

Using non-UI Flows as API Service Wrapper

Table of Contents

- 1 Objective
- 2 Do the following task items
 - 2.1 Register for IP information service
 - 2.2 Build flow that returns jackpot data based on IP address
 - 2.2.1 Retrieve the jackpot data
 - 2.2.2 Set IP and use the registry connector
 - 2.2.3 Lookup jackpot for country and complete flow
 - 2.3 Testing flow in DaVinci
 - 2.3.1 Adjust the emoji variable to pull the proper attribute
 - 2.3.2 Run a final test to verify emoji returned as expected
 - 2.3.3 Document your code with some annotations
 - 2.4 Integrate flow into application
 - 2.4.1 Deal with loading the jackpot data
 - 2.4.1.1 Define a policy flow in the DaVinci application
 - 2.4.2 Handle data returned from flow

1 Objective

The flows you've created within DaVinci to this point in the training have been UI driven. A user interacts with the flow and progresses through various screens while DaVinci integrates with various systems and services in the background.

In this exercise, you'll explore how DaVinci can accelerate development when integrating with backend services and APIs, enriching the overall user experience.

Keeping with the theme of our Online Gaming site, let's examine a flow that returns the weekly jackpot information for users' country and display it on the homepage to help entice users to register.

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

In order to support this flow and provide some data you will use the base Glitch application that you remixed from as the source for the Jackpot data. More on that later in the exercise.

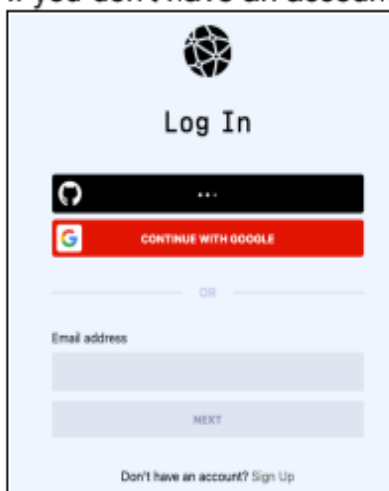
You will need to register for an IP service with a developer trial account. This will be used to retrieve the country of the user by using the IP address that the browser provides.

2.1 Register for IP information service

In this exercise you will be using a connector available in DaVinci that returns information about a particular IPv4 IP address. Information on this connector can be found in the Ping integrations directory at ipregistry.co.

In order to use this connector you will need an API key and therefore must have an account. You can signup for a free account that will allow you to make up to 100,000 API calls before you need to pay for the service. This is more than sufficient for this training and after you complete the training you can even delete the account.

- 1 In a browser tab **enter** the following URL or click the link.
 - <https://dashboard.ipregistry.co/overview>
 - If you don't have an account already with the service you will be prompted to register.



- Note that you can use a Google or GitHub account instead of creating a new one. Saves having to create yet another account and password.
- 2 You will receive an email to activate your account with *ipregistry*, make sure to look for it and **activate** the account.

3 Click on **API Keys** in the left menu once you have logged into the dashboard.



4 **Copy** the API key to a temporary editor window for later use. Or you can come back to this page

5 You can review the API documentation and even try it out on this page.

6 The **Overview** menu will take you to a display that shows how many API calls you made and have left.



7 Continue with the next section.

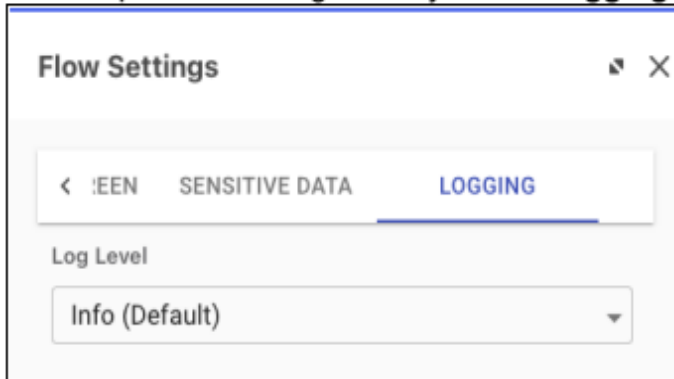
2.2 Build flow that returns jackpot data based on IP address

Now it is time to build out the flow that will fetch the jackpot data and use the IP service to get country information. This flow will wrap these two services and return the country, currency, and jackpot amount to the caller. This information will then be displayed on the application in a later section of this exercise.

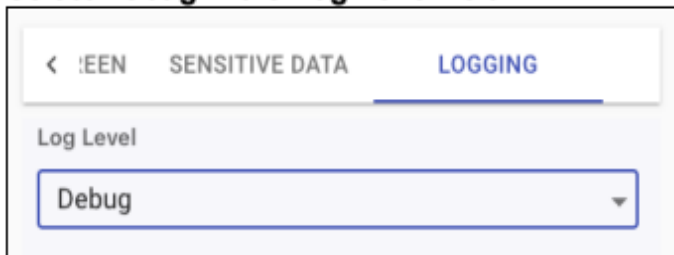
Building a DaVinci flow with no UI is really no different than building one with UI. It is all about

linking together nodes to provide the end result that you want for the calling application.

- 1 **Open** your DaVinci Administrator console in a browser tab.
- 2 Select **Flows** from the left menu.
- 3 Click **Add Flow** button to create a new **Blank Flow**
 - 3.1 **Name** your flow as
 - **Retrieve Jackpot Data via API**
 - 3.2 Click **Create** button.
- 4 Select **Flow Settings** to access the flow settings.
- 5 In the top tabs scroll right until you see **Logging** tab, then select it.



- 6 Select **Debug** in the **Log Level** field.



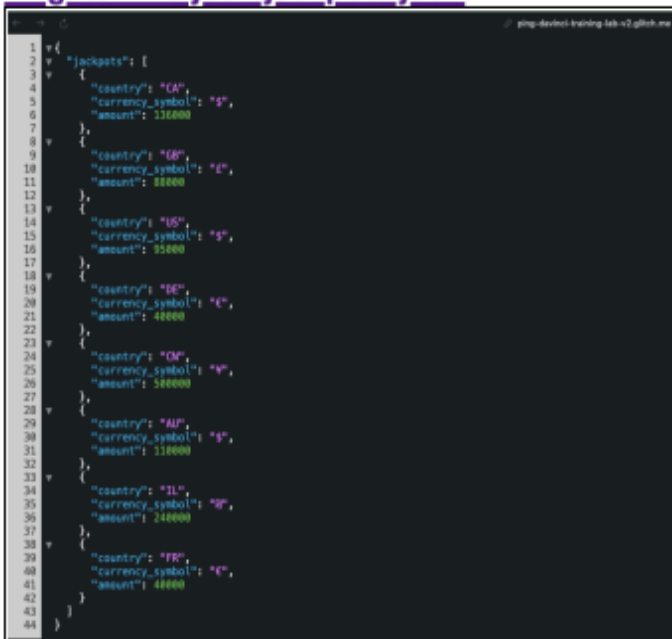
- In this exercise you will need to debug it to see the data being passed and returned by the nodes to understand and properly configure your flow.
 - As a general rule enabling debug when first creating a flow is not a bad habit to get into, as you may need to troubleshoot your flow. Once it is done and ready for production then you can turn debug off.
 - You will be doing this by using the Analytics option of DaVinci.
- 7 Click **Save** button to save the change.
 - 8 Click on the **Cancel** button to close the dialog.
 - 9 Continue with the next section.

2.2.1 Retrieve the jackpot data

The first thing to do is to retrieve the Jackpot data for all countries. Normally this would probably be some sort of API that would be called, but for simplicity it is going to be a JSON file hosted in base application. Once this data is retrieved then it can be parsed to retrieve the data for a specific country.

The data file does reside as part of your application, but in order to be able to use that file you need to have a paid Glitch account and your application must be boosted. If you can do that then you are welcome to use your own application file.

- 1 Click **+** button in the bottom left of the flow canvas.
- 2 Search for **var** then select **Variables** connector.
 - The variables connector will hold some initial data needed for other nodes in the flow and also to help in testing the flow.
- 3 Click the **Variables** node you just placed on your flow to configure it as follows:
 - 3.1 Select **Flow Instance Variable** for the capability.
 - 3.2 Add a field called **ipAddress** and make it of type **string**.
 - This will be used to provide a specific IP address to test the flow in DaVinci, before you add it to your application.
 - 3.3 Add a field called **jackpot_source_file** and make it of type **string**.
 - This will be the pointer to the JSON file that contains the data.
 - 3.4 You may want to apply your configuration for the node up to this point.
- 4 Lets take a look at the file, open a browser tab to the URL <https://ping-davinci-training-lab-v2.glitch.me/json/jackpots.json>



```

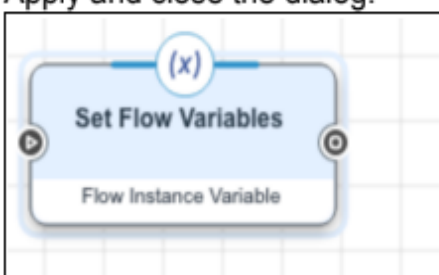
1 {
2   "jackpots": [
3     {
4       "country": "CA",
5       "currency_symbol": "$",
6       "amount": 110000
7     },
8     {
9       "country": "GB",
10      "currency_symbol": "£",
11      "amount": 80000
12     },
13     {
14       "country": "US",
15       "currency_symbol": "$",
16       "amount": 95000
17     },
18     {
19       "country": "GB",
20       "currency_symbol": "£",
21       "amount": 40000
22     },
23     {
24       "country": "OR",
25       "currency_symbol": "¥",
26       "amount": 500000
27     },
28     {
29       "country": "AU",
30       "currency_symbol": "$",
31       "amount": 110000
32     },
33     {
34       "country": "IL",
35       "currency_symbol": "₪",
36       "amount": 240000
37     },
38     {
39       "country": "FR",
40       "currency_symbol": "€",
41       "amount": 40000
42     }
43   ]
44 }

```

- Depending on your browser and browser extensions the view may look different, basically this is a JSON file with some test data that will be used by your flow.
- 5 **Copy and paste** the above URL in your flow as the **Value** for **jackpot_source_file** variable.



- 6 In the **Settings** tab of the node set the **Node Title** to
 - **Set Flow Variables**
- 7 Apply and close the dialog.



- 8 From the **Set Flow Variables** node draw a line and then add an **Http** node to your flow.
- 9 Click the **Http** node you just placed on your flow to configure it as follows:

9.1 Select **Make REST API Call** for the capability.

9.2 For the **URL** field add the **Flow Instance Variables** of **jackpot_source_file**

URL * indicates required

jackpot_source_file

HTTP Method *

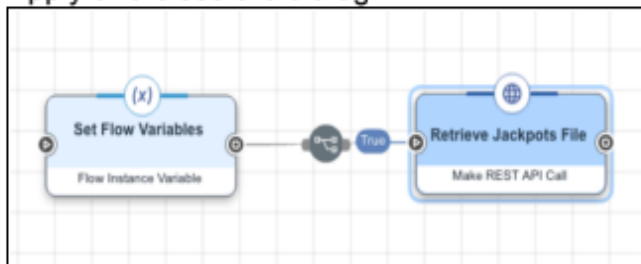
GET

9.3 The method will be **GET**, which is the default and you will take the rest of the defaults.

9.4 In the **Settings** tab of the node set the **Node Title** to

- **Retrieve Jackpots File**

9.5 Apply and close the dialog.



10 From the **Make REST API Call** node draw a line and then add an **Functions** node to your flow.

11 Click the **Functions** node you just placed on your flow to configure it as follows:

11.1 Select **A is Empty** for the capability.

11.2 For the **Value A** field add the **Flow Instance Variables** of **ipAddress**

- If the IP address is not provided as a variable then it will be retrieved from the flow at runtime.

11.3 Toggle **Check undefined/null** on.

TRIGGER:

A is Empty

GENERAL SETTINGS

Value A indicates required

ipAddress

Check undefined/null

String

11.4 In the **Settings** tab of the node set the **Node Title** to

- **Is IP Empty**

11.5 Apply and close the dialog.

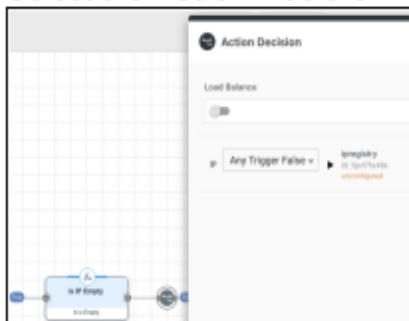


12 Continue with the next section.

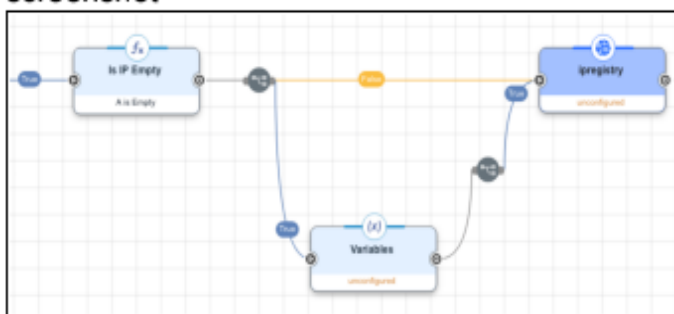
2.2.2 Set IP and use the registry connector

Now you have the Jackpot data in your flow and you started to verify if the IP exists. Now it is time to setup the call to the IP Registry to retrieve the country information assigned to the IP.

- 1 From the **Is IP Empty** node draw a line and then add an **ipregistry** node to your flow.
- 2 Drag the **ipregistry** node a bit to the right to give it some space.
- 3 Select the **Action Decision** node and change the decision to **Any Trigger False**



- 4 From the **Action Decision** node draw a line and then add an **Variables** node to your flow, it should be **All Triggers True**
- 5 From **Variables** draw a line to **ipregistry** node, at this point your flow should be as shown in screenshot



- 6 Click the **Variables** node in your flow to configure it as follows:
 - 6.1 Select **Flow Instance Variable** for the capability.

6.2 Add the **ipAddress** as the field with **Global** of **ip**

Flow Instance Variable

GENERAL SETTINGS >

* indicates required

Set Variables Edit + Field

Variable Name ipAddress (... x v)

ip Global

companyId (string)

flowId (string)

flowVersionId (string)

interactionId (string)

policyId (string)

ip (string)

6.3 In the **Settings** tab of the node set the **Node Title** to

- **Set IP Address**

6.4 Apply and close the dialog.



7 Click the **ipregistry** node in your flow to configure it as follows:

7.1 Select **Single Ip Lookup** for the capability.

7.2 Click the **gear** icon to configure the connector.

7.3 In the dialog popup enter (or paste) your **API Key** from the IP registry service you registered for in the earlier section of this exercise.

ipregistry Details

Company ID: 18d96224-c210-44c0-be4b-6405e0245b5a
Connection ID: da0c1046c62bacb987fd94557a52aecd

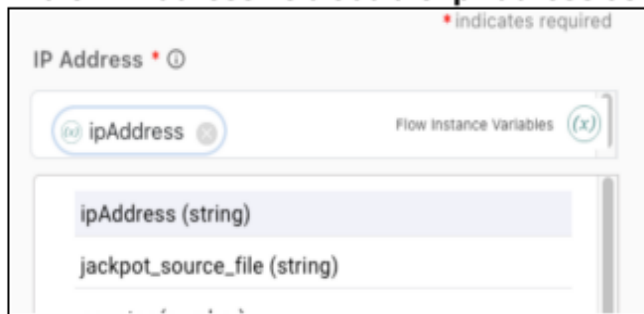
GENERAL CAPABILITIES

* indicates required

API Key *

7.4 Click **Apply** button to save the change and then **Close** button to close the dialog.

7.5 In the **IP Address** field add the **ipAddress** as the variable from **Flow Instance Variables**



7.6 In the **Settings** tab of the node set the **Node Title** to

- **IP Registry**

7.7 Apply and close the dialog.

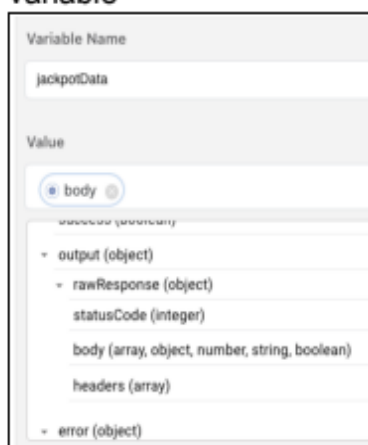


8 Continue with the next section.

2.2.3 Lookup jackpot for country and complete flow

Now it is time to complete the flow with retrieving the jackpot based on the country for the IP and then providing the final answer to the calling application. The final response will be JSON data that can then be displayed by the application.

- 1 From the **IP Registry** node draw a line and then add an **Functions** node to your flow.
- 2 Click the **Functions** node you just placed on your flow to configure it as follows:
 - 2.1 Select **Custom Function** for the capability.
 - 2.2 Click **Add** button under **Variable Input List** to add an input variable with following options
 - 2.2.1 **Value** variable reference from node **Retrieve Jackpots File** under output select **body** variable



2.2.2 Variable Name as

- **jackpotData**

2.2.3 Make sure that the **Variable Name** is **jackpotData** as it may get changed to body when you selected the value.

2.2.4 Select **Data Type** as **Object** from the dropdown2.3 Click **Add** button under **Variable Input List** to add an input variable with following options2.3.1 **Variable Name** as

- **code**

2.3.2 **Value** variable reference from node **IP Registry** under **country** select **code** variable

- Use the below image as a guide.

Input Variable 2:

Variable Name
code

Value
code

country (object)

- area (number)
- borders (array)
- calling_code (string)
- capital (string)
- code (string)

Data Type
String

2.3.3 The **Data Type** is the default of **String**

2.4 You may want to apply your changes at this point.

2.5 In the **Code** field below the input variables remove the existing sample code and **copy** and **paste** the following code:

```
module.exports = a = async ({ params }) => {
  const jackpotData = params.jackpotData;
  const countryCode = params.code;
  let retval;
  // iterate over each element in the array
  for (var i = 0; i < jackpotData.jackpots.length; i++) {
    let countryItem = jackpotData.jackpots[i];

    if (countryItem.country.toLowerCase() === countryCode.toLowerCase()) {
      return { 'jackpot': countryItem.amount, 'currency': countryItem.currency_symbol }
    }
  }
  return { 'jackpot': 0, 'currency': "unknown" }
}
```

- This code loops through the JSON file you reviewed earlier and searches for an entry that matches the country code of the IP.
- If none is found then 0 jackpot is returned.

```
Code
1 module.exports = a = async ({ params }) => {
2   const jackpotData = params.jackpotData;
3   const countryCode = params.code;
4   let retval;
5   // iterate over each element in the array
6   for (var i = 0; i < jackpotData.jackpots.length; i++) {
7     let countryItem = jackpotData.jackpots[i];
8     if (countryItem.country.toLowerCase() === countryCode.toLowerCase()) {
9       return { 'jackpot': countryItem.amount, 'currency': countryItem.currency_symbol }
10    }
11  }
12  return { 'jackpot': 0, 'currency': "unknown" }
13 }
```

2.6 In the **Output Schema** field **copy** and **paste** the following JSON schema:

```
{
  "output": {
    "type": "object",
    "properties": {
      "jackpot": {
        "type": "number"
      },
      "currency": {
        "type": "string"
      }
    }
  }
}
```

- The node will return an object with the jackpot amount and the currency of the jackpot.
- You are encouraged to review the code in both entries to understand what is going on and if you have questions ask the instructor.

2.7 In the **Settings** tab of the node

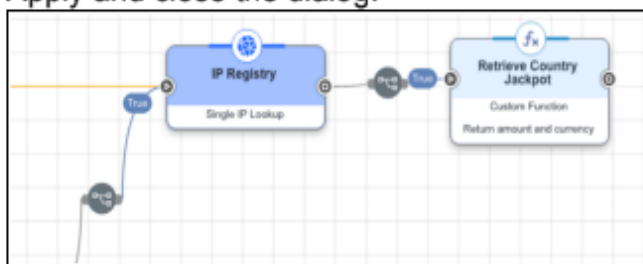
2.7.1 Set the **Node Title** to

- **Retrieve Country Jackpot**

2.7.2 Set the **Node Description** to

- **Return amount and currency**

2.8 Apply and close the dialog.

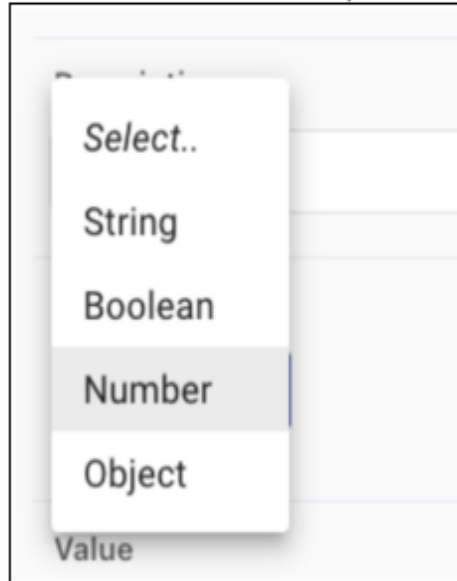


- 3 From the **Retrieve Country Jackpot** node draw a line and then add an **Http** node to your flow.
- 4 Click the **Http** node you just placed on your flow to configure it as follows:
 - 4.1 Select **Send Success JSON Response** for the capability.
 - This will return the data that the calling application expects.
 - You will define this data as additional fields in the response leveraging data from other nodes.
 - 4.2 Click **+ Field** button under **Additional Fields in the Response** to the data to be returned.
 - 4.2.1 Add **jackpot_country_code** as the variable and a data type of **String**
 - 4.2.2 **Value** is variable **code** from the **IP Registry** node.
 - 4.3 Click **+ Field** button under **Additional Fields in the Response** to the data to be returned.
 - 4.3.1 Add **jackpot_currency** as the variable and a data type of **String**
 - 4.3.2 **Value** is variable **currency** from the **Retrieve Country Jackpot** node.
 - 4.4 Click **+ Field** button under **Additional Fields in the Response** to the data to be returned.
 - 4.4.1 Add **jackpot_amount** as the variable and a data type of **String**
 - 4.4.2 **Value** is variable **jackpot** from the **Retrieve Country Jackpot** node.

4.5 Click **+ Field** button under **Additional Fields in the Response** to the data to be returned.

4.5.1 Add **httpStatusCode** as the variable

4.5.1.1 Select **Number** from dropdown list for **Data Type**



- The type is important so make sure to set it properly.
- If you miss it you can go to the **Variables** on the left menu and edit it there.

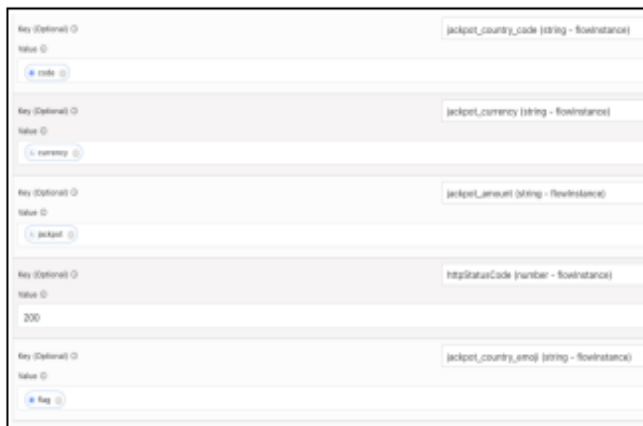
4.5.2 **Value** as **200** a constant value.

4.6 Click **+ Field** button under **Additional Fields in the Response** to the data to be returned.

4.6.1 Add **jackpot_country_emoji** as the variable and a data type of **String**

4.6.2 **Value** as variable **flag** from the **IP Registry** node.

- The value you need is part of the flag object, but is not available in the schema result for the service.
- You will figure it out later by debugging the code during testing of the flow.



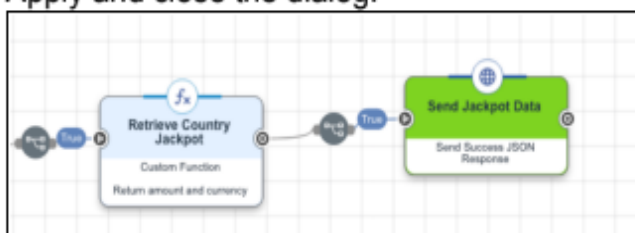
4.7 Apply your changes, always good to save as you go along.

4.8 In the **Settings** tab of the node set the **Node Title** to

- **Send Jackpot Data**

4.9 Select **light-green** as the **Node Background Color**

4.10 Apply and close the dialog.



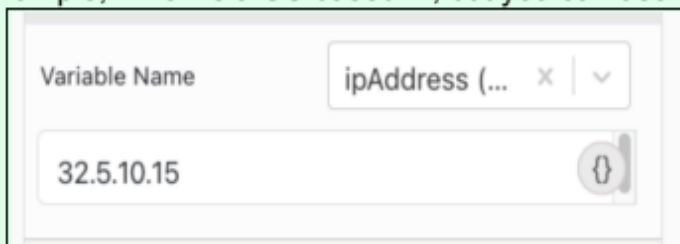
- 5 Don't forget to save your changes to the flow, if the save button is not greyed out.
- 6 Continue with the next section.

2.3 Testing flow in DaVinci

Now that the flow is built it is time to test it and see if it returns a result. You will not be able to see it in the browser as there is no UI component to this flow. You could of course build a wrapper test flow as you did before. But for this one you will use DaVinci Analytics to view the results of the flow and the data that is passed to the application. It was for this reason that you enabled debug logging at the start of this exercise.

Tip

In running through this test if your current IP address does not resolve you can set an IP address by setting it in the **Set Flow Variables** node setting **ipAddress**. You can use `32.5.10.15` for example, which is a US based IP, but you can use any that you know is valid.



- If you then want a dynamic value to be used during the application execution you will need to reset this variable back to a blank value.

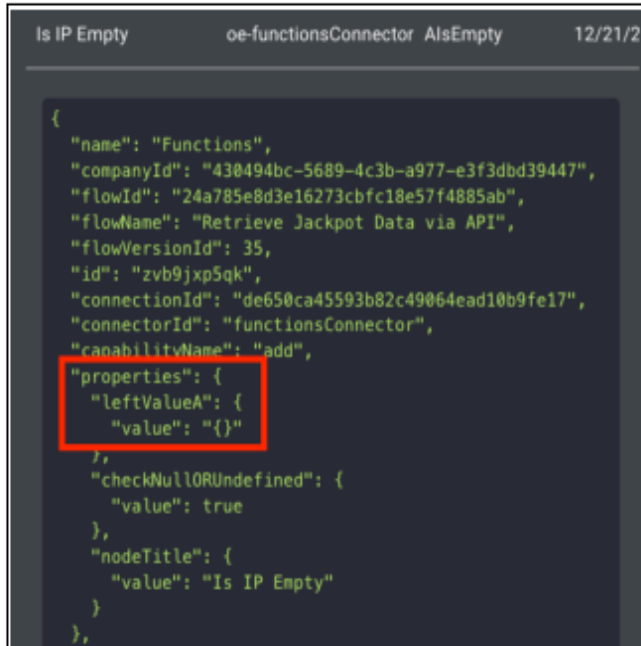
Note

In testing this flow if you don't get to the expected result you will need to use the Analytics to help debug the flow. Generally it will be selecting the wrong value for variables or typos in the variable name.

Once you find and fix the issue in your flow then run the test again. If needed reach out to the instructor for assistance.

- 1 Save and deploy your flow and then click **Try Flow** button.
- 2 When the flow completes you will get a message of **This screen type is not supported** in the browser tab where the flow executed.
 - 2.1 If instead you get an **Error** message *ipLookup error* as the response.
 - 2.2 The most likely cause is that the **Is IP Empty** node did not detect an empty value in the **ipAddress** variable.

2.3 Looking at the Analytics it shows that the variable value was an object with {} in it, see screenshot.

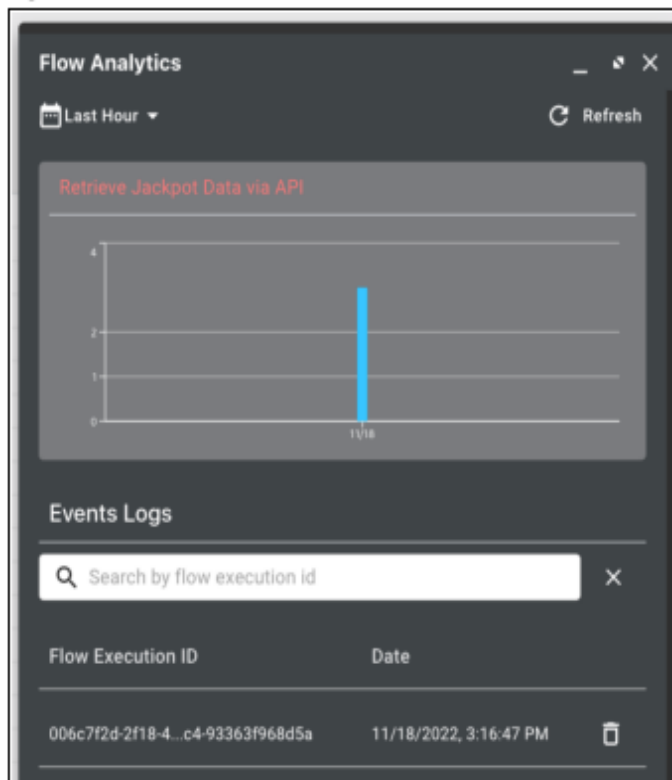


2.4 To resolve this go to the **Set Flow Variables** node set the value to some random value and then delete it.

2.5 Run the test again after saving and deploying your flow.

3 You can **close** the browser tab where your flow was executed, you will not need it open.

4 In your DaVinci tab where you have the flow open, click **Analytics** button in the bottom left of the flow.



- This will open a dialog to the right of your flow providing a list of the current flow executions for this flow.
- The first one in the list is the most current event execution, this is the one you are interested in at this time.

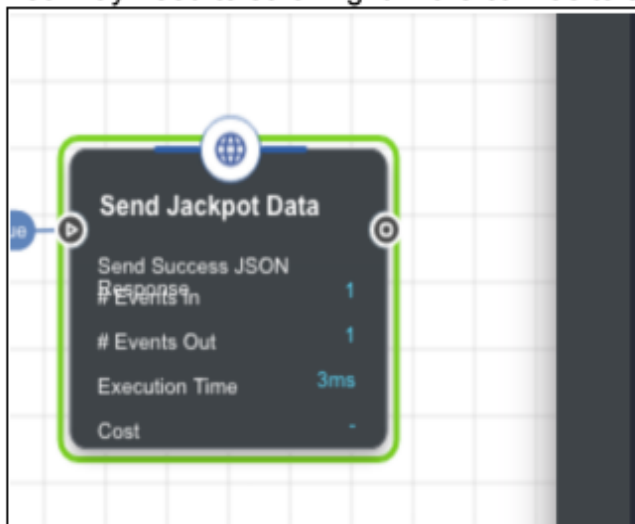
- 5 Click on the **first** event entry at the top of the list, you probably only have one unless you happened to do more than one execution.



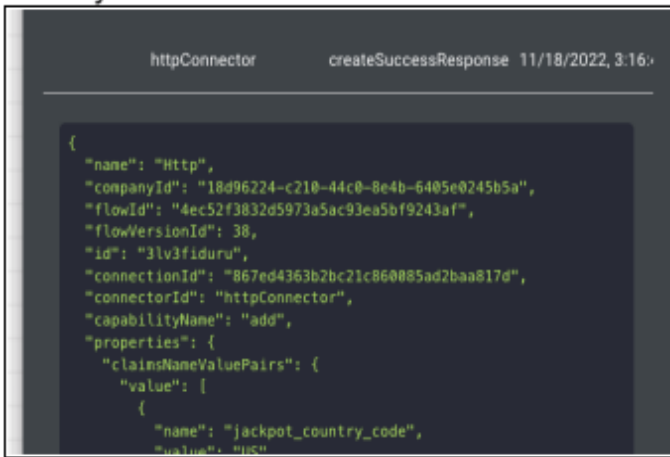
- When you select the event you will see the details of the flow and in the flow canvas the nodes will turn dark like the dialog; these are the nodes that were executed in the this execution.
- You can scroll down to and select the node by name and this will show in the canvas as the node you selected.
 - Or you can select the node on the canvas, which will scroll down the list to the first entry.
- There will be two entries for each node the request and the response, typically you are interested in the second one; the response.
- The entries labeled *oe* are for the orchestration engine, in general you are not interested in these events.

- 6 **Scroll** down the list of events and select the second entry for `httpConnector createSuccessResponse` this is your **Send Jackpot Data** node.

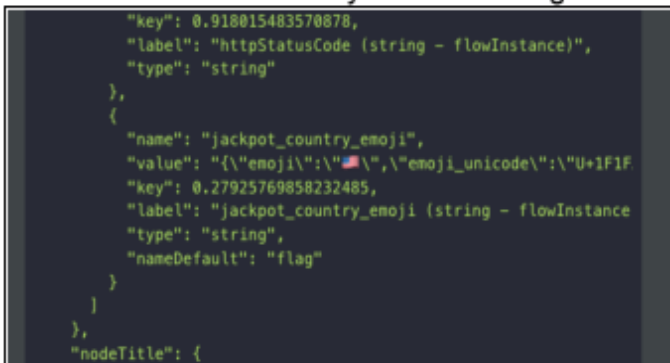
- You may need to scroll right in the canvas to see it, if you want.



7 When you click on it the details of the event will display under it.



8 Scroll down further until you see the flag data returned for the variable **jackpot_country_emoji**



8.1 Note that the **value** is a JSON object and that it contains an attribute called **emoji** this is the value that you want to display in the application.

8.2 You will address this in the following section.

9 Click the **X** at the top of the **Flow Analytics** dialog to close it.

10 Continue with the next section.

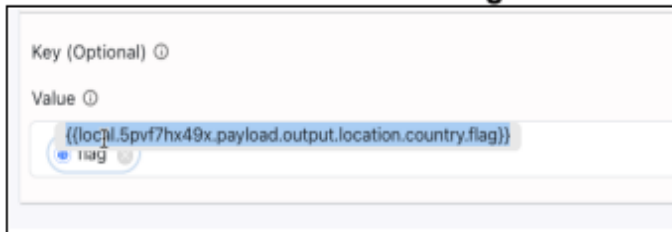
2.3.1 Adjust the emoji variable to pull the proper attribute

In reviewing the response sent by the flow in the previous section you identified the attribute you need from the flag object provided by the IP Registry call. In this section you will adjust your flow to capture this value, this will require looking at the expression generated internally, by DaVinci runtime, to reference the node and adjusting this expression.

1 Click **Send Jackpot Data** node to open the dialog, you will want to open the dialog to fill the browser:



2 Put the **mouse cursor** over the **flag** variable in the **jackpot_country_emoji** value.



- This will popup a value that shows the internal expression to reference this variable from the node.
 - The random value after `{{local.}` is the unique ID assigned to the node in your flow.
 - You can view these values in the canvas by selecting **Show Node ID** option in the flow configuration menu available from the top right.
- 3 Highlight the variable expression and **copy** it using the keyboard command.
 - 4 Paste this expression into a text editor to edit it as follows
 - 4.1 After **flag** and before the first **}** add the text **.emoji**
 - 4.2 Your expression should look something like the following, of course the node ID random value will be different.
 - `{{local.5pvf7hx49x.payload.output.location.country.flag.emoji}}`
 - 4.3 **Copy** this entire string from your editor.
 - 5 In the DaVinci flow delete the **flag** variable from **jackpot_country_emoji**
 - 6 **Paste** in the new value from your text editor, looking something like the following screenshot



- 7 Apply and close the dialog.
- 8 Continue with the next section.

2.3.2 Run a final test to verify emoji returned as expected

You adjusted your mapping in the previous section to write an expression to extract the emoji from the flag object returned by IP Registry service. Time to run one final test before adding this flow to your application..

- 1 Save, deploy and try your flow.
- 2 Once the **This screen type is not supported** appears in your browser test tab, **close** the tab.
- 3 Select **Analytics** in the DaVinci canvas for your flow.
- 4 Click on the **first** event entry at the top of the list, you probably only have one unless you happened to do more than one execution.
- 5 **Scroll** down the list of events and select the second entry for `httpConnector createSuccessResponse` this is your **Send Jackpot Data** node.

- 6 **Scroll** down in the response until you see the **jackpot_country_emoji**, there will be more than one listing but it should just have the emoji now:

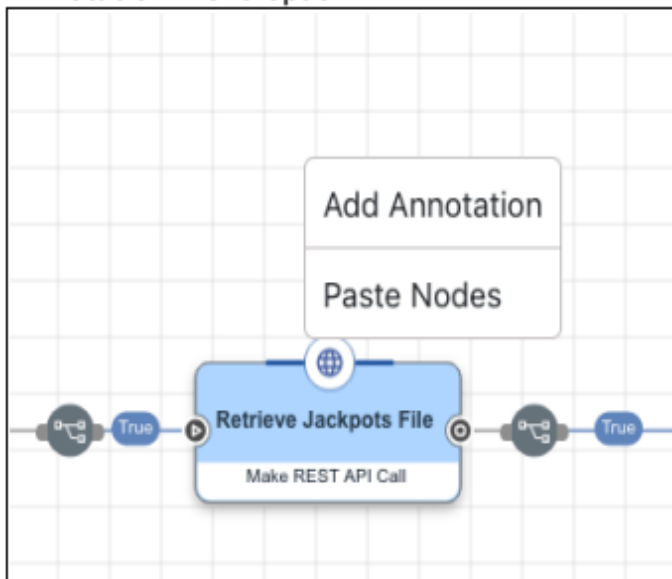
```
"success": true,
"parameters": {},
"additionalProperties": {
  "jackpot_country_code": "US",
  "jackpot_currency": "$",
  "jackpot_amount": 95000,
  "httpStatusCode": 200,
  "jackpot_country_emoji": "🇺🇸"
}
```

- The example we used earlier showed how it was evaluated in pulling the value from the node.
 - The above screenshot and shows what is returned as the response from the flow to your application.
- 7 Continue with the next section.

2.3.3 Document your code with some annotations

It is always good idea to document your flow to provide some overview of what the flow is doing in particular sections of your flow.

- 1 In your canvas somewhere above the **Retrieve Jackpots File** right mouse click then select **Add Annotation** menu option.



- 2 Click **annotation** node to configure as follows:

2.1 Annotation Text type

- **Retrieve Jackpot Data**

2.2 For Background Color select blue

2.3 In the **Width** field enter **1000**

Annotation Text
Retrieve Jackpot Data

Background Color
[Blue swatch]

Text Color
[Black swatch]

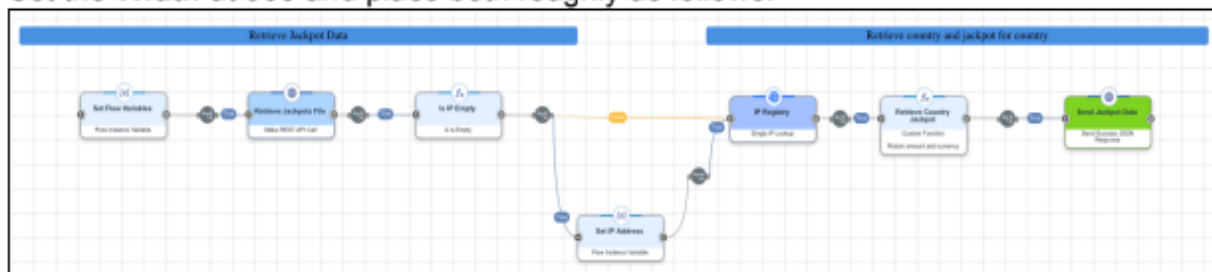
Width
1000

Height
[Empty field]

3 Copy the existing **annotation** and place it over the **Retrieve Country Jackpot** with the following text:

- **Retrieve country data and jackpot for the country**

4 Set the **Width** at **900** and place both roughly as follows:



5 Save and deploy your flow to have the most recent version for you application.

6 Continue with the next section.

2.4 Integrate flow into application

Now that you have your flow working and returning the required data it is time to integrate this into your application. The integration will involve adding some HTML to display the data along with the required JavaScript to call the flow. As with the other exercises the actual code will be provided for copy and paste. You will review the code that is doing the heavy lifting of calling DaVinci, it is already written.

You will be placing the output from the flow under the Register button on the application page.

Welcome to Beginner's Luck!

First visit, please click the **Register Now** below your free account

Returned? Great to see you back! Please click the Sign In button above

Register

1 In the **Glitch editor** open the file **index.html** to begin the edits of your application.

- 2 **Scroll** down until you find the button tag with an **id** of **btn-register** and a line after the closing button tag.

```

35- <button id="btn-register" class="margin-top">
36-   <i class="fa fa-user-plus" aria-hidden="true"></i> Register Now
37- </button>
38-
39- </td>

```

- 3 **Paste** following text at the line you just placed under the register button:

```
<div id="jackpot"></div>
```

- o This will be where the values returned by the flow will be placed.

```

35- <button id="btn-register" cl
36-   <i class="fa fa-user-plus"
37- </button>
38-   <div id="jackpot"></div>
39-
40- </td>

```

- 4 The code to make the API call to invoke the flow is in the **server.js** file, **open** it to review. This code is executed server side.

- 4.1 You will find a block of code with the following text:

- `app.post("/fetchJackpotData"`

```

36 // example of calling DaVinci flow as API.
37 // Calls the flow and then returns the returned data back to calling page
38- app.post("/fetchJackpotData", function (request, response) {
39-   const apiURL =

```

- 4.2 This code defines an endpoint in your application that can be invoked to fire off a call to DaVinci to invoke a flow.

- 4.3 This is similar to the `fetchDaVinciToken` endpoint that is used by the `loadwidget` function reviewed in an earlier exercise.

- 4.4 The code uses variables for the DaVinci API Endpoint and Company ID values needed.

```

39   const apiURL =
40     process.env.DAVINCI_API_URL +
41     "v1/company/" +
42     process.env.DAVINCI_COMPANY_ID +
43     "/policy/" +

```

- 4.5 The `request.body.policyId` is going to pull the DaVinci policy ID from the request when the endpoint is called.

```

43     "/policy/" +
44     request.body.policyId +
45     "/start";
46

```

- 5 A little bit further down you will see a fetch block.

```

57
58 /*** Call Jackpot flow to receive country specific jackpot ***/
59 fetch(apiURL, requestOptions)
60   .then((response) => response.text())
61   .then((result) => {
62     var jsonData = JSON.parse(result);
63     if (
64       jsonData.additionalProperties 56
65       jsonData.additionalProperties.httpStatusCode === 200
66     ) {
67       result = jsonData.additionalProperties;
68       console.log(result);
69       response.send(result);
70     } else {
71       console.log("Error retrieving jackpot data!");
72     }
73   })
74   .catch((error) => console.log("error", error));
75 });

```

- 5.1 This is what makes the actual call to DaVinci.

5.2 It returns back the additional properties from the flow, which is the JSON data needed to display on the page.

```
67     result = jsonData.additionalProperties;
68     console.log(result);
69     response.send(result);
70 } else {
```

6 You will be adding the call to the endpoint later in this exercise.

7 Continue with the next section.

2.4.1 Deal with loading the jackpot data

Now it is time to deal with the code that will invoke the method that calls your flow to load the jackpot data on your page.

1 In the **index.html** file find the **body** tag and in the **onload** attribute you will add **loadJackpotData()**, should a comma after the **initialize()** call, as follows:

```
</head>

<body onload="initialize(), loadJackpotData()">
  <div class="top-bar">
    <table class="content-holder">
```

2 At the bottom of **script** tag with all your functions you will add new function as follows:

```
function loadJackpotData() {
  //load jackpot data
  const policyId = "";

  const data = { policyId: policyId };

  console.log(JSON.stringify(data));

  fetch("/fetchJackpotData", {
    method: "POST",
    body: JSON.stringify(data),
    headers: { "Content-Type": "application/json" },
  })
    .then((response) => response.json())
    .then((data) => {
      // get the response from the server POST request
      console.log(JSON.stringify(data));
      loadJackpotSuccess(data);
    });
}
```

```
113- function loadJackpotData() {
114-   //load jackpot data
115-   const policyId = "";
116-
117-   const data = { policyId: policyId };
118-
119-   console.log(JSON.stringify(data));
120-
121-   fetch("/fetchJackpotData", {
122-     method: "POST",
123-     body: JSON.stringify(data),
124-     headers: { "Content-Type": "application/json" },
125-   })
126-     .then((response) => response.json())
127-     .then((data) => {
128-       // get the response from the server POST request
129-       console.log(JSON.stringify(data));
130-       loadJackpotSuccess(data);
131-     });
132- }
```

- This call passes in the policy ID and the callback function, essentially invoking the flow..

- You will deal with the call back function later.
- You can use the PRETTIER option at the top of the Glitch editor to have it reformat your code.

3 The `fetch("/fetchJackpotData", {` calls the endpoint in your application passing it the policy ID.

```
120
121-   fetch("/fetchJackpotData", {
122-     method: "POST",
123-     body: JSON.stringify(data),
124-     headers: { "Content-Type": "application/json" },
125-   })
126-   .then((response) => response.json())
127-   .then((data) => {
128-     // get the response from the server POST request
129-     console.log(JSON.stringify(data));
130-     loadJackpotSuccess(data);
131-   });
```

4 At the end it calls the `loadJackpotSuccess` with the data returned from the flow.

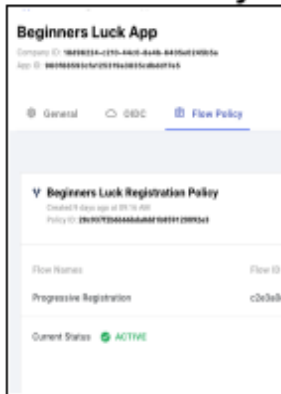
- You will deal with this function in the following sections.

5 Continue with the next section.

2.4.1.1 Define a policy flow in the DaVinci application

As you saw in the previous section you need a flow policy ID to invoke your flow. You have not defined this yet and will now. You will do this in the application you defined in DaVinci before you used for the registration flow. An application can be used to invoke multiple flows and typically you will only need to define one that is linked to your application that is integrated with DaVinci.

- 1 In DaVinci **console** select **Applications** from the left menu.
- 2 Click **Edit** button in **Beginners Luck App** to open the configuration.
- 3 Click **Flow Policy** tab



- 4 Click **Add Flow Policy** button to the left of the page.
- 5 In the **Name** field type
 - **Jackpot API Policy**
- 6 Select **Retrieve Jackpot Data via API** flow from the list

6.1 Click **Latest Version** checkbox.

6.2 Click **Create Flow Policy** button.

7 Click **Save Flow Policy** button.

8 **Copy** the **Policy ID** value for the new policy you just created.

9 In the `loadJackpotData` you defined in the `index.html` code in the Glitch editor **paste** the value.

```

112
113- function loadJackpotData() {
114-   //load jackpot data
115-   const policyId = "28[REDACTED]f4";
116-
117-   const data = { policyId: policyId };
118- }

```

- It needs to be inside the quotes
- This will invoke your new flow.

10 You can close the application in the DaVinci browser tab.

11 Continue with the next section.

2.4.2 Handle data returned from flow

The last bit of code that is needed is handling the actual data that is returned by the flow and then presenting this on the page.

In this section you will first setup the function to simply display the data in a JavaScript alert, allowing you to verify the call is made. You will then be add the required HTML to display it on the page.

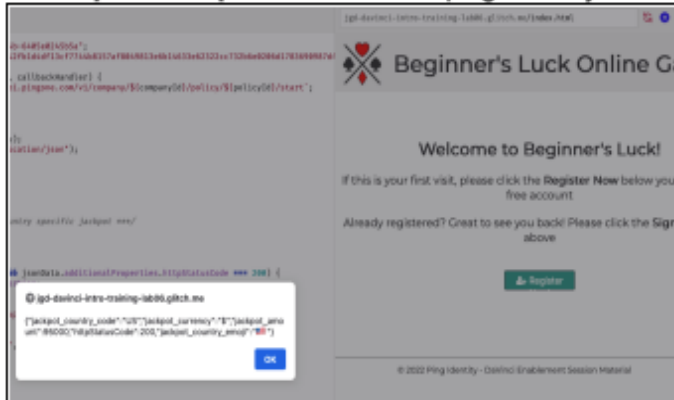
You will be able to view the code by using the preview pane in the Glitch editor, allowing you to test it without having to invoke the entire application. page in a separate browser tab.

- 1 **Copy** the following function definition and place it after the **loadJackpotData** function you placed in the **script** block of your **index.html** file.

```
function loadJackpotSuccess(data) {
  alert(JSON.stringify(data));
}
```

```
134~ function loadJackpotSuccess(data) {
135~   alert(JSON.stringify(data));
136~ }
137~ </script>
```

- 2 In the **preview pane** reload the page and you should get an alert popup



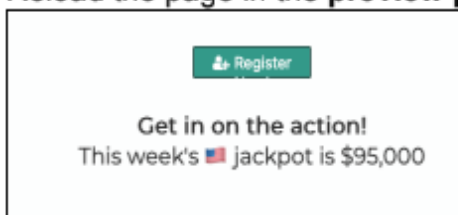
- This indicates that your flow was executed and the data was returned to your application.
 - The data is displayed in the alert popup.
- 3 Now it is time to add HTML code to the function to display the data:
 - 3.1 In the **index.html** comment out the **alert** in the function by adding **//** in front of it
 - 3.2 **Copy** the following HTML code and **paste** it after the commented out line for the alert:

```
const element = document.getElementById('jackpot')
element.innerHTML = `
  <p class="jackpot"><strong>Get in on the action!</strong><br/>
  This week's ${data.jackpot_country_emoji} jackpot is $
  ${data.jackpot_currency}${data.jackpot_amount.toLocaleString()}</p>`
```

- This references the div tag you placed under the register button earlier.
- It extracts the data from the request using variable references.

```
133~ function loadJackpotSuccess(data) {
134~   // alert(JSON.stringify(data));
135~   const element = document.getElementById("jackpot");
136~   element.innerHTML = `
137~     <p class="jackpot"><strong>Get in on the action!</strong><br/>
138~     This week's ${data.jackpot_country_emoji} jackpot is $
139~     ${data.jackpot_currency}
140~     ${data.jackpot_amount.toLocaleString()}</p>`;
141~   }
142~ }
143~ </script>
```

- 4 Reload the page in the **preview pane** to see the result:



- The country of course will depend on your IP address.
- 5 You will want to turn off the **preview pane** in Glitch so that your flow is not being called all the time the editor refreshes.

- 6 If you did hardcode an **ipAddress** at the start of your flow, you may want to remove that now. Or leave it, up to you.
- 7 You have completed this exercise.