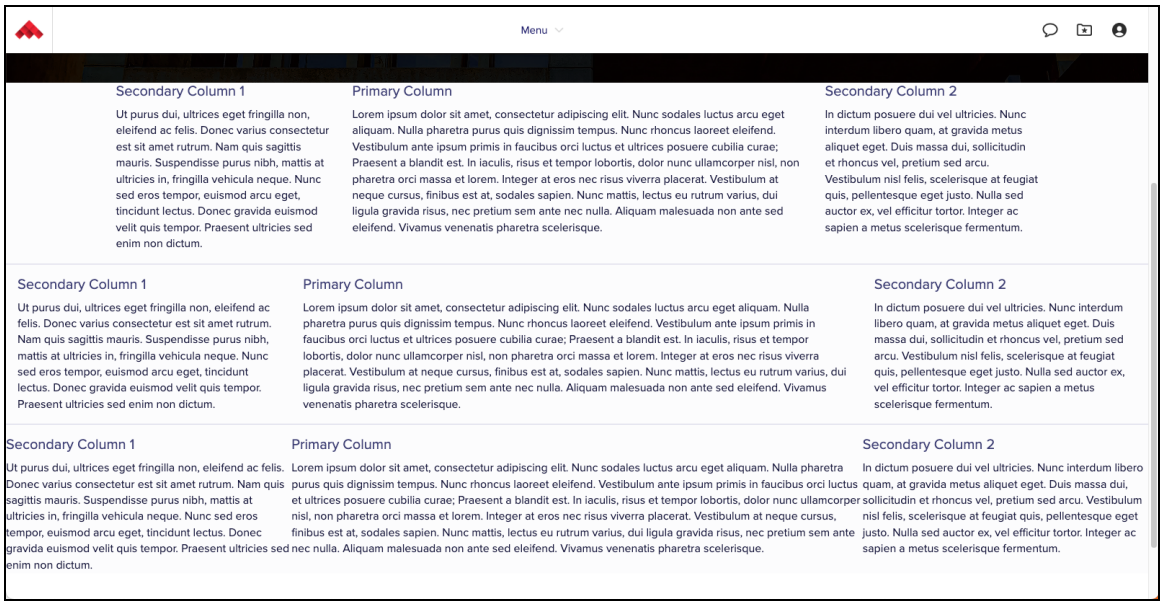
Note: This JSON creates a responsiveThreeColumn layout. Many customers want data stored in 3 columns.

6. Change the `contentContainerWidth` to cycle through `narrow`, `medium`, `wide`, and end with `full` to see the differences between each version.
7. Add a divider element after the existing `responsiveThreeColumn` element then copy and paste the element below the divider and add the property: `"sideMargins": "none"`, to the new `responsiveThreeColumn` element.

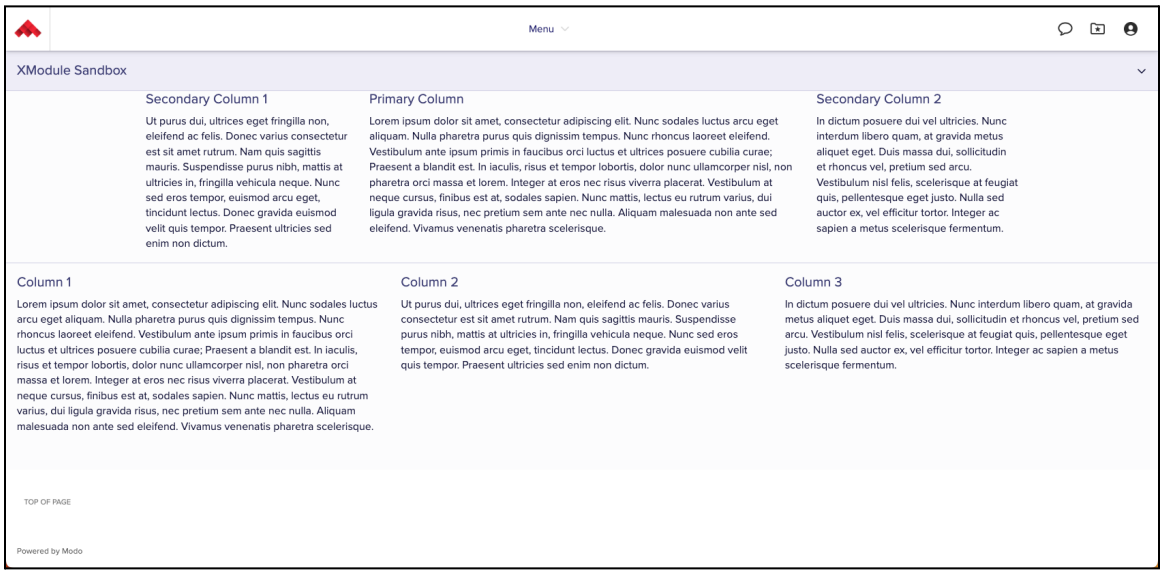


Notice the difference between the default settings for the responsiveThreeColumns element when sideMargins are set to “none”. There is still some spacing between the sides of the screen. This has to do with something called gutters that determines the horizontal padding between columns.

8. Copy and paste the divider and bottom `responsiveThreeColumns` element and add **`“gutters”: false,`** to the new `responsiveThreeColumns` object.



9. Shrink the page to cause the columns to collapse. Notice how the Secondary Column 1 becomes the top column. You can change this using the `responsiveBehavior` property. Setting this to `primaryFirst` causes the `primaryColumn` to remain as the focus for your users when the screen shrinks.
10. Copy and paste the last divider and responsiveThreeColumn items then change it to a responsiveTwoColumn element type, change secondaryColumn1 to secondaryColumn and delete the secondaryColumn2 object, delete the sideMargins and gutters properties, and press **Preview**.
11. You can change which side the secondaryColumn appears on using the primarySide property.
12. Delete the header in the sandbox and everything except the first three items in the content array from the Sandbox, change the third item to a simpleThreeColumn element type, change the column name properties to column1, column2, and column3 then **Preview** your changes.



13. Resize the window to see what happens to the columns as you reduce your screen size.
14. Use Google Developer tools and the Device toolbar to emulate different devices and see the screen size differences.

Congratulations! You have experimented with contentContainerWidth and header properties and tried out the different columns available for building full page layout compatible content.

Task 2: Using the responsiveVisibility property

Context: The `responsiveVisibility` property allows you to show or hide elements based on the screens viewport. The JSON contains the layout for all versions of the elements when it is created. As you change the viewport size elements automatically show or hide based on the `responsiveVisibility` display properties. This is especially powerful for customers using the desktop portal as they may choose to change their browser width and still need the designs to work in each area of viewable space.

Goal: Experiment with the responsiveVisibility property and see different ways it can be used.

Task Outline:

- Take the code from the responsive visibility support document and paste it into the sandbox
- Replace the JSON from the Sandbox with JSON found in step 5 below
- Create another hero element using the dragon_cave_wide.png which is hidden for **xsmall**, **small** and **medium** page sizes and set the initial hero section to be hidden for **large** and **xlarge** page sizes

Steps:

1. Visit the Responsive Visibility support document at the following link and copy the code from the provided code block: https://support.modolabs.com/support/solutions/articles/13000097638-responsive-visibility
2. Open the Sandbox and replace all the contents with the JSON copied from the above support article then click Preview to show the example.

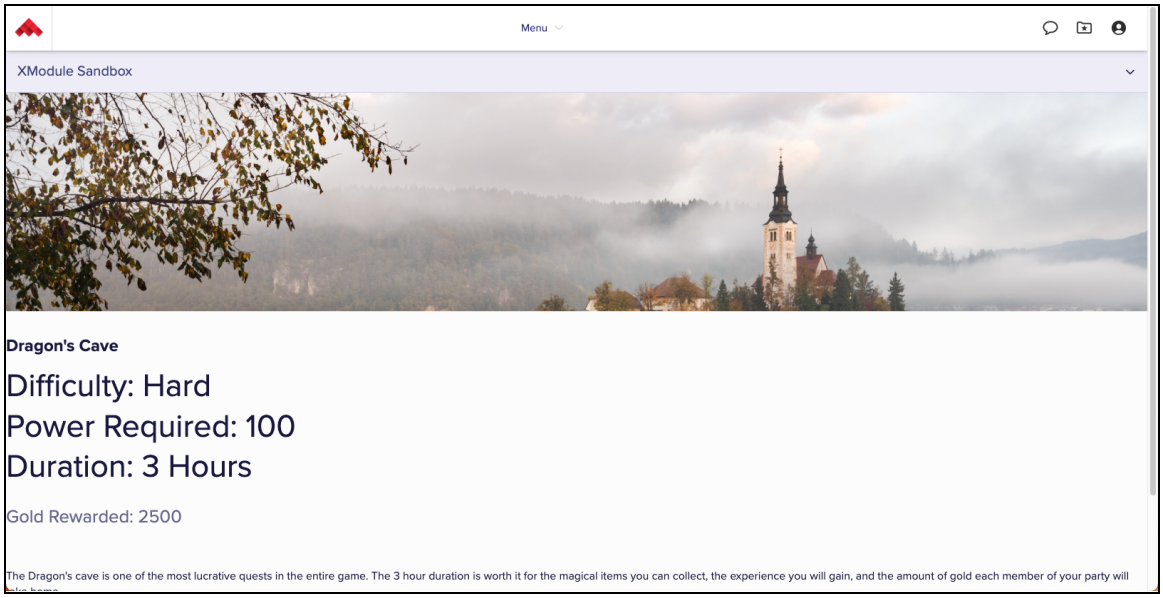
3. Shrink the page to see how the left navigation becomes a dropdown when the page content shrinks.
4. Expand the JSON and set all of the responsiveVisibility settings to false on the dropdownSelect element and remove them from the list element then Preview the page and adjust the screen to see how it responds to different screen sizes.
5. Delete the JSON in the Sandbox and replace it with the JSON snippet below and click Preview:

```
{
    "metadata": {
        "version": "2.0"
    },
    "contentContainerWidth": "wide",
```

```
    "content": [
        {
            "elementType": "detail",
            "id": "image_hero",
            "title": "Difficulty: Hard<br>Power Required: 100<br>Duration: 3 Hours",
            "description": "Gold Rewarded: 2500",
            "body": "The Dragon's cave is one of the most lucrative quests in the
entire game. The 3 hour duration is worth it for the magical items you can collect, the
experience you will gain, and the amount of gold each member of your party will take
home.",
            "label": "Dragon's Cave",
            "thumbnailBorderRadius": "1rem",
            "hero": [
                {
                    "elementType": "image",
                    "url":
"https://s3.us-west-2.amazonaws.com/static.modolabs.com/xmodule_certification/images/dr
agon_cave_narrow.png"
                }
            ]
        }
    ]
}
```

Note: The default action for an image is to fill the entire width of the element that contains it. Since the image is a hero image, it takes the entire width of the page, which is great on smaller page sizes but means you have a lot of image to scroll through if you want to see the content. We can fix this by having two versions of the image and using responsiveVisibility to determine which version to show.

6. Add responsiveVisibility settings to hide the element on medium or largers page sizes.
7. Copy the image element in the hero section and paste it as another item in the hero array with the image URL set to https://s3.us-west-2.amazonaws.com/static.modolabs.com/xmodule_certification/images/dragon_cave_wide.png and set responsiveVisibility to hide the content on small and xsmall page sizes.
8. Preview your changes and make sure you see different images on each page size.



Congratulations! You have successfully implemented responsiveVisibility in your JSON elements. You may only need this for rare occasions as many of the responsive properties of your Modo application automatically handle multiple screen sizes. It is just a great tool to have in your toolkit!

Lab 8: Miscellaneous topics

Some important XModule concepts do not fit nicely into a specific category so they have been gathered in one final group of tasks. These tasks include adding XModule based items as Bookmarks / Favorites, setting additional screen-level properties, using redirection, creating and consuming cookies, debugging your XModules with XLogger, and using the `barcodeButton` element.

Task 1: Collage screen XComponent blocks

Some people mistakenly believe an XComponent is a group of XModule elements that perform a specific function. This is incorrect. An XComponent is eXternal / eXtensible content that is embedded into a non-XModule page. These elements can be added to the header and footer section of many pages and specific element types can be added to both Collage and Mondrian publish page types. These tasks reinforce best practices when creating XComponents.

Context: This is the easiest transition for us to make into the world of XComponents. Collage screens are the most flexible of all publish screen types which means most of the XModules we have created to this point can be embedded into a Collage screen XComponent. In fact, multiple XComponents can be added to a single Collage page to build an entire portal experience. This is especially helpful when you have out-of-the-box datasources as well as XModule based content that needs to be combined into one page.

Goal: Add XModule based content as an XComponent block in a Collage page.

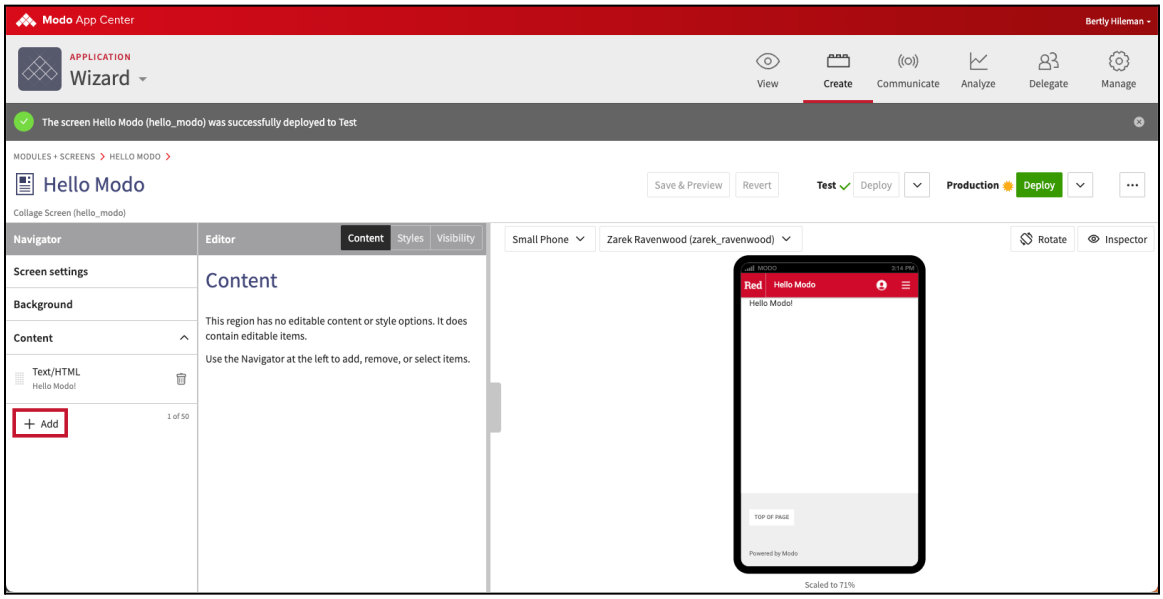
Task Outline:

- Get the JSON for a hero element then create a {xmod-prefix}-lab8 Lambda function with Function URL and return the Hero element JSON from that Lambda function

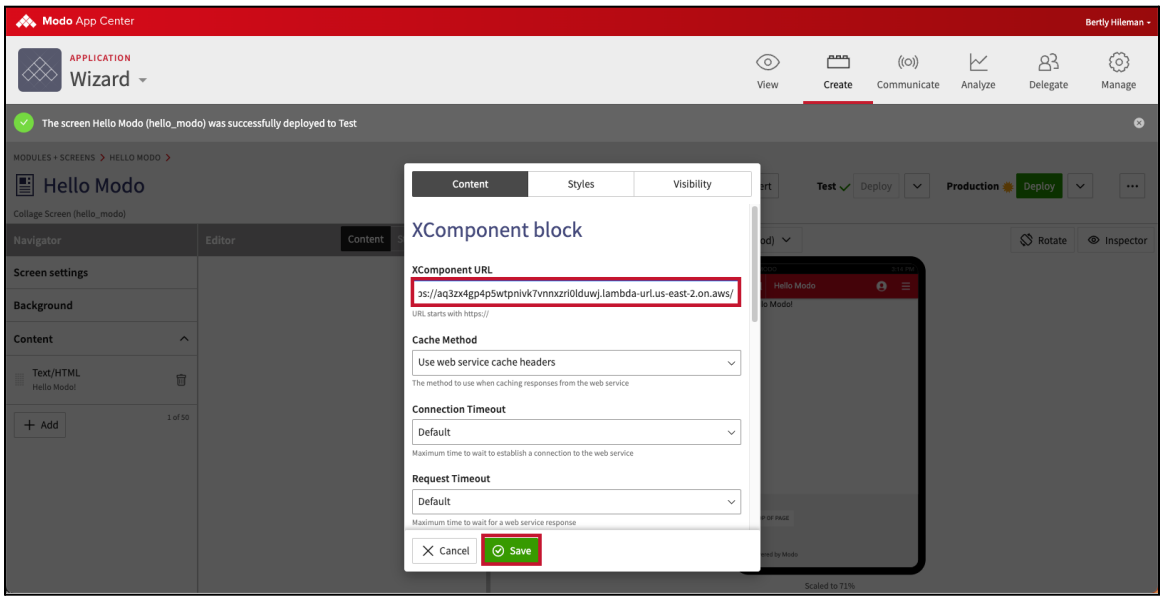
- Create an XComponent on your Hello Modo screen with the function URL from the element above

Steps:

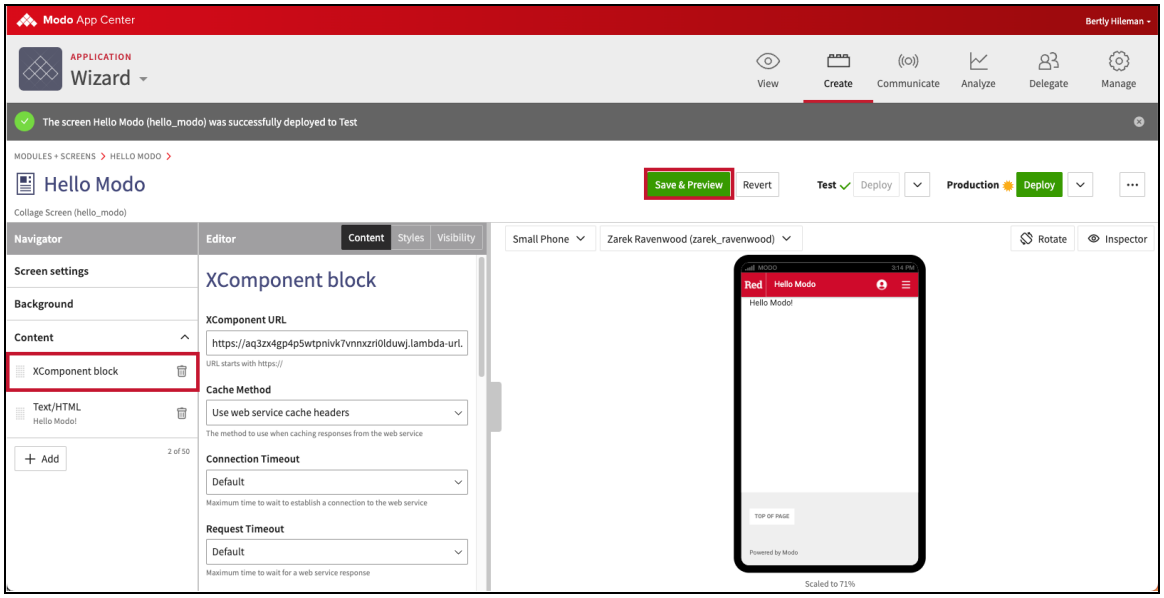
1. Get a hero element from the following page, isolate it and copy the Sandbox JSON to your clipboard: https://xmodule.modolabs.net/default/xmodule_ui_examples/page?relativePath=.%2Fhero.json
2. Create a new Lambda function named **{xmod-prefix}-lab8** with Function URL, log the HTTP Request, create an xmJson variable, set the variable to the contents from your clipboard, return your xmJson variable as the body of your response and Deploy your changes.
3. Go to your Modo App Center admin and find your Hello Modo module and screen, expand the Content section and click Add.



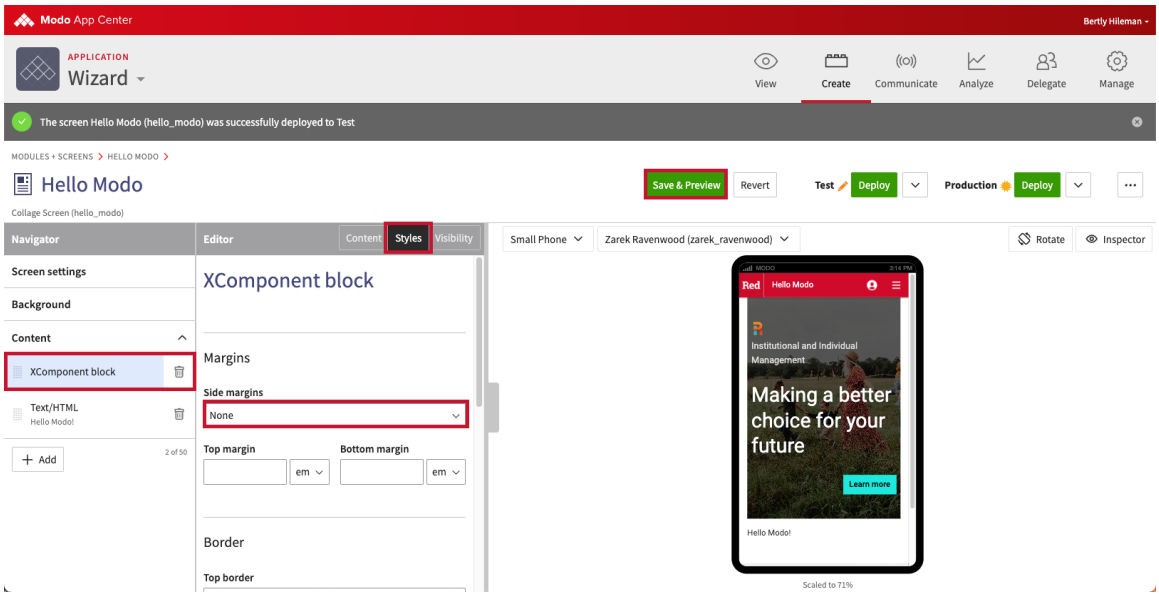
4. Select **XComponent** block and paste the Function URL for **XComponent URL**.



5. Drag the XComponent block to the top of the content section and click Save & Preview.



6. You can adjust the margins on the XComponent block by selecting the **XComponent block**, clicking **Styles** in the Editor, selecting **None** for **Side margins**, then clicking **Save & Preview**.



7. Deploy the page to **Test** and Preview your Hello Modo screen.

   Note: if the page does not display in a way you think is correct, remove the `contentContainerWidth` setting and test again.
8. You can add additional elements to the current Lambda function to get additional content into your XComponent.

Congratulations! You have added XModule content to your publisher screen and can take it even further to personalize the experience using information from the JWT.

Task 2: Redirects and Cookies

Context: There may be times you want to use XModule to redirect people to a new action within their Modo application or to some external site. This can be useful for providing a Post/Redirect/Get pattern where you

want to accept the data from a form and then redirect back to the form using GET to reduce duplicate submissions. Also, as stated in the Redirection XModule support article, “this becomes particularly useful when the URL for the destination page needs to be dynamic.” This could be using information from the JWT to dynamically build a deep linking URL that is pointed towards an external system.

Cookies can be used throughout your XModule development to store temporary data and reduce trips to backend systems. They can also be paired with redirection to hide information from the query string and add it to the JWT payload. In this task

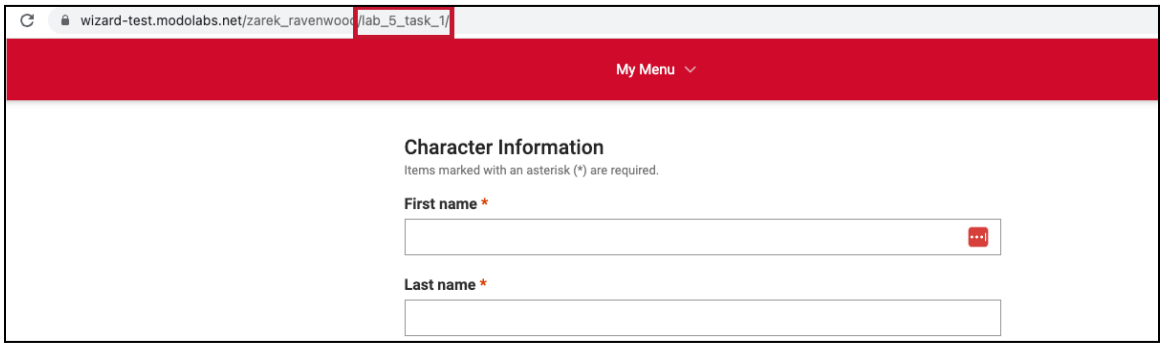
Goal: Implement XModule based redirection with cookies to store the banner information

Task Outline:

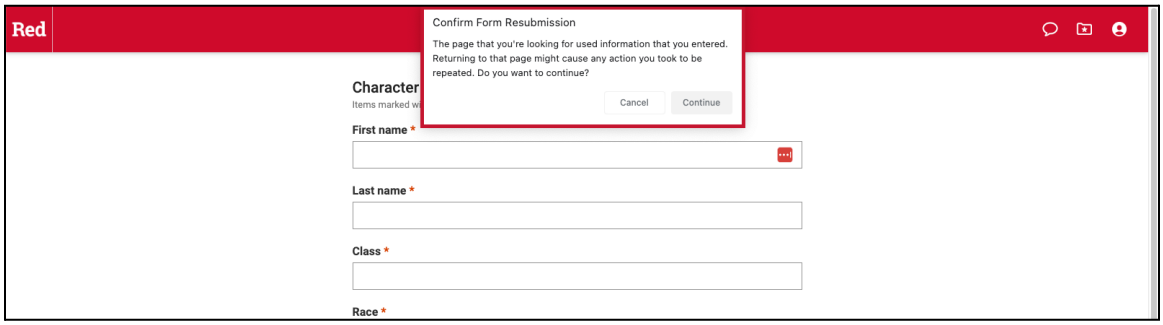
- Use the form created in Lab 5: Task 1, change the form requestMethod to POST to see what happens when you press refresh on a page when a form has been submitted
- Change the form back to GET and do the same thing
- Modify the form once it has been submitted to create a cookie with a banner message the form submission
- Redirect to the form creation page
- Check for the cookie and apply a banner then clear the cookie so the banner does not appear again

Steps:

1. Open your {xmod-prefix}-lab5-task1 Lambda function and change the form’s **requestMethod** to **POST** then open the **Lab 5: Task 1** Module preview page to view the XModule.



2. Submit the form, then refresh the page using **Ctrl/Cmd + R** or by pressing the refresh button and look for the **Confirm Form Resubmission** dialog.

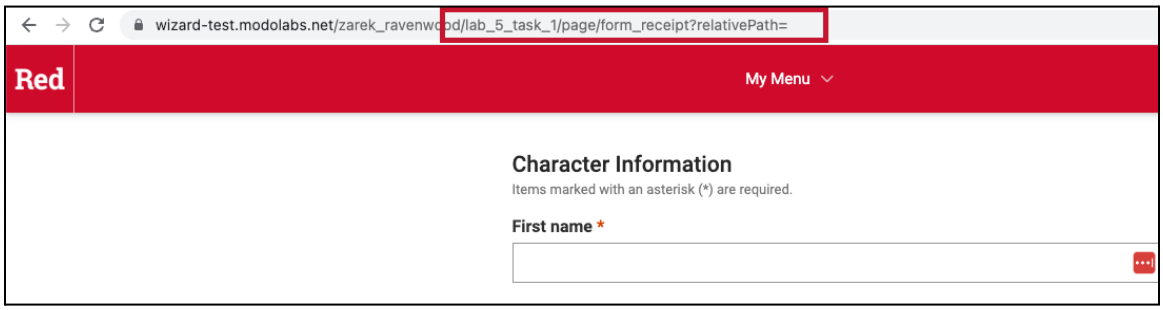


3. Cancel the form and open the **{xmod-prefix}-lab5-task1** Lambda function.
4. Look at the most recent CloudWatch logs. You should see back-to-back POST events.
5. Change the form’s requestMethod to GET. Load the form and refresh it to see what happens.

Note: There is no Confirm Form Resubmission popup because the items are appended to the query string parameters. This would effectively submit the form and then cause a resubmission.

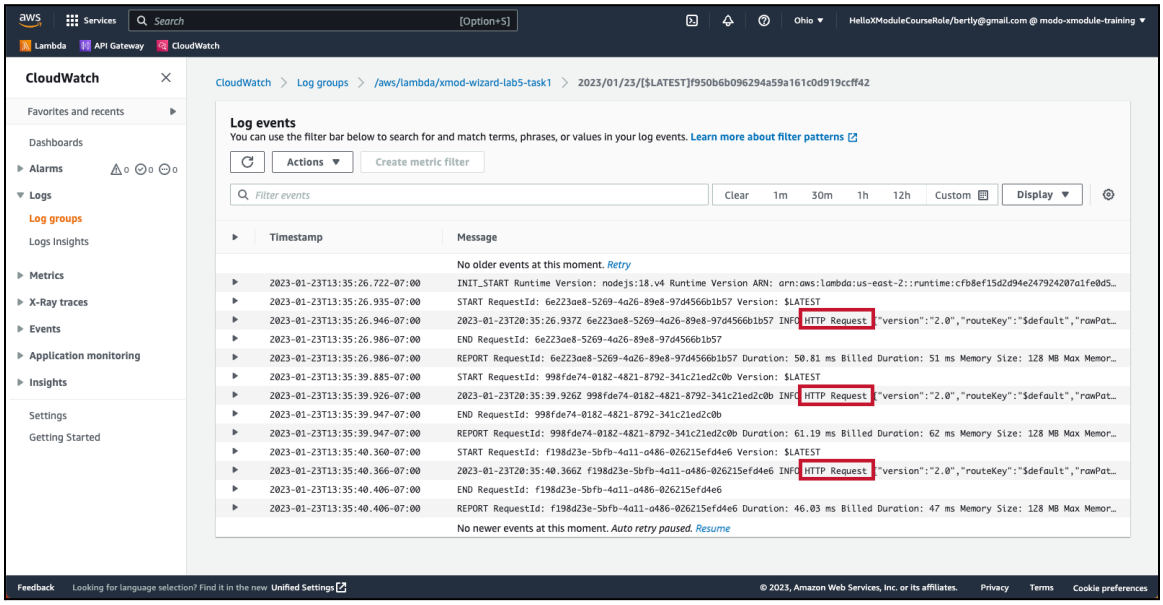
6. Change your **{xmod-prefix}-lab5-task1** function by doing the following:
    a. Set the xmJson variable to a base JSON object that has metadata and version only.
    b. Add the form data to a variable named xmForm. Add the form to the xmJson content array.
    c. **Deploy** your changes and test.
7. Create a condition to check for query string parameters in the HTTP Request and only return the xmForm JSON to the content array if there are no query string parameters.
8. If queryStringParameters exist, this means the form has been submitted, so we will create a cookie for the banner message we want to send then redirect to the same page with no queryParameters.
    a. Read the Redirect support page: https://support.modolabs.com/support/solutions/articles/13000088953-redirection
    b. Read the Cookies page: https://support.modolabs.com/support/solutions/articles/13000088954-cookies
9. Cookies uses the JWT so we need to enable that on Lab 5: Task 1 and we need to add support for cookies. Open Lab 5: Task 1 Module in your Admin console and do the following to the Module:
    a. Select Application Key for JWT Key
    b. Click to check Enable Sessions and Session Cookies
    c. Save your changes
    d. Deploy your changes to test.
    e. Preview the changes you made to the page.
10. View the CloudWatch logs to see if you are getting a JWT then put that JWT in JWT.io to see the values contained in your JWT. The image below shows that cookies are now added to the JWT payload.

11. In your **{xmod-prefix}-lab5-task1** Lambda, add cookies creation and redirection if the query string parameters exist and **Deploy** and Test your changes.

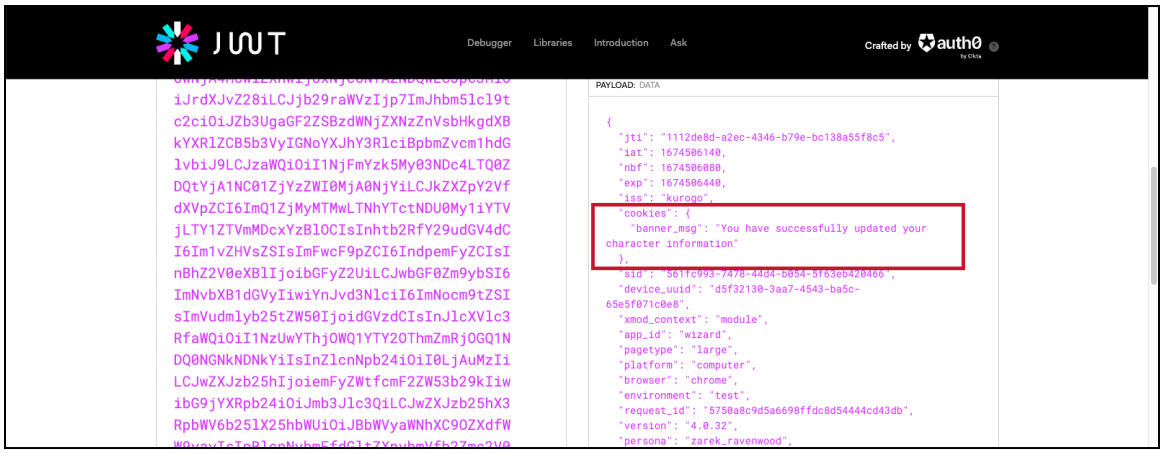


Note: The URL no longer has the query string parameters used to submit the form. This is exactly what we were hoping for.

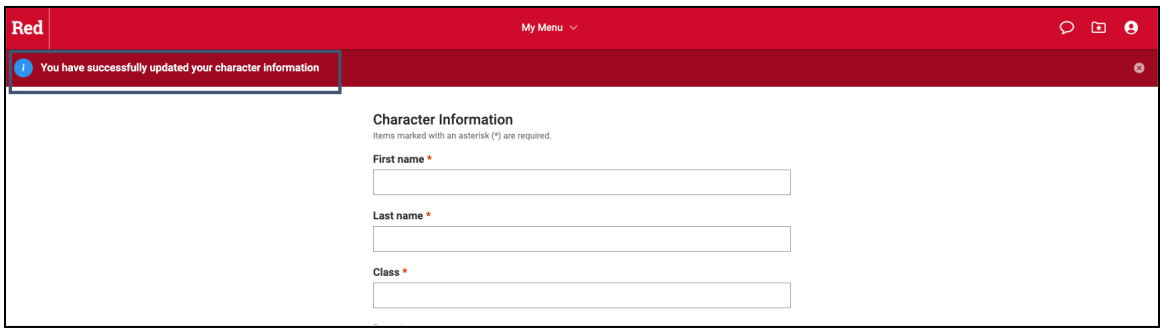
12. View the latest CloudWatch logs and verify that you have 3 different HTTP Requests



13. Expand the latest HTTP Request, copy and paste the JWT in JWT.io to check whether the cookie was set properly.



14. Add the `jsonwebtoken` and `jwk-to-pem` Lambda layers to your Lambda function and copy and paste the code that uses those layers to decode the payload from your **lab4-task3** Lambda function.
15. Check to see if there are cookies in the payload, if they exist, you should add a banner message then clear the banner message cookies so it does not appear the next time you load the page.
16. **Deploy** and Test your changes.



Congratulations! You have successfully implemented redirection in accordance with PRG patterns and you have added and viewed a banner message using cookies.

Task 3: Other screen-level properties

Context: There are a variety of additional screen-level properties that help you control your page. The following task helps you explore some of those options.

Goal: Explore the remaining screen-level properties and their outcomes

Task Outline:

- Implement and test the following screenlevel properties
    - contentStyle
    - contentBackgroundImage
    - contentBackgroundColor
    - bodyBackgroundImage
    - bodyBackgroundColor
    - footerTextColor
    - hideBackToTop
    - backToTopBackgroundColor
    - backToTopTextColor

Steps:

1. Copy and paste the code below into the XModule Sandbox.

```
{
    "metadata": {
        "version": "2.0"
    },
    "contentContainerWidth": "narrow",
    "contentStyle": "focal",
    "contentBackgroundImage": {
        "url":
"https://images.unsplash.com/photo-1625152785620-f1c191a941e8?ixid=MnwxMjA3fDB8MHxwaG90
by1wYWdlfHx8fGVufDB8fHx8&ixlib=rb-1.2.1&auto=format&fit=crop&w=1051&q=80",
        "alt": "Cruz Martinez, Unsplash",
        "overlayType": "solid"
    },
    "contentBackgroundColor": "lightgray",
    "content": [
        {
            "elementType": "divider",
            "borderColor": "transparent"
        },
        {
            "elementType": "container",
```

```
        "id": "textblock_margin",
        "content": [
            {
                "elementType": "list",
                "textblockMargin": "none",
                "heading": {
                    "heading": "textblockMargin: none",
                    "headingLevel": 3,
                    "headingTextColor": "grey",
                    "marginTop": "loose"
                },
                "items": [
                    {
                        "title": "List item A",
                        "link": {
                            "relativePath": ""
                        }
                    },
                    {
                        "title": "List item B",
                        "link": {
                            "relativePath": ""
                        }
                    },
                    {
                        "title": "List item C",
                        "link": {
                            "relativePath": ""
                        }
                    }
                ]
            },
            {
                "elementType": "list",
                "textblockMargin": "xxtight",
                "heading": {
                    "heading": "textblockMargin: xxtight",
                    "headingLevel": 3,
                    "headingTextColor": "grey",
                    "marginTop": "loose"
                },
                "items": [
                    {
                        "title": "List item A",
                        "link": {
                            "relativePath": ""
```

```
                    }
                },
                {
                    "title": "List item B",
                    "link": {
                        "relativePath": ""
                    }
                },
                {
                    "title": "List item C",
                    "link": {
                        "relativePath": ""
                    }
                }
            ]
        },
        {
            "elementType": "list",
            "textblockMargin": "xtight",
            "heading": {
                "heading": "textblockMargin: xtight",
                "headingLevel": 3,
                "headingTextColor": "grey",
                "marginTop": "loose"
            },
            "items": [
                {
                    "title": "List item A",
                    "link": {
                        "relativePath": ""
                    }
                },
                {
                    "title": "List item B",
                    "link": {
                        "relativePath": ""
                    }
                },
                {
                    "title": "List item C",
                    "link": {
                        "relativePath": ""
                    }
                }
            ]
        },
```

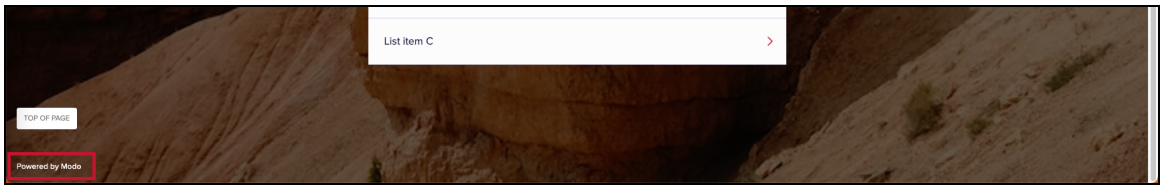
```
            {
                "elementType": "list",
                "textblockMargin": "tight",
                "heading": {
                    "heading": "textblockMargin: tight",
                    "headingLevel": 3,
                    "headingTextColor": "grey",
                    "marginTop": "loose"
                },
                "items": [
                    {
                        "title": "List item A",
                        "link": {
                            "relativePath": ""
                        }
                    },
                    {
                        "title": "List item B",
                        "link": {
                            "relativePath": ""
                        }
                    },
                    {
                        "title": "List item C",
                        "link": {
                            "relativePath": ""
                        }
                    }
                ]
            },
            {
                "elementType": "list",
                "textblockMargin": "medium",
                "heading": {
                    "heading": "textblockMargin: medium",
                    "headingLevel": 3,
                    "headingTextColor": "grey",
                    "marginTop": "loose"
                },
                "items": [
                    {
                        "title": "List item A",
                        "link": {
                            "relativePath": ""
                        }
                    },
```

```
                {
                    "title": "List item B",
                    "link": {
                        "relativePath": ""
                    }
                },
                {
                    "title": "List item C",
                    "link": {
                        "relativePath": ""
                    }
                }
            ]
        },
        {
            "elementType": "list",
            "textblockMargin": "loose",
            "heading": {
                "heading": "textblockMargin: loose",
                "headingLevel": 3,
                "headingTextColor": "grey",
                "marginTop": "loose"
            },
            "items": [
                {
                    "title": "List item A",
                    "link": {
                        "relativePath": ""
                    }
                },
                {
                    "title": "List item B",
                    "link": {
                        "relativePath": ""
                    }
                },
                {
                    "title": "List item C",
                    "link": {
                        "relativePath": ""
                    }
                }
            ]
        },
        {
            "elementType": "list",
```

```
                    "textblockMargin": "xloose",
                    "heading": {
                        "heading": "textblockMargin: xloose",
                        "headingLevel": 3,
                        "headingTextColor": "grey",
                        "marginTop": "loose"
                    },
                    "items": [
                        {
                            "title": "List item A",
                            "link": {
                                "relativePath": ""
                            }
                        },
                        {
                            "title": "List item B",
                            "link": {
                                "relativePath": ""
                            }
                        },
                        {
                            "title": "List item C",
                            "link": {
                                "relativePath": ""
                            }
                        }
                    ]
                }
            ]
        }
    ]
}
```

2. Notice how the backgroundImage overrides the backgroundColor. Cut the line with the backgroundImage and Preview your changes.
3. Change the **contentBackgroundColor** to **bodyBackgroundColor** and click Preview. Notice that the page is white except for the very bottom of the page that now has the `lightgray` background color.
4. Change the **contentStyle** to **nonfocal** and click **Preview**.
5. Paste the contentBackgroundImage code from the JSON above back into the Sandbox below the bodyBackgroundColor and change contentBackgroundImage to bodyBackgroundImage and click **Preview**.

   Note: The image is in the background for the entire page and with the contentStyle sent to nonfocal you can see the entirety of the image.
6. Add `"footerTextColor": "white",` below the bodyBackgroundImage property and Preview the changes.



7. Add `"backToTopBackgroundColor": "darkred"`, and `"backToTopTextColor": "white"`, as additional screen level properties and click **Preview**.

8. Add `"hideBackToTop": true,` to the screen level properties and click Preview and make sure the Back to Top button disappears.

Congratulations! You have successfully worked with the rest of the screen level properties. There are other examples of these screen properties at the following link: https://xmodule.modolabs.net/default/xmodule_ui_examples/page?relativePath=..%2Fscreen-properties%2Fscreen-contentStyle-focal.json

Task 4: Request Logger

Context: Debugging your XModule can be a troublesome task especially when you do not have access to server logs. Thankfully Modo provides Request Logger to intercept the communication between your Modo application and your web service. Request Logger can show you the outgoing HTTP request and the returning HTTP response as well as showing you the values included in your JWT. If it encounters errors those errors are included in the data returned from the web service. Request Logger is session specific so only the person initiating the request can see the request / response information.

Note: Using Request Logger creates a negative impact on your XModule performance so it should only be used in the Test environment and should be removed when no longer needed. If errors are only occurring in the production environment, you can enable it briefly to generate the appropriate logs but you should disable it as soon as you have valid log information.

Goal: Enable Request Logger and use it to test out the HTTP Request and Response for an XModule

Steps:

1. Follow the steps outlined in the following support article to configure Request Logging on your Lab 5: Task 1: https://support.modolabs.com/support/solutions/articles/13000098877-request-logger

Congratulations! You have followed our support article on setting up Request Logger. We are working on making our support articles super useful like this one. If you have any suggestions or feedback please let us know!

Task 5: Barcode Button

Context: Customers often want to use the QR code scanner to get a value from a barcode then send that value to a backend service for processing. The `barcodeButton` element allows you to configure this quickly.

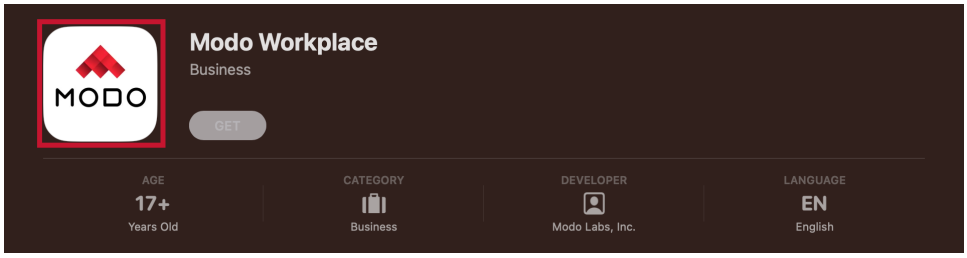
Goal: Use Modo Campus / Modo Workplace to test the `barcodeButton` element.

Task Outline:

- Download and install **Modo Campus** on your Mobile device and use your application identifier to load the application
- Create a **{xmod-prefix)-lab8-task5** Lambda function that creates a barcodeButton element
- Create a new resource in your API gateway and point it to your lab8-task5 Lambda you just created (make sure Lambda proxy integration is checked!!)
- Create a **Lab 8: Task 5** XModule and point it to your new API gateway resource
- Create a QR code using the following URL that has a quest ID as the QR code value: https://www.qr-code-generator.com/
- Handle the barcodeButton's event and send the data to the quest details page using redirect or rebuild the quest detail functionality in this Lambda function.
- **Optional**: Now that we have QR codes redirecting to the quest details, how do you handle incorrect quest IDs in the QR Codes?

Steps:

1. If you don't already have it downloaded, search for Modo Workplace on your mobile device's app store and install it on your mobile device.



2. Go to the following URL to view the information about the Barcode Button and copy the Basic example information to your clipboard: https://support.modolabs.com/support/solutions/articles/13000098179-barcode-button#barcode-button-0-0
3. Use the knowledge you have gained and the patterns we have demonstrated in this course to complete the task as outlined in the Task Outline!

Congratulations! You have completed all of the miscellaneous tasks and have been exposed to the breadth of XModule

Lab 9: Final Project!

Context: We want you to take everything you have learned throughout this training and create a player portal for our fantasy game. It should be a desktop first design that also transitions to a mobile experience. Your character's personalized dashboard should be view only and provide the following quick glimpse into your party:

- Show quests and be able to filter the results by text and by selecting a category.
    - Your current quests
    - Your available quests
    - Your unavailable quests because your party does not have enough power
- Show members of your party
- Show list of party status (skills, feats)
- Show party total gold
- Show characters who are not part of your party
- Show your character (thumbnail) and stats / info about your character

Additionally, using the same XModule link, you should be able to do the following by clicking different items from the dashboard:

- Ability to add up to 4 other characters to your party
- Edit your character's attributes - name and race but not gold or power and save these changes to the Modo Communicate Directory (Note: Logging out as the character will change those attributes to the default stored the Directory data)
- Provide quick navigation back to the dashboard at any time
- Optional
    - Add a random quest selector to choose an available quest for you
    - Add start and complete quest to improve power and add gold to your party (gold should be split evenly between party members)
    - Anything else you think to build an engaging experience

Resources:

-

Appendix

The appendix contains data that may help you if you get stuck in the lab guide. Often this includes all the code necessary to successfully complete the lab. Each group of code is labeled for the Lab and Task it applies to.

Lab 3: All Tasks

```
import jwkPem from 'jwk-to-pem';
import jwt from 'jsonwebtoken';
const key =
{"kid":"bc4078c2-4b69-4493-aa84-5c0d1341ec73","kty":"RSA","n":"1iD8ja4VntFVHtProcTgrdYX
ze0Hlsh8FSRgbKJGmhVUYASlkfIBYVqS335dysqcQTJ63zDYj7S0eJo_b5p3-QBp2dWmj8Qs8xs4FgQWKNYrCKu
1ax3VbLwTHgLwNuFhJIFzpVFCQ34PBhiz6Lt32mcFnG__ybAkGoWpRTHeLDG73DHeYezxVkAXFASHpLvI1zvSKw
DAye3r7MDkgOlWuq59dLXEm7fNHnVStc_-B8flqZ0pMLTWroOF5qdh5rNqK3mCOE82vs2LYiIbRki_iHAiMaHyO
SiiyI4Wpuv-_n8SJlNgiGGZo6DBxbowOk3gwzELnC_LLlcY2wI6Sfq2uQ","e":"AQAB"};

export const handler = async(event) => {
    console.log('HTTP Request', JSON.stringify(event));

    let xmList ={
        "elementType": "list",
        "listStyle": "grouped",
        "items": []
    };

    if ('headers' in event){
        for (const key in event.headers){
            xmList.items.push({
                "label": key,
                "title":event.headers[key]
            });
        }
    }

    let xmCardSet = {
        "elementType": "cardSet",
        "id": "sizes",
        "size" : "small",
        "marginTop": "medium",
        "items": []
    };

    let xmNametag = {
        "elementType": "nameTag",
```

```
        "id": "custom_style",
        "nameTagStyle": "horizontal",
        "name": "Ada Imani",
        "nameFontSize": "large",
        "nameFontWeight": "bold",
        "description": "B2B Manager",
        "imageHeight": "large",
        "imageWidth": "large",
        "imageBorderRadius": "medium",
        "image": {
            "url": "https://source.unsplash.com/i2hoD-C2RUA",
            "alt": "Photo of Ada Imani"
        },
        "accessoryButton": {
            "actionStyle": "emphasized",
            "title": " Contact ",
            "borderRadius": "full"
        }
    };

    let payload = null;

    if ('headers' in event && 'modojwt' in event.headers){
        payload = jwt.verify(event.headers.modojwt.split(" ")[1],jwkPem(key));
        console.log("Payload", payload);
    }

    if (payload != null){
         for (let key in payload){
             xmCardSet.items.push(        {
                "elementType": "contentCard",
                "title": key,
                "description": payload[key].toString()
            });
         }
         if ('name' in payload){
             xmNametag.name = payload.name;
             xmNametag.description = `<b>Class:</b> ${payload.class}<br><b>Race:</b>
${payload.race}<br><b>Gold:</b> ${payload.gold}<br><b>Power:</b>
${payload.power}<br><b>Skills:</b> ${payload.skills.join(" &#9752; ")}`;
                xmNametag.image = { "url": payload.photo, "alt": `Photo of
${payload.name}` };
         }
     }

    let xmJson =  {
```

```
        "metadata": {
            "version": "2.0"
        },
        "contentContainerWidth": "narrow",
        "content": [
            {
                "elementType": "divider",
                "borderColor": "transparent"
            },

            {
                "elementType": "tabs",
                "tabStyle": "folder",
                "tabs": [
                    {
                        "title": "HTTP Headers",
                        "content": [
                            xmList
                            ]
                    },
                    {
                        "title": "Payload Values",
                        'content': [
                            xmCardSet
                            ]
                    },
                    {
                        "title": "Character Nametag",
                        "content": [
                            xmNametag
                            ]
                    }
                ]
            }
        ]
    };

    const response = {
        statusCode: 200,
        body: JSON.stringify(xmJson),
    };
    return response;
};
```

Lab 4: Task 1

The following Javascript was used in the screenshots to prove functionality of these examples:

```
import data from './quests.mjs';

export const handler = async(event) => {
    console.log("HTTP Request", JSON.stringify(event));

    let xmJson = null;

    xmJson = {
    "metadata": {
        "version": "2.0"
    },
    "contentContainerWidth": "narrow",
    "content": [
    ]
};

    if ('pathParameters' in event && event.pathParameters == null) {
        let xmStatusList = {
                "elementType": "statusList",
                "id": "status_no_details",
                "heading": "Quest Statuses",
                "marginTop": "none",
                "listStyle": "padded",
                "items": [
                ]
            };

        if (data != null) {
            for (let i = 0; i < data.quests.length; i++) {
                xmStatusList.items.push({
                        "title": data.quests[i].name,
                        "statusText": data.quests[i].status.charAt(0).toUpperCase() +
data.quests[i].status.slice(1),
                        "status": data.quests[i].status == "available" ?
data.quests[i].status : "unavailable",
                        "statusDescriptor": `and pays ${data.quests[i].gold}`,
                        "link": {
                            "relativePath": `/details/${data.quests[i].id}`
                        }
                    });
            }
        }
```

```
        xmJson.content.push(xmStatusList);
    } else if ('pathParameters' in event && 'id' in event.pathParameters) {

        const quest = data.quests.find( ({ id }) => id === event.pathParameters.id );

        xmJson.content.push({
            "elementType": "detail",
            "id": "image_thumbnail",
            "title": quest.name,
            "description": `Difficulty: ${quest.difficulty}<br>Power Required:
${quest.recommended_power}<br>Duration: ${quest.duration}`,
            "byline": `Gold Rewarded: ${quest.gold} `,
            "body": quest.description,
            "thumbnail": {
                "url": quest.photo,
                "alt": quest.name
            },
            "thumbnailSize": "large",
            "thumbnailBorderRadius": "1rem"
        });
    }

    const response = {
        statusCode: 200,
        body: JSON.stringify(xmJson),
    };
    return response;
};
```

Lab 4: Task 2

This task was for showing how to load content via AJAX during a page load. This example includes a NodeJS function setTimeout(); to induce additional latency to the function.

```
import { setTimeout } from "timers/promises";

export const handler = async (event) => {
    console.log("HTTP Request", JSON.stringify(event));

    //Creates a grouped list with one item that has the current date as the title of
the lone item
    function getListWithDateTime() {
        return {
            "elementType": "list",
            "listStyle": "grouped",
```

```
            "id": "basic_list",
            "items": [{
                "title": new Date().toString()
            }]
        };
    }

    let xmJson = {
        "metadata": {
            "version": "2.0"
        }
    };

    //This is the AJAX request which is noted by having an ajax queryStringParameter
being set to true
    if ('queryStringParameters' in event && event.queryStringParameters != null &&
'ajax' in event.queryStringParameters && event.queryStringParameters.ajax == 'true') {
        await setTimeout(1000); //introduced fake latency of making an additional call
to get this data

        //ajax responses use regionContent instead of the traditional content array
        xmJson.regionContent = [

            getListWithDateTime()
        ];
    }
    else {

        //standard content added, creates a sideBySide element, sets the heading
element for both left and right sides then adds a list item to the left side and uses
ajax to load the right side
        xmJson.content = [{
            "elementType": "sideBySide",
            "left": {
                "preferredWidth": "50%",
                "content": [{
                        "elementType": "blockHeading",
                        "heading": "Loaded in intial call"
                    },
                    getListWithDateTime()
                ]
            },
            "right": {
                "preferredWidth": "50%",
                "content": [{
                        "elementType": "blockHeading",
```

```
                        "heading": "Loaded with AJAX"
                    },
                    {
                        "elementType": "container",
                        "content": {
                            "ajaxRelativePath": "?ajax=true"
                        }
                    }
                ]
            }
        }];
    }

    console.log(xmJson);
    const response = {
        statusCode: 200,
        body: JSON.stringify(xmJson),
    };
    return response;
};
```

Task 4: Lab 3

```
import jwkPem from 'jwk-to-pem';
import jwt from 'jsonwebtoken';
const key =
{"kid":"e17ae40d-d6c0-450f-9aac-2f565b3b1ca6","kty":"RSA","n":"3FOcqs0U3r7r8D_QnxiLgiCJ
2gV30uJFlJOC8OE9GGtlslKckoEHgji7iOrbtOEWGzHhZEOAG7E05iAHKDV-ZohAMruicSS3bd7xLw_8pCswlvd
0JpB4vjgGmwh1DqwanVlIYh6XD1Tllu5hfD6R89xHmz_akB6RBV-vDbErPqOxAs66LV8REDsc6FxdBOYcCHu9Qq
jsnDgUmoEuZVzeAozxBvGv5-krHoIHZHgMjvCK-ljS1egpBwS-GjbDcYMSEmsabuFhEdEvBLqP_cZG8Qo609Xy1
13YcjFFj0duRYbCBSW_Z_JOKpGXEagIzRd1hR258zsGzuwluaIpsGXXVQ","e":"AQAB"};

export const handler = async(event) => {
    console.log("HTTP Request", JSON.stringify(event));

    let xmJson =  {
    "metadata": {
        "version": "2.0"
    },
    "content": [
        {
            "elementType": "html",
            "html": "<h2>Geolocation Data</h2>"
        },
        {
            "elementType": "list",
            "listStyle": "grouped",
```

```
            "items": {
                "ajaxRelativePath":
"./?page=geolocationUpdate_ajaxUpdate&showLoading=true",
                "ajaxOnFirstLoad": false,
                "ajaxGeolocationEnabled": true,
                "ajaxGeolocationContinuous": false
            }
        }
    ]
};

    if ('queryStringParameters' in event && event.queryStringParameters != null &&
'page' in event.queryStringParameters && event.queryStringParameters.page ==
'geolocationUpdate_ajaxUpdate') {

        let payload = null; //create a variable to hold the decoded payload values

        //check that headers exist in event and that authorization is in event.headers
        if ('headers' in event && 'authorization' in event.headers) {
            payload = jwt.verify(event.headers.authorization.split(" ")[1],
jwkPem(key));
            console.log("Payload", payload);
        }

        let lat = '42.3940474', lon = '-71.1448423', statusCode = '0', time = new
Date().toString();

        if (payload != null && 'geolocation' in payload) {
            lat = payload.geolocation.lat != null ? payload.geolocation.lat.toString()
: lat;
            lon = payload.geolocation.lon != null ? payload.geolocation.lon.toString()
: lon;
            statusCode = payload.geolocation.status_code != null ?
payload.geolocation.status_code.toString() : statusCode;

        }

        xmJson.regionContent = [
            {

                "title": "<b>Latitude:</b> " + lat

            },

            {
```

```
                "title": "<b>Longitude:</b> " + lon

            },

            {

                "title": "<b>Status Code:</b> " + statusCode
            },

            {

                "title": "<b>Region Last Updated:</b> " + time

            }
        ];
    }

    const response = {
        statusCode: 200,
        body: JSON.stringify(xmJson),
    };
    return response;
};
```

Lab 5: Task 1 - POST Method

```
export const handler = async(event) => {
    console.log("HTTP Request", JSON.stringify(event));

    let xmJson = {
    "metadata": {
        "version": "2.0"
    },
    "contentContainerWidth": "narrow",
    "content": [
        {
            "elementType": "form",
            "id": "sample_form",
            "heading": {
                "heading": "Character Information",
                "headingLevel": 2,
                "description": "Items marked with an asterisk (*) are required."
            },
            "items": [
                {
```

```
            "elementType": "formInputText",
            "name": "s1_first",
            "label": "First name",
            "required": true
        },
        {
            "elementType": "formInputText",
            "name": "s1_last",
            "label": "Last name",
            "required": true
        },
        {
            "elementType": "formInputText",
            "name": "s1_class",
            "label": "Class",
            "required": true
        },
        {
            "elementType": "formInputText",
            "name": "s1_race",
            "label": "Race",
            "required": true
        },
        {
            "elementType": "formInputSegmented",
            "name": "s1_age",
            "label": "Age",
            "options": {
                "Under 25": "Under 25",
                "25-40": "25-40",
                "41-60": "41-60",
                "61+": "61 or over"
            },
            "fullWidth": true
        },
        {
            "elementType": "formInputAssistedSelect",
            "name": "s1_animal",
            "label": "Animal companion",
            "options": {
                "": "",
                "alpaca": "Alpaca",
                "ant": "Ant",
                "bird": "Bird",
                "cat": "Cat",
                "dog": "Dog",
```

```
                "dragon": "Dragon",
                "emu": "Emu",
                "ferret": "Ferret",
                "fish": "Fish",
                "goat": "Goat",
                "guinea": "Guinea pig",
                "hedgehog": "Hedgehog",
                "horse": "Horse",
                "kangaroo": "Kangaroo",
                "lizard": "Lizard",
                "mouse": "Mouse",
                "pig": "Pig",
                "rabbit": "Rabbit",
                "rat": "Rat",
                "salamander": "Salamander",
                "seal": "Seal",
                "shark": "Shark",
                "sheep": "Sheep",
                "snake": "Snake",
                "spider": "Spider",
                "turtle": "Turtle",
                "unicorn": "Unicorn",
                "weasel": "Weasel"
            }
        },
        {
            "elementType": "formInputRadio",
            "label": "Preferred contact",
            "preamble": "What's the best way to reach you?",
            "name": "s1_contact",
            "options": {
                "email": "Email",
                "phone": "Phone",
                "text": "Text",
                "discord": "Discord",
                "other": "Other"
            },
            "nested": true,
            "progressiveDisclosureItems": {
                "email": [
                    {
                        "elementType": "formInputEmail",
                        "name": "s1_email",
                        "label": "Email address",
                        "required": true
                    }
```

```
                ],
                "phone": [
                    {
                        "elementType": "formInputPhone",
                        "name": "s1_phone",
                        "label": "Phone number"
                    }
                ],
                "text": [
                    {
                        "elementType": "formInputPhone",
                        "name": "s1_text",
                        "label": "Phone number for text"
                    }
                ],
                "discord": [
                    {
                        "elementType": "formInputText",
                        "name": "s1_discord",
                        "label": "Discord username"
                    }
                ],
                "other": [
                    {
                        "elementType": "formInputTextarea",
                        "name": "s1_contact_other",
                        "label": "Describe the best way to contact you",
                        "rows": 4
                    }
                ]
            }
        }
    ],
    "buttons": [
        {
            "elementType": "formButton",
            "name": "s1_reset",
            "title": "Reset",
            "buttonType": "reset",
            "actionStyle": "destructiveQuiet",
            "minWidth": "8rem"
        },
        {
            "elementType": "formButton",
            "name": "s1_submit",
            "title": "Submit",
```

```
                    "buttonType": "submit",
                    "actionStyle": "constructive",
                    "minWidth": "8rem"
                }
            ],
            "trackDirtyStateButtonNames": [
                "s1_submit"
            ],
            "buttonsHorizontalAlignment": "center"
        }
    ]
};

    const response = {
        statusCode: 200,
        body: JSON.stringify(xmJson),
    };
    return response;
};
```

Lab 5: Task 1 - GET Method with banner

```
export const handler = async(event) => {
    console.log("HTTP Request", JSON.stringify(event));

    let xmJson = {
    "metadata": {
        "version": "2.0"
    },
    "contentContainerWidth": "narrow",
    "content": [
        {
            "elementType": "form",
            "id": "sample_form",
            "requestMethod": "GET",
            "heading": {
                "heading": "Character Information",
                "headingLevel": 2,
                "description": "Items marked with an asterisk (*) are required."
            },
            "items": [
                {
                    "elementType": "formInputText",
```

```
            "name": "s1_first",
            "label": "First name",
            "required": true
        },
        {
            "elementType": "formInputText",
            "name": "s1_last",
            "label": "Last name",
            "required": true
        },
        {
            "elementType": "formInputText",
            "name": "s1_class",
            "label": "Class",
            "required": true
        },
        {
            "elementType": "formInputText",
            "name": "s1_race",
            "label": "Race",
            "required": true
        },
        {
            "elementType": "formInputSegmented",
            "name": "s1_age",
            "label": "Age",
            "options": {
                "Under 25": "Under 25",
                "25-40": "25-40",
                "41-60": "41-60",
                "61+": "61 or over"
            },
            "fullWidth": true
        },
        {
            "elementType": "formInputAssistedSelect",
            "name": "s1_animal",
            "label": "Animal companion",
            "options": {
                "": "",
                "alpaca": "Alpaca",
                "ant": "Ant",
                "bird": "Bird",
                "cat": "Cat",
                "dog": "Dog",
                "dragon": "Dragon",
```

```
                    "emu": "Emu",
                    "ferret": "Ferret",
                    "fish": "Fish",
                    "goat": "Goat",
                    "guinea": "Guinea pig",
                    "hedgehog": "Hedgehog",
                    "horse": "Horse",
                    "kangaroo": "Kangaroo",
                    "lizard": "Lizard",
                    "mouse": "Mouse",
                    "pig": "Pig",
                    "rabbit": "Rabbit",
                    "rat": "Rat",
                    "salamander": "Salamander",
                    "seal": "Seal",
                    "shark": "Shark",
                    "sheep": "Sheep",
                    "snake": "Snake",
                    "spider": "Spider",
                    "turtle": "Turtle",
                    "unicorn": "Unicorn",
                    "weasel": "Weasel"
                }
            },
            {
                "elementType": "formInputRadio",
                "label": "Preferred contact",
                "preamble": "What's the best way to reach you?",
                "name": "s1_contact",
                "options": {
                    "email": "Email",
                    "phone": "Phone",
                    "text": "Text",
                    "discord": "Discord",
                    "other": "Other"
                },
                "nested": true,
                "progressiveDisclosureItems": {
                    "email": [
                        {
                            "elementType": "formInputEmail",
                            "name": "s1_email",
                            "label": "Email address",
                            "required": true
                        }
                    ],
```

```
                    "phone": [
                        {
                            "elementType": "formInputPhone",
                            "name": "s1_phone",
                            "label": "Phone number"
                        }
                    ],
                    "text": [
                        {
                            "elementType": "formInputPhone",
                            "name": "s1_text",
                            "label": "Phone number for text"
                        }
                    ],
                    "discord": [
                        {
                            "elementType": "formInputText",
                            "name": "s1_discord",
                            "label": "Discord username"
                        }
                    ],
                    "other": [
                        {
                            "elementType": "formInputTextarea",
                            "name": "s1_contact_other",
                            "label": "Describe the best way to contact you",
                            "rows": 4
                        }
                    ]
                }
            }
        ],
        "buttons": [
            {
                "elementType": "formButton",
                "name": "s1_reset",
                "title": "Reset",
                "buttonType": "reset",
                "actionStyle": "destructiveQuiet",
                "minWidth": "8rem"
            },
            {
                "elementType": "formButton",
                "name": "s1_submit",
                "title": "Submit",
                "buttonType": "submit",
```

```
                    "actionStyle": "constructive",
                    "minWidth": "8rem"
                }
            ],
            "trackDirtyStateButtonNames": [
                "s1_submit"
            ],
            "buttonsHorizontalAlignment": "center"
        }
    ]
};

if (event.requestContext.http.method.toLowerCase() == 'post' ||
('queryStringParameters' in event && event.queryStringParameters != null && 's1_submit'
in event.queryStringParameters)) {
    xmJson.metadata.banners = [
        {
            "message": "You have successfully submitted your character information",
            "type": "info"
        }
    ];
}

    const response = {
        statusCode: 200,
        body: JSON.stringify(xmJson),
    };
    return response;
};
```

Lab 5: Task 2

```
import data from './locations.mjs'

export const handler = async (event) => {
    console.log("HTTP Request", JSON.stringify(event));

    console.log("Locations Data", JSON.stringify(data));

    function buildFloors(floors) {
        let floorRadio = {
            "elementType": "formInputRadio",
            "name": "floor",
```

```
            "label": "Floor",
            "options": {}
        };

        for (let i = 0; i < floors.length; i++) {
            floorRadio.options[floors[i]] = floors[i];
        }

        return [floorRadio]; //contents must be in an array
    }

    function buildBuildings(buildings) {

        let buildingsSelect = {
            "elementType": "formInputSelect",
            "name": "building",
            "label": "Building",
            "nested": true,
            "required": true,
            "options": { },
            "progressiveDisclosureItems": {
                "main_street": [{
                    "elementType": "formInputRadio",
                    "name": "floor",
                    "label": "Floor",
                    "options": { }
                }]
            }
        };

        for (let i = 0; i < buildings.length; i++) {
            let key = buildings[i].id, val = buildings[i].name;
            buildingsSelect.options[buildings[i].id] = buildings[i].name;
            buildingsSelect.progressiveDisclosureItems[buildings[i].id] =
buildFloors(buildings[i].floors);
        }
        console.log("Building info", JSON.stringify(buildingsSelect));
        return [buildingsSelect];
    }

    function buildPgItems(region) {

        let citySelect = {
            "elementType": "formInputSelect",
            "name": "city",
            "label": "City",
```

```
            "nested": true,
            "required": true,
            "options": { },
            "progressiveDisclosureItems": { }
        };

        let locations = data[region].locations;

        for(let i = 0; i < locations.length; i++) { // loc in data[region].locations) {
            citySelect.options[locations[i].id] = locations[i].name;
            citySelect.progressiveDisclosureItems[locations[i].id] =
buildBuildings(locations[i].buildings);
        }

        return [citySelect];
    }

    let mainSelect = {
                "elementType": "formInputSelect",
                "name": "select",
                "nested": true,
                "label": "Select region",
                "options": {},
                "progressiveDisclosureItems": {}
    };

    //build main list and set disclosure items
    for (var key in data) {
        mainSelect.options[key] = key;
        mainSelect.progressiveDisclosureItems[key] = { "ajaxRelativePath": "?region=" +
key  };
    }

    let xmJson = {
        "metadata": {
            "version": "2.0"
        },
        "contentContainerWidth": "narrow",
        "content": [{
            "elementType": "form",
            "heading": "Quest Locations",
            "id": "select",
            "items": [
                mainSelect
            ]
        }]
```

```
    };

    //if we are looking for the region, we can add regionContent to our JSON and build
out the progressive disclosure items.
    if ('queryStringParameters' in event && 'region' in event.queryStringParameters) {
        xmJson.regionContent = await buildPgItems(event.queryStringParameters.region);
    }

    const response = {
        statusCode: 200,
        body: JSON.stringify(xmJson),
    };
    return response;
};
```

Lab 5: Task 3

```
import data from "./timezones.mjs";

export const handler = async(event) => {
    console.log("HTTP Request", JSON.stringify(event));

    let xmJson = {
    "metadata": {
        "version": "2.0"
    },
    "contentContainerWidth": "narrow",
    "content": [
        {
            "elementType": "divider",
            "borderColor": "transparent"
        },
        {
            "elementType": "form",
            "id": "assisted_select",
            "items": [
                {
                    "elementType": "formInputAssistedSelect",
                    "name": "assisted_select",
                    "label": "Assisted select",
                    "value": "",
                    "useAjaxSearching": true,
                    "ajaxSearchingRelativePath": "",
                    "searchFilterParameter": "search",
```

```
                    "minimumInputLength": 2
                },
                {
                    "elementType": "formInputAssistedMultiselect",
                    "name": "assisted_multiselect",
                    "label": "Assisted multi-select",
                    "description": "Select one or more",
                    "useAjaxSearching": true,
                    "ajaxSearchingRelativePath": "",
                    "searchFilterParameter": "search",
                    "minimumInputLength": 2
                }
            ]
        }
    ]
};

    if ('queryStringParameters' in event && 'search' in event.queryStringParameters) {
        let tzList = {}, searchString =
event.queryStringParameters.search.toLowerCase();
        for (let [key, value] of Object.entries(data.items)) {
            if (key.toLowerCase().includes(searchString) ||
value.toLowerCase().includes(searchString)) {
                tzList[key] = value;
            }
        }

        xmJson.elementFields = {
            "options": tzList
        };
    }

    const response = {
        statusCode: 200,
        body: JSON.stringify(xmJson),
    };
    return response;
};
```

Lab 6: Task 1

```
import data from './directory.mjs';

export const handler = async(event) => {
    console.log("HTTP Request", JSON.stringify(event));
```

```
let xmJson = {
    "metadata": {
        "version": "2"
    },
    "content": [
        {
            "elementType": "blockHeading",
            "heading": "Directory"
        },
        {
            "elementType": "form",
            "relativePath": "",
            "items": [
                {
                    "elementType": "formInputText",
                    "label": "Search Employees",
                    "name": "search_info",
                    "placeholder": "Search for name, occupation, or phone"
                }
            ],
            "events": [
                {
                  "eventName": "change",
                  "action": "ajaxUpdate",
                  "targetId": "employee_list",
                  "ajaxRelativePath": "",
                  "propagateArgs": true
                },
                {
                  "eventName": "submit",
                  "action": "ajaxUpdate",
                  "targetId": "employee_list",
                  "ajaxRelativePath": "",
                  "propagateArgs": true
                }
            ]
        },
        {
            "elementType": "list",
            "heading": "Type to show list",
            "id": "employee_list"
        }
    ]
};
```

```
    if ('queryStringParameters' in event && event.queryStringParameters != null &&
'search_info' in event.queryStringParameters) {
                let searchString =
event.queryStringParameters.search_info.toLowerCase(); //lowercase everything so we can
compare and find
        let directoryList = [];

        for(let d of data) {
            let fullName = `${d.firstName.toLowerCase()} ${d.lastName.toLowerCase()}`;
            if (fullName.includes(searchString) ||
d.occupation.toLowerCase().includes(searchString) || d.phone.includes(searchString)) {
                directoryList.push({
                   "title": `${d.firstName} ${d.lastName}`,
                   "description": `${d.occupation} &#9752; ${d.email} &#9752;
${d.phone}`
                });
            }
        }

        xmJson.elementFields = {
            "items": directoryList,
            "heading": `${directoryList.length} people found`
        };

    }

    const response = {
        statusCode: 200,
        body: JSON.stringify(xmJson),
    };
    return response;
};
```

Lab 6: Task 2

```
import data from "./quests.mjs";

export const handler = async(event) => {
    console.log("HTTP Request", JSON.stringify(event));

    let xmJson = {
    "metadata": {
        "version": "2.0"
    },
```

```
    "contentContainerWidth": "wide",
    "content": [],
    "elementFields": {}
};

    let xmFilter = {
            "elementType": "form",
            "id": "select",
            "items": [
                {
                    "elementType": "formInputSelect",
                    "name": "filter",
                    "label": "Filter results",
                    "options": [
                        {
                            "label": "Show all",
                            "value": "none"
                        },
                        {
                            "label": "Difficulty",
                            "value": [
                                {
                                    "value": "easy",
                                    "label": "easy"
                                },
                                {
                                    "value": "medium",
                                    "label": "medium"
                                },
                                {
                                    "value": "hard",
                                    "label": "hard"
                                }
                            ]
                        },
                        {
                            "label": "Status",
                            "value": [
                                {
                                    "value": "available",
                                    "label": "available"
                                },
                                {
                                    "value": "busy",
                                    "label": "busy"
                                }
```

```
                        ]
                    }
                ],
                "events": [
                    {
                        "eventName": "change",
                        "action": "ajaxUpdate",
                        "targetId": "quest_cards",
                        "ajaxRelativePath": "",
                        "propagateArgs": true
                    }
                ]
            }
        ]
    };

    let xmCardSet = {
            "elementType": "cardSet",
            "heading": "Quests",
            "id": "quest_cards",
            "marginTop": "medium",
            "items": [

            ]
        };

    function getCards(filter) {
        let xmCards = [];

        for (let q of data.quests) {
            if (filter == null || filter == 'none' || filter.includes(q.status) ||
filter.includes(q.difficulty)) {
                xmCards.push({
                    "elementType": "contentCard",
                    "size": "small",
                    "imageStyle": "fullbleedGradient",
                    "image": {
                        "url": q.photo,
                        "alt": q.name
                    },
                    "title": q.name,
                    "label": q.status,
                    "labelTextColor": q.status == 'available' ?
"theme:available_focal_text_color" : "theme:alarm_focal_text_color",
                    "description": `Difficulty: ${q.difficulty}<br>Gold earned:
${q.gold.toString()}`
```

```
                });
            }
        }

        return xmCards;
    }

    if ('queryStringParameters' in event && event.queryStringParameters!= null &&
'filter' in event.queryStringParameters) {
        xmJson.elementFields = {
            "items": getCards(event.queryStringParameters.filter)
        };

    } else {
        xmCardSet.items = getCards();

        xmJson.content.push(xmFilter); //add the form as the first thing
        xmJson.content.push(xmCardSet); //add the cardset as the second thing
    }

    const response = {
        statusCode: 200,
        body: JSON.stringify(xmJson),
    };
    return response;
};
```

Lab 6: Task 3

```
export const handler = async(event) => {
    console.log("HTTP Request", JSON.stringify(event));

    let xmJson = {
        "metadata": {
            "version": "2.0"
        }
    };

    if ('body' in event && event.body != null) {
        xmJson.elementFields = {
            image: {
                "icon": "confirm",
                "iconColor": "theme:confirmation_focal_text_color",
                "alt": "Complete"
```

```
            },
            collapsed: true
        };

    } else {
        let content = [
            {
                "elementType": "divider",
                "marginTop": "none",
                "marginBottom": "none"
            },
            {
                "elementType": "collapsible",
                "title": "Vivamus mattis consectetur",
                "id": "task1",
                "titleFontSize": "1.1875rem",
                "image": {
                    "icon": "confirm",
                    "iconColor": "theme:confirmation_focal_text_color",
                    "alt": "Complete"
                },
                "imageWidth": "2rem",
                "imageHeight": "2rem",
                "collapsed": true,
                "borderTopStyle": "none",
                "borderBottomStyle": "none",
                "headingPaddingTop": "tight",
                "headingPaddingBottom": "tight",
                "marginTop": "none",
                "marginBottom": "none",
                "content": [
                    {
                        "elementType": "container",
                        "paddingLeft": "2.75rem",
                        "paddingRight": "2rem",
                        "paddingBottom": "xtight",
                        "content": [
                            {
                                "elementType": "html",
                                "html": "Praesent commodo cursus magna, vel scelerisque
nisl consectetur et"
                            }
                        ]
                    }
                ]
            },
```

```
            {
                "elementType": "divider",
                "marginTop": "none",
                "marginBottom": "none"
            },
            {
                "elementType": "collapsible",
                "id": "task2",
                "title": "Vestibulum id ligula porta felis euismod semper",
                "titleFontSize": "1.1875rem",
                "image": {
                    "icon": "next",
                    "iconColor": "theme:tertiary_text_color",
                    "alt": "Incomplete"
                },
                "imageWidth": "2rem",
                "imageHeight": "2rem",
                "collapsed": false,
                "borderTopStyle": "none",
                "borderBottomStyle": "none",
                "headingPaddingTop": "tight",
                "headingPaddingBottom": "tight",
                "marginTop": "none",
                "marginBottom": "none",
                "content": [
                    {
                        "elementType": "container",
                        "paddingLeft": "2.75rem",
                        "paddingRight": "2rem",
                        "paddingBottom": "xtight",
                        "content": [
                            {
                                "elementType": "html",
                                "html": "Nullam id dolor id nibh ultricies vehicula ut
id elit. Donec ullamcorper nulla non metus auctor fringilla. Cras justo odio, dapibus
ac facilisis in, egestas eget quam."
                            },
                            {
                                "elementType": "buttonContainer",
                                "marginTop": "xtight",
                                "horizontalAlignment": "right",
                                "buttons": [
                                    {
                                        "elementType": "linkButton",
                                        "size": "small",
                                        "borderRadius": "full",
```

```
                                    "actionType": "emphasized",
                                    "title": "Complete the task",
                                    "icon": "next",
                                    "iconPosition": "right",
                                    "events": [
                                        {
                                            "eventName": "click",
                                            "action": "ajaxUpdate",
                                            "targetId": "task2",
                                            "ajaxRelativePath": "",
                                            "postData": {
                                                "complete": true
                                            }
                                        },
                                        {
                                            "eventName": "click",
                                            "action": "close",
                                            "targetId": "task2"
                                        },
                                        {
                                            "eventName": "click",
                                            "action": "open",
                                            "targetId": "task4"
                                        }
                                    ]
                                }
                            ]
                        }
                    ]
                }
            ]
        },
        {
            "elementType": "divider",
            "marginTop": "none",
            "marginBottom": "none"
        },
        {
            "elementType": "collapsible",
            "id": "task3",
            "title": "Cras justo dapibus ac facilisis in egestas eget quam -
valete! 🎉",
            "titleFontSize": "1.1875rem",
            "image": {
                "icon": "confirm",
                "iconColor": "theme:confirmation_focal_text_color",
```

```
                    "alt": "Complete"
                },
                "imageWidth": "2rem",
                "imageHeight": "2rem",
                "collapsed": true,
                "borderTopStyle": "none",
                "borderBottomStyle": "none",
                "headingPaddingTop": "tight",
                "headingPaddingBottom": "tight",
                "marginTop": "none",
                "marginBottom": "none",
                "content": [
                    {
                        "elementType": "container",
                        "paddingLeft": "2.75rem",
                        "paddingRight": "2rem",
                        "paddingBottom": "xtight",
                        "content": [
                            {
                                "elementType": "html",
                                "html": "Nullam id dolor id nibh ultricies vehicula ut
id elit."
                            }
                        ]
                    }
                ]
            },
            {
                "elementType": "divider",
                "marginTop": "none",
                "marginBottom": "none"
            },
            {
                "elementType": "collapsible",
                "id": "task4",
                "title": "Parturient montes, nascetur ridiculus mus",
                "titleFontSize": "1.1875rem",
                "image": {
                    "icon": "next",
                    "iconColor": "theme:tertiary_text_color",
                    "alt": "Incomplete"
                },
                "imageWidth": "2rem",
                "imageHeight": "2rem",
                "collapsed": true,
                "borderTopStyle": "none",
```

```
                "borderBottomStyle": "none",
                "headingPaddingTop": "tight",
                "headingPaddingBottom": "tight",
                "marginTop": "none",
                "marginBottom": "none",
                "content": [
                    {
                        "elementType": "container",
                        "paddingLeft": "2.75rem",
                        "paddingRight": "2rem",
                        "paddingBottom": "xtight",
                        "content": [
                            {
                                "elementType": "html",
                                "html": "<p>Nulla vitae elit libero, a pharetra augue.
<a href='https://example.com'>Maecenas sed diam</a> eget risus varius blandit sit amet
non magna. Parturient montes, nascetur ridiculus mus. Lorem ipsum dolor sit amet,
consectetur adipiscing elit.</p><ul><li>Integer posuere erat a ante venenatis
dapibus</li><li>Posuere velit aliquet</li><li>Donec sed odio dui</li><li>Fusce dapibus,
tellus ac cursus commodo</li></ul><p>Donec sed odio dui. Vivamus sagittis lacus vel
augue laoreet rutrum faucibus dolor auctor. Donec ullamcorper nulla non metus.</p>"
                            },
                            {
                                "elementType": "buttonContainer",
                                "marginTop": "xtight",
                                "horizontalAlignment": "right",
                                "buttons": [
                                    {
                                        "elementType": "linkButton",
                                        "size": "small",
                                        "borderRadius": "full",
                                        "actionType": "emphasized",
                                        "title": "Do the thing",
                                        "icon": "next",
                                        "iconPosition": "right",
                                        "events": [
                                            {
                                                "eventName": "click",
                                                "action": "ajaxUpdate",
                                                "targetId": "task4",
                                                "ajaxRelativePath": "",
                                                "postData": {
                                                    "complete": true
                                                }
                                            },
                                            {
```

```
                                        "eventName": "click",
                                        "action": "close",
                                        "targetId": "task4"
                                    },
                                    {
                                        "eventName": "click",
                                        "action": "open",
                                        "targetId": "task5"
                                    }
                                ]
                            }
                        ]
                    }
                ]
            }
        ]
    },
    {
        "elementType": "divider",
        "marginTop": "none",
        "marginBottom": "none"
    },
    {
        "elementType": "collapsible",
        "id": "task5",
        "title": "Duis mollis, est non commodo luctus, nisi erat porttitor
ligula",
        "titleFontSize": "1.1875rem",
        "image": {
            "icon": "next",
            "iconColor": "theme:tertiary_text_color",
            "alt": "Incomplete"
        },
        "imageWidth": "2rem",
        "imageHeight": "2rem",
        "collapsed": true,
        "borderTopStyle": "none",
        "borderBottomStyle": "none",
        "headingPaddingTop": "tight",
        "headingPaddingBottom": "tight",
        "marginTop": "none",
        "marginBottom": "none",
        "content": [
            {
                "elementType": "container",
                "paddingLeft": "2.75rem",
```

```
                        "paddingRight": "2rem",
                        "paddingBottom": "xtight",
                        "content": [
                            {
                                "elementType": "html",
                                "html": "Nulla vitae elit libero, a pharetra augue.
Donec sed odio dui. Vestibulum id ligula porta felis euismod semper. Etiam porta sem
malesuada magna mollis euismod. Fusce dapibus, tellus ac cursus commodo, tortor mauris
condimentum nibh, ut fermentum massa justo sit amet risus."
                            },
                            {
                                "elementType": "buttonContainer",
                                "marginTop": "xtight",
                                "horizontalAlignment": "right",
                                "buttons": [
                                    {
                                        "elementType": "linkButton",
                                        "size": "small",
                                        "borderRadius": "full",
                                        "actionType": "emphasized",
                                        "title": "Ipsa scientia potestas est",
                                        "icon": "next",
                                        "iconPosition": "right",
                                        "events": [
                                            {
                                                "eventName": "click",
                                                "action": "ajaxUpdate",
                                                "targetId": "task5",
                                                "ajaxRelativePath": "",
                                                "postData": {
                                                    "complete": true
                                                }
                                            },
                                            {
                                                "eventName": "click",
                                                "action": "close",
                                                "targetId": "task5"
                                            },
                                            {
                                                "eventName": "click",
                                                "action": "open",
                                                "targetId": "task6"
                                            }
                                        ]
                                    }
                                ]
```

```
                        }
                    ]
                }
            ]
        },
        {
            "elementType": "divider",
            "marginTop": "none",
            "marginBottom": "none"
        },
        {
            "elementType": "collapsible",
            "id": "task6",
            "title": "Maecenas faucibus mollis interdum",
            "titleFontSize": "1.1875rem",
            "image": {
                "icon": "next",
                "iconColor": "theme:tertiary_text_color",
                "alt": "Incomplete"
            },
            "imageWidth": "2rem",
            "imageHeight": "2rem",
            "collapsed": true,
            "borderTopStyle": "none",
            "borderBottomStyle": "none",
            "headingPaddingTop": "tight",
            "headingPaddingBottom": "tight",
            "marginTop": "none",
            "marginBottom": "none",
            "content": [
                {
                    "elementType": "container",
                    "paddingLeft": "2.75rem",
                    "paddingRight": "2rem",
                    "paddingBottom": "xtight",
                    "content": [
                        {
                            "elementType": "html",
                            "html": "<p>Curabitur blandit tempus porttitor. Donec
sed odio dui. Nullam quis risus eget urna mollis ornare vel eu leo. Lorem ipsum dolor
sit amet, consectetur adipiscing elit.</p><p>Nulla vitae elit libero, a pharetra augue.
Maecenas sed diam eget risus varius blandit sit amet non magna.</p>"
                        },
                        {
                            "elementType": "buttonContainer",
                            "marginTop": "xtight",
```

```
                            "horizontalAlignment": "right",
                            "buttons": [
                                {
                                    "elementType": "linkButton",
                                    "size": "small",
                                    "borderRadius": "full",
                                    "actionType": "emphasized",
                                    "title": "Go do the thing",
                                    "icon": "next",
                                    "iconPosition": "right",
                                    "events": [
                                        {
                                            "eventName": "click",
                                            "action": "ajaxUpdate",
                                            "targetId": "task6",
                                            "ajaxRelativePath": "",
                                            "postData": {
                                                "complete": true
                                            }
                                        },
                                        {
                                            "eventName": "click",
                                            "action": "close",
                                            "targetId": "task6"
                                        }
                                    ]
                                }
                            ]
                        }
                    ]
                }
            ]
        },
        {
            "elementType": "divider",
            "marginTop": "none",
            "marginBottom": "none"
        }
    ];

    xmJson.content = content;
}

const response = {
    statusCode: 200,
    body: JSON.stringify(xmJson),
```

```
    };
    return response;
};
```

Lab 7: Task 1

```
{
    "metadata": {
        "version": "2.0"
    },
    "header": [
        {
            "elementType": "hero",
            "height": "fluid",
            "contentContainerWidth": "medium",
            "backgroundImage": {
                "url":
"https://images.unsplash.com/photo-1591111126332-77daa792aad4?ixid=MnwxMjA3fDB8MHxwaG90
by1wYWdlfHx8fGVufDB8fHx8&ixlib=rb-1.2.1&auto=format&fit=crop&w=3320&q=80",
                "overlayType": "gradient",
                "overlayGradientStartColor": "rgba(0,0,0,0.9)",
                "overlayGradientStartPosition": 33,
                "overlayGradientAngle": 0
            },
            "content": [
                {
                    "elementType": "heroImage",
                    "imageSize": "58px",
                    "horizontalAlignment": "center",
                    "marginTop": "3rem",
                    "image": {
                        "url": "https://static.modolabs.com/james/images/headshot.png",
                        "alt": "Headshot of a man",
                        "borderWidth": "4px",
                        "borderColor": "#ffffff",
                        "borderRadius": "full"
                    }
                },
                {
                    "elementType": "heroHeading",
                    "textAlignment": "center",
                    "textColor": "#fff",
                    "fontSize": "2.5rem",
```

```
                    "marginTop": ".5rem",
                    "marginBottom": "none",
                    "responsiveScaling": true,
                    "heading": "Welcome, John Smith"
                },
                {
                    "elementType": "heroBody",
                    "fontSize": "xsmall",
                    "textAlignment": "center",
                    "textColor": "#fff",
                    "marginTop": "none",
                    "marginRight": "1rem",
                    "marginBottom": "2rem",
                    "marginLeft": "1rem",
                    "responsiveScaling": true,
                    "body": "You're scheduled to be in office at Boston, Floor 17
today"
                },
                {
                    "elementType": "heroButtons",
                    "horizontalAlignment": "center",
                    "marginBottom": "3rem",
                    "buttons": [
                        {
                            "elementType": "linkButton",
                            "backgroundColor": "rgba(0,0,0,0.25)",
                            "borderColor": "#00baff",
                            "borderWidth": "2px",
                            "title": "Cancel Reservation",
                            "textColor": "#00baff"
                        },
                        {
                            "elementType": "linkButton",
                            "backgroundColor": "#00baff",
                            "borderColor": "#00baff",
                            "borderWidth": "2px",
                            "title": "   Check in   ",
                            "textColor": "#000000"
                        }
                    ]
                }
            ]
        }
    ],
    "contentContainerWidth": "full",
    "content": [
```

```
        {
            "elementType": "responsiveThreeColumn",
            "primaryColumn": {
                "content": [
                    {
                        "elementType": "html",
                        "html": "<h2>Primary Column</h2><p>Lorem ipsum dolor sit amet,
consectetur adipiscing elit. Nunc sodales luctus arcu eget aliquam. Nulla pharetra
purus quis dignissim tempus. Nunc rhoncus laoreet eleifend. Vestibulum ante ipsum
primis in faucibus orci luctus et ultrices posuere cubilia curae; Praesent a blandit
est. In iaculis, risus et tempor lobortis, dolor nunc ullamcorper nisl, non pharetra
orci massa et lorem. Integer at eros nec risus viverra placerat. Vestibulum at neque
cursus, finibus est at, sodales sapien. Nunc mattis, lectus eu rutrum varius, dui
ligula gravida risus, nec pretium sem ante nec nulla. Aliquam malesuada non ante sed
eleifend. Vivamus venenatis pharetra scelerisque.</p>"
                    }
                ]
            },
            "secondaryColumn1": {
                "content": [
                    {
                        "elementType": "html",
                        "html": "<h2>Secondary Column 1</h2><p>Ut purus dui, ultrices
eget fringilla non, eleifend ac felis. Donec varius consectetur est sit amet rutrum.
Nam quis sagittis mauris. Suspendisse purus nibh, mattis at ultricies in, fringilla
vehicula neque. Nunc sed eros tempor, euismod arcu eget, tincidunt lectus. Donec
gravida euismod velit quis tempor. Praesent ultricies sed enim non dictum.</p>"
                    }
                ]
            },
            "secondaryColumn2": {
                "content": [
                    {
                        "elementType": "html",
                        "html": "<h2>Secondary Column 2</h2><p>In dictum posuere dui
vel ultricies. Nunc interdum libero quam, at gravida metus aliquet eget. Duis massa
dui, sollicitudin et rhoncus vel, pretium sed arcu. Vestibulum nisl felis, scelerisque
at feugiat quis, pellentesque eget justo. Nulla sed auctor ex, vel efficitur tortor.
Integer ac sapien a metus scelerisque fermentum.</p>"
                    }
                ]
            }
        },
        {
            "elementType": "divider"
        },
```

```
        {
            "elementType": "responsiveThreeColumn",
            "sideMargins": "none",
            "primaryColumn": {
                "content": [
                    {
                        "elementType": "html",
                        "html": "<h2>Primary Column</h2><p>Lorem ipsum dolor sit amet,
consectetur adipiscing elit. Nunc sodales luctus arcu eget aliquam. Nulla pharetra
purus quis dignissim tempus. Nunc rhoncus laoreet eleifend. Vestibulum ante ipsum
primis in faucibus orci luctus et ultrices posuere cubilia curae; Praesent a blandit
est. In iaculis, risus et tempor lobortis, dolor nunc ullamcorper nisl, non pharetra
orci massa et lorem. Integer at eros nec risus viverra placerat. Vestibulum at neque
cursus, finibus est at, sodales sapien. Nunc mattis, lectus eu rutrum varius, dui
ligula gravida risus, nec pretium sem ante nec nulla. Aliquam malesuada non ante sed
eleifend. Vivamus venenatis pharetra scelerisque.</p>"
                    }
                ]
            },
            "secondaryColumn1": {
                "content": [
                    {
                        "elementType": "html",
                        "html": "<h2>Secondary Column 1</h2><p>Ut purus dui, ultrices
eget fringilla non, eleifend ac felis. Donec varius consectetur est sit amet rutrum.
Nam quis sagittis mauris. Suspendisse purus nibh, mattis at ultricies in, fringilla
vehicula neque. Nunc sed eros tempor, euismod arcu eget, tincidunt lectus. Donec
gravida euismod velit quis tempor. Praesent ultricies sed enim non dictum.</p>"
                    }
                ]
            },
            "secondaryColumn2": {
                "content": [
                    {
                        "elementType": "html",
                        "html": "<h2>Secondary Column 2</h2><p>In dictum posuere dui
vel ultricies. Nunc interdum libero quam, at gravida metus aliquet eget. Duis massa
dui, sollicitudin et rhoncus vel, pretium sed arcu. Vestibulum nisl felis, scelerisque
at feugiat quis, pellentesque eget justo. Nulla sed auctor ex, vel efficitur tortor.
Integer ac sapien a metus scelerisque fermentum.</p>"
                    }
                ]
            }
        },
        {
            "elementType": "divider"
```

```
        },
        {
            "elementType": "responsiveThreeColumn",
            "sideMargins": "none",
            "gutters": false,
            "primaryColumn": {
                "content": [
                    {
                        "elementType": "html",
                        "html": "<h2>Primary Column</h2><p>Lorem ipsum dolor sit amet,
consectetur adipiscing elit. Nunc sodales luctus arcu eget aliquam. Nulla pharetra
purus quis dignissim tempus. Nunc rhoncus laoreet eleifend. Vestibulum ante ipsum
primis in faucibus orci luctus et ultrices posuere cubilia curae; Praesent a blandit
est. In iaculis, risus et tempor lobortis, dolor nunc ullamcorper nisl, non pharetra
orci massa et lorem. Integer at eros nec risus viverra placerat. Vestibulum at neque
cursus, finibus est at, sodales sapien. Nunc mattis, lectus eu rutrum varius, dui
ligula gravida risus, nec pretium sem ante nec nulla. Aliquam malesuada non ante sed
eleifend. Vivamus venenatis pharetra scelerisque.</p>"
                    }
                ]
            },
            "secondaryColumn1": {
                "content": [
                    {
                        "elementType": "html",
                        "html": "<h2>Secondary Column 1</h2><p>Ut purus dui, ultrices
eget fringilla non, eleifend ac felis. Donec varius consectetur est sit amet rutrum.
Nam quis sagittis mauris. Suspendisse purus nibh, mattis at ultricies in, fringilla
vehicula neque. Nunc sed eros tempor, euismod arcu eget, tincidunt lectus. Donec
gravida euismod velit quis tempor. Praesent ultricies sed enim non dictum.</p>"
                    }
                ]
            },
            "secondaryColumn2": {
                "content": [
                    {
                        "elementType": "html",
                        "html": "<h2>Secondary Column 2</h2><p>In dictum posuere dui
vel ultricies. Nunc interdum libero quam, at gravida metus aliquet eget. Duis massa
dui, sollicitudin et rhoncus vel, pretium sed arcu. Vestibulum nisl felis, scelerisque
at feugiat quis, pellentesque eget justo. Nulla sed auctor ex, vel efficitur tortor.
Integer ac sapien a metus scelerisque fermentum.</p>"
                    }
                ]
            }
        },
```

```
        {
            "elementType": "divider"
        },
        {
            "elementType": "simpleThreeColumn",
            "column1": {
                "content": [
                    {
                        "elementType": "html",
                        "html": "<h2>Column 1</h2><p>Lorem ipsum dolor sit amet,
consectetur adipiscing elit. Nunc sodales luctus arcu eget aliquam. Nulla pharetra
purus quis dignissim tempus. Nunc rhoncus laoreet eleifend. Vestibulum ante ipsum
primis in faucibus orci luctus et ultrices posuere cubilia curae; Praesent a blandit
est. In iaculis, risus et tempor lobortis, dolor nunc ullamcorper nisl, non pharetra
orci massa et lorem. Integer at eros nec risus viverra placerat. Vestibulum at neque
cursus, finibus est at, sodales sapien. Nunc mattis, lectus eu rutrum varius, dui
ligula gravida risus, nec pretium sem ante nec nulla. Aliquam malesuada non ante sed
eleifend. Vivamus venenatis pharetra scelerisque.</p>"
                    }
                ]
            },
            "column2": {
                "content": [
                    {
                        "elementType": "html",
                        "html": "<h2>Column 2</h2><p>Ut purus dui, ultrices eget
fringilla non, eleifend ac felis. Donec varius consectetur est sit amet rutrum. Nam
quis sagittis mauris. Suspendisse purus nibh, mattis at ultricies in, fringilla
vehicula neque. Nunc sed eros tempor, euismod arcu eget, tincidunt lectus. Donec
gravida euismod velit quis tempor. Praesent ultricies sed enim non dictum.</p>"
                    }
                ]
            },
            "column3": {
                "content": [
                    {
                        "elementType": "html",
                        "html": "<h2>Column 3</h2><p>In dictum posuere dui vel
ultricies. Nunc interdum libero quam, at gravida metus aliquet eget. Duis massa dui,
sollicitudin et rhoncus vel, pretium sed arcu. Vestibulum nisl felis, scelerisque at
feugiat quis, pellentesque eget justo. Nulla sed auctor ex, vel efficitur tortor.
Integer ac sapien a metus scelerisque fermentum.</p>"
                    }
                ]
            }
        }
```

```
    ]
}
```

Lab 7: Task 2

Responsive Visibility on Images

```
{
    "metadata": {
        "version": "2.0"
    },
    "contentContainerWidth": "full",
    "content": [
        {
            "elementType": "detail",
            "id": "image_hero",
            "title": "Difficulty: Hard<br>Power Required: 100<br>Duration: 3 Hours",
            "description": "Gold Rewarded: 2500",
            "body": "The Dragon's cave is one of the most lucrative quests in the
entire game. The 3 hour duration is worth it for the magical items you can collect, the
experience you will gain, and the amount of gold each member of your party will take
home.",
            "label": "Dragon's Cave",
            "thumbnailBorderRadius": "1rem",
            "hero": [
                {
                    "elementType": "image",
                    "url":
"https://s3.us-west-2.amazonaws.com/static.modolabs.com/xmodule_certification/images/dr
agon_cave_wide.png",
                    "alt": "Dragon's Cave image",
                    "responsiveVisibility": {
                        "xsmall": false,
                        "small": false,
                        "medium": true,
                        "large": true,
                        "xlarge": true
                    }
                },
                {
                    "elementType": "image",
                    "url":
"https://s3.us-west-2.amazonaws.com/static.modolabs.com/xmodule_certification/images/dr
agon_cave_narrow.png",
                    "responsiveVisibility": {
```

```
                    "xsmall": true,
                    "small": true,
                    "medium": false,
                    "large": false,
                    "xlarge": false
                }
            }
        ]
    }
]
}
```

```
export const handler = async(event) => {
    console.log("HTTP Request", JSON.stringify(event));

    let xmJson = {
    "metadata": {
        "version": "2.0"
    },
    "contentContainerWidth": "narrow",
    "content": [
        {
            "elementType": "hero",
            "height": "fluid",
            "contentContainerWidth": "medium",
            "backgroundImage": {
                "url":
"https://images.unsplash.com/photo-1542037104857-ffbb0b9155fb?ixid=MnwxMjA3fDB8MHxwaG90
by1wYWdlfHx8fGVufDB8fHx8&ixlib=rb-1.2.1&auto=format&fit=crop&w=3350&q=80",
                "alt": "Image of a family. Photo by
https://unsplash.com/@jessicarockowitz",
                "overlayType": "solid"
            },
            "content": [
                {
                    "elementType": "heroImage",
                    "imageSize": "1.65rem",
                    "image": {
                        "url":
"https://static.modolabs.com/james/images/logo_generic_R.png",
                        "alt": "Generic logo"
                    },
                    "marginTop": "3rem",
                    "marginBottom": "xxtight",
                    "horizontalAlignment": "left"
                },
                {
                    "elementType": "heroSubheading",
                    "subheading": "Institutional and Individual Management",
                    "textColor": "rgba(255,255,255,0.8)",
                    "textAlignment": "left",
                    "marginTop": "xxtight"
                },
                {
                    "elementType": "heroHeading",
                    "heading": "Making a better choice for your future",
                    "fontSize": "2.5rem",
```

```
                    "textColor": "#ffffff",
                    "textAlignment": "left"
                },
                {
                    "elementType": "heroButtons",
                    "horizontalAlignment": "right",
                    "marginBottom": "3rem",
                    "buttons": [
                        {
                            "elementType": "linkButton",
                            "backgroundColor": "#1ee8dc",
                            "borderColor": "#1ee8dc",
                            "borderWidth": "2px",
                            "title": "Learn more",
                            "textColor": "#000000"
                        }
                    ]
                }
            ]
        }
    ]
};
    const response = {
        statusCode: 200,
        body: JSON.stringify(xmJson),
    };
    return response;
};
```

Lab 8: Task 2

Updated the Lab 5: Task 1 Lambda function with the following code:

```
import jwkPem from 'jwk-to-pem';
import jwt from 'jsonwebtoken';
const key =
{"kid":"e17ae40d-d6c0-450f-9aac-2f565b3b1ca6","kty":"RSA","n":"3FOcqs0U3r7r8D_QnxiLgiCJ
2gV30uJFlJOC8OE9GGtlslKckoEHgji7iOrbtOEWGzHhZEOAG7E05iAHKDV-ZohAMruicSS3bd7xLw_8pCswlvd
0JpB4vjgGmwh1DqwanVlIYh6XD1Tllu5hfD6R89xHmz_akB6RBV-vDbErPqOxAs66LV8REDsc6FxdBOYcCHu9Qq
jsnDgUmoEuZVzeAozxBvGv5-krHoIHZHgMjvCK-ljS1egpBwS-GjbDcYMSEmsabuFhEdEvBLqP_cZG8Qo609Xy1
13YcjFFj0duRYbCBSW_Z_JOKpGXEagIzRd1hR258zsGzuwluaIpsGXXVQ","e":"AQAB"};

export const handler = async(event) => {
    console.log("HTTP Request", JSON.stringify(event));

    let xmJson = {
    "metadata": {
        "version": "2.0"
    }
};
    let xmForm = {
            "elementType": "form",
            "requestMethod": "GET",
            "id": "sample_form",
            "heading": {
                "heading": "Character Information",
                "headingLevel": 2,
                "description": "Items marked with an asterisk (*) are required."
            },
            "items": [
                {
                    "elementType": "formInputText",
                    "name": "s1_first",
                    "label": "First name",
                    "required": true
                },
                {
                    "elementType": "formInputText",
                    "name": "s1_last",
                    "label": "Last name",
                    "required": true
                },
                {
                    "elementType": "formInputText",
                    "name": "s1_class",
                    "label": "Class",
```

```
            "required": true
        },
        {
            "elementType": "formInputText",
            "name": "s1_race",
            "label": "Race",
            "required": true
        },
        {
            "elementType": "formInputSegmented",
            "name": "s1_age",
            "label": "Age",
            "options": {
                "Under 25": "Under 25",
                "25-40": "25-40",
                "41-60": "41-60",
                "61+": "61 or over"
            },
            "fullWidth": true
        },
        {
            "elementType": "formInputAssistedSelect",
            "name": "s1_animal",
            "label": "Animal companion",
            "options": {
                "": "",
                "alpaca": "Alpaca",
                "ant": "Ant",
                "bird": "Bird",
                "cat": "Cat",
                "dog": "Dog",
                "dragon": "Dragon",
                "emu": "Emu",
                "ferret": "Ferret",
                "fish": "Fish",
                "goat": "Goat",
                "guinea": "Guinea pig",
                "hedgehog": "Hedgehog",
                "horse": "Horse",
                "kangaroo": "Kangaroo",
                "lizard": "Lizard",
                "mouse": "Mouse",
                "pig": "Pig",
                "rabbit": "Rabbit",
                "rat": "Rat",
                "salamander": "Salamander",
```

```
                "seal": "Seal",
                "shark": "Shark",
                "sheep": "Sheep",
                "snake": "Snake",
                "spider": "Spider",
                "turtle": "Turtle",
                "unicorn": "Unicorn",
                "weasel": "Weasel"
            }
        },
        {
            "elementType": "formInputRadio",
            "label": "Preferred contact",
            "preamble": "What's the best way to reach you?",
            "name": "s1_contact",
            "options": {
                "email": "Email",
                "phone": "Phone",
                "text": "Text",
                "discord": "Discord",
                "other": "Other"
            },
            "nested": true,
            "progressiveDisclosureItems": {
                "email": [
                    {
                        "elementType": "formInputEmail",
                        "name": "s1_email",
                        "label": "Email address",
                        "required": true
                    }
                ],
                "phone": [
                    {
                        "elementType": "formInputPhone",
                        "name": "s1_phone",
                        "label": "Phone number"
                    }
                ],
                "text": [
                    {
                        "elementType": "formInputPhone",
                        "name": "s1_text",
                        "label": "Phone number for text"
                    }
                ],
```

```
                    "discord": [
                        {
                            "elementType": "formInputText",
                            "name": "s1_discord",
                            "label": "Discord username"
                        }
                    ],
                    "other": [
                        {
                            "elementType": "formInputTextarea",
                            "name": "s1_contact_other",
                            "label": "Describe the best way to contact you",
                            "rows": 4
                        }
                    ]
                }
            }
        ],
        "buttons": [
            {
                "elementType": "formButton",
                "name": "s1_reset",
                "title": "Reset",
                "buttonType": "reset",
                "actionStyle": "destructiveQuiet",
                "minWidth": "8rem"
            },
            {
                "elementType": "formButton",
                "name": "s1_submit",
                "title": "Submit",
                "buttonType": "submit",
                "actionStyle": "constructive",
                "minWidth": "8rem"
            }
        ],
        "trackDirtyStateButtonNames": [
            "s1_submit"
        ],
        "buttonsHorizontalAlignment": "center"
    };

//if we have query string parameters we should set a cookie
if ('queryStringParameters' in event && event.queryStringParameters != null ) {
    xmJson.metadata.cookies = [{
            "name": "banner_msg",
```

```
                "path": "/",
                "value": "You have successfully updated your character information"
            }];

        xmJson.metadata.redirectLink = {
            "relativePath": ""
        };
    } else {
        xmJson.content = [xmForm];
    }

    let payload = null; //create a variable to hold the decoded payload values

    //check that headers exist in event and that authorization is in event.headers
    if ('headers' in event && 'authorization' in event.headers) {
        payload = jwt.verify(event.headers.authorization.split(" ")[1], jwkPem(key));
        console.log("Payload", payload);
    }

    //check for cookies in playload and make sure banner_msg exists
    if (payload != null && 'cookies' in payload && payload.cookies != null &&
'banner_msg' in payload.cookies) {

        //create a banner message with the cookie
        xmJson.metadata.banners = [
            {
                "message": payload.cookies.banner_msg,
                "type": "info"
            }
        ];

        //since the banner message has been created, we need to delete the cookie so we
don't show the message again unless they create another submission
        xmJson.metadata.cookies = [{
                "name": "banner_msg",
                "path": "/",
                "value": null
            }];
    }

    const response = {
        statusCode: 200,
        body: JSON.stringify(xmJson),
    };
    return response;
};
```