

Add DaVinci Widget to Existing Web Application

Table of Contents

- 1 Objective
- 2 Do the following task items
 - 2.1 Add DaVinci library to application
 - 2.2 Create a place for rendering the flow
 - 2.3 Review JavaScript code that invokes flow
 - 2.4 Connect it to the button in application
 - 2.4.1 Time to test your code

1 Objective

Presently your flow is the only object displayed on the page. Most of the time, you'll want to render your flow within an existing application. In this exercise you'll embed the Registration flow but only show it as a modal popup once a user presses the Register Now button.

In prior exercises this was discussed at a high level and references to the [documentation](#) were provided. In this exercise you will dive a bit more into the code needed to complete the integration.

Tip

All the number instruction steps in each section must be followed to successfully complete this exercise. Any bullet points or images are for example only unless called out in a numbered step.

Tip

You will be using Glitch for editing the code and testing with the your flow changes. Before you start a lab or need to access Glitch, ensure that you are logged into your Glitch account and accessing the current project.

2 Do the following task items

Specifically in this exercise you will be doing the [widget method](#) because you are integrating into an existing Web application.

As stated above you are going to dive further into the code, but do not fear it is going to be a matter of copy and paste, followed by an explanation of the code. The general approach to integrating to

an application is:

- 1 Import the DaVinci JavaScript library - this is the code that makes the actual calls to the DaVinci API to invoke your flow
- 2 Create a div container to hold the flow contents - leverage the power of HTML to allow you to reference placement in your application
- 3 Create a JavaScript method to call the flow - leveraging the library you imported
- 4 Connect the click event of the button in your application to render the flow method - in other words provide that user experience you want DaVinci to provide

This is what you will be accomplishing in the following sections of this exercise.


2.1 Add DaVinci library to application

In order to use the [widget method](#) you need to import the library into your application. The URL for this library is:

- <https://assets.pingone.com/davinci/latest/davinci.js>

But the first thing will be to swap out your index.html page to the new page that reflects the application that you will modify. Once this is done then you will customize it to use the DaVinci library.

- 1 In a browser tab **open** your **Glitch** project that you created in the earlier exercise.
 - All the changes will be made in this application, nothing needs to be done in DaVinci.
 - You will be copying some information from DaVinci or the existing page, your choice.

- 2 In the editor select the **index.html** file then click the  icon next to the name to display the menu options you can use on the file.



- 3 Click **Rename** to rename the file.
 - Note that the rename does allow you to essentially move the file to another folder by adding a folder name followed by / and the the file name.
 - You don't need to do this for this exercise, just a tip for future reference in using Glitch.

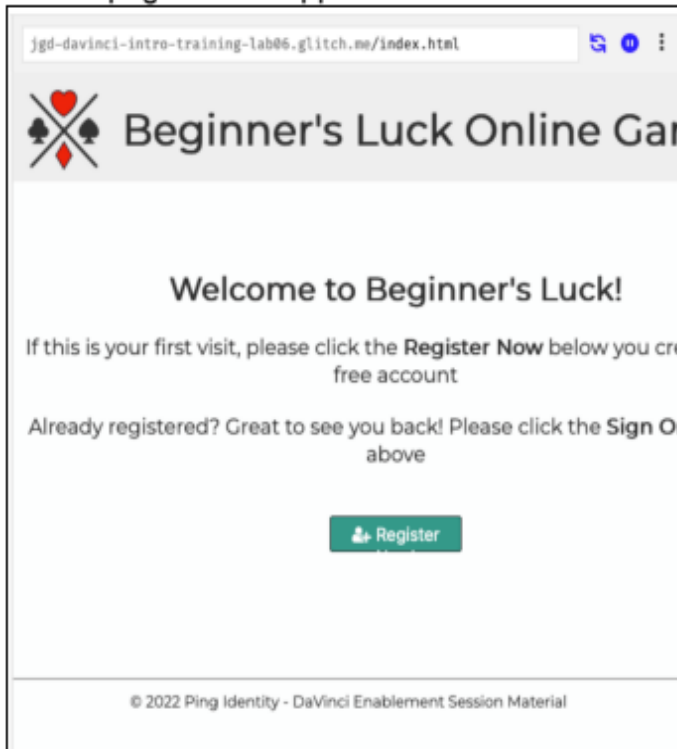
4 Change the file name to lab06-index.html

```
views/  
lab06-index.html  
lab08-index.html  
.env
```

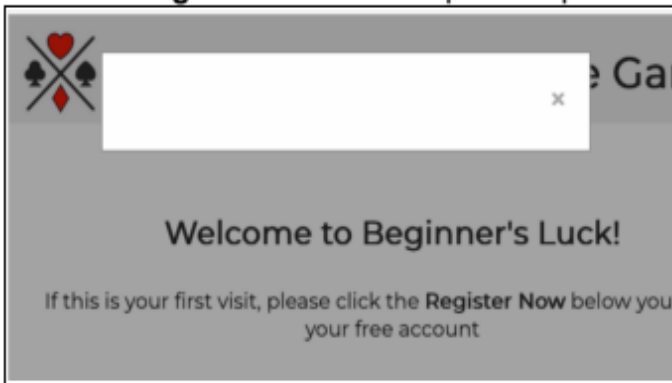
- If the preview pane is open in Glitch you will see an exception, because the index.html file does not exist, ignore it.

5 Select the lab08-index.html file and rename the file to index.html, this will now be your application

```
views/  
index.html  
lab06-index.html  
.env
```

6 Reload the preview pane you to see the application that you will customize; make sure that the address in preview shows index.html and not lab08-index.html or it maybe blank as that is the default page for the application. The look and of the page will be like the following screenshot.

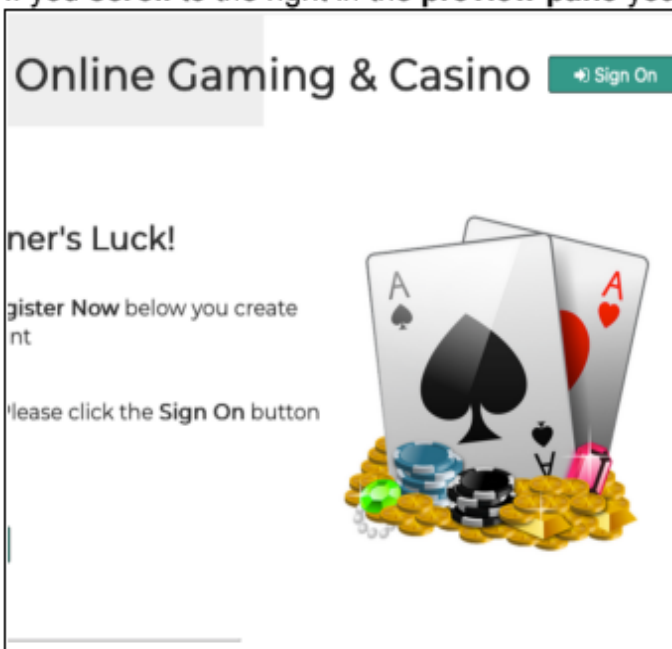
7 Click the **Register** button in the preview pane.



- An empty modal will popup over the application page. This is where the flow will be rendered for interaction with the user.

8 Click **x** to close the modal.

9 If you **scroll** to the right in the **preview pane** you will see a Sign On button.



- This will be part of future exercises where you will build a flow and integrate into the application.

10 Make sure that you are at the top of the **index.html** file in the editor.

11 The code to load the DaVinci library will be placed below the existing link elements and above the existing script element.

```

7      />
8      <link rel="stylesheet" href="/css/glitch.css" />
9      <link rel="stylesheet" href="/css/site.css" />
10     <link rel="stylesheet" href="/css/flow-updated.css" />
11
12     <script type="text/javascript">
13         // Set as .env variables

```

12 **Copy** the following code:

```

<!-- PingOne DaVinci Javascript library -->
<script type="text/javascript" src="https://assets.pingone.com/davinci/latest/davinci.js"
></script>

```


- 13 Depending on the region your tenant is deployed to you may need to adjust the url in the above code. If your tenant is **not** in North America, then change it as follows:

```
APAC
  assets.pingone.com is assets.pingone.asia
EU
  assets.pingone.com is assets.pingone.eu
Canada
  assets.pingone.com is assets.pingone.ca
```

- 14 **Paste** this code in the **editor** below the last **link** element.

```
10 <link rel="stylesheet" href="/css/flow-updated.css" />
11
12<!-- PingOne DaVinci Javascript library -->
13<script
14  type="text/javascript"
15  src="https://assets.pingone.com/davinci/latest/davinci.js"
16></script>
17
18<script type="text/javascript">
19  // Set up any variables
```

- You may need to space it out to align, although not required but does make it look nice.
- You can also optionally use the *Prettier* button at the top of the editor next to the file name to have Glitch editor reformat the file for you.
- This code simply imports the code that is the DaVinci library into your application when the page is loaded in a browser.
- Changes to the file are saved automatically by the Glitch editor.

- 15 Continue with the next section.

2.2 Create a place for rendering the flow

The flow needs to be rendered by the DaVinci library on your page, in this case the modal that will popup. In order to do this you need to indicate a spot in your HTML that can be used by the library to do that rendering. The best way to accomplish this is with a `div` element that has an `id` attribute. The `id` does need to be unique in your HTML page.

- 1 **Scroll** down towards the bottom of the `index.html` file where you will find `div` element with an `id` of `modalPopup`, this is the registration popup.

```
150 </div>
151
152<!-- PopUp Modal Dialog where the DaVinci Flow will be placed -->
153<div id="modalPopup" class="modal">
154  <div class="modal-content">
155    <div id="closeBtn" class="close">&times;</div>
156  </div>
157</div>
158</body>
159</html>
```

- 2 Your code will need to be a child of the `div` element with a **class** of `modal-content`
- It will go right under the `div` with `id="closeBtn"`, as shown in the above screenshot.
 - You may want to insert a line to make it easier to paste it in.
- 3 **Copy** the following code:

```
<div id="flow-widget" class="flow-widget"></div>
```

4 Paste this code in the editor below **div** with **id="closeBtn"**

```

151
152- <!-- PopUp Modal Dialog where the DaVinci Flow will be placed -->
153- <div id="modalPopup" class="modal">
154-   <div class="modal-content">
155-     <div id="closeBtn" class="close">6times;</div>
156-     <div id="flow-widget" class="flow-widget"></div>
157-   </div>
158- </div>
159 </body>
160 </html>

```

- You may need to space it out to align, although not required but does make it look nice.
- This code provides a place where the flow will be rendered. The **id** will be passed to the DaVinci library when you call the render function.

5 Continue with the next section.

2.3 Review JavaScript code that invokes flow

In this section you will not actually write all the JavaScript code, there is a bit of, it will be provided for you. You will configure it to point to your DaVinci application used on the other page from previous lab. You will also review the code to understand what it is doing.

The code will leverage the variables you updated in the earlier exercise. The only thing that needs to be copied is the policy ID from your old code. You will do that in the next section. This section is to just review the provided code that you would need, but is already provided.

- 1 **Scroll** to the top of the **index.html** file, up to the second script tag, which is below the one you inserted in previous section.
- 2 This script tag contains a few functions that are used in this application and you will be adding more to it in future exercises.
- 3 Just below the script tag there are two constants defined as template variables.

```

17
18- <script type="text/javascript">
19   // Set as .env variables
20   const companyId = "<%= companyId %>";
21-   /* TENANT ENDPOINT - .env variable DAVINCI_API_URL */
22   const flowURL = "<%= flowURL %>";
23

```

- This is similar to what you reviewed in an earlier exercise and loads the variables from the .env page of your project.

4 **Scroll** down until you see the **loadwidget** function definition.

```

58- function loadwidget(policyId, renderComponent) {
59-   var requestOptions = {
60-     method: "GET",
61-   };
62

```

- This is essentially the same function in the other HTML page that loaded the widget using the DaVinci code.
- You could of placed this code in a separate JavaScript file and loaded at runtime but for simplicity in this lab you will keep the code all in one file.
- It will grow, so separating is not a bad idea, and in a more complicated application with multiple developers that would be a preferred approach.

- 4.1 Note the **policyID** parameter passed to the function, which is the flow you want to render.
- 4.2 Note the **renderComponent** parameter passed to the function, which is where you want to be rendered on the page.
- 4.3 This makes the function reusable to render other flows in the application.

5 A little further down in the function you will see **fetch("/fetchDaVinciToken")** code block.

```

63-  /** Retrieve DaVinci Token */
64-  fetch("/fetchDaVinciToken", requestOptions)
65-    .then((response) => response.json())
66-    .then((responseData) => {
67-      var props = {
68-        config: {
69-          method: "runFlow",
70-          apiRoot: flowURL,
71-          accessToken: responseData.access_token,
72-          companyId: companyId,
73-          policyId: policyId,
74-        },
75-        useModal: false,
76-        successCallback,
77-        errorCallback,
78-        onCloseModal,
79-      };
80-      /** Invoke DaVinci Widget */

```

- This is the same code you reviewed in the earlier exercise that calls the server-side code in your application to get a DaVinci token.
- It sets up the props object required to invoke the DaVinci widget with the policy ID and of course the token.

6 And a little further down you will see **davinci.skRenderScreen** code block.

```

83-
84-    console.log(props);
85-    davinci.skRenderScreen(
86-      document.getElementById(renderComponent),
87-      props
88-    );

```

- This makes the actual call to render widget, using the code you added a link to at the start of this exercise.
- It takes the renderComponent parameter to find the element on the current page.
 - This a pointer to the `div` you setup for it, earlier in the exercise.
- The props object is passed to the function, which includes the policy ID and DaVinci token

7 Continue with the next section.

2.4 Connect it to the button in application

Now it is time for the last step and that is to write code that will link the register button to `loadwidget` function you reviewed in the previous section. This provides the final link needed so that you can invoke the flow in the application.

1 In the **Glitch** editor continue with the **index.html** file.

2 **Scroll** to the bottom of **script** you were just looking at, there are other functions in this code.

```

94-    }
95-
96-    function errorCallback(error) {
97-      console.log(error);
98-    }
99-
100-    function onCloseModal() {
101-      console.log("onCloseModal");
102-    }
103-  }
104-  </script>
105-  </head>
106-
107-  <body onload="initialize()">

```

- You will add your new function at the end of this block just before the closing `</script>` tag.

3 Copy the following code:

```
// Load DaVinci Registration Flow
function loadRegistrationFlow() {
  const policyId = "";
  const divComponent = "flow-widget";
  loadwidget(policyId, divComponent);
}
```

4 Paste this code in the editor above </script> tag

```
99
100~      function onCloseModal() {
101~          console.log("onCloseModal");
102~      }
103~  }
104~
105~      // Load DaVinci Registration Flow
106~      function loadRegistrationFlow() {
107~          const policyId = "";
108~          const divComponent = "flow-widget";
109~          loadwidget(policyId, divComponent);
110~      }
111~  </script>
112~  </head>
```

- You may need to space it out to align, although not required but does make it look nice.
- This code calls the `loadwidget` function passing it the placement element and the policy ID for your DaVinci application.

5 Fill the const values to be passed in the function call.

5.1 **policyId** use either from your DaVinci application or the values in the **lab06-index.html** copy between the quotes.

5.2 **divComponent** type **flow-widget** this is the `id` you provided for your `div` component earlier.

```
// Load DaVinci Registration Flow
function loadRegistrationFlow() {
  const policyId = "28[REDACTED]3"
  const divComponent = "flow-widget"
  loadwidget(policyId, divComponent)
}
```

6 If you scroll up a bit you will see a function call for when the register button is clicked that calls the function `loadRegistrationFlow` you just added to your application.

```
// Create 'Register Now' button click handler
var registerBtn = document.getElementById("btn-register");
registerBtn.onclick = function() {
  // Display the modal popup div
  modalPopup.style.display = "block";
  loadRegistrationFlow()
}
```

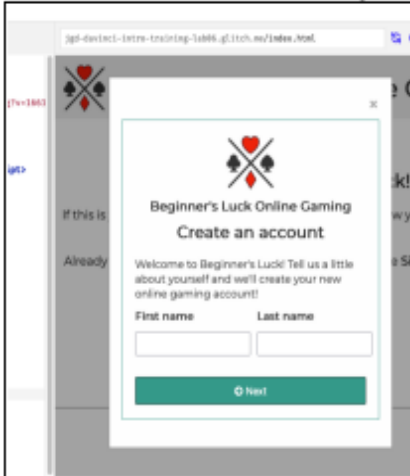
- At this point everything is in place to invoke your flow in the modal popup.

7 Continue with the next section.

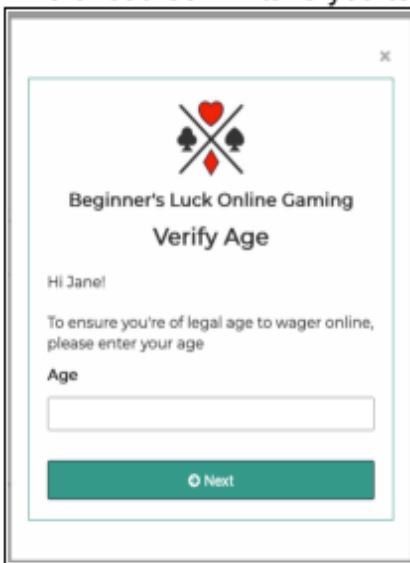
2.4.1 Time to test your code

All the code is in place to execute your flow when the register button is clicked. You can test this in the preview pane or open a new tab to preview it. Your choice. The examples that follow will use the preview pane.

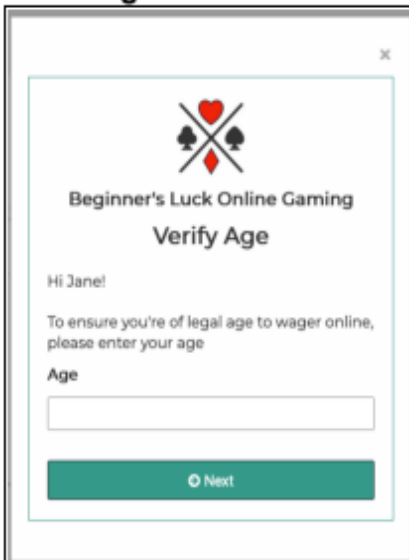
- 1 In the **preview** click **Register** button.
- 2 The modal should render your flow:



- 3 Enter a first and last name then click **Next** button.
- 4 This of course will take you to the age page.



- 5 Click the **x** to close the modal popup.

6 Click **Register** button.

- This took you back to the age page in your flow.
- Sometimes you want this behavior in your application.
- But you don't in this particular case, and it is easy to resolve.
- You may notice that it may only briefly appear and then return back to the first page, so the cleanup is being done. Still not a bad idea to add the code, so that even the brief appearance is not there.

7 In your **index.html** page in the **script** block, find the function `closeModalPopup`

```
// Close the modal popup
function closeModalPopup() {
  // Hide the modal popup div
  modalPopup.style.display = "none";
}
```

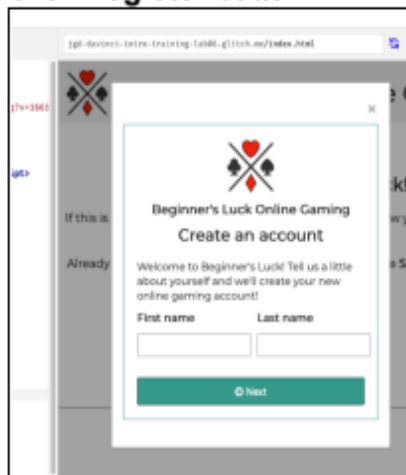
8 **Copy** the following code:

```
davinci.cleanup(document.getElementById("flow-widget"));
```

9 **Paste** this code in the **editor** below the last line `closeModalPopup` function

```
// Close the modal popup
function closeModalPopup() {
  // Hide the modal popup div
  modalPopup.style.display = "none";
  davinci.cleanup(document.getElementById("flow-widget"));
}
```

- You may need to space it out to align, although not required but does make it look nice.
 - This code calls the `cleanup` function passing it the placement element.
 - This function is provided by the DaVinci library you imported.
- 10 In the **preview** click **Register** button.
 - 11 The modal should render your flow.
 - 12 Enter a first and last name then click **Next** button.
 - 13 This of course will take you to the age page.
 - 14 Click the **x** to close the modal popup.

15 Click Register button.

- You are now at the start of your flow.

16 You have completed this exercise.