

eBook

# The Ultimate Guide to Building Applications with FlowFuse Dashboard for Node-RED

Published June 2024

# Introduction

*“Easy things should be easy, and hard things should be possible” is the premise of the FlowFuse Dashboard for Node-RED*

For years, the original Node-RED Dashboard has been a cornerstone for building user-friendly dashboards for Node-RED flows. While it served its purpose well enough, its underlying technology (Angular v1) is no longer actively maintained. This presented an opportunity to create something better. Rebuilt from the ground up, Flowfuse Dashboard, incorporates the best features and feedback from its predecessor, to provide a platform that makes it easy to visualize and interact with your data like never before.

Whether you're a seasoned developer or just getting started, Flowfuse Dashboard empowers users to build full-stack applications. The Dashboard intuitive interface allows you to drag and drop widgets, buttons, and other elements to create rich UIs without writing a single line of code. This significantly reduces development time and allows non-technical users to build powerful applications. Going beyond simple data visualization, FlowFuse Dashboard lets you create interactive dashboards with features like buttons and user-specific views. This empowers users to explore data, filter results, and take actions directly within the dashboard.

Node-RED actively fosters a growing community of developers. This means a constantly expanding library of widgets and nodes, ensuring you have the tools to build any dashboard you can imagine. Like the original, the FlowFuse Dashboard embraces the open-source spirit and thrives on community collaboration. It remains licensed under Apache 2.0, making it accessible and adaptable to your specific needs.

The Flowfuse team is deeply invested in the Node-RED community and the success of this Dashboard for the Node-RED ecosystem. Joe Pavitt, Head of UX & Design at FlowFuse, was instrumental in developing the original Node-RED Dashboard. This experience ensures a product that is purpose-built for Node-RED and leverages the strengths of the platform.

In this ebook, we will touch on topics that will show you how to easily build stunning dashboards that unlock the power of your data and take your Node-RED applications to the next level.

# Table of Content

Installing and Setting up FlowFuse Dashboard	3
Creating a Data Chart	11
Exploring Dashboard Widgets	16
Styling Your Dashboards	28
Adding Custom Rich Components to Your UI	32
Personalised Multi-User Dashboards	34
Displaying Logged-in Users	41
About FlowFuse	44

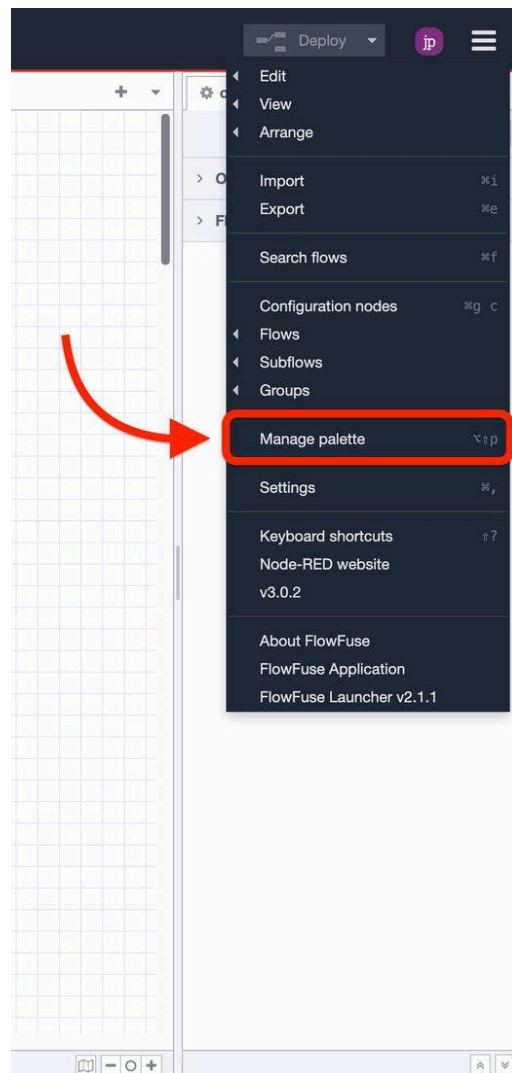
## Chapter 1

# Installing and Setting up the Dashboard

It only takes a few steps to get started with [FlowFuse](#)'s Node-RED Dashboard which is available in the Node-RED Palette Manager.

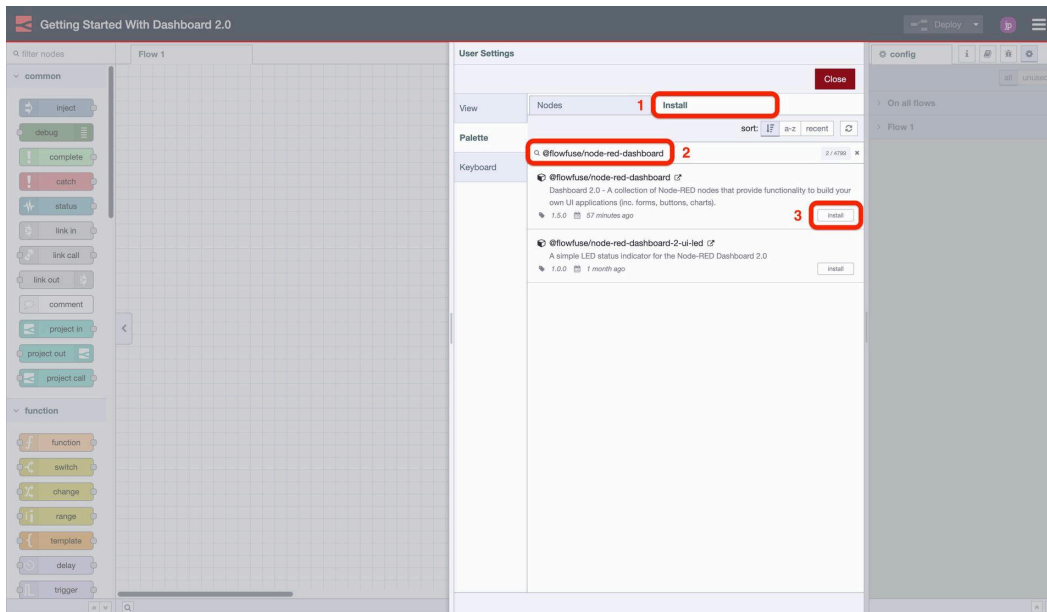
### Step 1: "Manage Palette"

1. Click the Node-RED Settings (top-right)
2. Click "Manage Palette"



## Step 2: Search & "Install"

1. Switch to the "Install" tab
2. Search for `@flowfuse/node-red-dashboard`
3. Click "Install"



## Step 3: Adding your first widgets

With the nodes installed, getting started is as easy as choosing a node from the Palette (the left-hand side list of nodes) in Node-RED, and dropping it onto your canvas.

You can drop in a **ui-button**, click "Deploy" and then see the button running live in your user interface.

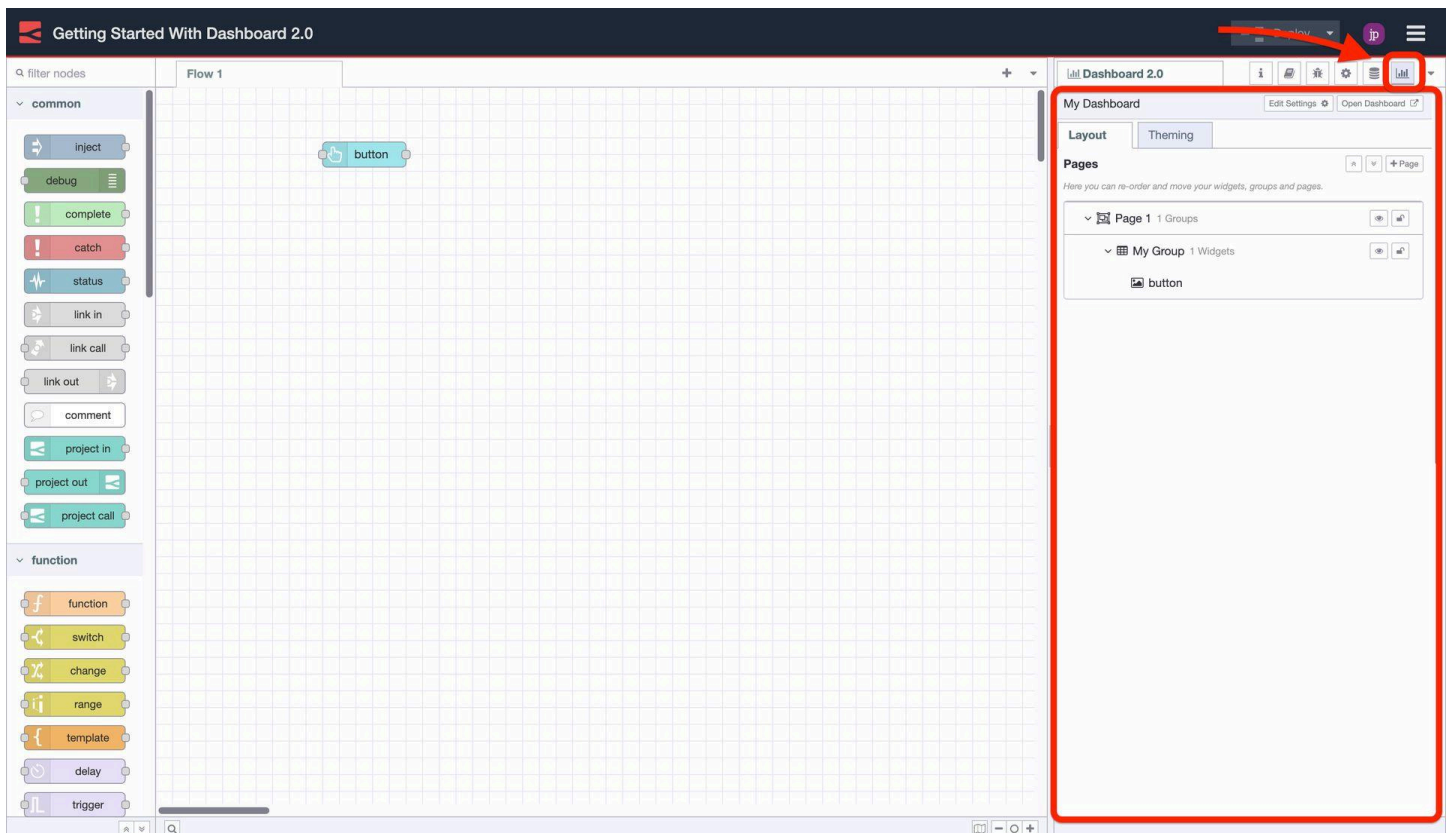
The Dashboard will automatically setup some underlying configurations for you (visible in the right-side menu):

- **ui-base**: Each instance of Node-RED that uses the Dashboard must have a single **ui-base** element (we're hoping to add support for multiple in the future). This element contains all of the global settings for your Dashboard instance.

- **ui-page:** A single Dashboard (**ui-base**) can consist of multiple pages, and can be navigated to using the left-side sidebar. Each page is then responsible for displaying a collection of **ui-group** elements.
- **ui-group:** Each group contains a collection of widgets, and can be used to organize your Dashboard into logical sections.
- **ui-theme:** Each **ui-page** can be assigned a given theme. Your "Themes" provide control over the aesthetic of your Dashboard like color, padding and margins.

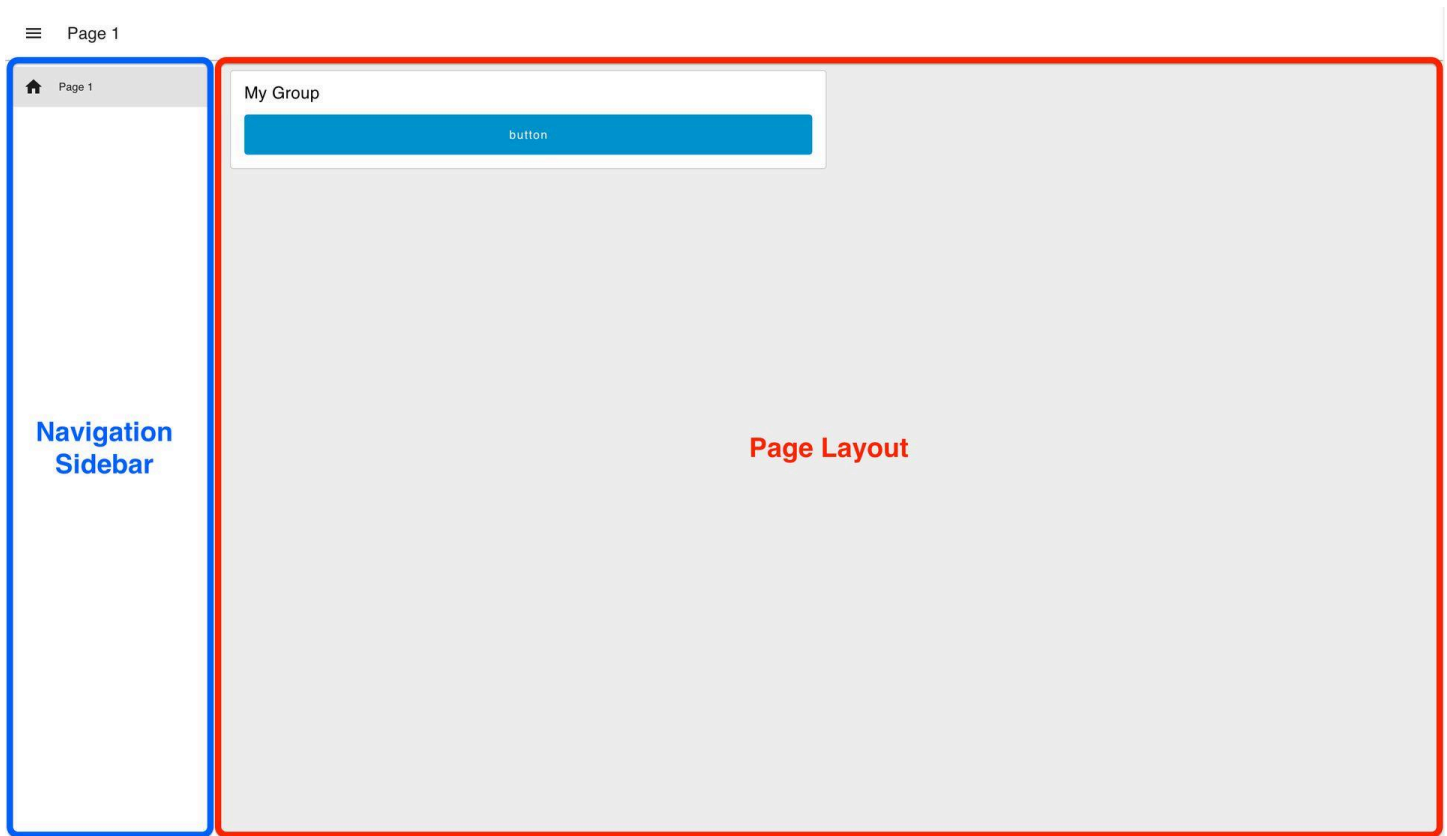
## Step 4: Configuring your layout

The Dashboard adds a dedicated sidebar to Node-RED to provide a centralized view of your pages, groups and widgets. From here you can add new pages and groups, modify existing settings, and re-order content to your liking.



When defining your layout options, we break the choice into two sections:

- **Page Layout:** Controls how the ui-groups are presented on a given page in your application.
- **Navigation Sidebar:** Defines the left-side navigation style, defined at the ui-base level.



## Page Layout

Currently, there are three different options for page layout:

- **Grid:** This is the default layout for a page, and uses a 12-column grid system to layout your ui-groups. Widths of groups and widgets define the number of columns they will render in. So, a "width" of 6" would render to 50% of the screen. Grid layouts are entirely responsive and will adjust to the size of the screen. [More details on Grid can be found here](#)
- **Fixed:** Each component will render at a fixed width no matter the screen size. The "width" property is converted to a fixed pixel value (multiples of 48px by default). [More details on Fixed can be found here.](#)
- **Notebook:** This layout will stretch to 100% width, up to a maximum width of 1024px, and will centrally align. It is particularly useful for storytelling (e.g. articles/blogs) or analysis type user

interfaces (e.g. Jupyter Notebooks), where you want the user to digest content in a particular order through scrolling. [More details on Notebook can be found here.](#)

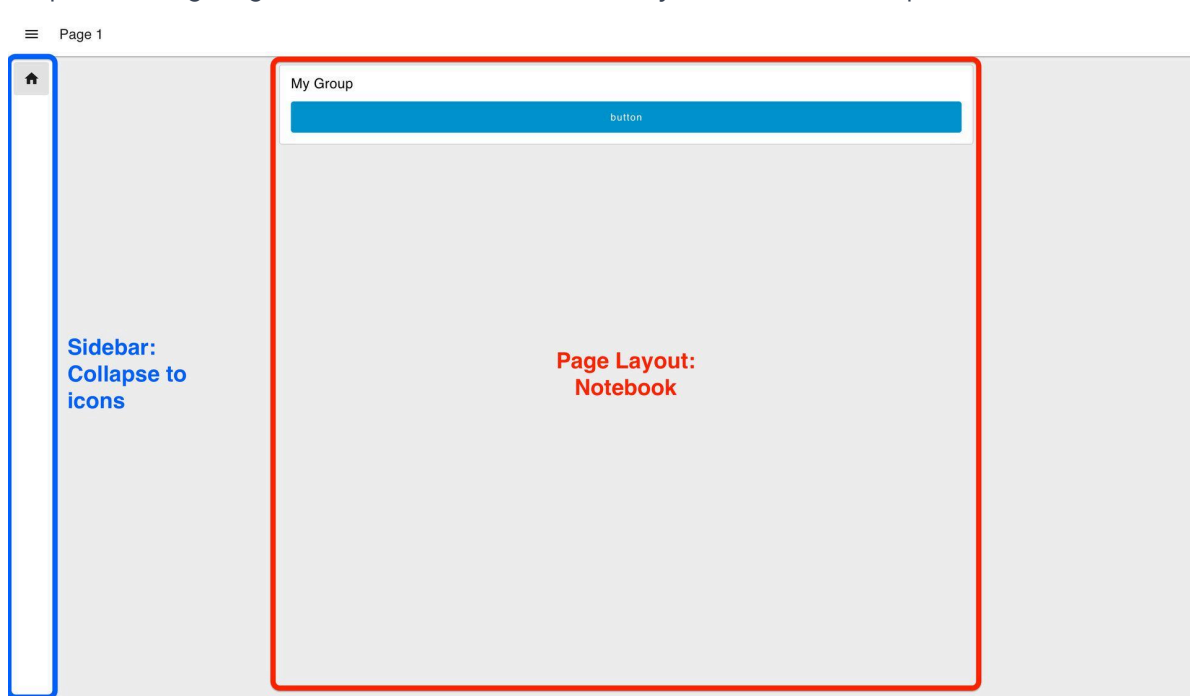
## Navigation Sidebar

The Dashboard offers various options for the appearance of the navigation sidebar:

- **Collapsing:** When the sidebar is opened the page content will adjust with the width of the sidebar.
- **Fixed:** The full sidebar will always be visible, and the page content will adjust to the width of the sidebar.
- **Collapse to Icons:** When minimized, users can still navigate between pages by clicking on the icons representing each page in the sidebar.
- **Appear over Content:** When the sidebar is opened, the page is given an overlay, and the sidebar sits on top.
- **Always Hide:** The sidebar will never show, and navigation between pages can instead be driven by [ui-control](#).

## Define Your Layout

In our example, we're going to switch to a "Notebook" layout, with a "Collapse to Icons" sidebar:





## Step 5: Adding More Widgets

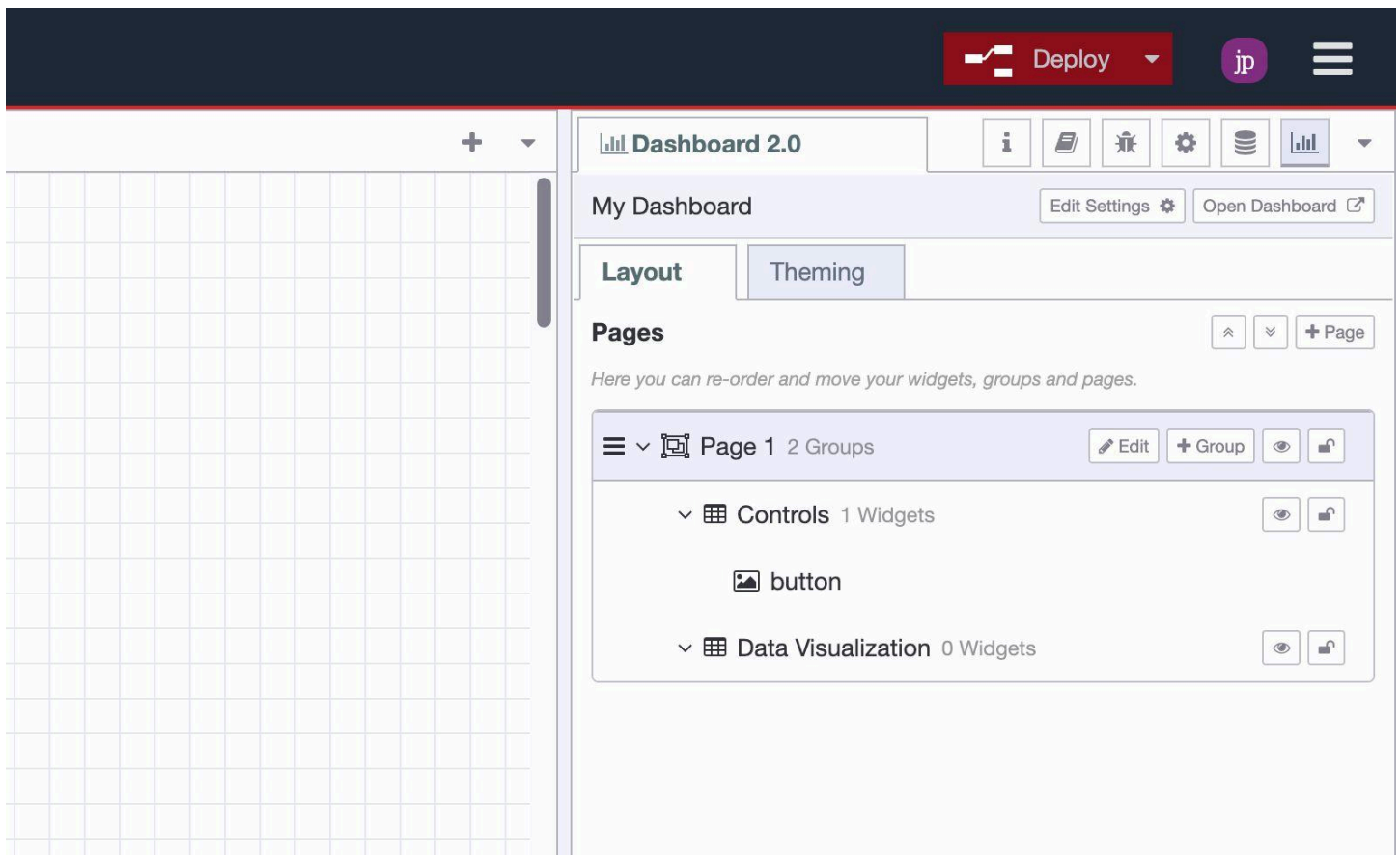
Now, let's build a quick example to demonstrate how we can wire nodes together, and visualize the output from a **ui-slider** onto a **ui-chart**.

### Adding a Group

In the Node-RED Editor's Dashboard sidebar, we're going to do the following:

1. Edit "My Group" and rename it to "Controls"
2. Create a new "Group" in your existing page called "Data Visualization"

You'll now see the two groups listed under "Page 1". "Controls" with a single **ui-button** and "Data Visualization" with no widgets.

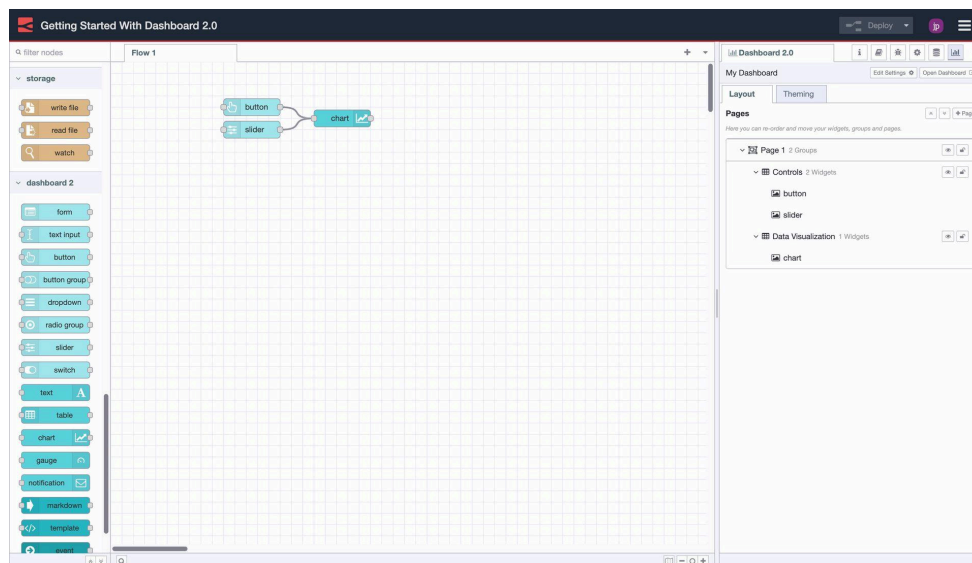


## Connecting New Nodes

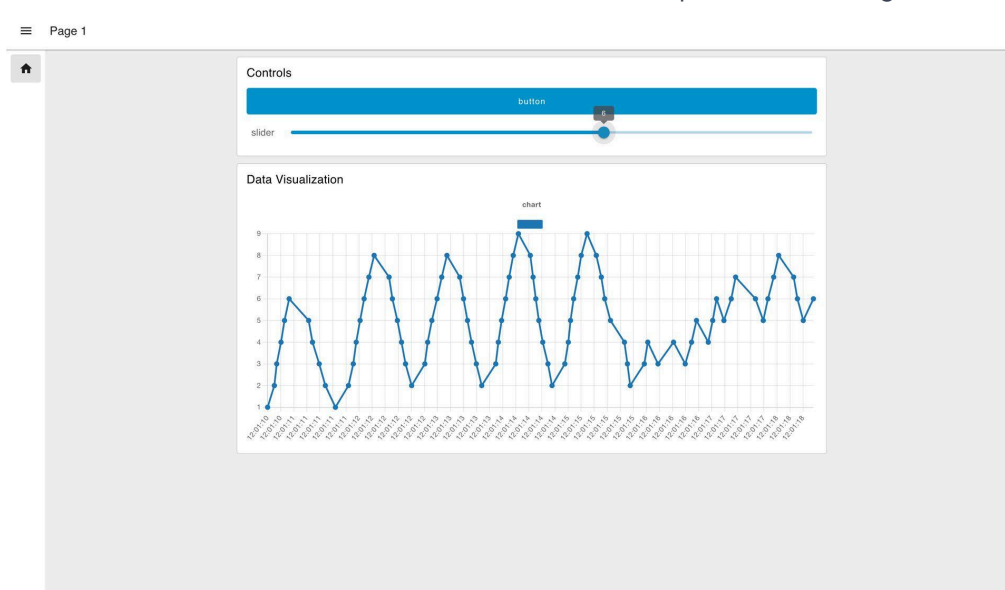
Now we will add two new widgets; UI Chart and UI Slider, which we can do by dropping them from the left-side Palette and onto our canvas.

We'll need to double-click each new node and confirm which "Group" we want to add this node into. In this case, we'll add the ui-slider to the "Controls" group, and the ui-chart to the "Data Visualization" group.

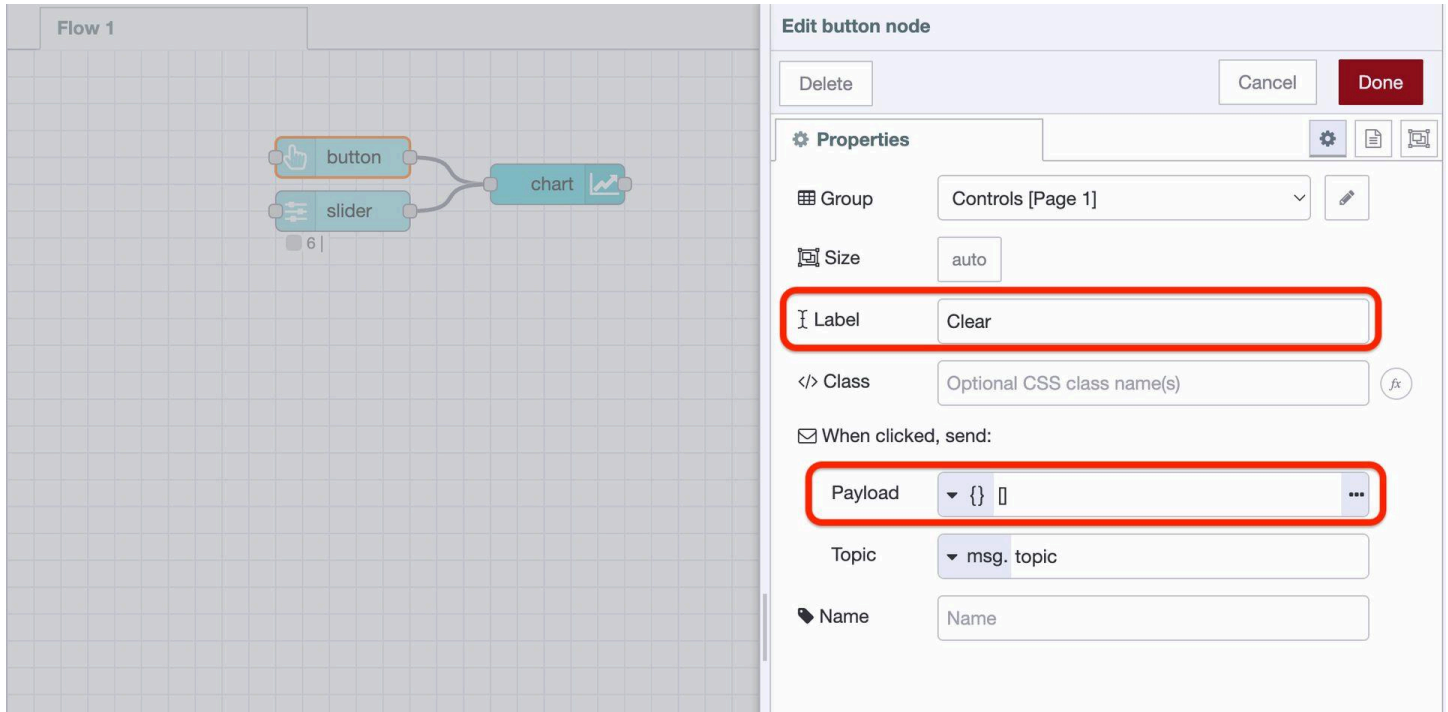
We're also going to connect the output from both the ui-slider and ui-button to the input of the ui-chart:



Now, when we view our Dashboard, we can see the ui-slider output is def straight into our ui-chart:



The final step we're going to make is to modify our ui-button. We're going to rename it to "Clear", and configure it's "Payload" option to send a JSON payload of `[]`, which, when sent to the ui-chart will clear the chart of all data.



With all of this together, you have a functional Dashboard!

## Chapter 2

# Creating a Data Chart

There are many different ways to get data for a Node-RED dashboard. One common way is to use a REST API.

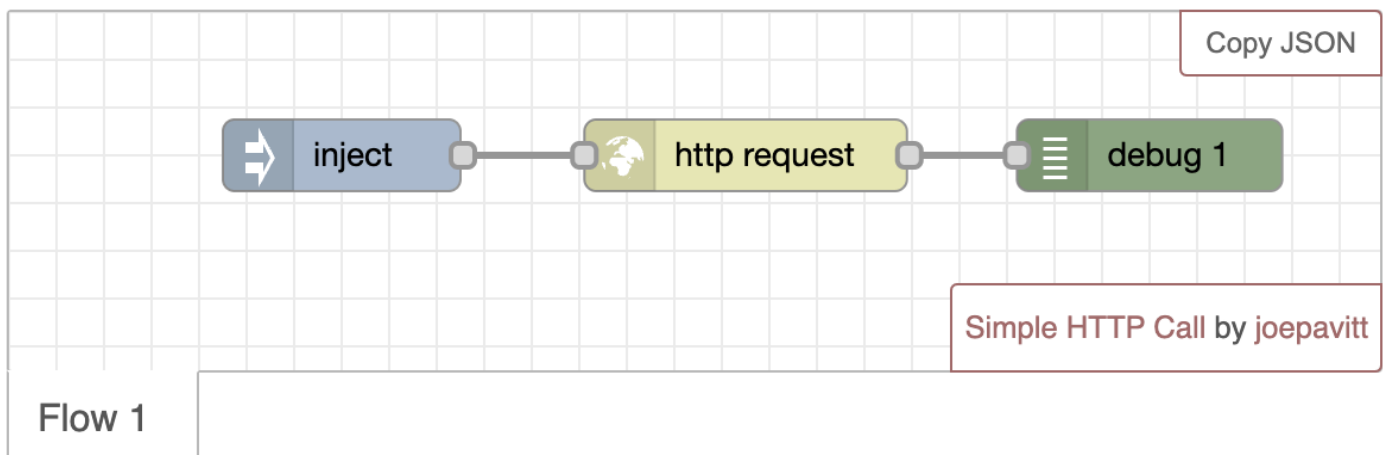
A REST API is a set of web services that allow developers to interact with a server and its resources. To get data from a REST API, you can use the HTTP nodes in Node-RED to send HTTP requests to the API and receive the data in the response. Below is a quick guide of the process.

First, let's consider what data we want to get. For a dashboard chart it's nice if there were at least two dimensions to it. In this case, we'll create a chart for the number of downloads for a certain NPM package.

### Getting the data

Reading the data for a package is done through a HTTP GET request, for example: <https://api.npmjs.org/downloads/range/last-month/@flowforge/node-red-dashboard>.

A simple flow to achieve this would be



Where we paste the URL from above into the settings panel:

### Edit http request node

DeleteCancelDone

#### Properties

Method

GET

URL

https://api.npmjs.org/downloads/range/last-mont

Payload

Ignore

When running this flow you'll see a blob of text in the Debug pane. This is a great first start, but a blob isn't useful for the rest of the flow.

We need to parse the data as JSON. While the [JSON node](#) would work, the HTTP request node can do this natively. Let a parsed JSON object the Return settings of the HTTP request node.

So now we got the data, and a little more than we need, so let's change the message output to keep only what we're interested in; payload.downloads. To do this, we'll use the [change node](#).

#### Rules

Set

▼ msg. payload

to the value

▼ msg. payload.downloads

☐ Deep copy value

## Building the Dashboard

Once you have your Dashboard up and running, drag in the chart node that's available after installing the dashboard package and make sure it's input comes from the configured change node.

Before hitting the deploy button the dashboard itself needs configuring:

First add configuration for the ui-group:



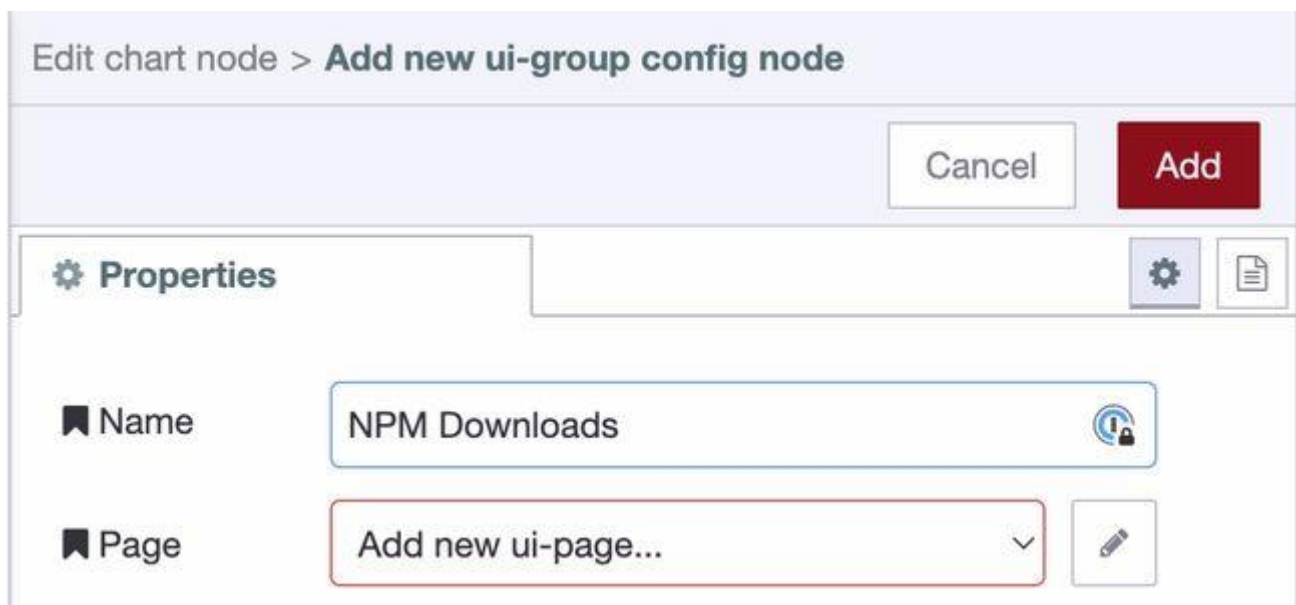
**Edit chart node**

Delete Cancel Done

**Properties**

Group Add new ui-group...

To setup the ui-group correctly you'll need to add configuration for the ui-page:



**Edit chart node > Add new ui-group config node**

Cancel Add

**Properties**

Name NPM Downloads

Page Add new ui-page...

To create the UI page it requires another 2 config settings, ui-base, and the theming through ui-theme.

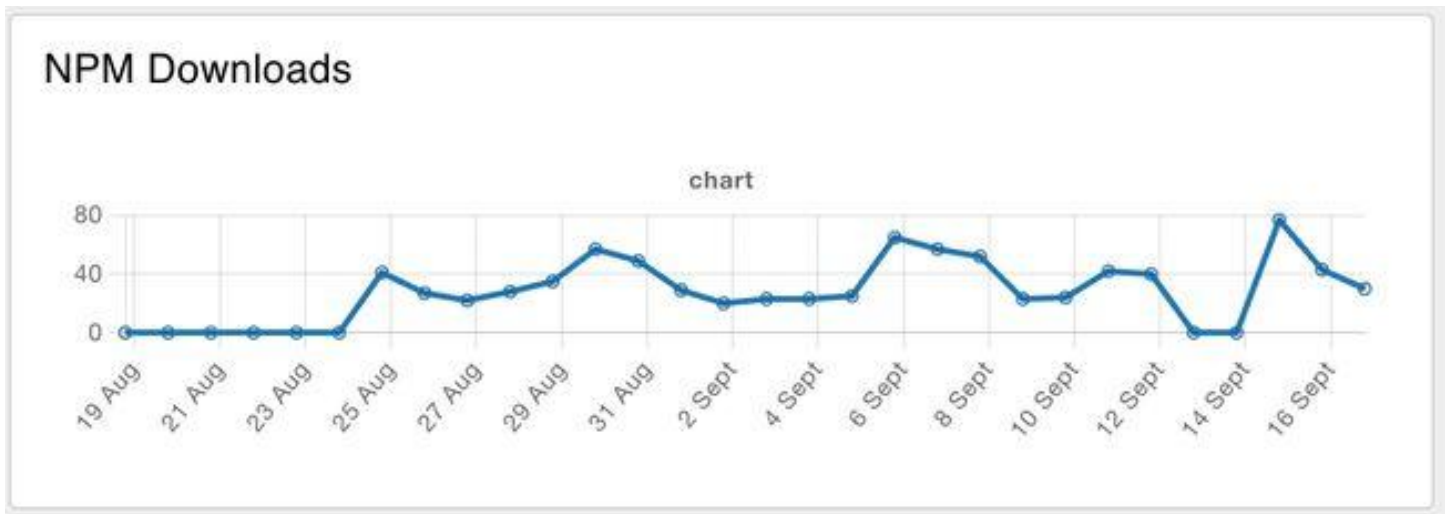
The default theme is great, so just accept that, and save all dialogs to continue the chart creation.

## Normalizing the data

The data for the chart needs to be changed before we can show it. The messages should have a x and y key. So let's prepare the data with a combination of the [Split](#) and change node.

The Split node with the default configuration allows to 30 elements of the array to be mapped individually. The change node will set the payload.x and payload.y on the message:

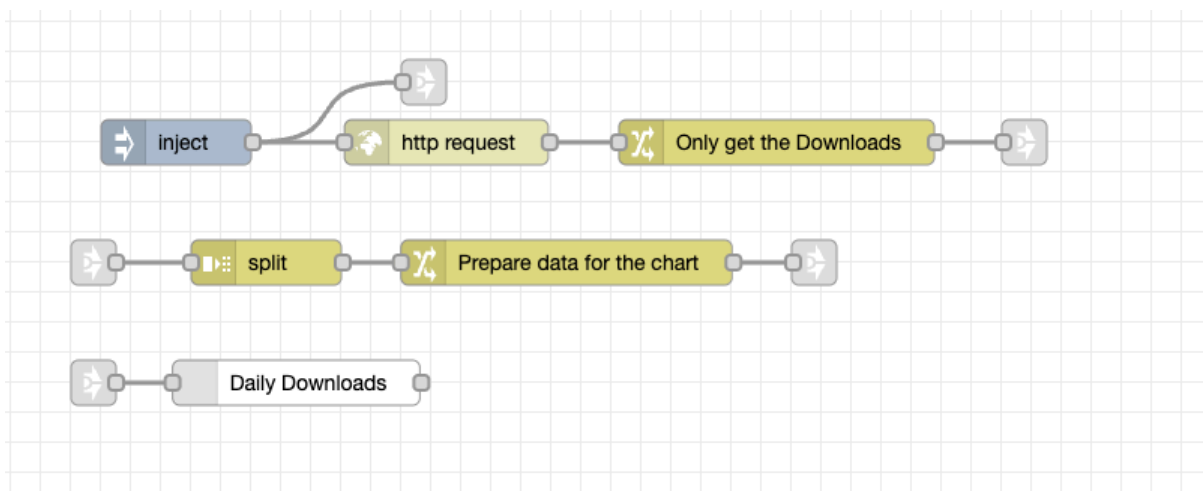
Connect the change node output to a new chart node, and voila:



### Keeping the data up-to-date

While we created a chart and it has some data, there's one more thing to explain. How can the data be kept up-to-date? It's straight forward to have the Inject node [run every night](#), but the chart would now have multiple data points for the same day. This paints multiple lines on top of each other. While that works, the hover of the chart will display the duplication and it's wasteful.

So before we update the chart we need to send a message to the chart where the [payload is \[\]](#). That way the chart is emptied first, and right afterwards it will receive the new data to write.





# Exploring Widgets

Widgets are the building blocks for creating a user interface in the Dashboard. There are a variety of widgets like forms, templates, buttons, and others to make different parts of your interface.

Let's create a basic application to input expenses and income. This will then be displayed in a chart and table for analysis. The application will utilize a wide range of widgets available in the Dashboard.

## Adding Forms

For the income and expense submission, we'll incorporate a form using the ui-form widget.

1. Drag the ui-form widget onto the canvas.
2. Double-click on it to access various widget properties and select the ui-group where it should render.
3. Add "date", "description", "amount", and "note" form elements by clicking the +element button at the bottom left.

**Edit form node**

Delete Cancel Done

**Properties**

Name: Income Submission Form

Group: Income Submission Form [New Income]

Size: auto

Label: optional label

Class: Optional CSS class name(s)

**Form elements**

Label	Name	Type	Required	Rows	Remove
Date	date	Date	<input checked="" type="checkbox"/>		
Description	description	Text	<input checked="" type="checkbox"/>		
Amount	amount	Number	<input checked="" type="checkbox"/>		
Note	note	Text	<input type="checkbox"/>		

+ element

**Buttons**

submit clear

☐ Place the form elements in two columns

Enabled

Once you've added the income submission form, repeat the process to add an expense submission form on another ui-page and ui-group. For more information on ui-form, refer to the [ui-form docs](#).

**Edit form node**

Delete Cancel Done

**Properties**

Name: Expense Submission Form

Group: Expense Submission Form [New Expense]

Size: auto

Label: optional label

Class: Optional CSS class name(s)

**Form elements**

Label	Name	Type	Required	Rows	Remove
Date	date	Date	<input checked="" type="checkbox"/>		
Description	description	Text	<input checked="" type="checkbox"/>		
Category	category	Text	<input checked="" type="checkbox"/>		
Amount	amount	Number	<input checked="" type="checkbox"/>		
Note	note	Text	<input type="checkbox"/>		

+ element

**Buttons**

submit clear

☐ Place the form elements in two columns

Enabled

## Storing Form Data

The ui-form widget emits a payload object with key-value pairs of form elements upon submission. We'll store this data in a global context, If you are not familiar with Node-RED context, refer to [Understanding Node-RED](#) variables.

1. Drag a function node onto the canvas and add the following code. This will store the submission in the income global context variable, and then modify msg.payload to pass on a notification to any further connected nodes.

JavaScript

```
// Retrieve the existing 'income' array from the global context, or initialize it as an empty array if it doesn't exist  
  
let income = global.get('income') || [ ] ;
```

```
// Push the incoming payload along with a 'type' property set to "income" into the 'income' array

income.push({
  ...msg.payload,
  type: "income",
});

// Store the updated 'income' array back into the global context
global.set('income', income);

// Set the message payload to a confirmation message for notification
msg.payload = "Thank you for submitting income!";

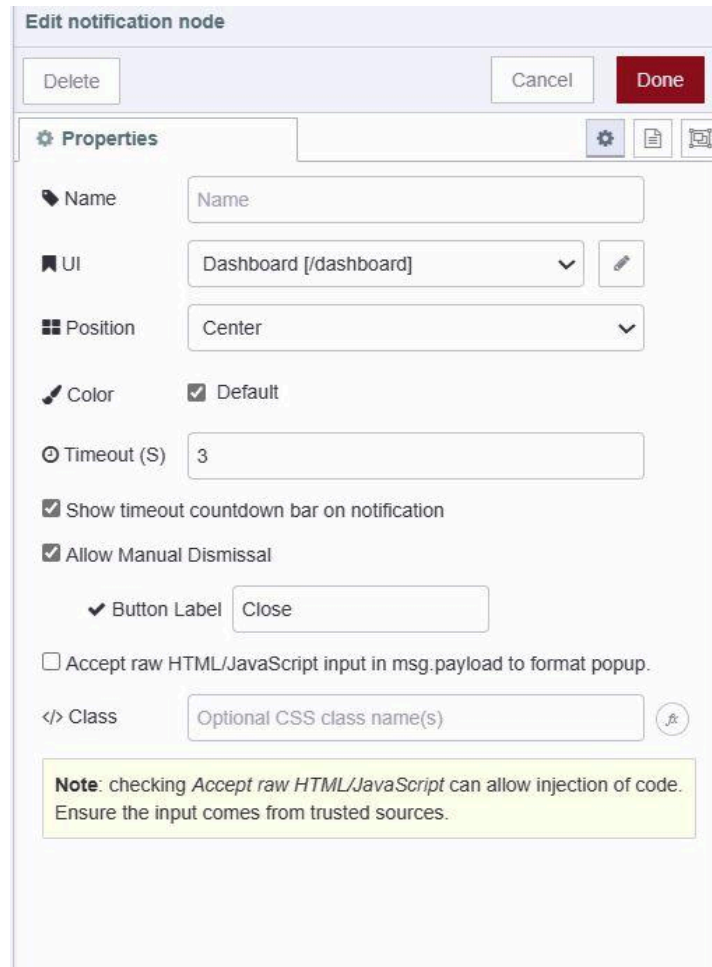
// Return the modified message
return msg;
```

Similarly, you can do this for storing expense data submitted using the expense submission form.

## Displaying Notifications

For displaying notifications on the dashboard, we'll utilize the ui-notification widget, which emits notifications to the user's dashboard. It accepts msg.payload which should be a string format or raw HTML/JavaScript for custom formatting.

1. Drag the ui-notification widget onto the canvas.
2. Set the position property to center. You can also adjust colors or notification timeout by modifying the color and timeout properties. Please take a look at the [ui-notification docs](#) for more information on ui-notification.



**Edit notification node**

Delete Cancel Done

**Properties**

Name

UI  ▼

Position  ▼

Color ☒ Default


Timeout (S)

☒ Show timeout countdown bar on notification

☒ Allow Manual Dismissal

☒ Button Label

☐ Accept raw HTML/JavaScript input in msg.payload to format popup.

</> Class  

**Note:** checking Accept raw HTML/JavaScript can allow injection of code. Ensure the input comes from trusted sources.

## Listening for events

In the Dashboard, the ui-event widget allows you to listen to user behavior or events. It does not render any content or components on the dashboard. Currently, this widget only listens for page views (\$pageview) and leave (\$pageleave) events.

With this, we can listen for page view and page leave events and trigger tasks based on those events. For instance, in our application, we will be displaying a table containing income and expense data, along with a chart. We'll update them when navigating to a new page or leaving a page.

1. Drag an ui-event widget onto the canvas.
2. Double-click on it and select the correct ui-base of your application.

For more information on ui-event refer to [ui-event docs](#).

## Retrieving Income-Expense Data

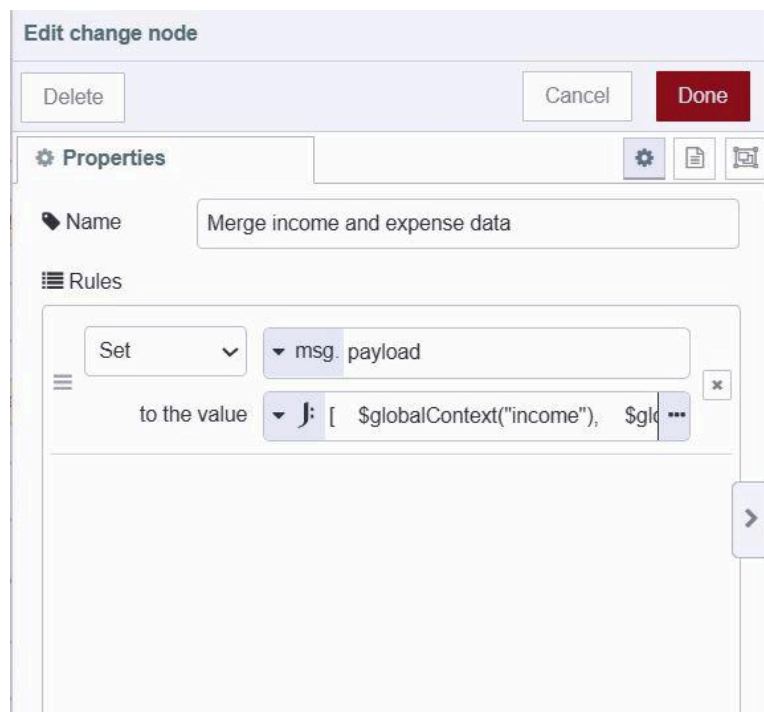
In our income-expense application, we will display the income and expenses in a single table.

1. Drag a change node onto the canvas.
2. Set msg.payload to the JSONata expression below, which merges the income and expense arrays.

JavaScript

```
[$globalContext('income'), $globalContext('expense')]
```

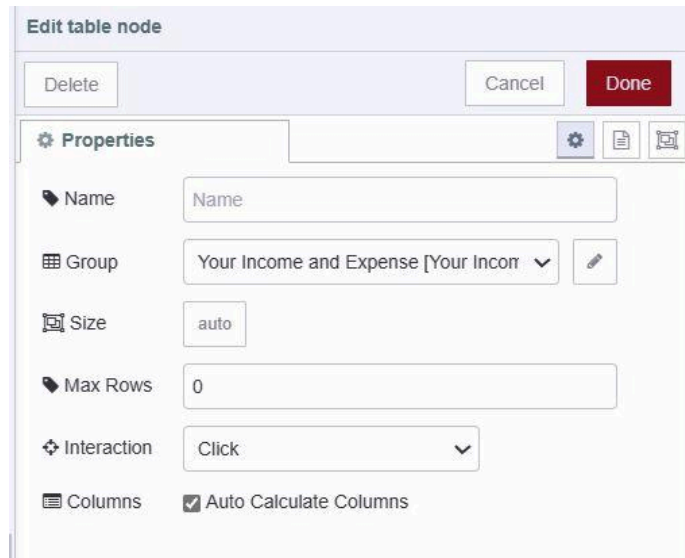
3. Connect the output of the ui-event widget to the input of the change node.



## Displaying Data on the Table

To display data on the table, we use the ui-table widget in the Dashboard. This widget accepts an array of objects as input. The columns in the table correspond to the properties of the objects within the array, and each row represents a different object with values corresponding to those properties.

1. Drag a ui-table widget onto the canvas.
2. Create a new ui-page and ui-group for it.
3. Connect the output of the change node to the input of the ui-table widget.

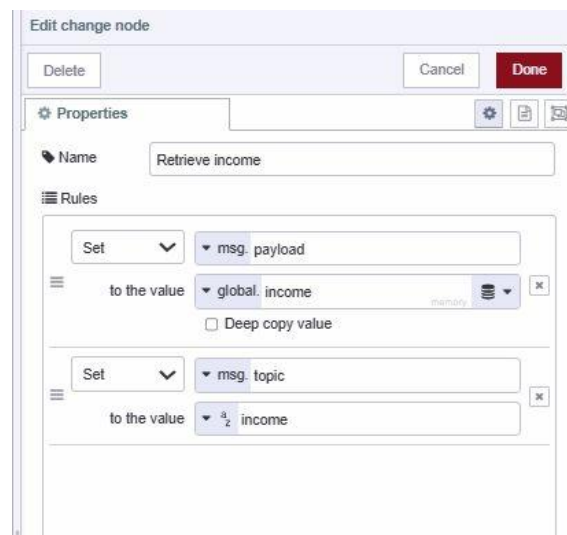


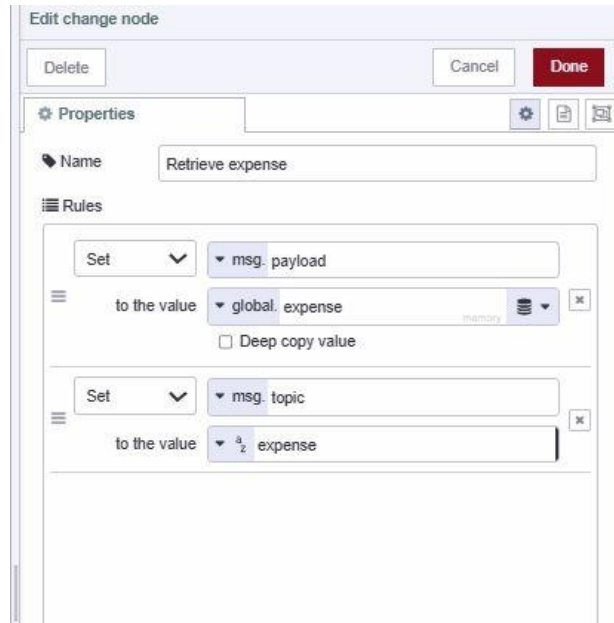
For more information on ui-table refer to [ui-table docs](#)

### Calculating total category-wise

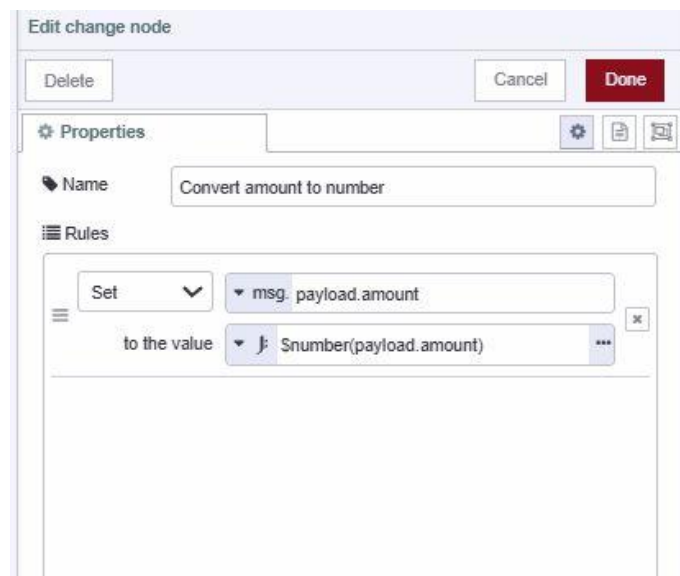
In our application, we will display data on the chart, showing the total income and total expenses for analysis. In this section, we will calculate the total expenses and income using the function node.

1. Drag the two change node onto the canvas.
2. For the first Change node Set msg.payload to global.income and msg.topic to "income" and give it name "retrive income". For the second Change node, set msg.payload to global.expense and msg.topic to "expense" and give that second change node name "retrieve expense".





3. Drag a Split node onto the canvas.
4. Drag the Change node onto the canvas and set msg.payload.amount to the JSONata expression \$number(payload.amount) and give it name "Convert amount to number".



5. Drag a Join node onto the canvas, select mode as reduced expression, and set the Reduce exp to \$A + payload.amount. Set Initial value to 0, and Fix-up exp to \$A. Give this join node the name "Calculate total". This function operates similarly to using the javascript reduce method on an array to calculate the sum of its values. \$A stores the accumulated value, and with every incoming message payload, it adds the payload.amount value to it, for more details on this refer to the [core node docs on join node](#).

Edit join node

Delete Cancel Done

Properties

Mode: reduce sequence

Reduce exp: `$.SA+payload.amount`

Initial value: `0`

Fix-up exp: `$.SA`

☐ Evaluate in reverse order (last to first)

Name: Calculate total

This mode assumes this node is either paired with a *split* node or the received messages will have a properly configured `msg.parts` property.

7. Drag another join node onto the canvas set mode to manual, combine each to complete message, to create to array and After a number of message parts to 2 and give it name "combine two objects into array".

Edit join node

Delete Cancel Done

Properties

Mode: manual

Combine each: complete message

to create: an Array

Send the message:

- After a number of message parts: 2
- After a timeout following the first message: seconds
- After a message with the `msg.complete` property set

Name: Combine two object into one array



8. Connect the output of the ui-event widget to the input of the Change node named "Retrieve Income" and "Retrieve Expense". Then, connect the outputs of the "Retrieve Income" and "Retrieve Expense" Change nodes to the input of the Split node.

9. Next, connect the output of the Split node to the Change node named "Convert Amount to Number". Afterward, connect the output of that Change node to the input of the Join node named "Calculate Total". Finally, connect the output of the "Calculate Total" Join node to the input of the Join node named "Combine Objects into Array".

## Displaying data on the chart

To display charts on the dashboard, we have to use the ui-chart widget which allows us to display different types of charts on a dashboard including linear, bar, scatter, etc. This accepts an array and object as input.

1. Drag a ui-chart widget onto the canvas.
2. Double-click on the widget and select Type as bar.
3. Configure the series to category and the y-axis to amount. This configuration informs the chart that the amount property of the input objects will be plotted on the y-axis and category to the x-axis of the chart.
4. Connect the output of the join node named "Combine Objects into Array" to the ui-chart widget's input.

The screenshot shows the 'Edit chart node' configuration window. It includes fields for Name, Group, Label, Class, Type, Action, X-Axis Type, Y-Axis (min/max), Series (key: category), Y (key: amount), and Series Colors. The 'Type' is set to 'Bar' and 'X-Axis Type' is 'Categorical'. The 'Series' is 'key: category' and 'Y' is 'key: amount'. The 'Action' is 'Append'. There are also 'min' and 'max' input fields for the Y-axis. At the bottom, there is a 'Series Colors' section with a grid of color swatches and an 'Enabled' checkbox.

## Adding custom footer with ui-template

With the ui-template widget, we can add a custom component to our app using Vue.js. It also allows adding custom CSS for the dashboard and lot of other things. For more information refer to [ui-template docs](#).

Using this widget, we will add a footer to our application.

1. Drag an ui-template widget onto the canvas.
2. Set the widget type (scoped UI) that will render this widget on the entire dashboard, eliminating the need to add separate footers for each page of the dashboard.
3. Insert the following vue.js code in the ui-template widget.

JavaScript

```
<template>

<!-- Footer Component -->

<div class="footer">

  <!-- Description of the Income-Expense Tracker -->

  <div>

    Welcome to our comprehensive income expense tracker! Take control of your finances by monitoring your income and expenses effortlessly. Our user-friendly interface makes it simple to record transactions, categorize expenses, and analyze your financial trends. With real-time insights into your spending habits, you can make smarter financial decisions and achieve your money goals faster.

  </div>

  <!-- Copyright Information -->

  <div class="copyright">

    <!-- Display Current Year and Copyright Information -->

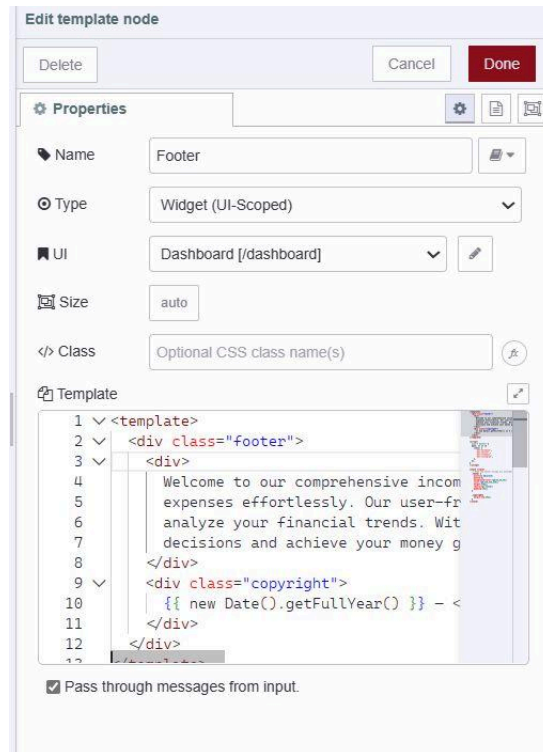
    2024 — <strong>Vuetify</strong>

  </div>

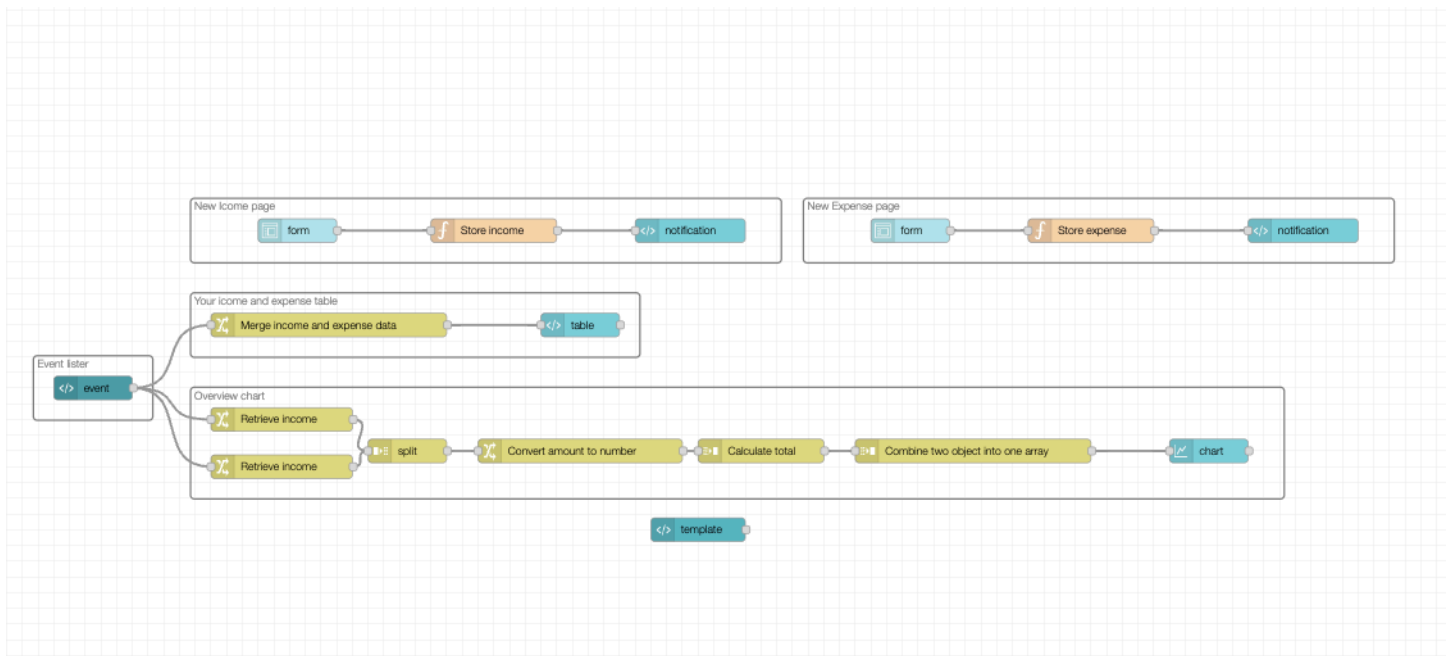
</div>
```

```
</div>
</template>
<style scoped>
/* Make the footer occupy all available space */
.footer {
  position: absolute;
  bottom: 0;
  background-color: rgb(26,26,26);
  color: rgb(238,238,238);
  height: 130px;
  text-align: center;
  padding: 14px;
}

.copyright{
  margin-top: 10px;
}
</style>
```



## Deploying your application flow



1. Deploy the flow by clicking the top right Deploy button.
2. Locate the \*Open Dashboard button at the top-right corner of the Dashboard sidebar and click on it to navigate to the dashboard.

# Styling your Dashboards

Let's look at how you can easily style Dashboard elements effortlessly.

Setting the size for elements in the Dashboard is straightforward, but understanding the actual unit size in the size property can be a bit tricky.

It's important to note that the size of a single horizontal unit varies depending on the layout, but the vertical size of a single row is consistently 48px.

## Sizing Widgets within a Group

In any layout—Grid, Notebook, or Fixed—widgets within a group are sized using a unified approach. The size property assigned to widgets determines their width within the group. Each unit in the size property represents a fraction of the group's total width. This width is determined by an internal grid established by the group.

### Widget Sizing

Widgets are sized relative to the number of columns in the internal grid. For example, if a group has 4 columns and two widgets, and the first widget is set to 1 width while the second to 3 width, the first widget will occupy 25% of the group's width, and the second widget will occupy 75%.

Regardless of the layout type, the concept of sizing widgets within a group remains consistent. Whether it's the grid, notebook, or fixed layout, the same principles apply, ensuring uniformity in widget layout and design.

### Setting element size

To set the size of groups and widgets in the Dashboard, follow these steps:

1. Go to the Dashboard sidebar and click on the edit button next to the element you want to resize.
2. Adjust the size using the size property.

## Understanding Dashboard Themes

The theme is a collection of colors that control the look and feel of the widgets, groups, and other elements on the page.

In the Dashboard, when adding a page ( ui-page ) we can specify which theme it will use. By default, we have one theme in the Dashboard, we can add more themes using the Dashboard side panel.

### Theme properties

In the theme (ui-theme) configuration, there are two main sections:

1. Colors: Specify colors for Navigation, primary elements, page background, group backgrounds, and outlines.
2. Sizing: Define the gaps between groups, page padding, group outline radius, and gaps between widgets, all in pixels.

For additional information on the ui-theme settings, please refer to the [ui-theme documentation](#).

### Setting a new page theme

1. Navigate to the Dashboard sidebar and switch to the theme tab.
2. Click on the top-right "+theme" button to add a new theme.
3. After specifying colors and sizing click on the top right update button to save the theme.
4. Now switch to the layout tab and click on the edit button next to the page for which you want to set a new theme.
5. In the page config, select the newly added theme in the Theme field.

## Dashboard Navigation

### Setting the sidebar

1. Navigate to the Dashboard sidebar in the Node-RED editor
2. Click on the "Edit Settings" button located at the top left side of the Dashboard sidebar.
3. Select your preferred sidebar style from the "Style" field in the sidebar options section.

## Sidebar Navigation Options

In the Dashboard, we have 5 different navigation options for your application.

1. Collapsing: This is the default sidebar, when it's opened, the page content adjusts to the width of the sidebar.
2. Fixed: In this type, the sidebar is always visible and fixed on the left side, and the top menu icon is hidden. The page content adjusts to the width of the sidebar.
3. Collapse to icon: This type of sidebar is similar to the collapsible one, but when the sidebar is collapsed, you can still navigate through different pages as the page icons become visible.
4. Appear over content: When the sidebar is opened, the page is partially covered by a transparent layer, and the sidebar appears on top of this layer
5. Always hide: In this type, the sidebar is always hidden, and navigation between different pages can be achieved using the ui-control widget.

## Customizing your Dashboard further

In the Dashboard we can add classes to almost all widgets, pages, and groups and style them using CSS.

### Adding classes

1. To add classes to your widget, page, or group, you'll need to open its configuration
2. Find the 'Class' field and enter your class.

**Edit ui-page node**

Delete Cancel Update

**Properties**

Name My page

UI Dashboard [/dashboard]

Path /worldmap

Icon earth

Theme Another Theme

Layout Grid

</> Class Optional CSS class name(s) for page

**Default State**

Visibility Visible

Interactivity Active

Open Dashboard 2.0 Sidebar

annotely.com

## Writing custom CSS

In the Dashboard, the ui-template node allows you to write custom CSS. In the template node, you can add CSS for two different scopes:

1. Single Page: Selecting this allows you to specify CSS that is constrained to a single page of your dashboard.
2. All Pages: Selecting this allows you to define CSS that will apply across your whole dashboard.

To define your own CSS:

1. Drag an ui-template widget onto the canvas.
2. Double-click on it and select the scope within the type field.
3. If you select the "CSS (Single Page)" type, you'll then need to select the ui-page to which your custom class definitions will apply. If you select the "CSS (All Pages)" type, then you'll need to select the ui-base that includes those pages to which you want to add styling.
4. Now you can write your custom CSS within the ui-template.



## Chapter 5

# Adding custom rich components to your UI

Vuetify is a library of UI components using Vue. This saves developers of FlowFuse Dashboard a lot of time, but it can also help you, the end-user. As Vuetify is now included, it can be used to include any of their components.

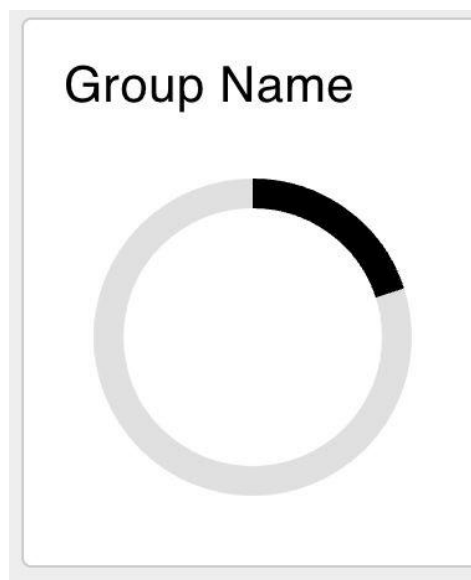
While going through the list of components on [Vuetify](#) there are several examples that aren't natively implemented in the Dashboard. One example is the [Progress circular](#) to build a countdown timer

The documentation explains which elements one can change, in this case, its the size and width. Having set those to the values you'd want in your dashboard, the HTML is generated for you as follows:

```
<v-progress-circular model-value="20" :size="128" :width="12"></v-progress-circular>
```

### Using the template node

Like the [template core node](#), the dashboard package comes with [a template node of its own](#). If we take the HTML from the Vuetify docs pages and copy it in a template node the spinner will show up on the dashboard.



## Dynamic templates

While a custom element on a page is cool, and shows you can inject arbitrary HTML on a Dashboard, it's even better if we can make the element dynamic.

Let's start with a first dynamic element. The quickest way to get that done is to have an [Inject](#) node output a random number every second.

Let's hook up an Inject, with msg.payload's output being a JSONata expression `$round($random() * 100)` to generate a random number. And let's make sure it sends a message every second.

Then we need to update the template node to the following snippet:

```
<v-progress-circular v-model="msg.payload" :size="128" :width="12"></v-progress-circular>
```

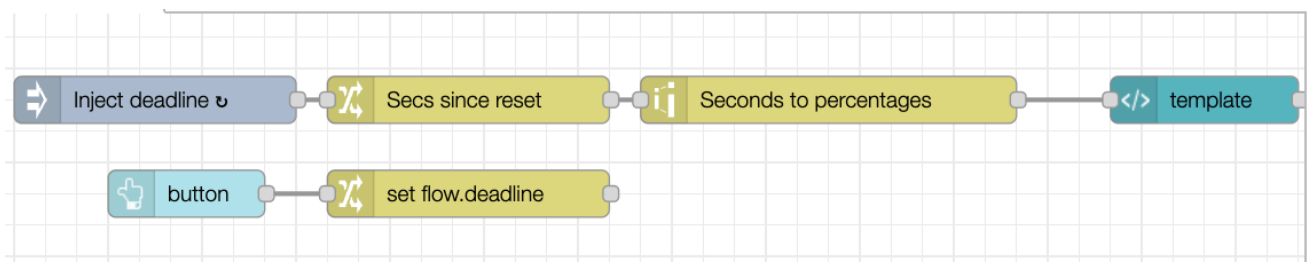
The difference is subtle but important. Instead of hard-coding the model-value to 20, the tag has changed name and it's set to msg.payload. The latter makes the value dynamic.

Changing model-value to v-model is due to leaking implementation details of the Dashboard. It uses VueJS to provide, among other features, easy updating of components. If components are dynamic, always use v-model. This allows VueJS to pick up changes made dynamically.

## Finishing the count down timer

A button would be great to reset the timer, and for the sake of this example we can hardcode the deadline to 1m from the button press. When dragging in a button node, connect it to a [change](#) node. In the change node set the flow variable flow.deadline to the timestamp. The Inject node from earlier needs updating to inject the flow.deadline. All that's left is calculating how many seconds passed, and normalizing 60 seconds to the range between 0-100.

The complete flow is:



## Chapter 6

# Personalised Multi-User Dashboards

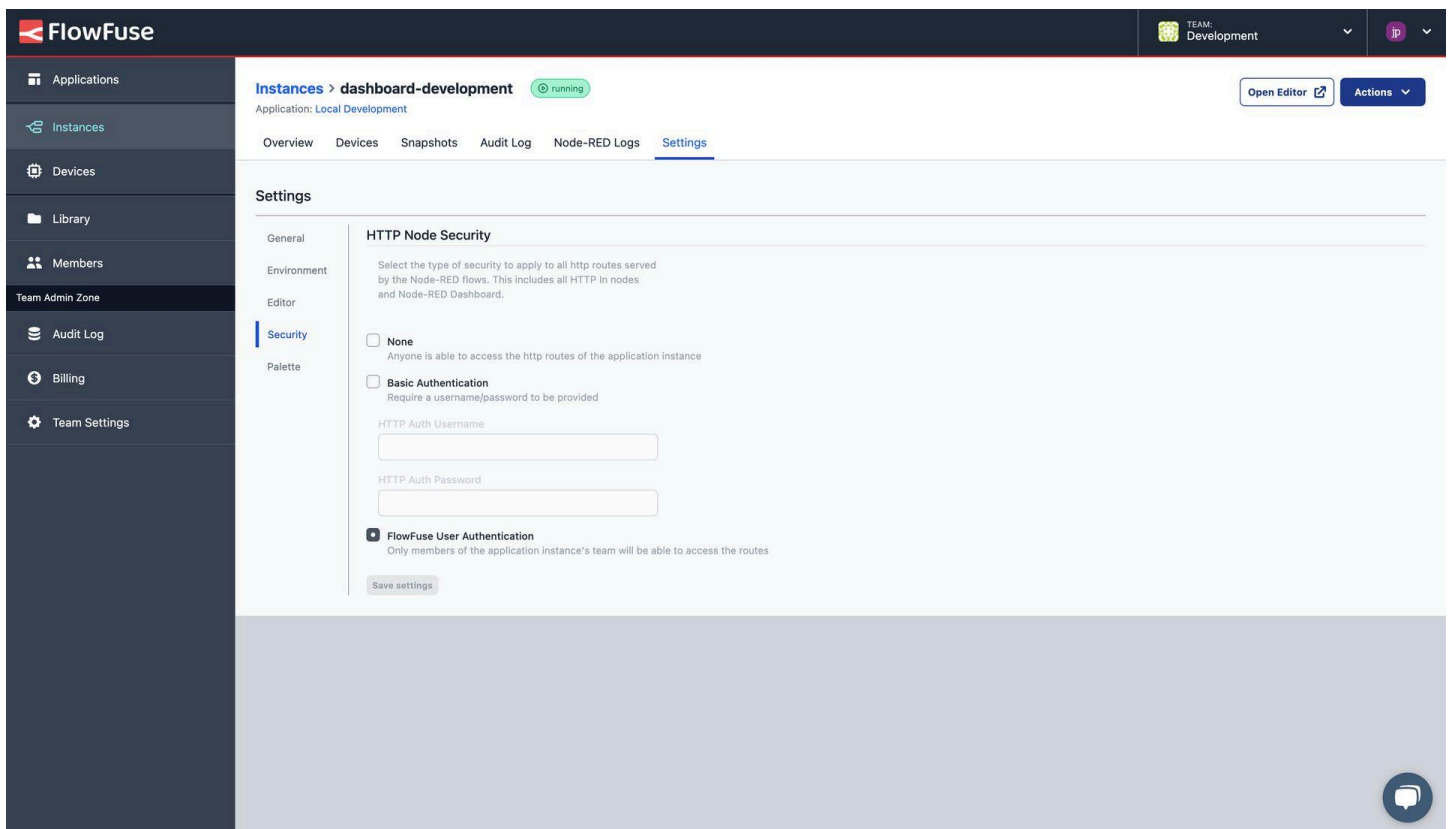
The original Node-RED Dashboard was built with a "single source of truth", no matter how many users interacted with the dashboard, each user would always see the same data. This is great for prototyping, or hobby projects, but as you scale up your Node-RED usage, you'll want to be able to have unique dashboard experiences for each user.

## Getting Started

To enable personalised multi-user dashboards, you'll need to be using FlowFuse, and complete two steps:

### Step 1: Enable "FlowFuse User Authentication"

All instances on FlowFuse can be configured with "FlowFuse User Authentication" in the "Security" Settings. This option requires any user that wants access to your Editor or dashboard to be authorized by FlowFuse first.



## Step 2: Install FlowFuse's User Addon

- FlowFuse Cloud

The Personalised Multi-User Dashboard plugin, [@flowfuse/node-red-dashboard-2-user-addon](#), is available in our [Certified Nodes](#) catalogue, accessible to our Teams and Enterprise customers.

Once the "FlowFuse User Authentication" option has been enabled on your instance, you can then install our plugin, [@flowfuse/node-red-dashboard-2-user-addon](#), through the "Manage Palette" option in the Node-RED Editor.

For your devices, we provide the necessary configuration and access token upon request, so that your Node-RED devices can also benefit from a Personalised Multi-user Dashboard.

- FlowFuse Self-Hosted

For all our Teams and Enterprise Self-Hosted customers who also want to use the Certified Nodes and the Multi-User Dashboard, we provide all necessary configurations upon request to get started..

### Using the Plugin

Once enabled, any messages emitted by a Dashboard node will contain a new `msg._client.user` object, e.g:

```
JavaScript
{
  "userId": "", // unique identifier for the user
  "username": "", // FlowFuse Username
  "email": "", // E-Mail Address connected to their FlowFuse account
  "name": "", // Full Name
  "image": "" // User Avatar from FlowFuse
}
```

Then, when running Node-RED Dashboard on FlowFuse, you'll have a new sidebar option in the Node-RED Editor, which allows you to control which node types allow for "client constraints".

Dashboard 2.0

i


Multi User Dashboard


Edit Settings


Open Dashboard


Layout

FF Auth

☒  Include Client Data

 This option includes client data in all messages transmitted by Dashboard 2.0 (e.g. Socket ID and information about any logged in user)

 Accept Client Constraints

 Defines whether the respective node type will use client data (e.g. socketid), and constrain communications to just that client.

☐ ui-text-input

☐ ui-button

☐ ui-dropdown

☐ ui-radio-group

☐ ui-slider

☐ ui-switch

☐ ui-text

☐ ui-table

☐ ui-chart

☒ ui-notification

☐ ui-markdown

☒ ui-template

☒ ui-control

In the original Node-RED Dashboard, this was *always* enabled for the [ui-notification](#) and [ui-control](#) nodes, whereby you could include [msg.socket](#) data and it would only then send that message to the specified client. For the FlowFuse Dashboard we've extended this concept so that as a Node-RED Developer, you can now include [msg.\\_client.user](#) data in any message sent to a the Dashboard node. Under the covers, our FlowFuse-exclusive plugin will then automatically filter messages to only send to the relevant user's connection.

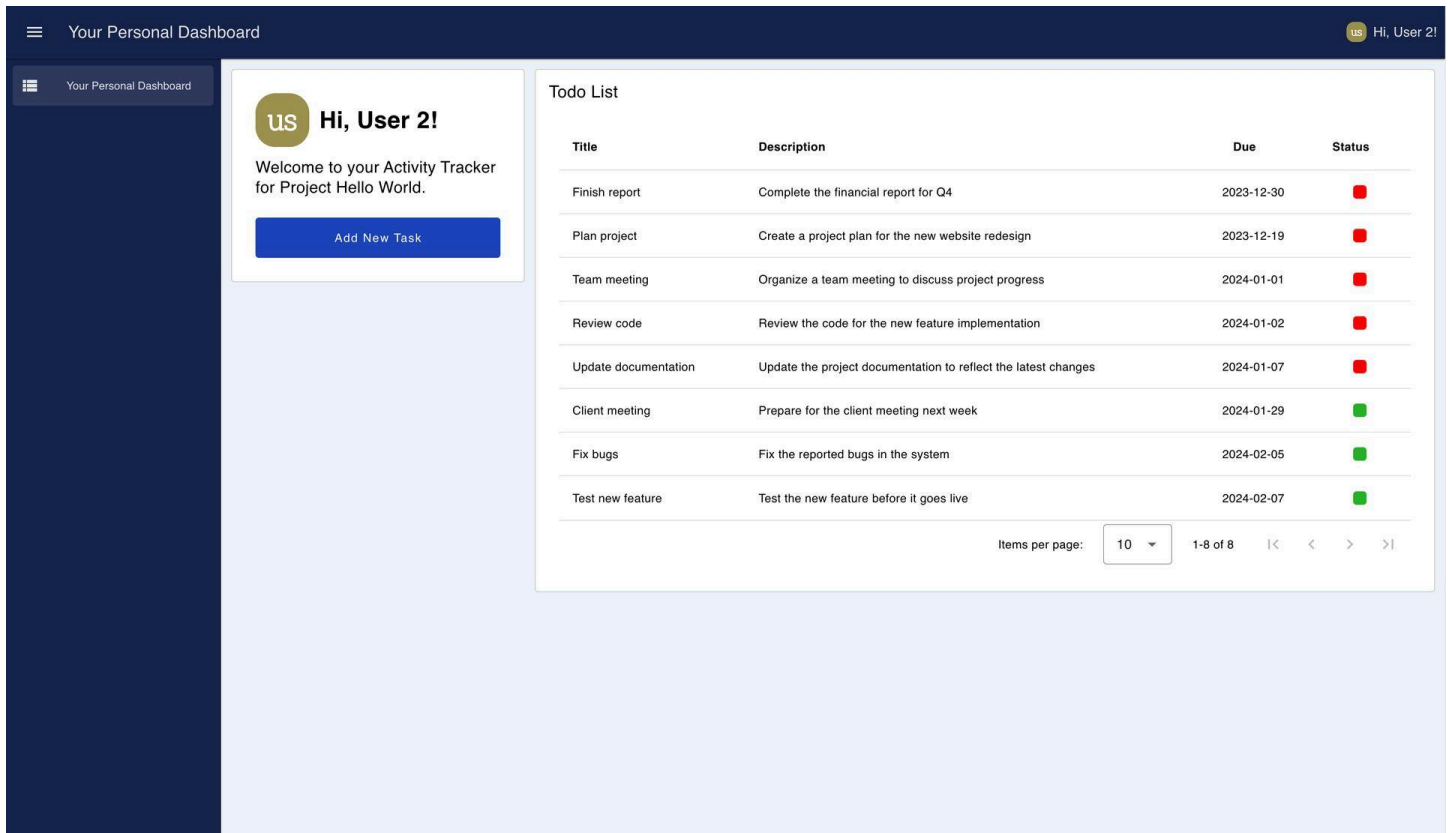
Utilising this feature, below you can see an example where we send data to a [ui-template](#) to render a custom table for each user. Under the covers this is a [ui-event](#) node (triggered on a page view), which then uses the [msg.\\_client.user](#) object to make a REST API call to retrieve a list of todo items for that specific user. We then wire the response into the [ui-template](#), which has been configured to "Accept Client Constraints", and so only sends this data to User 2's dashboard.

Title	Description	Due	Status
Finish report	Complete the financial report for Q4	2023-12-30	Red
Plan project	Create a project plan for the new website redesign	2023-12-19	Red
Team meeting	Organize a team meeting to discuss project progress	2024-01-01	Red
Review code	Review the code for the new feature implementation	2024-01-02	Red
Update documentation	Update the project documentation to reflect the latest changes	2024-01-07	Red
Client meeting	Prepare for the client meeting next week	2024-01-29	Green
Fix bugs	Fix the reported bugs in the system	2024-02-05	Green
Test new feature	Test the new feature before it goes live	2024-02-07	Green

Note too that we're also utilising the new [Teleport](#) option available in a [ui-template](#) which allows us to define content to show in the top-right of the dashboard, in this case, a little *"Hi {username}"* message.

## Example: Rendering Logged In User Data

In the previous example, you may have noticed that we're also displaying a welcome to the authenticated user on our dashboard, this means that we have access to the full User object within any `ui-template` that we render too.



Under the covers, we're appending our user object to the `msg` object, via the `SocketIO` auth option. We make the `socketio` object available via a computed setup variable, this means that we can access this data in any `ui-template` node, and render like so:

```
Unset
<template>
  <div>
    <h1>Hi, {{ setup?.socketio?.auth?.user?.name }}!</h1>
  </div>
</template>
```

To enable custom user-by-user content in a ui-template though, we must allow it to "Accept Client Constraints". This means that if a `.msg_client.user` value is included in any messages sent to a ui-template node, then the underlying SocketIO message will be filtered to only send to the relevant user's connection and no others.

## Admin Only Views

We can also now show/hide content based on the authenticated user.

We recently introduced the option to [set default visibility & interaction states](#). This was partly introduced because it's a good practice to set the default "Visibility" option for any admin-only pages to "Hidden", and then use a `ui-control` node to show the content only to the relevant admins.

Let's breakdown the above flow:

1. We wire a `ui-event` node (which emits each time a user views a page) into a switch node
2. Our switch node checks the `user.username` against a known list of admin users and branches "admin:" and "non-admin" users
3. For admin users, the `change` node defines a message for our `ui-control` node to dynamically show content, in this case an "Admin" page, when appropriate.

```
Unset
{
  "pages":{
    "show": ["Admin View"]
  }
}
```

All events going into `ui-control` are automatically filtered based on the `msg_client.user` object, so only the Admin users will receive the message to show the "Admin View" page, resulting in:





Further extensions of this could also check `ui-event` in case a non-admin user tries to access the `/admin` page directly, in which case we can utilise `ui-control` to navigate the user away from the page immediately. See the [ui-control documentation](#) for more details on this.

## Chapter 7

# Displaying Logged-in Users

Before we display logged-in user data on the dashboard, first we need to set up a login mechanism with FlowFuse for the Dashboard as per the previous chapter.

To display user information on the dashboard we will use Vue's [Teleport](#) feature to render content to a specific location in the DOM, we will display user information at the action bar's right-hand side.

1. Drag a ui-template widget onto the canvas.
2. Click on that node, and select type as "Widget (Ui-Scoped)". ( this allows us to render this ui-template at ui scoped which means I will not required to add separate ui-templates for different pages )
3. Copy the below vue snippet and paste that into the ui-template.

Unset

```
<template>
  <!-- Teleporting user info to #app-bar-actions, which is the ID of the action bars' right corners area -->
  <Teleport v-if="loaded" to="#app-bar-actions">
    <div class="user-info">
      <!-- Displaying user image -->
      
      <!-- Greeting the user -->
      <span>Hi, {{ setup.socketio.auth.user.name }}</span>
    </div>
  </Teleport>
</template>

<script>
export default {
  data() {
    return {
      // Flag to indicate if the component is loaded
      loaded: false
    };
  },
  mounted() {
```

```

    // This function is called when the component is inserted into the DOM.
    // Setting loaded to true here ensures the component is ready to access #app-bar-actions,
    // as it's now part of the same DOM structure.
    // Accessing it before mounted() would cause an error because the component wouldn't be initialized
    in the DOM yet.
    this.loaded = true; // Setting loaded to true to indicate that the component has been mounted
    successfully
  }
}
</script>

<style>
/* Styling for user info display */
.user-info
{
  display: flex;
  align-items: center;
  gap: 8px;
}
/* Styling for user avatar image*/
.user-info img
{
  width: 24px;
  height: 24px;
}
</style>

```

## Deploying the flow

1. With your flow updated to include the above, click the "Deploy" button in the top-right of the Node-RED Editor.
2. Navigate to <https://<your-instance-name>.flowfuse.cloud/dashboard>.



# About FlowFuse

FlowFuse is a platform that empowers any engineer; mechanical, electrical, or software engineer, to build, deploy, manage, and secure enterprise-grade applications. We enable IT experts and operational engineers to collaborate on data acquisition, merging of data streams, create data visualization, and interactive applications.

Our platform enhances the capabilities of Node-RED by providing security and compliance guardrails, increasing developer productivity, and centralized operations for scalability and reliability.

At FlowFuse we are committed to providing the best tools and support to seamlessly integrate into your existing business, offering an adaptable foundation for complex application development.

[Contact us](#) to find out how we can help develop innovative applications that revolutionize your operations.