Cortex Interaction Portal

Developer Guide

Contents

Versions

Date	Version	Notes
13/02/2023	0.1	Internal release
10/04/2023	1.0	First release

Preface

About this manual

This manual provides a general user guide to the use of Appgyver for developing UIs for processes automated with Cortex, including how-to, examples, best practices, and architecture considerations.

Audience

This document is intended for Cortex flow developers who intend to add UI capabilities to Cortex Innovation.

Related Material

Document	Version
Cortex Interaction Portal Deployment Guide	1.0
Cortex Interaction Portal Merging Guide	1.0
Cortex Interaction Portal User Guide	1.0

Abbreviations used in this Document.

UI   User Interface

UX   User Experience

CIP   Cortex Interaction Portal

1   Introduction

1.1   AppGyver Overview

The Cortex Interaction Portal is built using SAP AppGyver: a low-code web (and mobile) application building tool which offers pre-built components and the ability to create your own component templates. Also included is integrated logic to issue API requests, navigate pages, show data, and many more options.

> *AppGyver is offered in several different pricing tiers. At the time of writing the Community Edition includes all the core functionality required.*

1.2   AppGyver and Cortex

Using AppGyver's integrated logic and the Cortex REST API, AppGyver can be used to provide Human-in-the-Loop capabilities to Cortex. This is offered by:

1.   A generic AppGyver solution with the ability to be extended to fit further needs.
2.   A set of generic Cortex modules which handles authentication, service requests, process executions, and much more.

The Web Application itself offers two core methods of interacting with automation:

1.   **UI-Driven Process (Service Request)**
     a.   This is a service request catalogue including role-based access control, configurable from the admin settings within the Web App.
     b.   Each Service Request can consist of any number of UIs, and each UI can integrate with any number of Cortex Flows.
     c.   Every Cortex interaction from a UI will require its own flow, and AppGyver will handle the actions, responses, and navigation – essentially in control of 'orchestrating' the process.
     d.   These processes must be initiated by a user, from the relevant service request.

An example of a UI-Driven Process would be a traditional web-based wizard to break data entry into a series of screens, guiding the user through completing a task.

2.   **Process-Driven UI (Process Flow and User Tasks)**
     a.   This handles user interactions for a process which may or may not be triggered by a user.
     b.   The process is developed and run entirely in Cortex, only breaking for user entry if / when required.
     c.   Users can view the pending user interactions from a dashboard, which also supports role-based access control. This can be configured at an individual task level, providing much more fine-grain control of the process.
     d.   The components to interact with Cortex are all generic (instead of many bespoke flows for the UI), leaving just the end UIs to be created.

An example of a Process-Driven UI would be any process that requires a manual approval step as part of its otherwise fully automated execution.

The solution includes Role-Based Access Control via Active Directory, which is managed from the Cortex Interaction Portal (via Cortex flows).

2   UI Development Process

In this section, the overall process for making changes to the Cortex Interaction Portal solution, consisting of Cortex flows and AppGyver pages, is covered.

Complete the following steps:

1. Import the app into the AppGyver online platform.
    a. Navigate to the AppGyver online platform at **https://platform.appgyver.com/**.
    b. Log in using the credentials of the account to which the Appgyver app should be imported.
    c. Select the 'Create New' button.
    d. In the dialog which appears, select 'Import from file'.
    e. Select the package provided.
    f. Give the app a relevant name (such as Cortex Interaction Portal) and select 'Create'.

2. Make any changes or additions necessary to the app in the AppGyver online platform.
    a. Identify which Cortex flows are required, along with the input and output variables that should be used to pass data between them.
    b. Test the pages using the App Preview Portal.

3. In Cortex Gateway develop the required flows
    a. Test the flows using the debugger.

4. Build the app.
    a. Navigate to the AppGyver online platform at **https://platform.appgyver.com/**.
    b. Once logged in, open the App.
    c. Select a page (for example Home Page) and select Launch from the toolbar at the top.
    d. In the left-hand panel, select Distribute, then select Open Build Service.
    e. For the first build operation, the build should be configured:
        i.   Select Web App > Configure.
        ii. For the Hosted Domain, enter any value – this will not be used as we will be deploying directly to the server.
        iii. Select 'ZIP' as the Build Scheme.
        iv. Upload a favicon if desired, for example, the Cortex Logo or a company logo.
        v.   Nothing is required in the 'Permissions' sections.

     f.   Click Build.

         i.   Ensure ZIP is selected.

        ii.   Select the latest Client runtime version.

       iii.   Select 'No' for whether the app should be deployed to the cloud.

       iv.   Enter a version number, for example 0.0.1.

        v.   Click Build.

       vi.   Wait for an email to the account registered under the account that is logged in. This will contain the files to be deployed.

5. Deploy the app. Once the operation has completed, the .zip file can be downloaded. This can then be extracted and placed into an IIS Site on the App Server(s).

   a.   If the application has not been deployed previously, then complete the following steps:

   b.   In IIS, expand the server node and then expand Sites.

   c.   Select (or create) the Site you wish to configure it under, for example, expand 'Cortex' which also hosts Cortex Gateway.

   d.   Right click this and select 'Add Application', configuring as required.

         i.   For Physical Path, you should select the directory you wish to use (creating it if it does not already exist).

        ii.   For example: C:\inetpub\wwwroot\Cortex\CortexUI.

       iii.   Select or create an Application Pool as required, for example, in a development environment you could use the 'existing 'Cortex' application pool.

   e.   Navigate to the folder configured for the site in File Explorer and copy all the unzipped files across.

         i.   If the copy operation fails, they may first need to be copied to a directory local to your user (such as a folder in Downloads).

6. Package the Cortex flows and publish them.
7. Configure the pages as Service Requests or Process Driven UI pages using the steps in Section 3.1 and 3.2.
8. Create new (or update existing) processes using the 'Manage Processes' page.

3 Cortex Interaction Portal How-To

This section will focus on a variety of 'how-to' sections relating to Cortex Interaction Portal and its integration with processes automated with Cortex.
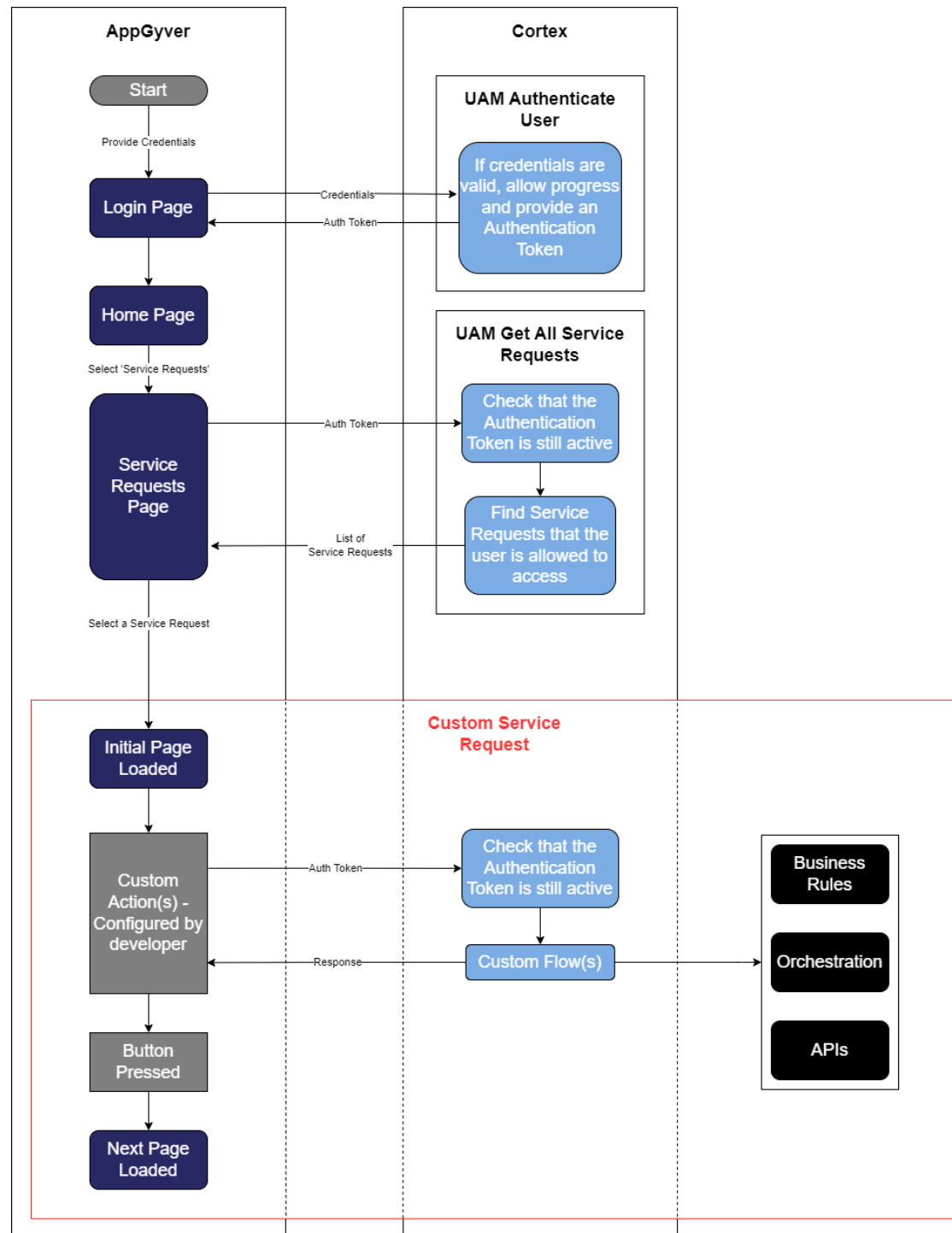
3.1 Creating a Service Request

For a service request, or 'UI-driven process', to be usable, it requires the following to be completed:

1. The service request to be registered, so that it is shown on the service request page.
2. An initial page to be created, which will be navigated to when the service request is selected. From this page, further pages may be navigated to as the developer wishes.
3. This page to be mapped to the service request, so that it is navigated to when the service request is selected.

A guide to creating a very simple service request page that may be selected from the service request catalogue is covered in the below steps. Additionally, the page logic will most likely call a Cortex flow, so this will be covered too.

The following diagram shows a high-level overview of how a service request operates, with interactions between the Interaction Portal and Cortex.

3.1.1   Creating an entry in Service Request Catalogue

Open the Cortex Interaction Portal with admin permissions, and from the home page, select 'Admin Settings, then 'Manage Service Requests'.

From here service requests and logical groups for them may be created as required. To create a Service Request, select 'New Service Request', and enter the following information:

1. **Name** – the displayed name of the Service Request
2. **Description** – Information about the Service Request. This will be displayed when the Service Request is selected on the Service Request page.
3. **Page Name** – used to map the Service Request to an AppGyver Page which would be opened when the Service Request is started.
   a. This is configured using AppGyver logic and is detailed later in the document.
4. **Allowed Roles** – the Active Directory groups which are allowed to start the Service Request
   a. Enter these by searching for roles in the list displayed.

📖 *This list is populated by the user groups given User or Admin roles in the 'Manage Settings' page.*

Select 'New Service Request' to create the service request, at which point it will be immediately accessible by users.

3.1.2 Creating a basic Service Request Page

In an AppGyver development environment, complete the following steps to create a new page:

1. Select the page name under the application title in the top left.
    a. This will open the page menu where any page in the application may opened and edited as required.
2. Select 'Add New Page'
3. Enter a page name.

For simplicity it is best to name the page the same as the 'Page Name' property that was configured in Section 3.1.1, but the page can be mapped accordingly once it has been created. To build an example page:
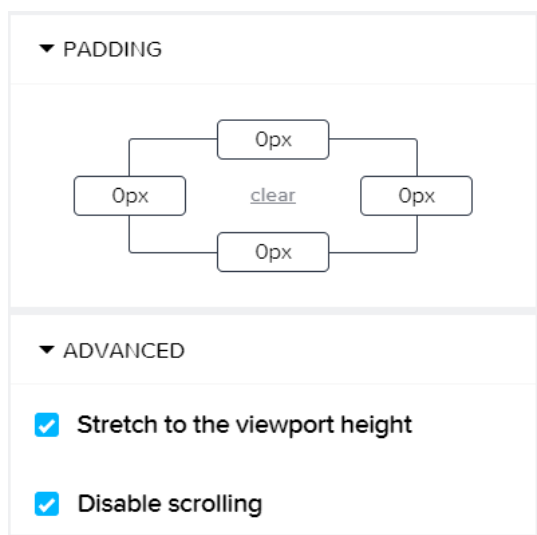
1. In the newly created page, delete the contents (2 text fields) and select 'Save'.
2. Open the 'Page Template' page from the page menu.
    a. Again, this may be opened by selecting the page name under the application title in the top left.
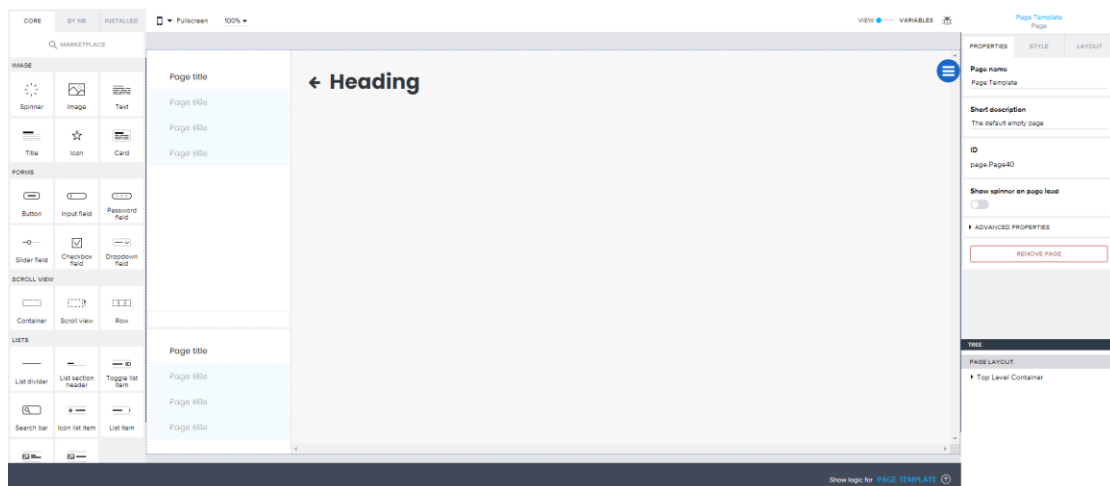
3. Everything is grouped under the 'Top Level Container' container in the right-hand tree panel.
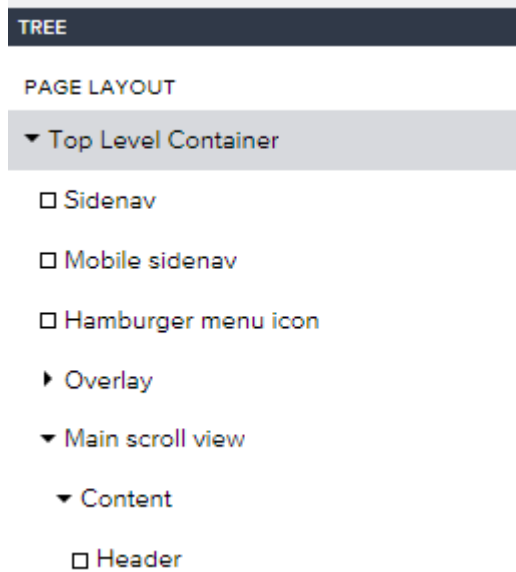

a. Select the 'Top Level Container' container, making sure that the whole page is highlighted, then copy and paste it onto the newly created page using CTRL+C and CTRL+V respectively.

b. With 'Page Layout' selected in the components tree, navigate to the 'Style' tab in the top right. Ensure that the padding is cleared and both advanced features are ticked.

c. Save the page.

4. Open another page, for example the Add Numbers example page.
5. All the functionality of the page should be grouped under the 'Content' container.
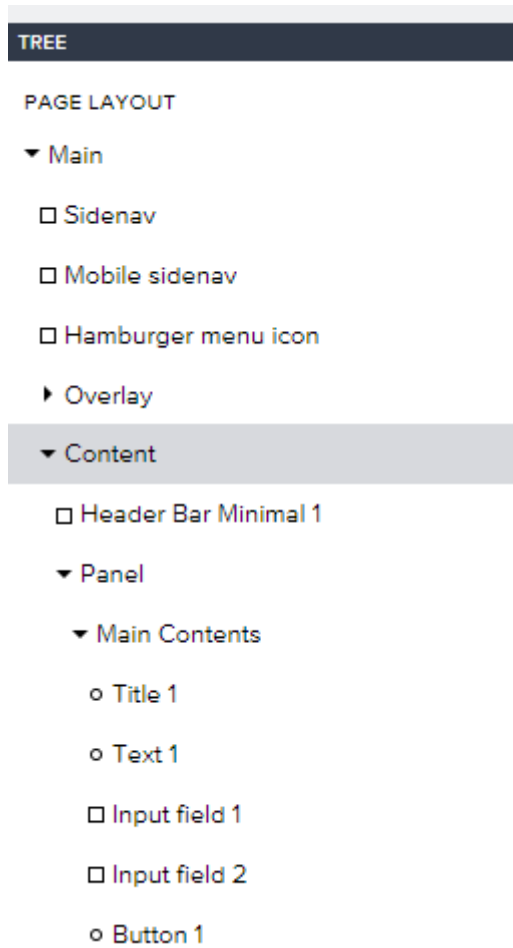
a. Copy the contents of this container and paste it onto the newly created page under its own 'Content' container.
b. Save the page.

📑 *Whenever a new page is created, it is recommended that the entirety of the 'Page Template' page is copied and pasted into it. Then, any page functionality should be created under the 'Content' container.*

6. The page content may now be edited as required.
    a. Remove the Number 1 and Number 2 text input fields.
    b. Select the title.
        i. Change the 'Page Name' property to 'Demo Service Request'.
        ii. Ensure that the 'Back Bar' property is set to 'True'.
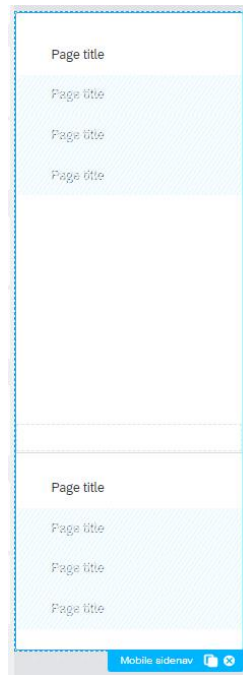
Page Name

$^{A}B_{C}$ Example Service Request

Back Bar

True

c. Change the title within the panel to 'Demo Service Request' also.
d. Delete the text in the panel.
e. Drag a new Input Field into the panel, select it and change the label to 'Name' in its properties.
f. Change the text on the button to 'Hello World'.

7. The page should look like this:

Page title
Page title
Page title
Page title

← **Demo Service Request**

Demo Service Request

Name

Type here...

Add These Numbers

Page title
Page title
Page title
Page title

8. Select the Mobile Sidenav element.

 a. Ensure that the 'Page Name' property is set to 'Service Requests', since this is a service request page.

 b. Leave the other properties as they are.

Now the basic page structure is complete, page logic and Cortex integration must be configured:

1. Change the view to Variables in the top-right.
   a. Select Page Variables instead of App Variables, then click Add Page Variable
   b. In the right-hand panel, change the variable name to 'Name' and ensure it is set as a Text value.

Note that variable names are case sensitive.

2. Switch back to the main view and configure the Name input field so its value is the variable you just created.
   a. You can select Data and Variables > Page Variables > Name.

Note that in some scenarios, especially when handling responses from Cortex, you cannot select the variable you need. The Formula option should then be used to reference a variable, since it does not restrict the options in this manner.

3.1.3   Mapping the Page to the Service Request

Open the page named 'Global Page' and expand the Logic panel at the bottom. This page logic governs what happens when events are raised by other pages. Here, the logic attached to the 'Open Service Request' event will be considered.

Double click on the 'Go To Service Request' flow function to view the contained logic.
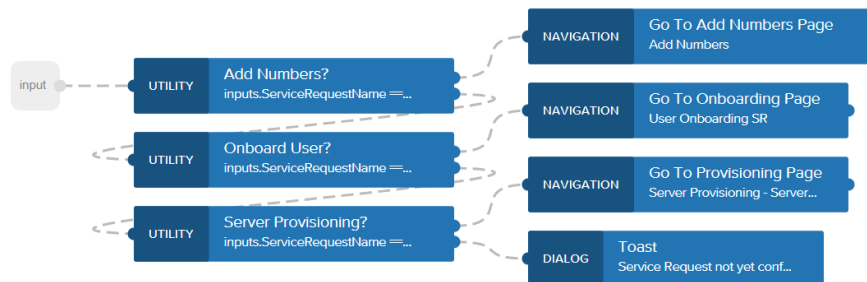
To map a new Service Request, duplicate one of the IF blocks and add it below the bottom one, connecting the bottom output of the previous IF block to its input.
Connect the blocks as follows:
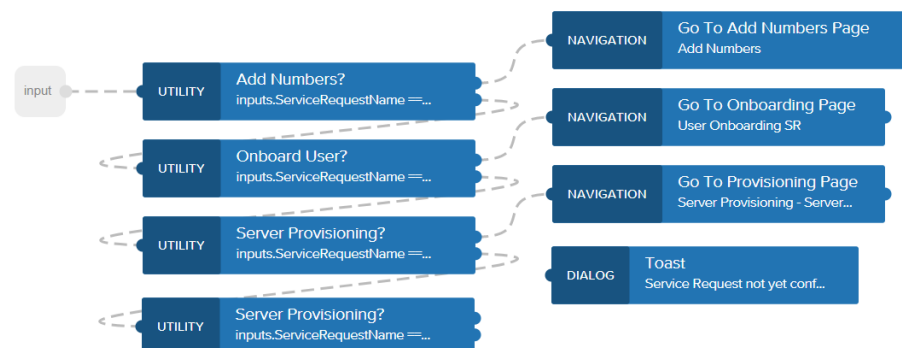
Click the new IF block and edit the formula to update the decision to match the new Ui Page Name:

```
inputs.ServiceRequestName ==  "Demo Service Request"
```

Update the Name accordingly (under Advanced).
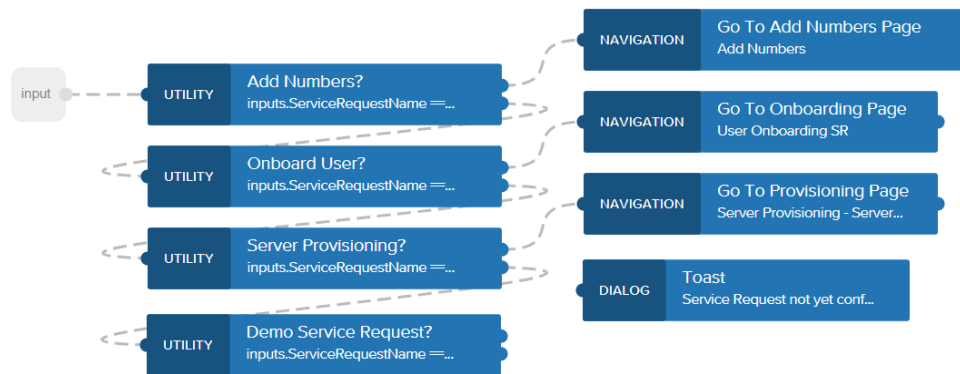
Connect the Toast Dialog block to the bottom output of this If Condition block.
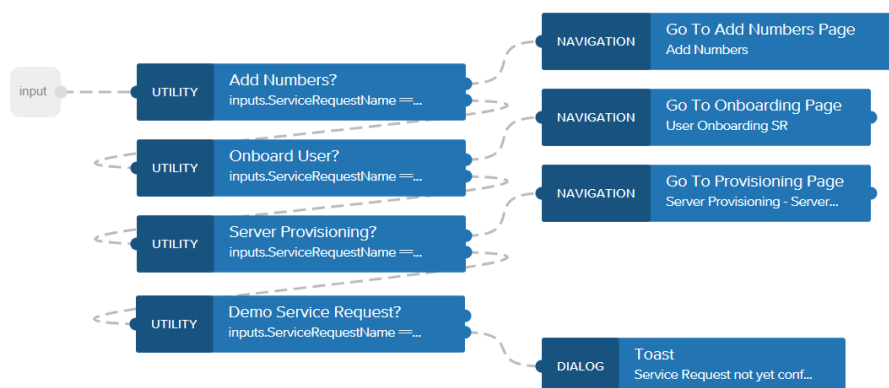
Connected the top output to a 'Open Page' block, reconfigured to navigate to the Service Request page.
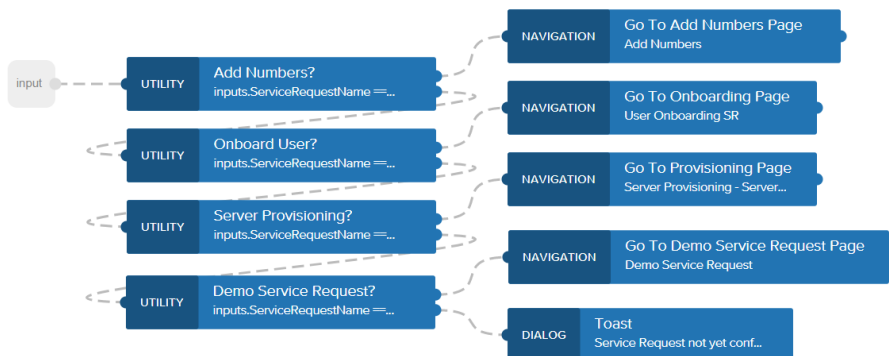
You can now test this from the AppGyver Preview, or by building the app and dropping the compiled files into the IIS website directory

3.1.4 Creating and calling a Cortex Flow

Here, we will outline a simple Cortex Innovation flow that has everything required to be called from a service request page in the Cortex Interaction Portal and return a value. From this example, it should be clear how to implement more complex logic.

3.1.4.1 Flow Structure

1. In Cortex Gateway, navigate to or create a group to store your Service Request UI flows.
2. Create a new flow, for example, 'Demo-Service-Request-Hello-World'.
3. Define your input and output variables.
   a. Create an input variable called authToken, another called name, and an output called message.
4. Connect a Run Flow block to the Start Flow block and set it to call the UAM-Validate-Token flow.
   a. Set the authToken input as the same authToken input variable you juts created, and enter a value in Outputs.
   b. Create a variable in which to save the response from the flown named tokenResponse. This may be created in the variables table, or from the property by typing a name and selecting 'Create <name>'.
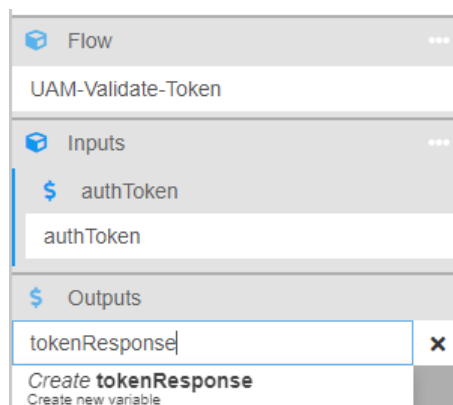
> 🏳 *When a user logs into the Cortex Interaction Portal web app, they are given an authentication token with a finite lifetime to uniquely identify their session using the app. This flow will retrieve the status of this token so that another flow can check whether or not the user should still have access to their session.*

> 🏳 *It is recommended that any flow called by an AppGyver page starts by calling this flow first, so that it can make sure that the user's access has not timed out or otherwise been revoked.*

5. Connect this block to an If True Exit Bottom decision block, which should be configured with the following expression:

```
($)tokenResponse.status == "Active"
```

    a. Connect the False branch to a Throw New Flow Exception block and ensure that the Default Exception Handler also has a Throw New Flow Exception block instead of the End Flow block.

6. Connect the True branch to a Set Variable block, which should be setting the output 'message' variable with the following expression:

```
$@"Hello {($)name}, welcome to the Cortex Interaction Portal"
```

7. Save and commit the flow. It should look like the below:
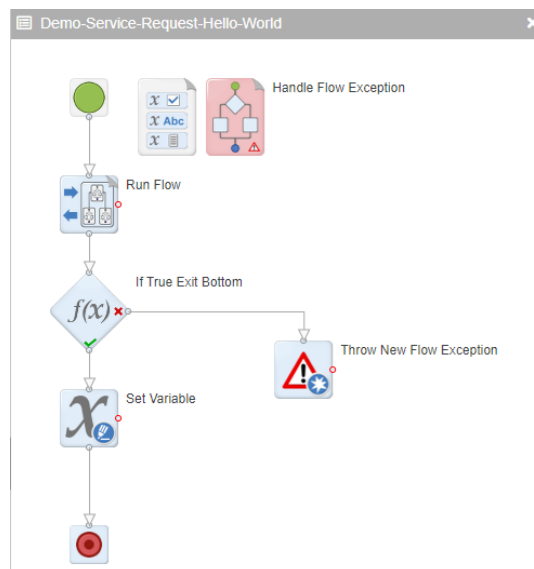
8. Include this flow in a package and publish it for the Cortex Interaction Portal to call.
    a. Click Settings > Packages, select the UserAccessManagement package and click 'Create New Version'.
    b. Tick the newly created flow, click Save, then click Publish.
    c. Ensure the 'Is Published' and 'Is Default Version' columns are both selected for the package:

3.1.4.2   Calling the Flow from the created Page

1. Ensure the Hello World button is selected and click on the Logic bar at the bottom to expand the bottom panel.

    a. You should see some logic already exists here:
        i. The 'Get Item from Storage' block will retrieve the user's authentication token, required when making a request to Cortex.
        ii. The flow function block labelled 'Add Numbers' is an implementation of the flow function named 'CortexFlowRequestSynchronous' and is used to make a request to Cortex.

2. Edit the CortexFlowRequestSynchronous flow function block.
    a. Firstly, rename this to 'Hello World' by using the Advanced section of the Properties panel.
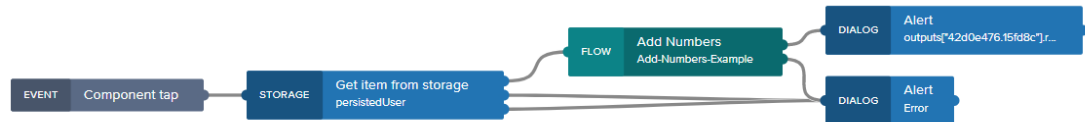   b. Edit the 'Flow Name' parameter and change 'Add-Numbers-Example' to the name of the new flow.
    c. Update the 'Body' Parameter to pass in the Auth Token and the Name variable linked to the input field:

```
{"authToken": outputs["Get item from
storage"].item.token, "name": pageVars.Name}
```

    d. Leave the 'Package Name' and 'Package Version' parameters as they are

3. Configure the Alert block to return the following:

```
outputs["Hello World"].resBodyParsed.message
```

4. Save the page.

📖 *Now, when a user enters their name into the page created earlier and presses the "Hello World" button, this flow will simply return a greeting including their name, which will be shown on the page.*

3.2   Creating a Process (Process-Driven UI)

In this section, the steps to create a simple Cortex flow will be covered. This flow will raise a task to the Cortex Interaction Portal web app, then wait for that task to be completed before continuing with its execution.

The following diagram shows a high-level overview of how a Process Driven UI operates, with interactions between the Interaction Portal and Cortex.
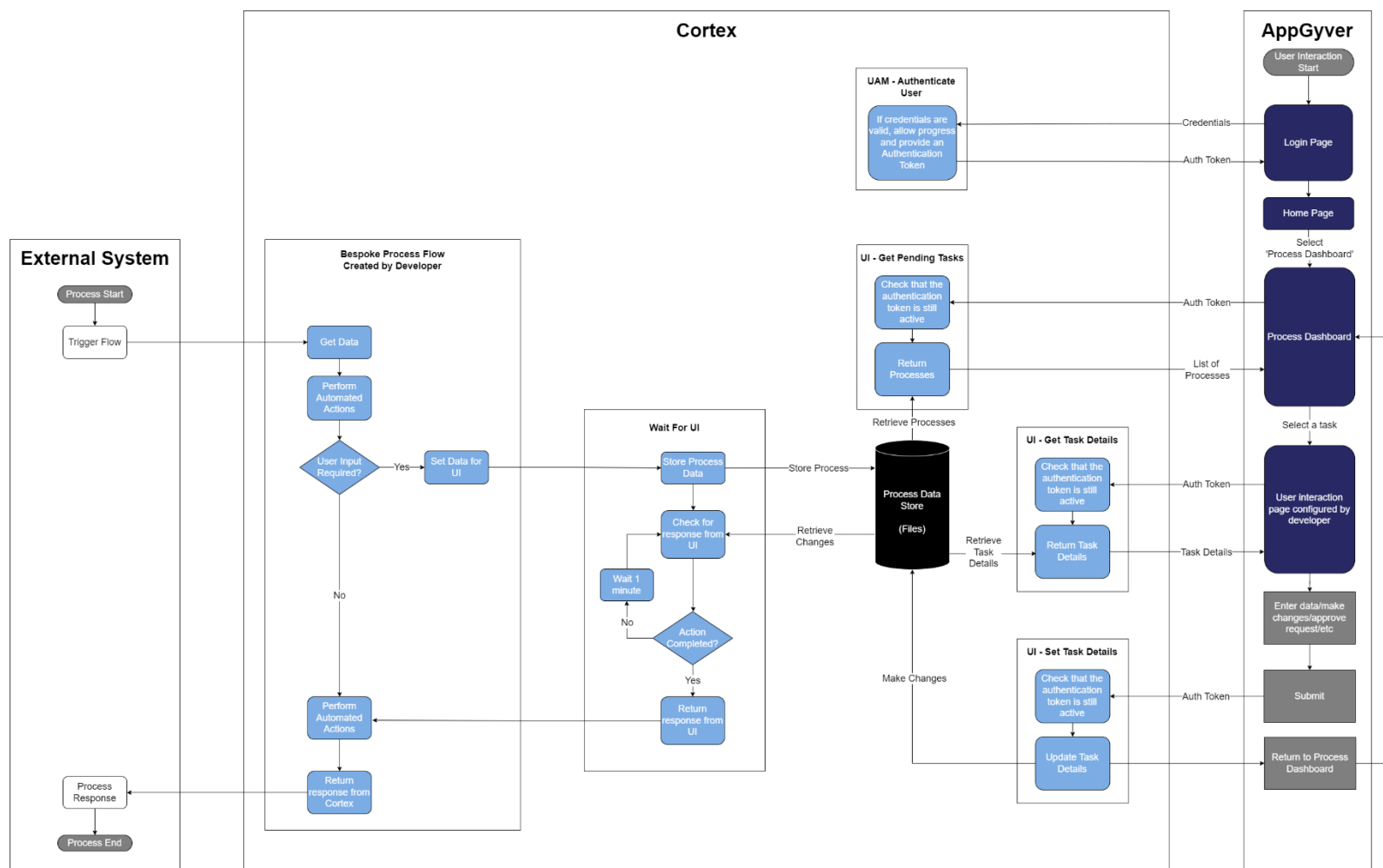
3.2.1 Creating a Process Flow

To start with, create a new flow in Cortex Gateway (for example 'My-Test-Process'). This will be built up to raise a task in the Cortex Interaction Portal web app.

3.2.1.1 Initialising the Process

1. Create a workspace called 'Initialise' and connect it to the 'start' block.
2. Call the 'Initialise-Process' flow using the Run Flow block, passing in the Process Name (e.g., Test Process) and optionally a list of Tags and a Reference to identify the execution.
   a. For now, configure tags as the below. IT is a tag which shows how processes can be grouped by a function or department, and test could relate to the process itself.

```
new List<string>()
{
    "IT",
    "Test"
}
```

   b. Reference can be empty if required – this will use an automatically generated Process ID (guid) instead.

3. Save the Outputs to a 'Response' variable local to the workspace you are working on, then use a Set Variable block to save the response to a global variable called 'LogData'.
   a. This should be using the variable reference 'Response.LogData'.

4. The flow should now look something like the below:
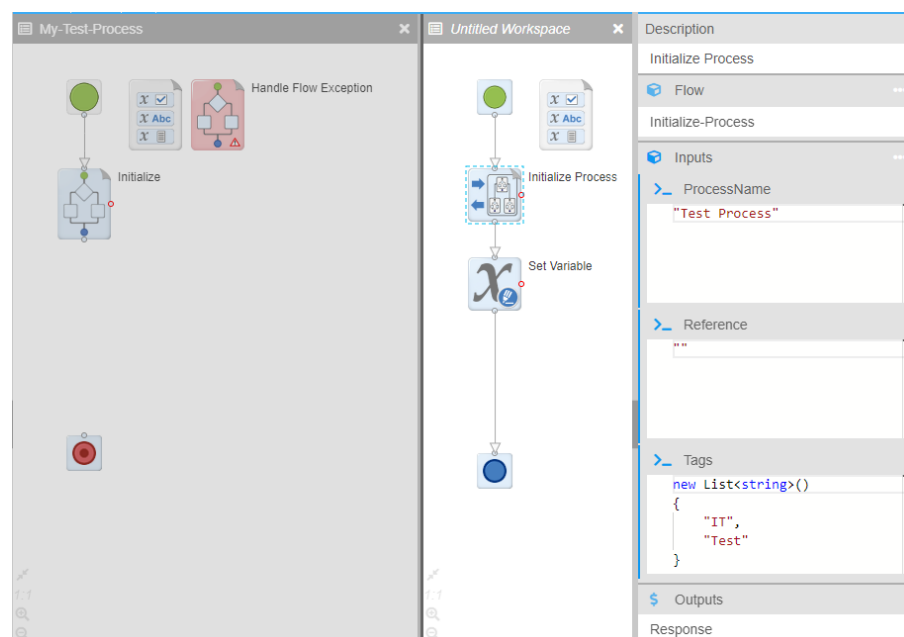
3.2.1.2 Logging Events

> 📑 *The logging model consists of one Process, many Stages, and many Events.*

> o *Process – The end-to-end course taken by this execution of the automation logic, with a definite start and end time.*

> o *Stages – Contained by the process, representing it's high level steps that have a definite start and end time.*

> o *Events – Contained by each stage, representing each action that is taken by the automation at single points in time.*

To log events as the flow takes actions, call the Log-Process-Data flow with the following inputs.

Input Variable	Type	Details	Example

Details	Structure	Optional
		Extra parameters to log alongside an event.
			{
			 "username": "user1"
			}

EventName	String	Name of the event to log	"Successful validation"
LogData	Structure	The Log Data variable returned from the Initialise-Process flow	LogData (variable reference)
StageName	String	Name of the stage to which the event will be logged. If the stage does not exist, then it will be created.	"Validate Order"

The output of this flow is not required.
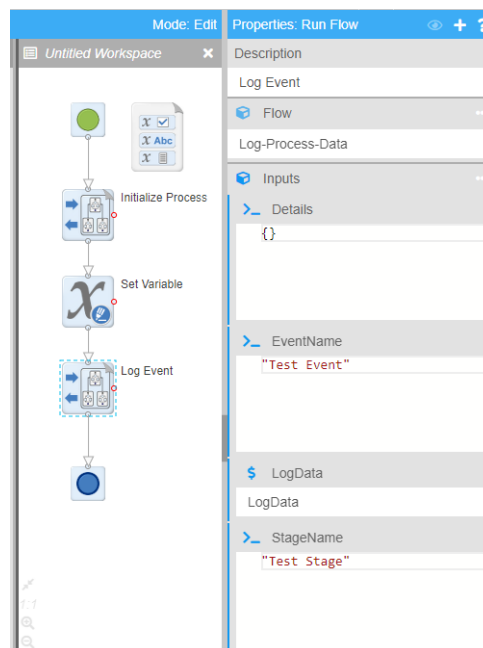
3.2.1.3 User Intervention - Task

To ensure that the flow's execution breaks for User Intervention:

1. Create a new workspace in the flow called 'UI' and connect it to the previous workspace.
2. Place a Set Variable block to create inputs to be passed to the AppGyver page that will be configured later, called UiInputs.

```
{
    "Param1": "Hello World",
    "Param2": DateTimeOffset.Now
}
```

3. Use the Run Flow block to call the 'Wait-For-UI' flow.
4. Configure the inputs as follows:

Input Variable	Type	Details	Example
Description	String	Optional description for the User Intervention, displayed in the dashboard	"This is a Dummy User Input to show how Process Intervention functions"
Parameters	Structure	Any parameters to pass through to the UI (structure)	UiInputs (variable reference)
ProcessData	Structure	The Log Data variable returned from the Initialise-Process flow	LogData (variable reference)
SLAHours	Int	Number of hours in which the task should be completed	10
TaskName	String	The name of the task that will be raised	"Dummy User Input"
Title	String	Optional title for the User Intervention, displayed in the dashboard	"Example User Interaction"
UiPageName	String	The name of the page the will be opened when a user handles this task	"Dummy UI"
UiTimeout	Int	If provided and > 0, the time in minutes before the UI will timeout and the main process flow will continue	0

5. Save the Outputs to a variable, for example UiResponse.
6. Place a decision block below this which will continue the flow is the Response Status is 'Completed' or throw an exception if not.

```
($)UiResponse.Status == "Completed"
```

7. At this point, the flow should look like the below:
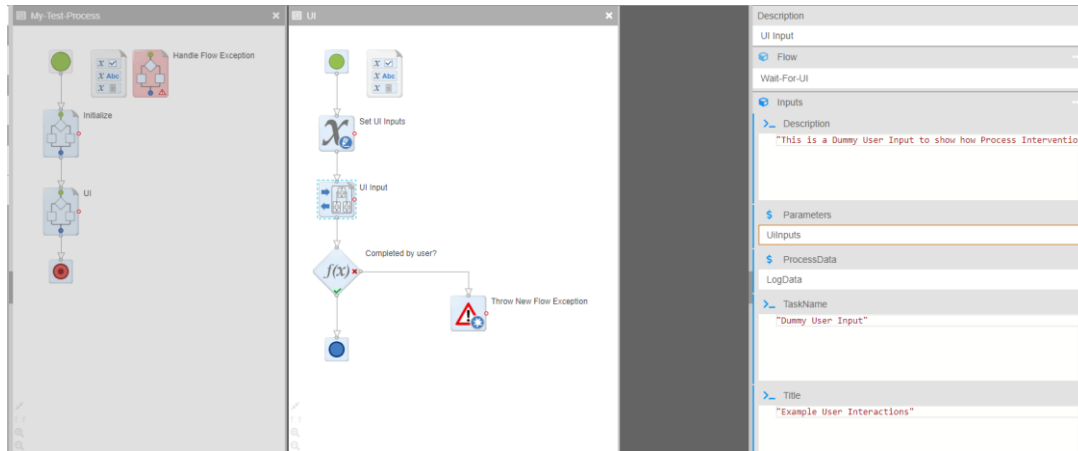
3.2.1.4 Ending the Process

1. Place another new workspace and call the 'End-Process' flow, passing in the LogData structure and the default Status value of 'Completed'.
2. Connect this workspace to the End Flow block at the top-level workspace.

3.2.1.5 Access Control Configuration

1. Login to the Cortex Interaction Portal web application with Admin permissions.
2. On the home page, click 'Manage Processes'.
3. Click the Define New Process button and enter the name of your process (from the 'Initialise-Process' call), for example, Test Process.
   a. Add Active Directory groups that should have access to this process.
4. Add a Task to this process, which should match the Task Name from the 'Wait-For-UI' call.
   a. Add Active Directory groups that should have access to this task if it is raised. These will be selectable from the list that were given access to the process above.
5. Click the Save button (above the search bar)
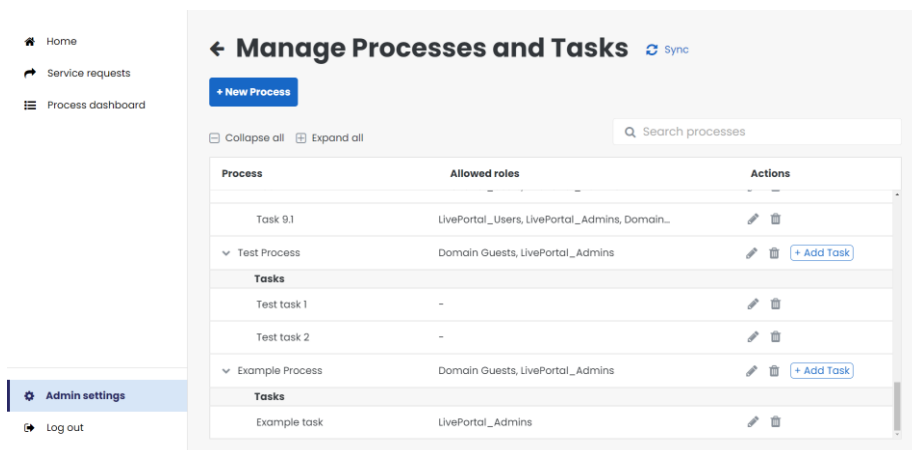
3.2.2 Creating the Task UI

The process flow created in the previous section will raise a task in the Cortex Interaction Portal web app. Selecting the task will enable this user to navigate to an AppGyver page where they may perform whatever actions are necessary to handle the task, and return data to the flow, which will then continue executing.

3.2.2.1 Creating the Page

1. Create a new page in AppGyver for the Task.

⚑ *The recommended naming convention is UI - <ProcessName> - <TaskName>, for example UI - Test Process - Dummy User Input.*

2. Delete the components that are automatically placed in the page.
3. As in section 3.1.2, copy the contents of the 'Page Template' page and paste them into the new page.
    a. Again, ensure the Page Layout styling is correct.
4. Select the 'Sidenav' element from the Tree in the bottom right.
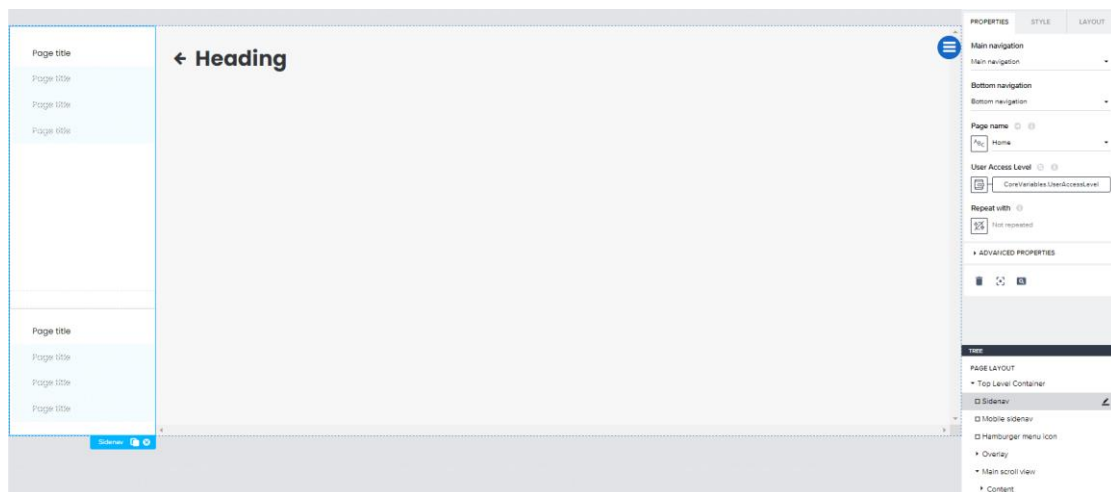
   a. Ensure that the 'Page Name' property is set to 'Process Dashboard', since this is a process UI page.
   b. Leave the other properties as they are.

5. Update the 'Mobile sidenav' element similarly.

6. The page layout and any additional logic can be configured as required, for example, you might show the input data retrieved and provide data entry inputs for the user to interact with the process, then send the data back to the process flow when they click a button.

    a. For this example, you can configure an input field mapped to a new Page Variable 'inputText'.

3.2.2.2   Get Input Data (on load)

Here, the page logic is configured such that when page loads, it fetches the process data from a file, where it is stored after being passed into the 'Wait-For-UI' flow.

1. In the logic panel, copy the Page Mounted logic from another Task UI. This ensures that the input data will be retrieved when the page loads.
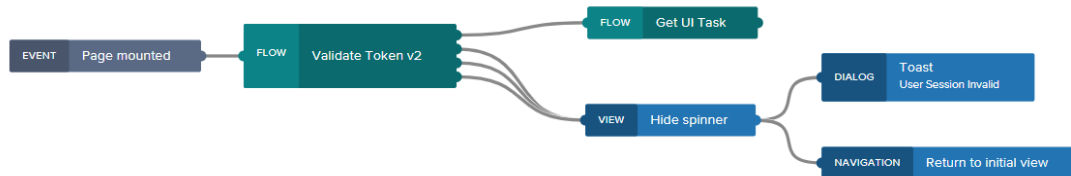
2. You will also need to add a Page Variable called 'TaskParameters' (with type 'any value') in the Page Variables section.

This ensures the associated data can be stored and displayed.

3. You will also need to update the AuthToken input to the 'Get UI Task' flow function – this should be:

```
outputs["Validate Token v2"].sessionDetails.authToken
```

3.2.2.3   Set Response Data (on completion)

Here, the data entered by the user on the AppGyver page is passed back to the flow so that it can continue executing.

1. The Update Process logic can also be copied from another UI page. This logic updates the process data with the If you copied the components across earlier, this should already be attached to the button.

2. If there is one, delete the 'Set UI Response block' – this usually sets a variable to pass in, but in this case, we can pass our JSON directly.
3. Selecting the 'Update UI Task' flow function block, configure the ResponseParams to pass in your text field by using the formula binding.

```
{"Text": pageVars.inputText}
```


4. You will also need to update the AuthToken input to the 'Update UI Task' flow function – this should be:

```
outputs["Validate Token v2"].sessionDetails.authToken
```

3.2.2.4 Mapping the Task UI to the Dashboard

Here, the page that was created for the task is mapped to the task dashboard, similar to mapping a page to a Service Request in Section 3.1.3. This ensures that when the task is raised, it can be navigated to by a user selecting the task.

1. On the 'Global Page' page, open the Logic panel and view the logic attached to the 'Open Process' event.

2. Double click the 'Go To Process Task' flow and view the logic.
3. Here the bottom IF block may be copied and connect it in the same manner.
   a. The true condition should connect to a Navigation block, configured to send the user to the page that was created in Section 3.2.2.1.
   b. The false condition should connect to the Toast block (or another IF block).
   c. You should configure the Formula accordingly for the IF block. For example:

```
(inputs.PageName == "Demo Service Request").
```

⚐ *The UI Page Name should be the 'UiPageName' input to the 'Wait-for-UI' Cortex flow which is called from the main process flow.*

Before

input	
UTILITY Onboarding - Department inputs.PageName == "Enter Ro..."	**NAVIGATION** Open page UI - Onboarding - Select Dep...
UTILITY Onboarding - Assets inputs.PageName == "Select A..."	**NAVIGATION** Open page UI - Onboarding - Assets
UTILITY Onboarding - Approval inputs.PageName == "Approve ..."	**NAVIGATION** Open page UI - Onboarding - Approval
UTILITY Order Management - Exception inputs.PageName == "Order Ma..."	**NAVIGATION** Open page Order Management - Exception
UTILITY Order Management - Data Missing inputs.PageName == "Order Ma..."	**NAVIGATION** Open page Order Management - Data Miss...
	DIALOG Toast UI Not Configured
	VIEW Hide spinner

After

input	
UTILITY Onboarding - Department inputs.PageName == "Enter Ro..."	**NAVIGATION** Open page UI - Onboarding - Select Dep...
UTILITY Onboarding - Assets inputs.PageName == "Select A..."	**NAVIGATION** Open page UI - Onboarding - Assets
UTILITY Onboarding - Approval inputs.PageName == "Approve ..."	**NAVIGATION** Open page UI - Onboarding - Approval
UTILITY Order Management - Exception inputs.PageName == "Order Ma..."	**NAVIGATION** Open page Order Management - Exception
UTILITY Order Management - Data Missing inputs.PageName == "Order Ma..."	**NAVIGATION** Open page Order Management - Data Miss...
UTILITY Order Management - Data Missing inputs.PageName == "Demo Ser..."	**NAVIGATION** Open page Demo Service Request
	DIALOG Toast UI Not Configured
	VIEW Hide spinner

3.2.3   Testing

1. Execute the flow in Cortex Gateway and ensure there are no exceptions – the execution should move to the Wait-For-UI loop.
   a. Set a breakpoint on the decision block in the UI state.
2. Open the Cortex Interaction Portal (either deployed to your server or via AppGyver preview)
   a. Optionally, keep the Chrome developer Tools open to diagnose any issues making requests to Cortex.
   b. Log in to the system if prompted.
3. Select 'Process Dashboard' from the home page.
   a. The dashboard should show at least 1 pending execution in the executions table.
   b. If there are many executions, as below, then sorting by 'Start Date' will show the most recently started one.

Reference	Process	Status	Start Date ∨	Pending Time	SLA Due Date	Action
ℹ CTX_OM_74035 / Speed not available - Exception	Order Management	Pending	2023-02-08 16:42:39	1 days		↗
ℹ CTX_OM_31208	Order Management	Cancelled	2023-02-08 16:23:07			
ℹ CTX_OM_80012 / Speed not available - Exception	Order Management	Pending	2023-02-08 16:22:46	1 days		↗
ℹ CTX_OM_92390 / Speed not available - Exception	Order Management	Pending	2023-02-08 16:22:22	1 days		↗
ℹ CTX_OM_66654	Order Management	Cancelled	2023-02-08 16:21:58			
ℹ CTX_OM_83224	Order Management	Cancelled	2023-02-08 16:19:01			
ℹ CTX_OM_59415	Order Management	Cancelled	2023-02-08 16:09:34			
ℹ CTX_OM_51337 / Speed not available - Exception	Order Management	Pending	2023-02-08 16:05:13	1 days		↗
ℹ CTX_OM_21284	Order Management	Cancelled	2023-02-08 16:01:30			
ℹ CTX_OM_49307	Order Management	Cancelled	2023-02-08 15:54:27			
ℹ CTX_OM_44823 / Speed not available - Exception	Order Management	Pending	2023-02-08 15:52:49	1 days		↗
ℹ CTX_OM_83203 / Speed not available - Exception	Order Management	Pending	2023-02-08 15:48:59	1 days		↗
ℹ CTX_OM_81155	Order Management	Cancelled	2023-02-08 15:43:12			

4. The view is filterable based on available statuses and processes.

5. If any data was logged, clicking the 'i' button will show these logs
6. Providing the access control was set up correctly, the Open Task button should be displayed – click this to navigate to the UI
   a. If the 'IF' statements were correctly configured in Section 3.2.2.4, the page will open. Otherwise, a notification will be shown.
   b. Enter some data in the text input field and click Confirm.
   c. The execution will now have a status of 'Running'.

7. Go back to Cortex – after a short time the execution will move to the breakpoint.
    a. Here you should see the response text from the UI, and the status of the task (Completed)

```
UiResponse: Structure
        Code ▾
1 ▾  {
2 ▾    "Response": {
3        "Text": "JOE"
4      },
5      "Status": "Completed"
6  }
```

8. Continue the execution.
    a. It should run through to the end.
    b. The execution will now have a status of 'Completed' and will disappear after 24 hours.

3.3 Direct Linking

When manual tasks are raised by an otherwise fully automated process, they are shown in the Process Dashboard. Direct linking allows the pages associated with a specific task that has been raised to be shared and accessed via a URL.

3.3.1 URL Structure

The URL takes the following form:

https://**<host>**.**<domain>**.com/**<app name>**/?ProcessID=**<process ID>**

The parameters should be configured as follows:

Parameter	Description	Example
Host	The server hosting the Cortex Interaction Portal	myserver
Domain	The domain of the server hosting the Cortex Interaction Portal	mydomain
App name	The name given to the Cortex Interaction Portal Usually this is 'CortexInteractionPortal'	CortexInteractionPortal
Process ID	The uuid of the process that the task has been raised as part of.	b7873633-9aab-494c-aa95-368332cf741f

Therefore, an example URL could be:

https://myserver.mydomain.com/CortexInteractionPortal/?ProcessID=b7873633-9aab-494c-aa95-368332cf741f

3.3.2 Sharing a link manually

To share a link to a task that is visible on the process dashboard, navigate to it and examine the reference column.

	Reference	Process	Status	Start Date ∨	Pending Time	SLA Due Date	Action
i	CTX_OM_74035 / Speed not available - Exception	Order Management	Pending	2023-02-08 16:42:39	42 days		↗
i	CTX_OM_80012 / Speed not available - Exception	Order Management	Pending	2023-02-08 16:22:46	42 days		↗
i	CTX_OM_92390 / Speed not available - Exception	Order Management	Pending	2023-02-08 16:22:22	42 days		↗
i	CTX_OM_51337 / Speed not available - Exception	Order Management	Pending	2023-02-08 16:05:13	42 days		↗
i	CTX_OM_44823 / Speed not available - Exception	Order Management	Pending	2023-02-08 15:52:49	42 days		↗
i	CTX_OM_83203 / Speed not available - Exception	Order Management	Pending	2023-02-08 15:48:59	42 days		↗
i	CTX_OM_87922 / Speed not available - Exception	Order Management	Pending	2023-02-08 15:39:01	42 days		↗
i	CTX_OM_98359 / Speed not available - Exception	Order Management	Pending	2023-01-31 16:33:06	50 days		↗
i	CTX_OM_48478 / Speed not available - Exception	Order Management	Pending	2023-01-31 16:33:01	50 days		↗
i	CTX_OM_47774 / Speed not available - Exception	Order Management	Pending	2023-01-31 16:32:58	50 days		↗
i	CTX_OM_48686 / OMS Exception - Exception	Order Management	Pending	2023-01-31 16:32:55	50 days		↗

1 2 3 4 5 6 7 8 9 10 11

The ID before the name of the task is the Process ID that can be used in the sharable URL. Using the example above, if a user was to navigate to the following URL, they will only be shown that particular task in the dashboard.

https://myserver.mydomain.com/CortexInteractionPortal/?ProcessID=CTX_OM_74035

	Reference	Process	Status	Start Date ∨	Pending Time	SLA Due Date	Action
i	CTX_OM_74035 / Speed not available - Exception	Order Management	Pending	2023-02-08 16:42:39	42 days		↗

1

3.3.3 Sharing a link automatically

Rather than sharing the link manually, it may be a good idea to have the flow that raises the task send an email notification to the user who should deal with the task. This notification could contain the URL that will directly link them to the task to deal with.

An example flow configuration can be shown below, where an email is sent directly before the task is raised.
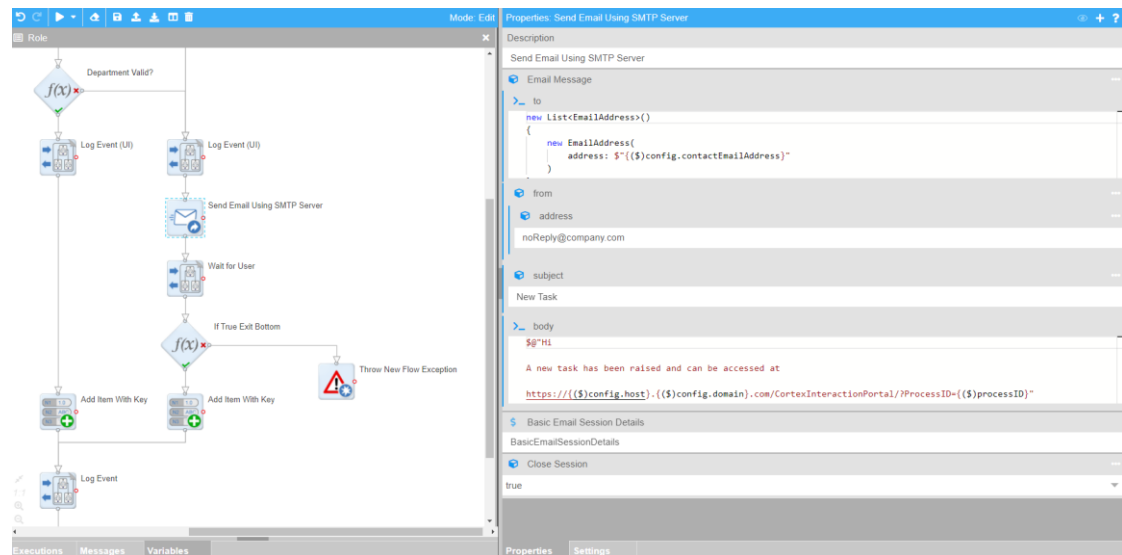
4   AppGyver How-To Guidance

This section will focus on a variety of miscellaneous 'how-to' sections relating to building pages in AppGyver.

4.1   UI Components

4.1.1   Conditional visibility

Often when building UIs, there is a requirement to make certain components visible only under certain conditions. There are two ways of doing this, both making use of the 'Visible' property which is configurable for all UI components:

4.1.1.1   Configure the visibility based on a formula

Here we will bind the 'Visible' property of a component to a formula, ensuring that a button is only visible on the first day of every month.

1. Select the component to which the visibility should be configured. In this case a button, but this process is extendable to any UI components.
2. On the right, the properties of this button will be displayed, by clicking 'Advanced Properties', the 'Visible' property that should be configured here will be revealed.
3. Select the button directly below the property name to configure the bindings.
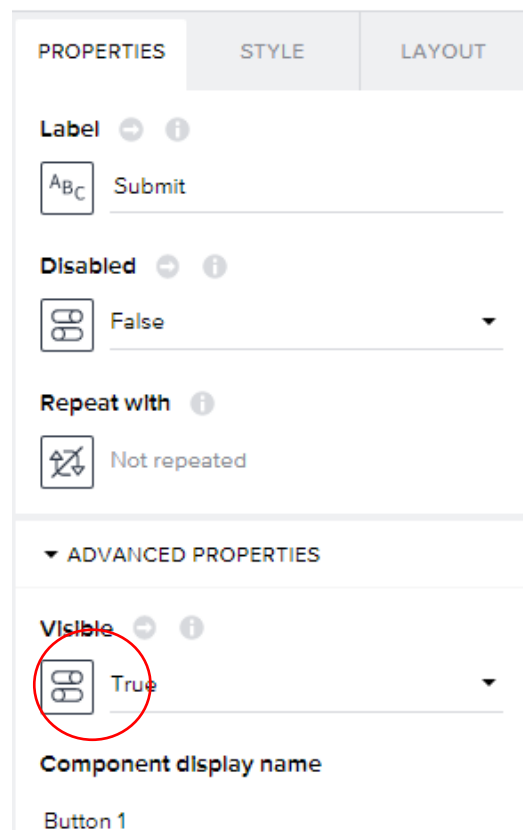
CORTEX

4. This will open the window for editing bindings for this property.

Bindings allow the user to set the property value to a datum from another source, not just a static value. This is analogous to changing the view on a Cortex block property between literal, variable, and expression.

5. Select 'Formula', then click the existing formula.

6. This will open the formula editor. A multitude of variables and functions with which to manipulate them are available to use by double-clicking them.
7. Enter the below formula into the formula editor. This will return true if the current date is the first of the month.

```
GET_DATETIME_COMPONENT(NOW(), "date") == 1
```

8. Select Save, then Save again, to close the bindings window. The 'Visible' property now has a visible formula binding.

9. Preview the web app. If today is the first day of the month, then the button will be visible, otherwise it will not be.

If today is not the first day of the month, then it may be worth updating the formula to display it on the current day of the month. For example, if it is currently the 4th, then update the formula so that the button is only shown on the 4th.

4.1.1.2   Configure the visibility based on a variable

Here we will bind the 'Visible' property of a component to a variable, ensuring that a button is only visible if another button is clicked.

1. Create a blank UI containing two button components inside a parent container.
2. Label the buttons 'Toggle Visibility' and 'Submit'. The UI should look like the below:
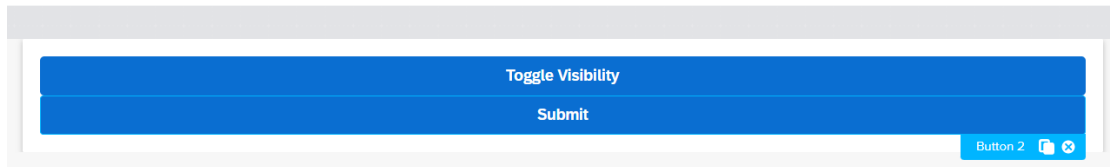
3. Select the View-Variables toggle, then Page Variables.
    a. Create a new page variable called 'isVisible'.
    b. Set the type to 'True/false'

4. Select the View-Variables toggle again and select the 'Submit' button.
5. Go to select the bindings button for the 'Visible' Property.
    1. Rather than selecting 'formula', as in the previous section, select 'Data and Variables'.
    2. Select 'Page Variable'
    3. Select the variable 'isVisible'
    4. Select 'Save'

6. Select the 'Toggle Visibility' button and open the logic workspace at the bottom of the screen.
7. Attach a 'Set Page Variable' block to the 'Component tap' event block
    1. Configure the 'Variable name' parameter to be the 'isVisible' variable created earlier.
    2. Bind the following formula to the 'Assigned value' property. This will set the value of isVisible is true if it false, and to false if it true.

```
IF(pageVars.isVisible, false, true)
```

8. Save the page and ensure it has been mapped to a service request following the steps in section 3.1.
9. Preview the app and launch the service request.
10. Only the 'Toggle Visibility' button should be visible and selecting it should show and hide the 'Submit' button.

📖 *Note that this variable can have its 'preview' value set to 'false' in order to hide the component while developing, but this will not affect its behaviour when deployed and used.*

4.1.2   Repeating UI components

It is often a requirement for a page to have components repeat many times in a dynamic fashion based on a list of data. For example, a table or list of data, or a progress bar denoting how far along in a UI driven process they are.
In AppGyver, this works similarly to a 'For Each' loop in Cortex.

1. In AppGyver, create a page or app variable with a type of 'List'.

⏴ *This may be populated with data from a user input, a response from calling a Cortex flow, passed as a parameter from another page, etc*

2. Place a component to repeat on the page, this could be a button, input, or even a container containing many other components.
3. Select this component and locate the 'Repeat with' parameter.
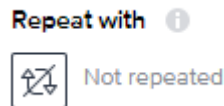
4. Binding this parameter to the list variable created earlier will cause the component to be rendered once for each element in the list.
5. The element of the list that a repetition of the component represents can be accessed in a formula by referencing *repeated.current*, similarly to the *CurrentIteration.Value* variable used in 'For Each' loop in Cortex.
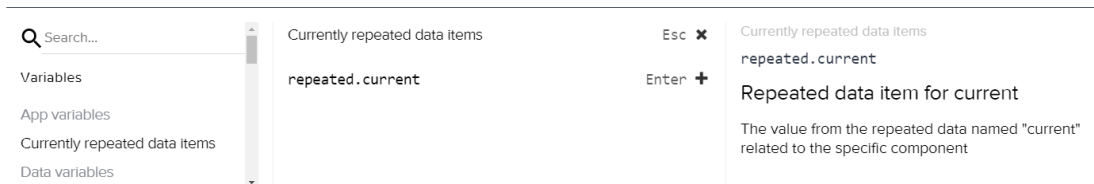
4.1.3   Dynamic Forms

Using conditional visibility and repeating UI components, it is possible to build up a dynamic web form based on a JSON specification returned by Cortex.

For example, a user may be registering a car with a car dealership on an AppGyver page, which calls a Cortex flow to return the information required by the dealership's systems to complete the registration.

```json
[
    {
            "label": "Make",
            "typeData": {
                    "typeName": "Dropdown",
                    "dropDownData": [
                            {
                                    "label": "Ford",
                                    "value": "Ford"
                            },
                            {
                                    "label": "Land Rover",
                                    "value": "LandRover"
                            },
                            {
                                    "label": "Volkswagen",
                                    "value": "Volkswagen"
                            }
                    ]
            },
            "defaultValue": "Ford"
    },
    {
            "label": "Condition",
            "typeData": {
                    "typeName": "Text"
            },
            "defaultValue": "No issues"
    },
    {
            "label": "Tax Required?",
            "typeData": {
                    "typeName": "Checkbox"
            },
            "defaultValue": true
    },
    {
            "label": "Value",
            "typeData": {
                    "typeName": "Number",
                    "minValue": 15000,
                    "maxValue": 100000
            },
            "defaultValue": 30000
    }
]
```

The above example specification defines a list of four input fields, each of a different type, along with all information required by AppGyver to render each field.

Here, the aim is to create an AppGyver page that turns the above specification into a web form, then saves the data entered by the user to an object variable. To be truly dynamic, if a fifth element is added to the above JSON specification, then a new form element should be added to the AppGyver page without any changes being made to it.

To implement the above, complete the above steps:

1. Create a new page in AppGyver and register a service request that will allow a user to navigate to it using the steps in Section 3.1.
2. Create the following page variables for the page.

Variable Name	Variable Value Type
AllParameters	Object
FormSpecification	List of texts
ParameterNamesGivenValue	List of texts
ParameterValue	Object with a 'text' property named 'Name' and an 'any value' property named 'Value'
ParameterValues	List of objects
ParametersGivenValue	Object

3. Select 'PAGE LAYOUT' on the component tree in the bottom right and open the logic panel.
4. Connect a 'Set page variable' block to the 'Page mounted' event.
   a. The 'Variable name' should be 'FormSpecification', as created earlier.
   b. The 'Assigned value' should be a formula, with the JSON above compressed and pasted as the value.

```
[{"label":"Make","typeData":{"typeName":"Dropdown","dropDownData":[{"label":"Ford","value":"Ford"},{"label":"Land Rover","value":"LandRover"},{"label":"Volkswagen","value":"Volkswagen"}]},"defaultValue":"Ford"},{"label":"Condition","typeData":{"typeName":"Text"},"defaultValue":"No issues"},{"label":"Tax Required?","typeData":{"typeName":"Checkbox"},"defaultValue":true},{"label":"Value","typeData":{"typeName":"Number","minValue":15000,"maxValue":100000},"defaultValue":30000}]
```

   a. This will ensure that the specification value is defined when the page loads.
   b. The logic should look as follows:
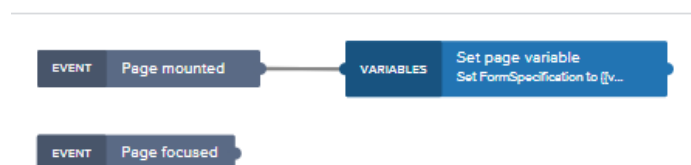
5. Add a container to the page.
6. Bind the 'Repeat with' property to a formula, simply referencing the FormSpecification page variable as follows:

```
pageVars.FormSpecification
```

7. Within this container, add an input field, a dropdown field, a slider field, and a checkbox field in any order.
8. Save the page and launch the app as in the preview portal.
9. Log into the app and navigate to the page from the service requests page. It should look like the below, with the set of input fields being repeated 4 times.

10. Return to the UI Canvas and select the input field

    a. To ensure that the component is displayed correctly, configure the properties as follows:

Label - formula with value:

```
repeated.current.label
```

Value - formula with value:

```
repeated.current.value
```

Placeholder text - formula with value:

```
repeated.current.defaultValue
```

Visible - formula with value:

```
repeated.current.typeData.typeName == "Text"
```

    b. To ensure that the data entered is saved correctly, configure the component logic as follows:

        i. Connect a 'Set page variable' block to the Component onChange event. Configure its properties as follows:

Variable name – Page Variable ParameterValue, with Page Variable Field ParameterValue.Name

Assigned Value – formula with value:

```
repeated.current.label
```

        ii. Connect another 'Set page variable' block to the one above. Configure its properties as follows:

Variable name – Page Variable ParameterValue, with Page Variable Field ParameterValue.Value

Assigned Value – formula with value:

```
self.value
```

        iii. Connect another 'Set page variable' block to the one above. Configure its properties as follows:

Variable name – Page Variable ParameterValues

Assigned Value – formula with value:

```
REMOVE_ITEM_BY_KEY(pageVars.ParameterValues,
"Name", repeated.current.label)
```

        iv. Connect a final 'Set page variable' block to the one above. Configure its properties as follows:

Variable name – Page Variable ParameterValues

Assigned Value – formula with value:

```
WITH_ITEM(pageVars.ParameterValues,
pageVars.ParameterValue)
```

11. Select the dropdown field.

    a. To ensure that the component is displayed correctly, configure the properties as follows:

Label - formula with value:

```
repeated.current.label
```

Option List - formula with value:

```
repeated.current.typeData.dropDownData
```

Selected Value - formula with value:

```
repeated.current.defaultValue
```

Visible - formula with value:

```
repeated.current.typeData.typeName == "Dropdown"
```

    b. To ensure that the data entered is saved correctly, configure the component logic as follows:

        i. Copy the four 'Set page variable' blocks from logic for the input field configured previously.

        ii. Paste them into the logic for the dropdown field and connect the first of them to the 'Component tap' field.

12. Select the slider field.

    a. To ensure that the component is displayed correctly, configure the properties as follows:

Label - formula with value:

```
repeated.current.label + ": " +
(FIND_BY_KEY(pageVars.ParameterValues, "Name",
repeated.current.label).Value)
```

Value - formula with value:

```
repeated.current.defaultValue
```

Minimum Value - formula with value:

```
repeated.current.typeData.minValue
```

Maximum Value - formula with value:

```
repeated.current.typeData.maxValue
```

Visible - formula with value:

```
repeated.current.typeData.typeName == "Number"
```

    b. To ensure that the data entered is saved correctly, configure the component logic as follows:

        i. Copy the four 'Set page variable' blocks from logic for the input field configured previously.

        ii. Paste them into the logic for the slider field and connect the first of them to the 'Component onChange' field.

13. Select the checkbox field.
    a. To ensure that the component is displayed correctly, configure the properties as follows:

    Label - formula with value:
    ```
    repeated.current.label
    ```
    Value - formula with value:
    ```
    repeated.current.defaultValue
    ```
    Visible - formula with value:
    ```
    repeated.current.typeData.typeName == "Checkbox"
    ```

    b. To ensure that the data entered is saved correctly, configure the component logic as follows:
        i. Copy the four 'Set page variable' blocks from logic for the input field configured previously.
        ii. Paste them into the logic for the checkbox field and connect the first of them to the 'Component onChange' field.

14. Below the repeated container, add a button component with a labelled 'Submit'.
15. In the logic for this button, connect five 'Set page variable' blocks in sequence.
    a. Configure the first as follows:

    Variable name – Page Variable ParametersGivenValue

    Assigned Value – formula with value:
    ```
    BUILD_OBJECT(pageVars.ParameterValues, "Name", "Value")
    ```
    b. Configure the second as follows:

    Variable name – Page Variable ParameterNamesGivenValue

    Assigned Value – formula with value:
    ```
    KEYS(pageVars.ParametersGivenValue)
    ```
    c. Configure the third as follows:

    Variable name – Page Variable AllParameters

    Assigned Value – formula with value:
    ```
    BUILD_OBJECT(pageVars.FormSpecification, "label",
    "defaultValue")
    ```
    d. Configure the fourth as follows:

    Variable name – Page Variable AllParameters

    Assigned Value – formula with value:
    ```
    OMIT_KEYS(pageVars.AllParameters,
    pageVars.ParameterNamesGivenValue)
    ```
    e. Configure the fifth as follows:

    Variable name – Page Variable AllParameters

    Assigned Value – formula with value:
    ```
    MERGE([pageVars.AllParameters,
    pageVars.ParametersGivenValue])
    ```

16. Connect an 'Alert' block to the sixth block and configure its properties as follows:
    Dialogue title - Text with value 'Entered Values'
    Dialogue message – Page Variable AllParameters

17. Save the page and launch the app as in the preview portal.

18. Log into the app and navigate to the page from the service requests page. It should look like the below, with the set of input fields that are listed in the JSON specification.

Make

| Ford | ⌄ |

Condition

No issues

☑ Tax Required?

Value: undefined

| Submit |

19. Enter some values, and press 'Submit'. A dialog will be displayed showing the variable 'AllParameter', which contains the details that were entered.
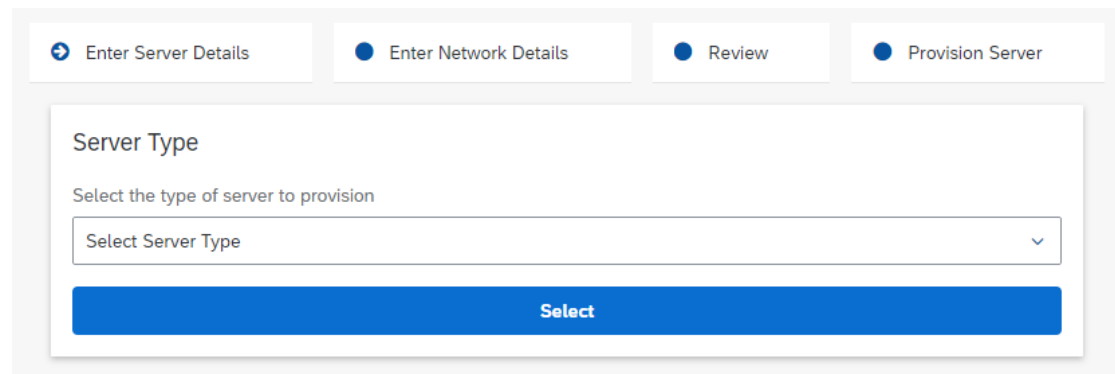
🏳 *Possible extensions to this implementation:*

- *Try re-ordering or adding new the elements in the specification JSON entered in step 4 above.*

  - *See how page changes when previewed without any changes to the page in the UI Canvas.*

  - *See how the data stored in the AllParameters variable changes once data are entered.*

- *Try returning the specification JSON from a Cortex flow.*

  - *Build a flow that returns a different specification JSON based on an input variable.*

  - *In this way, Cortex can define what data needs to be sent to it by a user without any changes being required in AppGyver.*

4.1.4 Progress Bar

During a UI driven process, where a user is guided through a sequence of steps with a sequence of pages, it can be worthwhile highlighting the point in the process that a user is currently at.
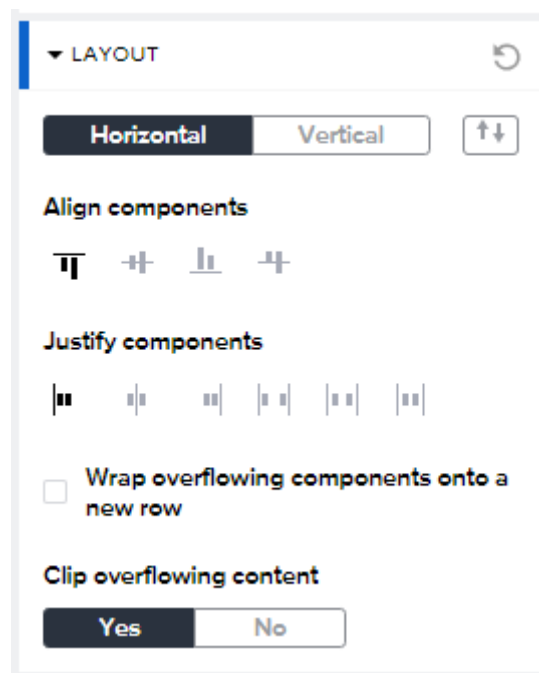
For example, in the 'Server Provisioning' example:

Complete the following steps to implement the above progress bar:

1. Create an App variable called '<process>Pages'. In this example, this will be referred to as the 'Pages' variable.
2. Insert a container component to the top of the first UI page in the sequence of Ui Driven Process pages.
3. With the container selected, go to the 'Layout' tab, and change the layout to 'Horizontal'.

4. Inside this container, add an 'Icon list item' component. Configure its properties as follows:

    Icon name – formula with value:

    ```
    IF(repeated.current.CurrentPage, {set: "fontAwesome", name:
    "arrow-circle-right"}, {set: "fontAwesome", name: "circle"})
    ```

    Assigned Value – formula with value:

    ```
    repeated.current.DisplayName
    ```

    Repeat with – 'Pages' App variable created in step 1

5. In the Page Layout logic, attach a 'Set App Variable' block to the 'Page Mounted' event.
    a. Bind the 'Variable name' property to the 'Pages' App variable.
    b. Bind the 'Assigned value' property to a 'List of values'.
    c. Add the following element.

    d. For each of the other pages in the process, add another element with their name, CurrentPage as 'False', and a display name.

6. For each of the other pages:
    a. Copy the container containing the icon list to the top
    b. Copy the set variable block and attach it to that page's 'Page Mounted' event.
    c. In the set variable block:
        i.   Ensure that every element has CurrentPage set to 'False'.
        ii. Set CurrentPage to 'True' in the element representing this page.

4.2   Logic

4.2.1   Debugging logic

Cortex Gateway has advanced features for debugging flow logic, however there are challenges when debugging logic defined in AppGyver. Here, some ways to go about this are explored.

4.2.1.1   Dialog Blocks

If there is some complex logic associated with a component, then there may be multiple variables having their values changed during its execution. At times, it may not be obvious what the value of each variable is or at what point in the execution certain actions are occurring.

In these scenarios it is useful to use Dialog blocks to display the values of variables, or to show what stage of the logic the current execution has reached.

For example, consider an example similar to the dynamic forms discussed in Section 4.1.3.
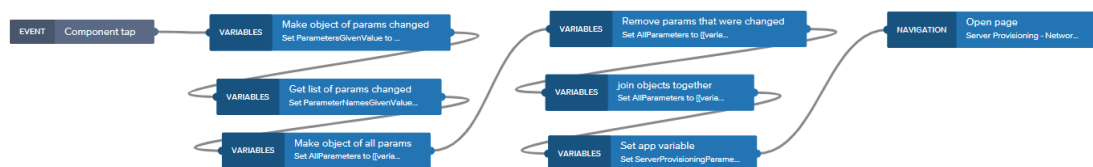
Here, there are multiple variables being manipulated, and if anything is not working as expected, then it may not be obvious where the issue lies. It may be helpful to check the value of the variable 'AllParameters' at a certain point during the logic execution. A Dialog block may be used for this, like the below:
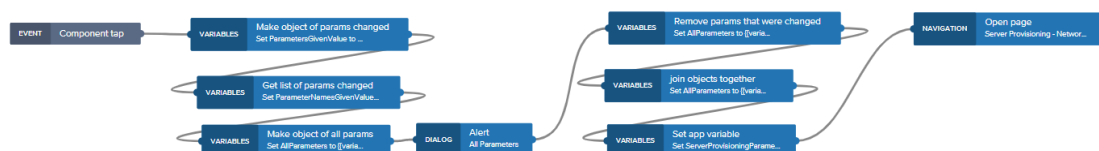
This new block has been configured to display a small window the user showing the value of the variable 'AllParameters' at this point in the logic's execution.


4.2.1.2   Chrome Developer Tools

Calling a Cortex flow from AppGyver can be challenging to debug, but if a Google Chrome-based browser is being used to view the Cortex Interaction Portal then the developer tools can be useful to gain better insight into what is occurring.

1. Start the Cortex Interaction Portal and log in.
2. Press the F12 to open developer tools.
3. Navigate to the Network tab, where a timestamp event will be occurring every second.
4. On the Cortex Interaction Portal, select Service Requests
5. Among the timestamps, other events will be shown, where the UI is calling Cortex flows.

6. If any requests have failed, they will be highlighted in red.

7. Selecting a request will show details of exactly what happened, including request headers, the body of the request, and the response received from Cortex.
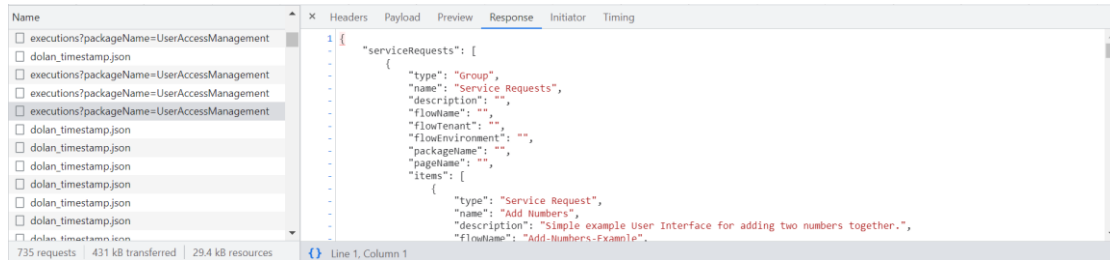
5   Best Practises

In this section there are some standards and guidelines it is recommended that people wishing to use AppGyver and Cortex together should adhere to.

5.1   AppGyver Development

5.1.1   Variables

5.1.1.1   App variables

These are to be used only for config or global app data that is required by multiple independent pages.

If App Variables are the only way to achieve something (e.g. storing data throughout the life of a Service Request) they can be used. However, ideally this data should be maintained in Cortex, using a GUID as a Page Parameter which can then be used to retrieve it from Cortex.

5.1.1.2   Page variables

These are to be used for any items of data required by a page.

All Process-Driven UI task UIs should have a Page Variable called 'TaskParameters' that can take any value. TaskParameters stores all data sent by the UI which can be displayed on the page.

5.1.1.3   Page parameters

These are to be used for one page to pass the minimum required data to another when navigating to it.

5.1.2 Logic

5.1.2.1 Flow Functions

Repeated functions should be wrapped in a flow function with inputs outputs as required.

5.1.2.2 Auth Token Validation

The 'Validate Token' flow function should be called first whenever a page loads or is navigated to, and whenever an action is taken.


5.1.2.3 Spinner Logic

The spinner icon is shown while pages load, so it is good practise to have it showing while pages are loading to ensure a smooth transition.
Consider when a button is clicked to navigate to a new page. The spinner should be shown immediately.
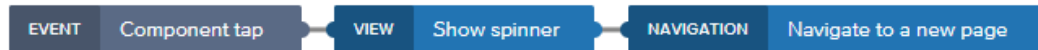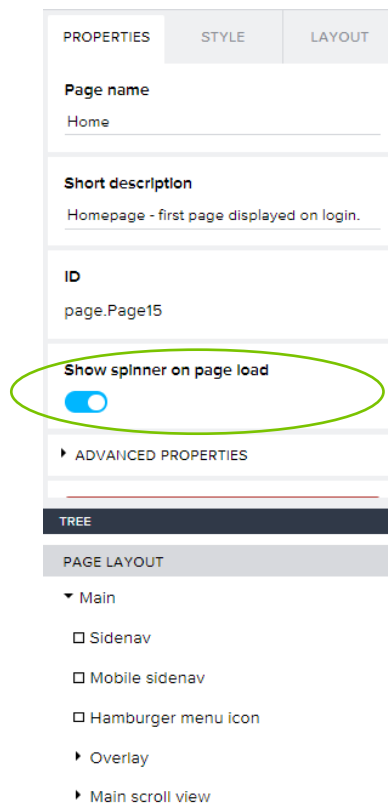

Then, on the new page, the page mount logic should validate the authentication token and perform all actions needed to populate the page before hiding the spinner to show the newly loaded page.
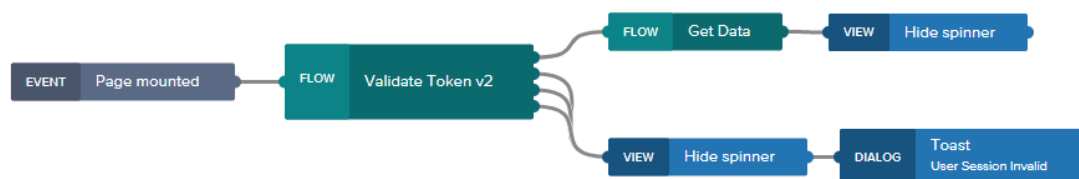
> ⚐ *The spinner should also be hidden before any error notifications are popped up.*

As well as this, ensure that the 'Show Spinner on Page Load' property is toggled on. This can be found in the right-hand panel when 'Page Layout' is selected from the page tree.

This ensures that from a user's perspective, they select the button to navigate to a new page, then the spinner is displayed right until a fully loaded page is displayed to them.

5.1.3   Ui Pages

5.1.3.1   Page Naming Conventions

Service Requests should be named according to the following scheme:

```
<Service Request Name> - <Task>
```

For example: Server Provisioning – Enter Server Details

> *Note that the page name is not the same as the 'Service Request Name' which is defined when managing the Service Requests (Admin function).*

Process UIs should be named according to the following scheme:

```
UI - <ProcessName> - <TaskName>
```

For example: *UI - Onboarding – Approval*

> *Note that the page name is not the same as the 'Task UI Name' which is used for mapping the task to the individual page.*

5.1.3.2   Other Best Practises

All pages should be created by copying the entirety of the 'Page Template' page and building the page functionality in the 'Content' container.

For an example of this being done, see section 3.1.2.

5.2 Cortex Development

5.2.1 Flow Hierarchy

The Cortex-Library flow group contains the following relevant flow groups:
1. **Cortex Interaction Portal** is designed for core functionality and examples.
2. **Manual Intervention** contains flows related to managing process and task execution and user interactions.
3. **User Access Management** contains flows related to authentication with Active Directory. This may be re-used in other modules as needed.

Any bespoke flows for UI interactions should be in a separate top-level group relevant to the solution or process, with sub-groups as required.

Ideally, all flows relating to a single service request should be within a single group inside this group structure, with the group being named something relevant to the Service Request.

5.2.2 Checking Authentication Tokens

The 'Check Token' flow should be called whenever a flow is triggered from AppGyver, ensuring that the provided token is still valid.
1. Therefore, AuthToken should be an input to every called flow.
2. If it is valid the flow should continue.
3. If not, it should return an exception in a consistent manner which will instruct AppGyver to return to the Login screen.
4. This should also validate that the user has the appropriate access: for most requests, either 'User' or 'Admin'.

5.2.3   Exceptions

Flow exceptions should generally be handled within the flow, triggering an exception to go to the 'Handle Flow Exception' workspace.

In this workspace, more logic can be added to generate a useful response based on the exception / status.

This should then be followed with a Throw New Flow Exception block which will ensure that the exception response is sent back from the API call.

This can include a status, message, code (if relevant), etc - the main thing is that AppGyver will get the response as an exception, allowing the AppGyver logic to branch accordingly.

For example: