UNQORK

No-code application development platform
that supports organizations in finance, insurance,
healthcare, and government.

Role

Technical + UX Writer

Responsibilities

Documentation for the Platform Curriculum Team

Internal documentation of software capabilities,
features, and API functions.

UX and Technical documentation for the Help Center and
public Learning Paths.

API Best Practices

Introduction

You covered many API-related topics in previous courses including the basic functions of API calls. You also learned about predefined Internal API calls at Unqork, such as how to Create and Update Submissions.

The possible uses for API calls extend much farther than creating or updating submissions in the Unqork Designer Platform. Configurators use API calls to process data in many ways. Remote executes keep data safe, keep proprietary processes hidden, and improve speed and security.

Endless opportunities present themselves when an application includes remote executes. Follow these API best practices to learn how to:

- Organize configurations.
- Standardize configurations so others can understand the purpose of the API call.
- Enable use of the Unqork Development Life Cycle Toolkit.

UNQORK

Best Practices Example: APIs

What You'll Learn

The following lesson teaches you how to:

- Create an API module.
- Use the API Specification snippet.
- Test your API call.
- Use the API Docs Dashboard tool.
- Use the Config Analysis Dashboard tool.
- Handle errors in API calls.

Creating Your API Module

Create a new module to build out your API. Keep in mind these best practices as you build:

- Include a notation in the title indicating that this is an API module. A common convention is **APPLICATION NAME: MODULE NAME (API)**.
- Add the **API** Field Tag to your module.
- Open the Settings for your module and switch on **Server Side Execution Only**. Save the updated settings.

UNQORK

Best Practices Example: APIs

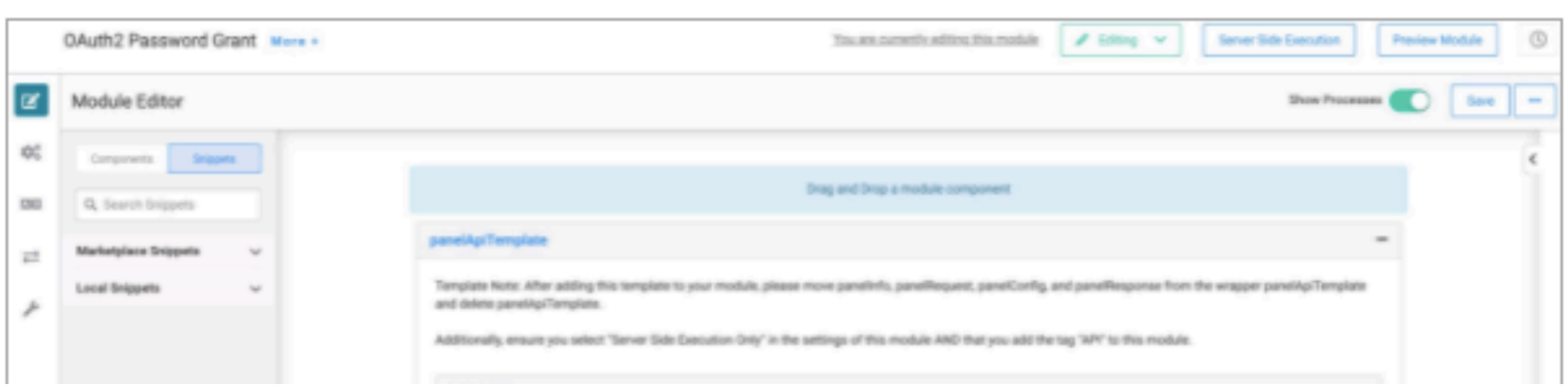
Using the API Specification Snippet

The API Specification snippet simplifies building an API in Unqork. It offers a standardized layout for modules that execute as APIs. Your API module stores in the API Docs Dashboard when you use the framework of this snippet. The Unqork Development Life Cycle Toolkit includes the API Docs Dashboard, which lists the API modules in your current environment and contains documentation for each API module. This documentation includes a description, the request parameters, and the response parameters.

To add the API Specification snippet to your module:

1. Click the **Snippets** button in left sidebar of Module Editor.
2. Enter **API Specification** in the search bar.
3. Drag and drop the **API Specification** snippet onto your canvas.
4. Remove the **panelInfo**, **panelRequest**, **panelConfig**, and **panelResponse** Panels from the **panelApiTemplate** Panel.
5. Delete the **panelApiTemplate** Panel.
6. **Save** your module.

The API Specification snippet looks like this in the Module Editor:



Testing the API Call

It is imperative that you test your configurations when you build in the Unqork Designer Platform, including when you build APIs.

Follow these steps to test your API call:

1. Test your application in Express View to see how the end-user experiences the front end of your application.
2. Use the Server Side Execution tool to test your API. The Server Side Execution tool is the best course of action to test an API since APIs run on the server.

   NOTE Learn more about the Server Side Execution tool: https://academy.unqork.com/professional-integration-troubleshooting/753826.

3. Run your API module as a remote execute, and reach your endpoint from wherever your application should call the API module. This ensures your API call works as expected.

Using the Life Cycle Toolkit

The UDLC Toolkit (Unqork Development Life Cycle Toolkit) helps you organize and visualize your projects. Use the API Specification snippet in your API module for access to extra tools in the UDLC Toolkit. The success of these tools depends on following consistent best practices in your modules.

NOTE The Professional Learning Path covers some of these tools. Learn more about these solutions tools here: https://academy.unqork.com/solutions-tools.

UNQORK

Best Practices Example: APIs

Config Analysis Dashboard Tool

Also take note of the Config Analysis Dashboard tool, which analyzes best practices across a variety of metrics in the Unqork Designer Platform.

(missing or bad snippet)

This tool lets you:

- View a detailed list of the configuration analysis requests submitted in the current environment.
- Submit new configuration analysis requests.
- Upload a configuration analysis file.
- View existing config analysis data.
- Update violation statuses to see which violations are in review.

A Filters Panel displays when you use the Config Analysis Dashboard tool. Notice the two parameters dedicated to API best practices when you filter for tests:

- **getApiSpecViolations** identifies the modules that execute as APIs but violate API specifications at Unqork.
- **getApiParamViolations** identifies request or response parameters in remote execute modules that violate API specifications at Unqork.

Handling Errors in an API Call

Mistakes happen in configurations despite best efforts! Practice configuring complex applications and troubleshooting errors to minimize mistakes over time. Issues still occur, and these tips for troubleshooting API calls and API best practices make error handling as easy as possible when they occur.

Errors Handling for Plug-in Calls

Every Plug-in component requires error triggers to handle configuration issues. These triggers alert you when a specific Plug-in breaks. The trigger sends you an error code associated with all API errors. You determine the issue from the code and accompanying message, which helps you fix the specific problem instead of blindly troubleshooting API calls.

Error Handling and Troubleshooting

More information about error handling can be found in the Expert Integration Troubleshooting (https://academy.unqork.com/expert-integration-troubleshooting) and Introduction to Error Handling (https://academy.unqork.com/introduction-to-error-handling) courses through Unqork Academy.

Summary

This lesson brings together the API best practices at Unqork. Review what you learned by answering these questions:

1. Which Trigger Type do you avoid when you configure Data and Event Processing components in your API module?
2. How do you test an API?
3. Which tools keep your APIs organized?

Setting Up a SOAP Call

Introduction

In the previous lesson, you learned that SOAP APIs only work with XML (eXtensible Markup Language). They also require proper syntax to function successfully. In this lesson, you'll learn the proper syntax for SOAP API calls and the proper syntax for XML.

What You'll Learn

In this lesson, you'll learn:

- Syntax of a SOAP call.
- Syntax of XML.

Syntax of a SOAP Call

SOAP calls must be encoded with XML. To function as intended, a SOAP call requires an Envelope, Header, and Body.

1. **Envelope** defines the XML document as a SOAP call.
2. **Header** displays information for the API call.
3. **Body** contains the bulk of the data for the call.

UNQORK

Conceptual Example: Setting Up SOAP API Calls

Syntax of XML

The earlier lesson on transforms outlines the proper syntax for an XML document. To reiterate the rules discussed in that lesson:

- **Prologue** is optional, but if it's used then it must be written first: **<?xml version="1.0" encoding="UTF-8"?>**
- **Tags** are the same as keys in a JSON structure.
- **Values** must be stored with an opening tag and a closing tag: **<tag>value</tag>**
- **Values** in an object or array must be properly nested: **<object><tag>value</tag></object>**
- XML **Elements** can have attributes, similar to HTML: **<element attribute=”value”></element>**
- Specific **Characters** cannot be used and instead require entity reference:

Name of Character	Symbol	XML Entity Reference
Less Than	<	>
Greater Than	>	>
Ampersand	&	&
Apostrophe	'	'
Quotation Mark	"	"

UNQORK

Conceptual Example: Setting Up SOAP API Calls

Here's an example of a SOAP call that uses XML:

XML Syntax	Description
<?xml version="1.0"?>	Optional Prologue
<soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelope">	Opening of Require Envelope
<soap:Header>	Opening of Required Header
{{Header Data Goes Here}}	Header Data
</soap:Header>	Closing Header
<soap:Body>	Opening of Required Body
{{Data Goes Here}}	Body Data
</soap:Body>	Closing Body
<soap:Envelope>	Closing Envelope

UNQORK

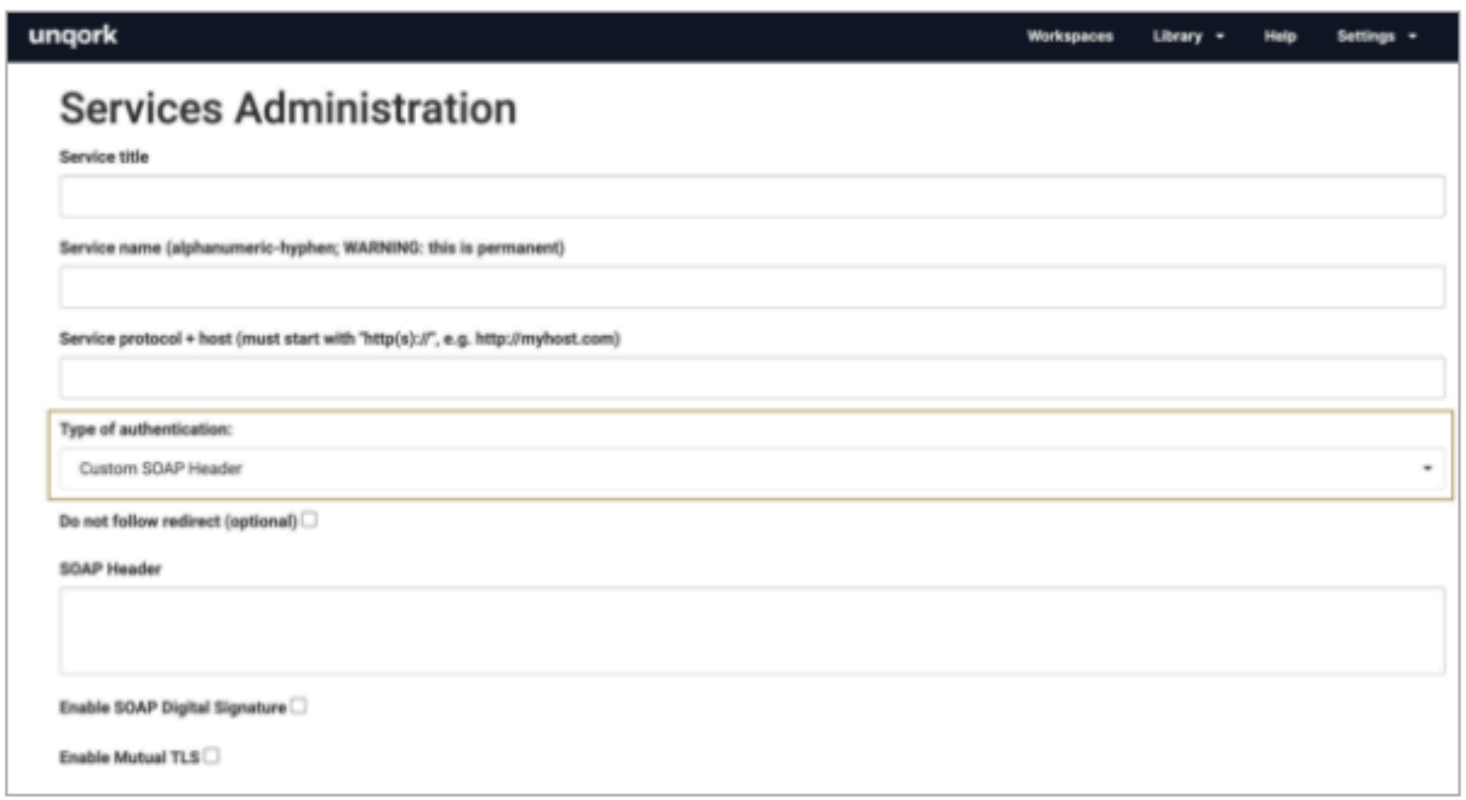
Conceptual Example: Setting Up SOAP API Calls

Setting Up Your XML Integration

To set up your integration, you'll begin the same way you set up any external service. Open up Services Administration from the Administration page.

Here's how to set up your XML integration:

1. Enter your information in the **Service title**, **Service name**, and **Service protocol + host** fields.
2. Select **Custom SOAP Header** from the **Type of Authentication** drop-down. The SOAP Header text box displays.
3. Fill in your SOAP header in the **SOAP Header** text box if necessary.



What you enter for your SOAP header depends on the call you make. By entering a SOAP header, you can keep the header in a single place. That means you can use the same header for multiple calls. The downside to the Service Administration SOAP Header is that it can only hold static values. For dynamic or changing values, you can enter your SOAP header in your transform body.

Use Case: Building Out Modal Actions

Introduction

Now that you know about modal pop-ups in a ViewGrid, let's walk through an example. Before you set up the dashboard, you'll create a Transform with Input Data to generate sample content. Then, you'll configure a panel, a button-click event, and a Data Workflow to display the sample data in the modal pop-up.

What You'll Learn

In this use case, you'll build a modal pop-up to display additional information in a ViewGrid.

How This Use Case Works

In this use case, you set up a Transform with sample data to generate the content for your Dashboard submissions. Then, you configure a panel to open as a modal and a button-click event to trigger the pop-up. Lastly, you map the outputs and set up a Data Workflow to display the sample data in the modal pop-up. To complete this use case, you must be familiar with configuring Hidden components, Initializer components, and Plug-in components. If needed, revisit the Introduction to Data and Events Processing and Introduction to APIs courses for a refresher.

UNQORK

Use Case Example: Building Out Modal Actions

Here are the basic steps of how this use case works:

1. A Transform with Input Data generates your sample data content.
2. The ViewGrid pulls and displays data from your Data Table.
3. A panel includes text fields for the end-user to input additional data.
4. Your panel is set to display as a modal with the additional data.
5. A rule connects a button-click event on your ViewGrid to the modal pop-up.
6. The modal opens with the additional data when the button is clicked.
7. The outputs are mapped in the ViewGrid, which populates the text fields in the modal pop-up.
8. A Data Workflow connects the collected address data and displays the information in the modal pop-up.

Here's how the completed module looks in the Module Editor:

UNQORK

Use Case Example: Building Out Modal Actions

What You'll Need

To set up this use case, you'll need:

- 1 Initializer component
- 1 Plug-In component
- 2 Hidden components
- 1 ViewGrid component
- 1 Decisions component
- 1 Panel component
- 5 Text Field components
- 1 Data Workflow component

To set up your Data Workflow, you'll need:

- 1 Input operator
- 3 Get operators
- 3 Output operators

UNQORK

Use Case Example: Building Out Modal Actions

Adding Sample Data

For this use case, you'll use a Transform with Input Data to generate your sample content. If you have questions about setting up a Transform, revisit the introductory lesson about [Transforms](#).

For this example, your Transform requires the following data:

```
{
  "customers":[
     {
        "firstName":"Arya",
        "lastName":"Stark",
        "Age":15
        "concerns":"White Walkers and being too small",
        "strengths":"Swords, sailing, and changing faces",
        "address":{
           "street":"123 Castle Ln",
           "city":"Winterfell",
           "state":"The North"
           }
       },
       {
        "firstName":"Hikaru",
        "lastName":"Sulu",
        "age":29,
        "concerns":"Terrorists, Thanos, Whiplash",
        "strengths":"Bravery and Navigation",
        "address":{
           "street":"The Helm",
           "city":"Main Deck",
           "state":"USS Enterprise"
           }
       },
       {
        "firstName":"Harry",
        "lastName":"Potter",
        "age":32,
        "concerns":"Ghost and the Supernatural",
        "strengths":"Courage and Magic",
        "address":{
           "street":"The cupboard under the Stairs",
           "city":"Little Whinging",
           "state":"Surrey"
           }
       },
       {
        "firstName":"Vivian",
        "lastName":"Banks",
        "age":40,
        "concerns":"Children's safety",
        "strengths":"Strong and Independent. ",
        "address":{
           "street":"251 N Bristol Ave",
           "city":"Los Angeles",
           "state":"California"
           }
       }
    ]
}
```

UNQORK

Use Case Example: Building Out Modal Actions

This serves as your sample submission data for the Dashboard. Now, you can manipulate how it looks visually in the Dashboard using a modal pop-up and Data Workflow.

After setting up your Transform, map the output in the **Outputs** table of your **Plug-In** component like this:

Property ID	Mapping	Option	Header
customers	jsonData.customers		

Displaying the Data

Now, let's set up your **ViewGrid** to display submission data in the dashboard. Your Transform data is the Input to your ViewGrid. Enter **customer** in the **id** column of the **Inputs** table.

Next, complete the **Display** table of your **ViewGrid** like this:

ID	Formula	Heading	Type	CSS	Button
firstName		First Name			
lastName		Last Name			
age		Age			

UNQORK

Use Case Example: Building Out Modal Actions

Finally, create a button to display on your Dashboard. Enter **Details** in the **Action** field of the ViewGrid. This button triggers the modal pop-up.

Configuring the Panel Component

Now let's add a **Panel** component. The panel stores information to display in your modal pop-up. Here are some important settings to note in this component:

Settings	Configuration
Display as a Modal	Yes (checked)
Dismiss Modal Event	closeDetails
Open Modal Event	openDetails

Then, configure the additional Text Field components that you want to display inside the panel. For this use case, drag and drop five Text Field components onto your canvas. Label your text fields like this:

Property ID	Label Text
concerns	Concerns
strengths	Strengths
street	Street
city	City
state	State

Configuring the Decision Component

Your ViewGrid button needs a rule connected to it so that the button-click event triggers the modal to open. Let's use a Decisions component to create this rule. Set the **Trigger Type** to **Watch**. Then, configure the **Inputs** table like this:

Property ID	Type
buttonClick	contains

Configure the **Outputs** table like this:

Property ID	Type
panelDetails	trigger

Configure the **Micro Decisions** table like this:

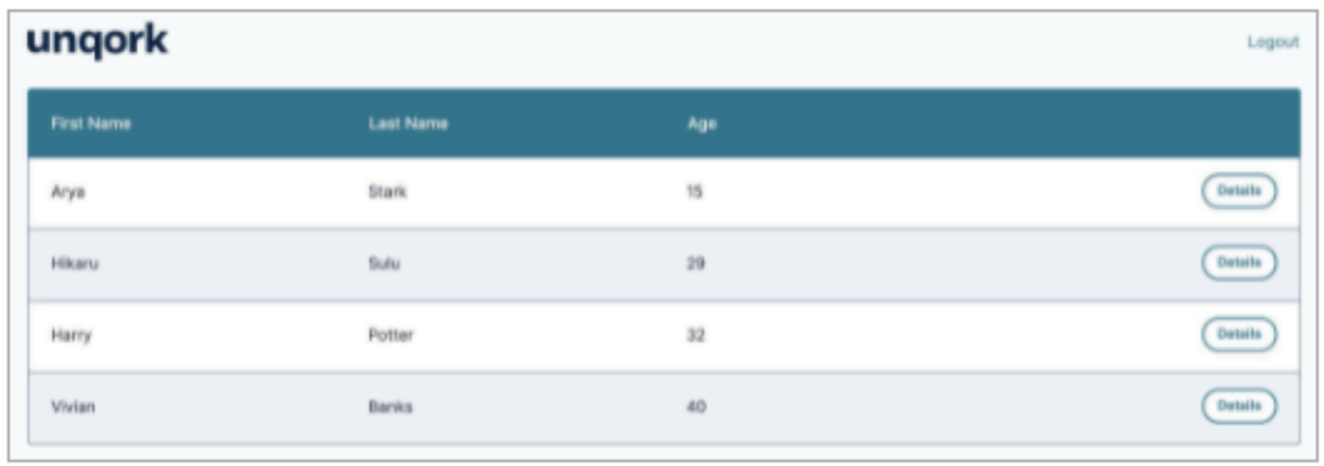
Input Values	Output Values
buttonClick	panelDetails_trigger
Details	openDetails

UNQORK

Use Case Example: Building Out Modal Actions

Save and Preview

Save the Decision component, save the module, and preview the module. Your Express View looks like this:



If you click the **Details** button, your modal opens and you'll see this:

The text fields are empty because you need to set up the mapping for your outputs. Let's go back into your ViewGrid and update the mapping in the **Outputs** table. The completed table looks like this:

ID	Mapping
concerns	concerns
strengths	strengths
address	address

Displaying the Dynamic Data

To include addresses in your modal pop-up, use a **Data Workflow** to auto-populate the modal with your end-user's submitted information. Before configuring your Data Workflow, make sure to add three **Hidden** components to store the output values.

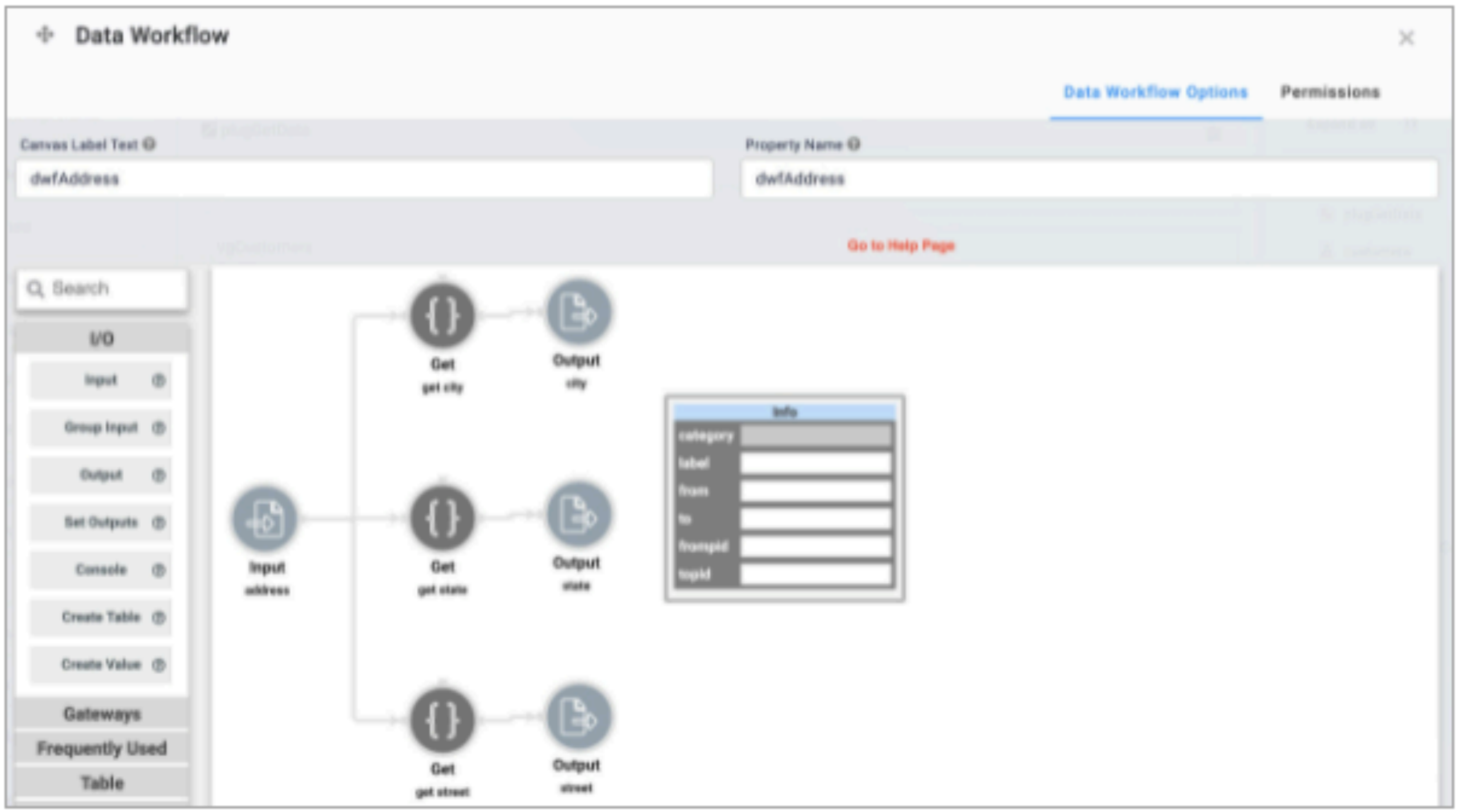
This Data Workflow requires three different types of operators. An **Input** operator brings in your original data. **Get** operators allow you to grab single data points from the large data set. Then, the Get operators pass that data through your Data Workflow. In this example, the Get operators pull the values for Street, City, and State entered by the end-user. These values display in the modal pop-up.

Finally, an **Output** operator connects to each Get operator. These operators save the new data to your three hidden components.

UNQORK

Use Case Example: Building Out Modal Actions

And here's how they look on your Data Workflow canvas:



Add your Data Workflow to the Outputs table in the **Decisions** component. Set the Type to **trigger**. Then, the Micro Decisions table auto-populates with the Data Workflow. In the column **Output Values**, enter GO.

UNQORK

Use Case Example: Building Out Modal Actions

Now, let's save and preview the module. Click the **Details** button again, and you see that all of the fields populate!

NOTE Modal pop-ups can create scope issues. This happens when the pop-up triggers more logic events like the data workflow in your use case. In this example, a data workflow inside of the modal pop-up doesn't execute properly. The data workflow must be outside the Panel component. A later course explores scope issues, but for now, let's keep it simple. If a logic component doesn't work, try to move the logic component into or out of the Panel component.

Lab

You can view this completed use case here: https://training.unqork.io/#/form/61390ed9e22f7649204c016a/edit

UNQORK

Technical Copywriting Example: University Partnership Vertical

Unqork is going to college!

This past year, Unqork began an exciting collaboration with universities to bring no-code onto campus. We went live in a classroom at **Kennesaw State University** and our course was met with enthusiasm and success.

Kennesaw State is a member of the University System of Georgia and is the second-largest university in the state. **Five of their professors completed our Train the Trainer (TTT) bootcamp**, and Unqork was incorporated into a course curriculum for the first time. Students learned how to use the platform to design solutions to real-life FinTech problems.

Now, we're looking forward to diversifying our work on campus. We're excited to team up with **Elavon**, a new corporate partner and the payment division of the **U.S. Bank.** Our thriving collaboration at Kennesaw State currently includes:

UNQORK

Technical Copywriting Example: University Partnership Vertical

- **Digital Payments Security Course**
  - 41 students registered with Unqork out of the 76 students enrolled in the course.
  - This course is connected with the far-reaching Georgia Fintech Academy, in collaboration with the University System of Georgia.

- **Experiential Learning in FinTech Course**
  - For this new course, Unqork co-partnered with **Elavon, the payment division of the U.S. Bank.**
  - Elavon creates use cases for students, solved using Unqork's platform.
  - The 42 students enrolled in the course will be registered with Unqork this semester.
  - **Seven Elavon employees, as well as two members of the U.S. Bank also registered with Unqork** in connection to this course.
  - This course is also connected with Georgia Fintech Academy.

- **Discussion of Expanded University Presence**
  - The **College of Computing and Software Engineering** is in the discovery phase of a partnership with Unqork.
  - The dean is interested in sending more professors through bootcamp.

UNQORK

Technical Copywriting Example: University Partnership Press Release

With Unqork at the center, **Emory University** is offering a senior seminar course about the disruptive potential of no-code platforms. Students pursuing a Bachelor of Business Administration degree have the opportunity to learn through experimentation, allowing them to conceptualize and then create enterprise-grade software solutions. With an **enrollment capacity of fifty students**, we're confident that this seminar is the start of a great relationship with Emory! Due to high demand and our expanding vision for classroom connections, the Unqork Customer Success team is creating scalable, repeatable processes to support this new university vertical as it evolves.

The reason for our enthusiasm around this project is clear: Unqork's college connections are exciting and mutually beneficial.

Democratizing enterprise-level application design through no-code allows a new generation of software designers to focus on experimentation and problem-solving. Their attention remains on finding creative, scalable solutions for clients, **all without writing a single line of code.**

UNQORK

Technical Copywriting Example: University Partnership Press Release