



Advanced Graph Building

The background of the slide is a dark blue gradient with a diagonal light blue stripe. Several dandelion seeds are scattered across the frame, some in sharp focus and others blurred, creating a sense of movement. The text "Training Overview" is centered in a white, sans-serif font.

Training Overview

Advanced Graph Building - Learning Outcomes

- ◀ Execution Layer Tasks and Roles
- ◀ Best practices in Kitewheel
- ◀ Testing
- ◀ Debugging
- ◀ Loop Node
- ◀ Exercises in Kitewheel
- ◀ Certification

Kitewheel Personas & Target Audience

Client

CHLOE



- Defines Business Requirements
- Defines KPIs, Goals and Metrics to track
- Tracks progress against goals on Journey Insights

Secondary

Strategy

SOPHIE



- Translates business requirements into Journeys Map
- Captures journey details, metrics and goals

Primary

Solution Design
&
Configuration

CHARLES



- Solution Design
- Identifies data sources
- Configures rules
- Creates outcomes
- Develop and test
- Deploy

Primary

Technical
&
Support

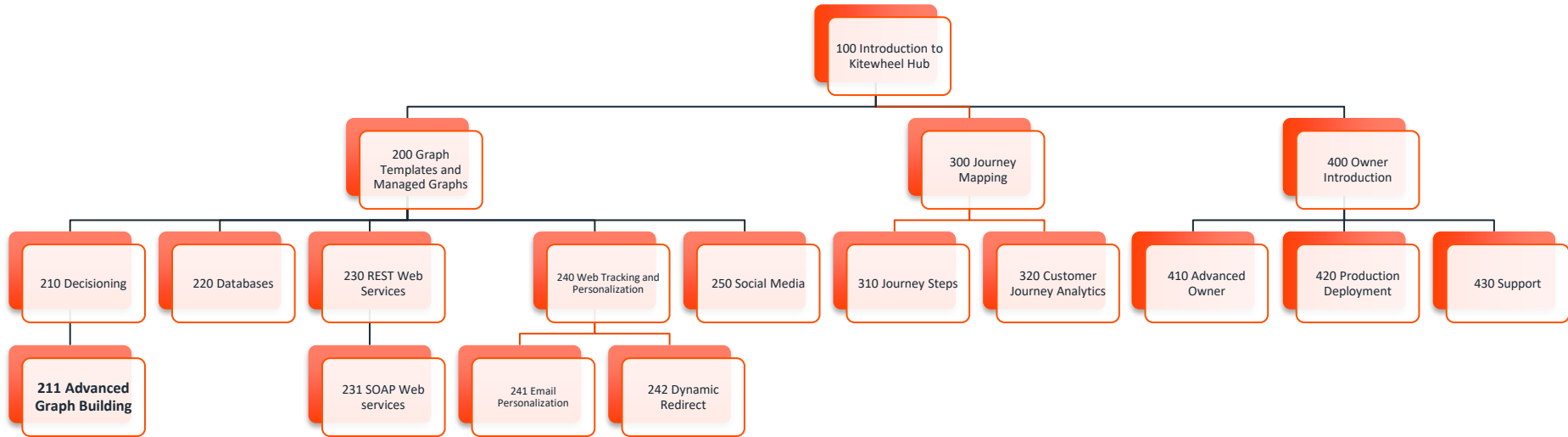
THOMAS



- Enable tech environment - servers, database
- Handle security and internet facing services
- Support accounts and projects

Secondary

Training Course Overview



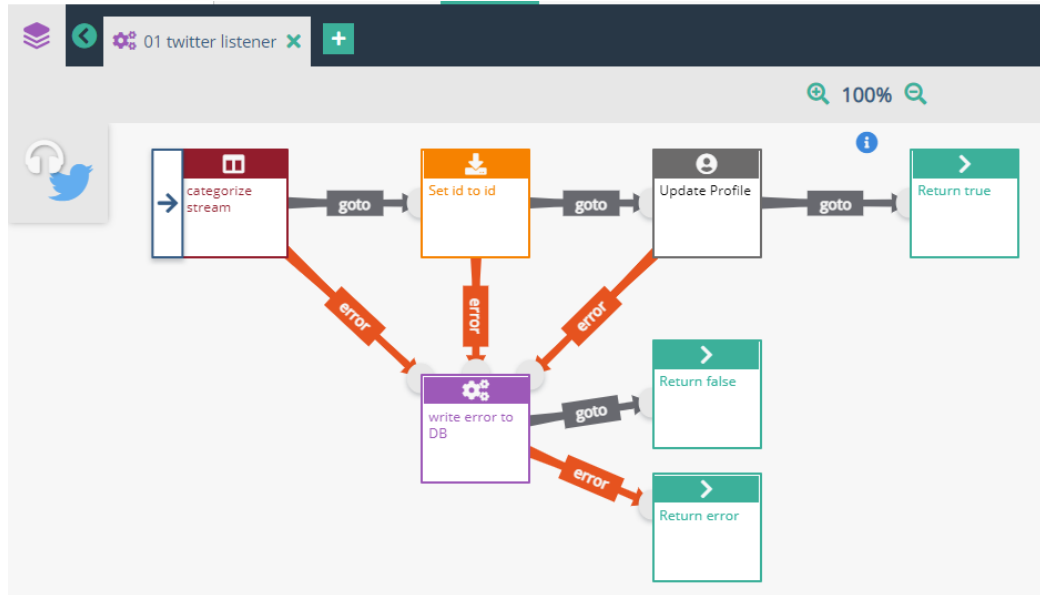
Execution Layer Tasks

◀ **Solution Implementation Design**

- ◀ Before starting a project or a graph, it is important to think about the solution design and how to integrate with existing systems and APIs
- ◀ The graph will have to integrate with the existing systems like customer databases, email service providers, APIs etc.
- ◀ Make a list of all the existing data sources and understand what data has to be accessed at what point in the solution to be more efficient

Execution Layer Tasks

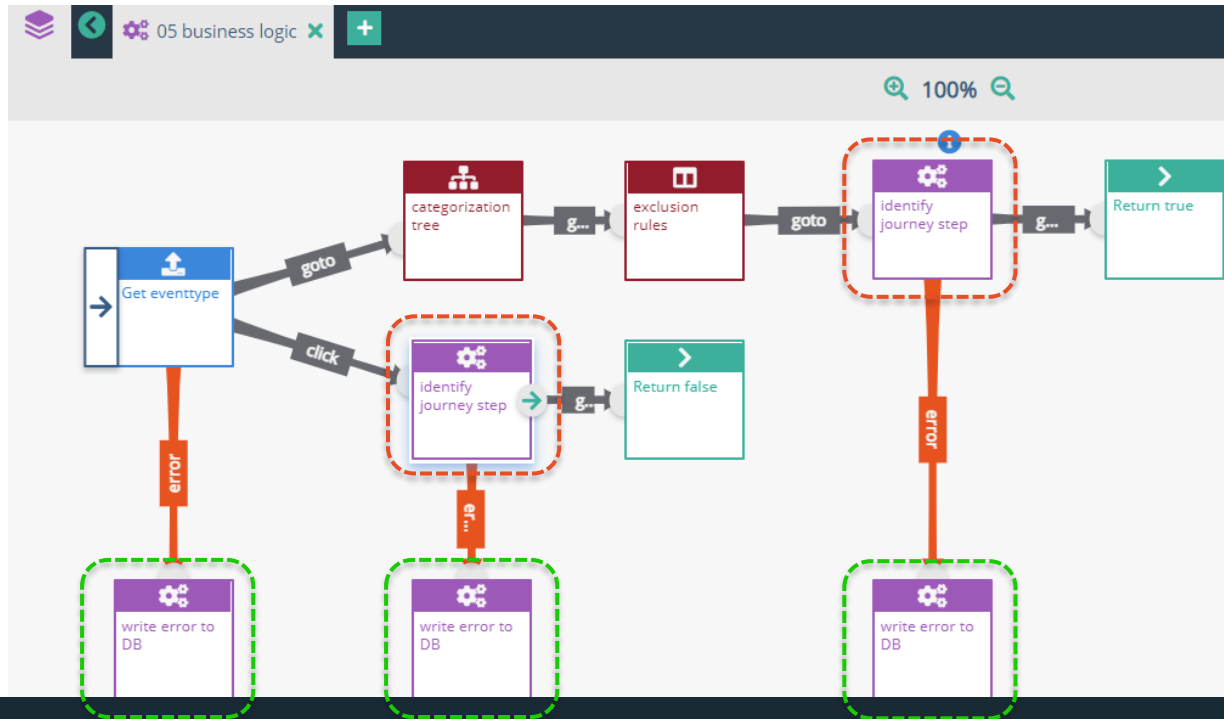
- ◀ As an example: In a Twitter graph, all the filters should be applied before reading the DB table to check if this is a new profile or an existing one.



Execution Layer Tasks

Reusable components

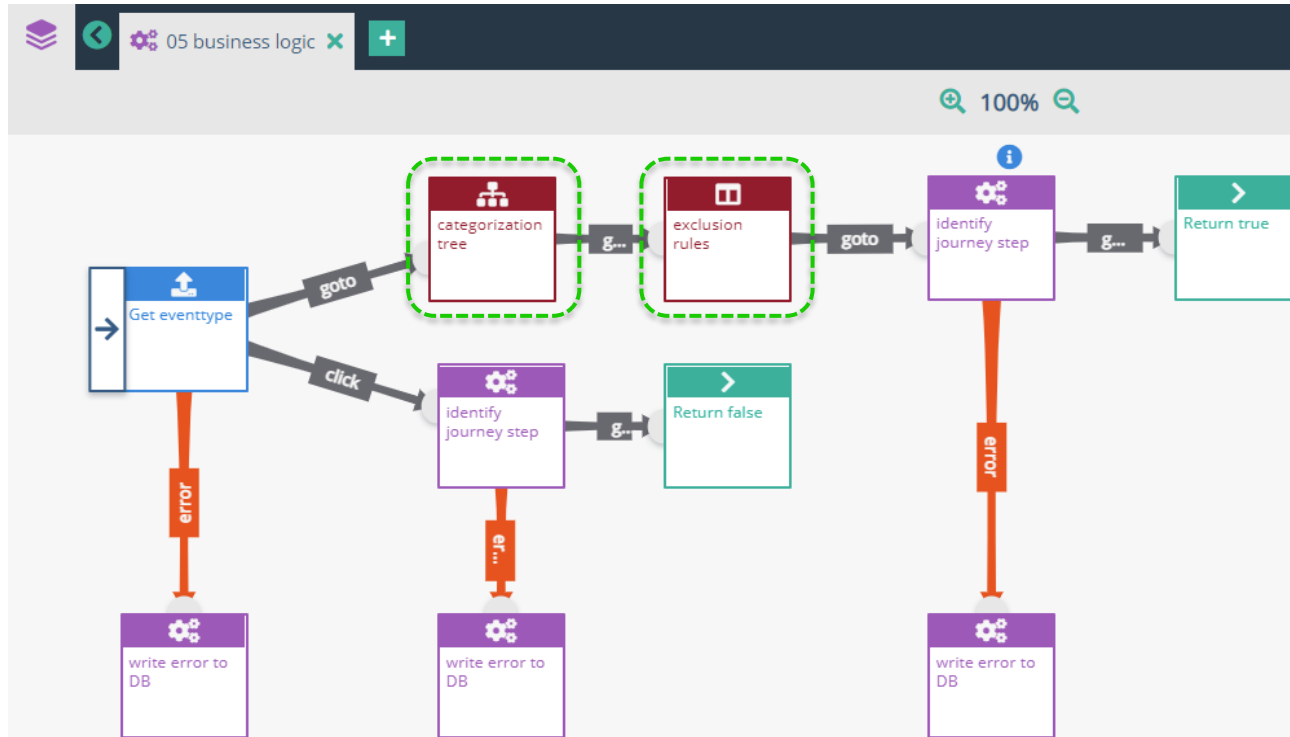
- Identify the bit of logic that can be made as generic as possible so that it can be reused in other graphs



Execution Layer Tasks

Custom components

- Identify any logic that is very specific and cannot be reused



Execution Layer Tasks

◀ Testing and Debugging

- ◀ Every component has to be tested on its own- Unit testing
- ◀ Then test with other nodes in the graph- Integration testing

◀ Production Support

- ◀ Every issue that appears after the project goes live, is categorized as production support
- ◀ Production issues can be related to an unhandled exception in the graph logic or to unexpected data formats or volume from the source. Support team can help with classification.

Best Practices

◀ Use Graph Templates where possible

- ◀ Graph Templates are graphs, schema and public variables which can be used and combined to build more complex graphs
- ◀ Templates are pre-packaged and created for some of the most common use cases in Kitewheel
- ◀ Check if any of the graph templates can be used in the graph that you are building

◀ Use tagging to find components

- ◀ Create tags while creating a new node, for ease of search and readability

◀ Using the comment node to inform the graph viewer

- ◀ Use comment nodes in the graph that has a description of the graph
- ◀ Use comments in the JavaScript node to describe what it does

Best Practices

< Identify reusable components

- < Identify bits of logic that can be made as generic as possible

< Smaller graphs

- < Make sure the graphs are small enough to be viewed in a single screen

< Determine which node to use to represent rules or make decisions

- < If then else statements to return either True or False - **IF Conditional**
- < Two attributes are used to make the decision – **2D table**
- < Multiple attributes are to be evaluated in an order to get multiple outputs - **Columnar table**
- < Combination of multiple attributes contributes to different decisions, helping in categorization - **Decision Tree**
- < When a logic or rule check cannot be implemented by any of the built-in decision nodes - **JavaScript**
- < Randomly select a variant for A/B or multivariate testing, Choose the winning variant that has been getting the most engagement, Reassign the probability of selecting a variant to favor the winner - **A/B Split**

Best Practices

◀ Using Application Parameters

- ◀ Application parameters, like access codes, secret keys, account IDs etc., that are project and environment specific
- ◀ Kitewheel records this data in a database table called appParams
- ◀ This data is read into persistent variables that cache it for a specified time

◀ Align schema and data model on naming convention

- ◀ Make sure the name of the fields in database tables and the schema variable names are same to avoid confusion
- ◀ Place schema variables used for a journey, under the same section

◀ Error branches on all decision, adaptor, and KIM nodes

- ◀ Add error branch and error handling to all decision-making nodes and Kitewheel Identity Manager nodes

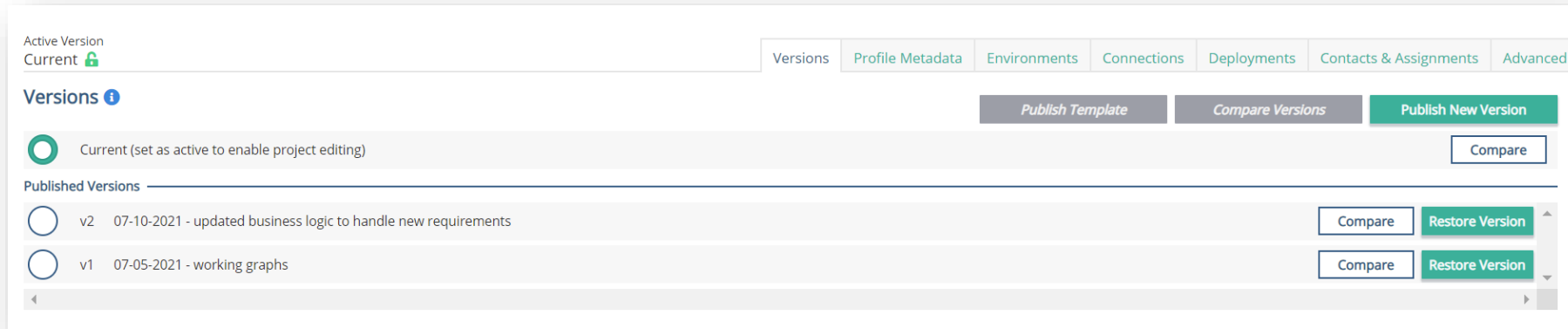
◀ Timestamp on KIM nodes

- ◀ Add the transaction's timestamp to all journey step and interaction nodes, so the accurate time is associated with the steps in the KIM

Best Practices

< Version everything

- < After testing individual nodes and the graph, create a Version of the Project
- < View any version of the project by selecting the **Restore Version** button beside the version
 - < Note: Create a version for your current changes before restoring a version, otherwise they will be lost
- < Make sure to give a meaningful name to the version along with the date (when version was created) and user initials (if multiple people are working on it)



Active Version
Current

Versions Profile Metadata Environments Connections Deployments Contacts & Assignments Advanced

Versions ⓘ Publish Template Compare Versions Publish New Version

Current (set as active to enable project editing) Compare

Published Versions

<input type="radio"/>	v2	07-10-2021 - updated business logic to handle new requirements	Compare	Restore Version
<input type="radio"/>	v1	07-05-2021 - working graphs	Compare	Restore Version

- < Older versions can be restored by clicking the Restore Version button. Any version after the restored version become archived under the restored version.

Testing

- ◀ Test everything!
- ◀ Unit test each node before combining them with any other node
 - ◀ Development environment
- ◀ Integration test components / sub graphs
 - ◀ Development / Stage environment
- ◀ End to end testing
 - ◀ Development / Stage
 - ◀ All rules need to be tested - correlate to Project Test Cases
 - ◀ Every rule should be hit
 - ◀ Test for exceptions and edge cases

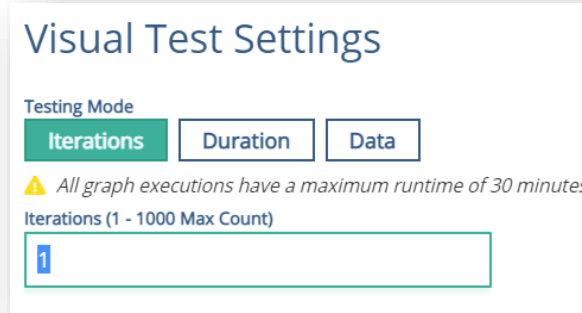
Testing

- ◀ Test Case results
 - ◀ Present to Strategy, Business and QA teams
 - ◀ Any changes to the graph need to be unit and regression tested
- ◀ Load testing
 - ◀ SLA needs to be defined
 - ◀ Always load test at least 5 to 10% of the total volume
- ◀ UAT / Smoke testing
 - ◀ Testing in the Production environment if possible
- ◀ Documentation
 - ◀ Everything that was built must be documented along with the test cases and load test results

Debugging

◀ Kitewheel hub provides 3 types of visual test settings:

- Iterations
- Duration
- Data (JSON)

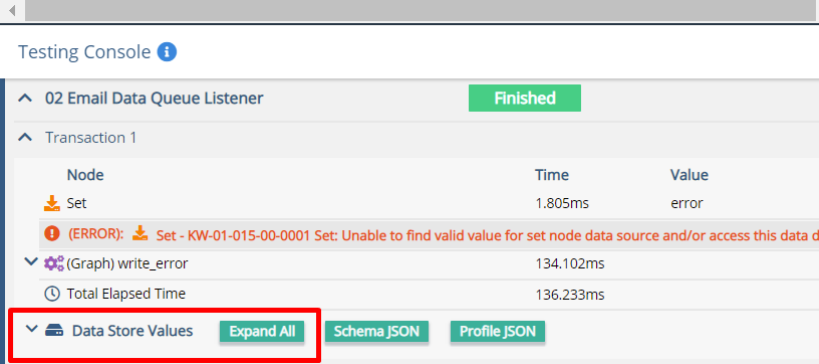


The screenshot shows a dialog box titled "Visual Test Settings". It features three tabs: "Iterations" (which is selected and highlighted in green), "Duration", and "Data". Below the tabs, there is a warning icon and the text "All graph executions have a maximum runtime of 30 minutes". Underneath, it says "Iterations (1 - 1000 Max Count)" and shows a text input field containing the number "1".

- ◀ Run 1 or 2 iterations to capture the input data for a listener graph
- ◀ Copy the JSON from the transaction
- ◀ Use JSON as input to test different rules and exceptions in the graph

Debugging

- ◀ Use JSON as input to test different rules and exceptions in the graph
- ◀ Use JSON to unit test complex components and nodes
- ◀ When testing the sub-graphs in a transaction, display the JSON as it exists- use this to watch the schema as it is transformed in the graph
- ◀ For example: if the graph has an error in sub-graph 3 capture the JSON from sub-graph 2 and run it through 3 to see the results



The screenshot shows a 'Testing Console' window. At the top, it says 'Testing Console' with an information icon. Below that, there's a section for '02 Email Data Queue Listener' with a 'Finished' status. Underneath is 'Transaction 1'. A table lists the nodes in the transaction:

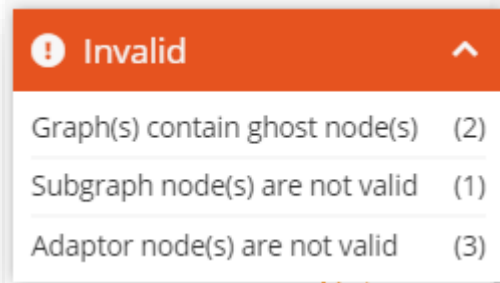
Node	Time	Value
Set	1.805ms	error
(Graph) write_error	134.102ms	
Total Elapsed Time	136.233ms	

Below the table, there's an error message: '(ERROR): Set - KW-01-015-00-0001 Set: Unable to find valid value for set node data source and/or access this data de'. At the bottom, there are three buttons: 'Data Store Values' (highlighted with a red box), 'Expand All', 'Schema JSON', and 'Profile JSON'.

Debugging

◀ Validate your graphs

- ◀ Invalid graphs will not be allowed to run
- ◀ The graph validator will highlight the broken node or link



- ◀ Error messages show up on the transaction history on the node that fails but the error could have been introduced earlier in the graph
- ◀ Common Error messages and Data Validation errors are documented in the online Kitewheel

Loop Node – Requirements

◀ To execute a subgraph multiple times in one position in the graph

You will need:

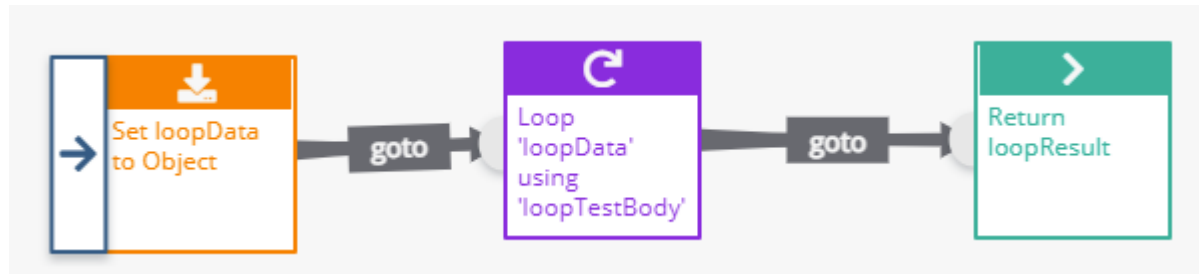
◀ A sub-graph

◀ Schema location

◀ Formatted JSON array of elements or objects

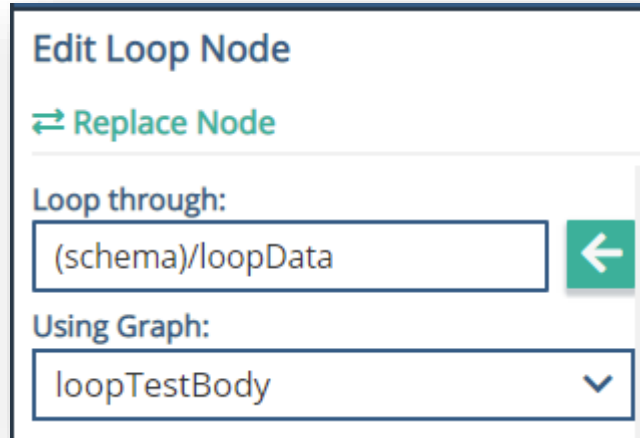
◀ Eg: "cars": ["Ford","BMW","Fiat"]

◀ If not an array it will be considered an array with 1 value



Loop Node – Creation

- ◀ Add a new node to the graph
 - ◀ Choose 'Loop Node'
- ◀ Select the loop node to add conditions
 - ◀ Schema location for the data
 - ◀ Drop down to choose a graph to loop through



Edit Loop Node

↔ Replace Node

Loop through:

(schema)/loopData ←

Using Graph:

loopTestBody ▾



Loop Node – Notes

- ⚡ The data location will be the same as where the array of data is when within the sub-graph
 - ⚡ But only the one value
- ⚡ Loop node cannot:
 - ⚡ Call it's own graph
 - ⚡ Use public variables – must use schema locations
- ⚡ Will process the array sequentially
- ⚡ Graph state is global so changes made in one sub-graph will be available for the next sub-graph
- ⚡ Testing console will show all iterations of the loop even though the graph will show just one transaction



Loop Node – Exercise

- ⏪ Create a loop that loops over the following array of words and concatenates them into one sentence.
- ⏪ ["The", "quick", "brown", "fox", "jumps", "over", "the", "lazy", "dog."]
- ⏪ Hints:
 - ⏪ JavaScript will be needed for the concatenation
 - ⏪ Have a separate location for the sentence location in the schema

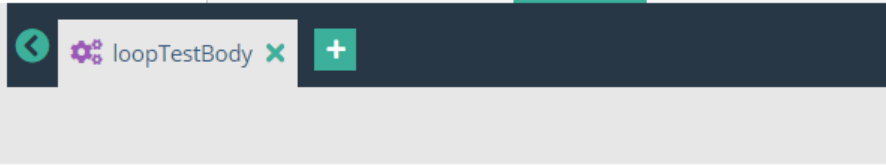
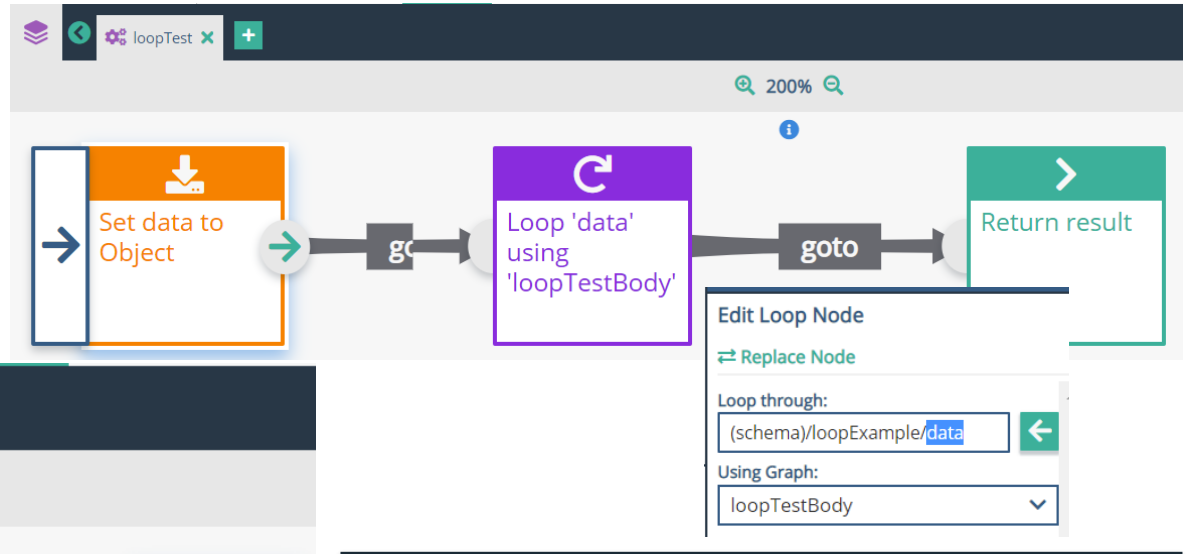


Loop Node – Exercise Solution

```
1 ["The", "quick", "brown", "fox", "jumps", "over", "the", "lazy", "dog-"]
```

loopExample

- > data
- > result



JavaScript Editor

+ Add Argument

```
function concatenateJS (newWord, outputResult) {  
  1 | return (outputResult || "") + " " + (newWord)  
}
```





















Loop Node – Exercise Solution

Testing Console 

Environment: d

Transaction 1

Node	Time	Value
 Set	0.249ms	true
^  (Loop) Loop	71.640ms	
∇  Iteration 1	9.385ms	
∇  Iteration 2	7.397ms	
∇  Iteration 3	7.581ms	
∇  Iteration 4	6.601ms	
\wedge  Iteration 5	6.694ms	
 (Script) concatenateJS	5.867ms	The quick brown fox jumps
 Get	0.233ms	jumps
 Return	0.196ms	The quick brown fox jumps
 Subgraph Elapsed Time	6.694ms	
∇  Data Store Values		
Expand All		
Schema JSON		
Profile JSON		
∇  Iteration 6	7.123ms	
∇  Iteration 7	6.541ms	
∇  Iteration 8	6.699ms	
∇  Iteration 9	6.933ms	
 Return	0.201ms	The quick brown fox jumps over the lazy dog.
 Total Elapsed Time	72.558ms	

Display JSON

```
1 {
2   "loopExample": {
3     "data": [
4       "The",
5       "quick",
6       "brown",
7       "fox",
8       "jumps",
9       "over",
10      "the",
11      "lazy",
12      "dog."
13    ],
14    "result": " The quick brown fox jumps over the lazy dog."
15  }
16 }
```



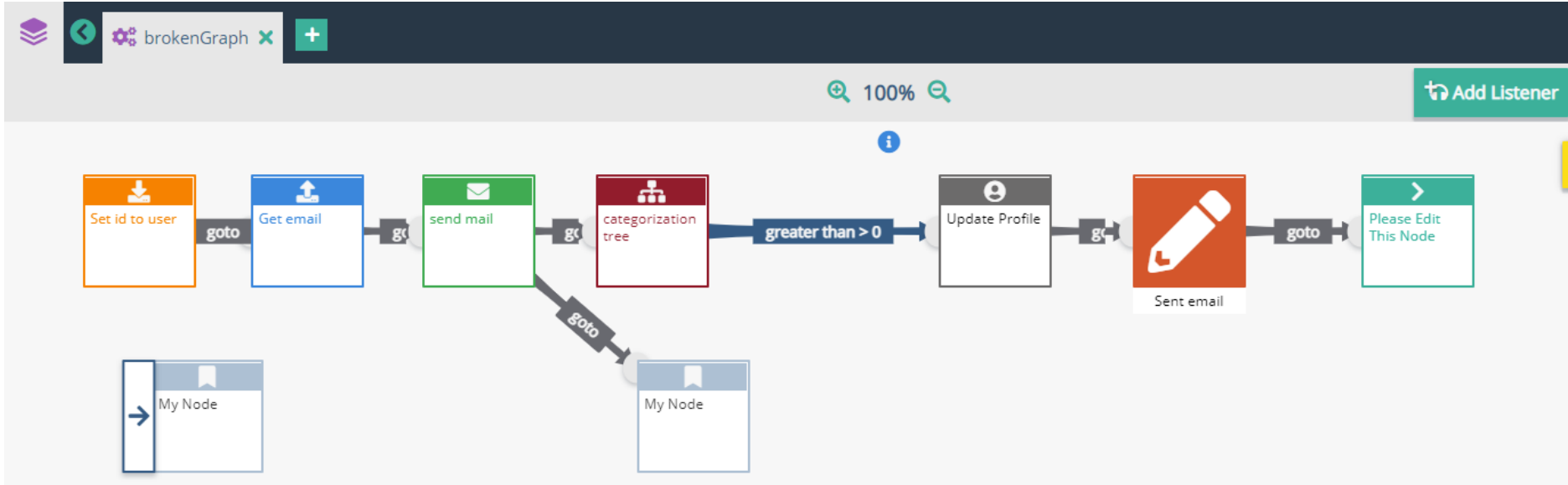
Debugging- Exercise

- ◀ Send email with coupon “THISISRANDOM” to all eligible customers and update their details in DB after sending the email. They should get their next email after 5 days.

HINT 1: - Pick all eligible records, add 5 days to get the next process time. Get their emailAddress and send them email. Update their nextProcessTime and lastProcessTime in the table – bg_customer

HINT 2: There are at least 10 edits/errors in this graph (see following slide)

Debugging- Exercise



Debugging- Exercise Solution

To fix this graph, we need to:

- Add a listener to get the information from a source
- Move the start node to the Set node, and delete the ghost node
- Move the Update Profile node to immediately after the set node (so we can access the Profile's information in the graph)
- Replace second ghost node with a Return node (or remove this ghost node)
- Add a GoTo link to the categorization tree node
- Add a value to the Return node
- Add error handling

Certification

- ⏪ How many graph templates can be used in a project?
- ⏪ Who is responsible to test the connections in a project?
- ⏪ What are the different types of testing that should be performed in Acquia Journey?
- ⏪ What are the steps to debug an Acquia Journey API graph?
- ⏪ What are the steps to debugging a Database Listener graph?
- ⏪ What format does the data have to be stored in to loop through?
- ⏪ What happens if it's not stored in that format?
- ⏪ Where can the data not be stored?
- ⏪ Can a graph loop through a graph which has a different loop on it?
- ⏪ How does the testing console show the loops?



Thank You