https://bloggedin.co.uk/creating-web-pages-with-node-red/

Creating web pages with Node-RED

Hopefully the quick examples in this post will give you enough information and insight for you to use in other projects as fundamentally this is what we try to enable at bloggedin; show you simple examples and how they may form part of a larger project you may wish to undertake.

A very simple web page

A picture containing text, indoor, toilet, tiled

Description automatically generated

With just 3 nodes we are able to serve a web page in Node-RED. First, we use a http in node from the palette and because we are serving a page, we leave the method as the default GET. The URL box is where we define our URL endpoint.

Graphical user interface, text, application, email

Description automatically generated

The entry of /home simply means that our URL path will be (if served locally) 127.0.0.1:1880/home.

This creates the URL route, but we need to add some content, we do this by providing a template node under the functions section of the palette.

Graphical user interface, text, application

Description automatically generated

Our third node is a http response node, this requires no configuration other than connecting it to our previous node. If we deploy this flow and browse to 127.0.0.1:1880/home we have our rendered web page.

Graphical user interface, text, application, chat or text message

Description automatically generated

Adding Style

More often than not all modern web pages are spruced up using CSS (Cascading Style Sheets) and you’ll be pleased to learn that in Node-RED we can also add CSS, now you could use inline CSS direct in the template but it is much more useful to separate out CSS as you would when building a website. Using the below flow we can show how we can achieve this quite simply in Node-RED.

A picture containing text, shoji, indoor, toilet

Description automatically generated

You will notice that the flow only needs the addition of another template node which we have named CSS inside this we have the following (yes, it is very basic CSS).

Graphical user interface, text, application

Description automatically generated

The important part to note here is the …property entry we have assigned the CSS to the payload.style which we can refer to by modifying our HTML template using mustache code.

Graphical user interface, text, application

Description automatically generated

Note the addition of line 4 where we use the {{{payload.style}}} to inject our style sheet.

Once deployed if we now go to our page, we have the following output (note I changed the route to home2 to show have both pages available for the article).

Graphical user interface, application

Description automatically generated

All simple stuff but especially useful if we want our Node-RED output to be different than using the standard UI dashboards.

Let us get formal by adding Forms!

A key constituent to any website is the ability to gather user data and to do this we typically use forms in HTML using GET and POST. Again, we can easily achieve this in Node-RED, the process is twofold; serving up the form and generating a URL to POST the data to.

A screenshot of a computer

Description automatically generated with low confidence

Using the above flow, we have created a route /webform using the GET method, we then use two templates (CSS and HTML to serve up our form) prior to passing it to a HTTP response node.

The CSS template is below, feel free to dive in and apply your own styling, it is passed in as before using the payload.style.

Graphical user interface, text, application

Description automatically generated

Our HTML is like any typical HTML form

Graphical user interface, text, application

Description automatically generated

The important parts of this code are in line 5 where we pass in the Style and line 13 where we set the POST action to /webformpost (no we haven’t created this yet) this will be where Node-RED picks up the data which we will come to in a minute. First let us have a look at how our form renders in the browser by deploying the flow and browsing to 127.0.0.1:1880/webform

Graphical user interface

Description automatically generated with low confidence

We have created input fields to capture firstname, lastname and age, we also have a submit button that currently will error as we have not built the route to process the submission. We will do that now.

Capturing the POST

To enable our submit button to work we need to create the route for webformpost that the form action refers to.

Diagram

Description automatically generated

Note now our http in node is now POST instead of GET.

Graphical user interface, text, application, email

Description automatically generated

To gather the data posted we use a JSON node from the parser section of the palette set with the following parameters:

Graphical user interface, text, application, email

Description automatically generated

This will put our form data into JSON which is very useful for backend processing which we will come to soon, but to give the user a response we can add a function to process what they have submitted and return the JSON.

Graphical user interface, text, application, email

Description automatically generated

When this is passed to a http response node the following output is received to confirm that our form is now working (although doing nothing at this stage in the backend other than returning our entries).

Graphical user interface, application

Description automatically generated

Upon submission we get the following response:

Graphical user interface, text, application

Description automatically generated

As you can see Node-RED has nicely gathered our data entry into JSON and JSON is the stuff that Node-RED eats for breakfast. What we need now is to do something with our data, a typical form will add data to a backend database so let us not break tradition and add in this functionality.

SQLite backend

In order to capture our data into a database we are going to need a database, so let us create one in Node-RED using SQLite (refer [here](https://bloggedin.co.uk/creating-apis-with-node-red/) to installing SQLite nodes, scroll to SQLite and creating a CRUD Database).

Diagram, timeline

Description automatically generated

Using inject nodes with SQL commands as the topic we can add a table and create our database ready to receive data.

Graphical user interface, text, application, email

Description automatically generated

You can see in the topic I have created a table contacts and set up fields for firstname, lastname and age as per our form. We deploy this and press the inject button once (creates our table for later use).

Processing our form

I have duplicated our form flow and called it webform2 to keep all the flows available and updated the POST action in the HTML to go to webformpost2 but other than that everything is the same.

When we now go to webform2 if we POST data it goes to webformpost2 which has the following flow:

Diagram

Description automatically generated

The top flow chain we will ignore as this is the same as previous whereby, we give the user a response showing the data submitted. The below chain is where we add the data to the database again with 3 simple nodes a JSON parser, a function and a SQLite Node.

Note the JSON node has a slightly different configuration:

Graphical user interface, text, application, email

Description automatically generated

This time we are converting it to a JavaScript Object so we can use it fully in a function to extract data.

The function detail is as follows:

Graphical user interface, text, application, email

Description automatically generated

You can see that we can refer to each submitted data by calling the field name appended to the payload. We add the extracted firstname, lastname and age to the message topic as part of SQL query that we pass into the SQLite node.

When deployed and we browse to webform2, enter data it is now injected into our SQLite database thus we are now producing “backend” processing.

Viewing our data!

Capturing data is an important part of any website but what about reversing the process? How do we extract data from our database and push it back to the frontend?

Let us take a look at how we can do this in Node-RED and thus complete the stack.

A picture containing timeline

Description automatically generated

The process is broken down as follows:

First, we create the view route as a http in set to GET with a URL of /viewdata. We then need a function to create our query for the database node in this example we will pull back all entries and display them as a table.

Graphical user interface, text, application, email

Description automatically generated

The function simply creates a topic with our query.

The payload from this after the SQLite database node is our JSON object, which when we add multiple submissions becomes an array which we can see in our debug output.

A picture containing text

Description automatically generated

Using a couple of templates one for CSS and one for HTML we can render our data back to the user in a table.

Graphical user interface, text, application, email

Description automatically generated

Graphical user interface, text, application

Description automatically generated

You can learn how to create tables in HTML from online resources such as [www.w3schools.org](http://www.w3schools.com/), this post is not intended for teaching HTML but from the above code the key take home is lines 10 to 16 this is where we inject all our array items from the payload in their own rows to the table. This is much like a loop through data in other languages embedded in the HTML code.

Using the mustache code on the payload we can refer to each item in the payload (firstname, lastname, age) and put them in table dimensions inside a row. The Payload (loop) does this for all the entries pulled back from the query.

When deployed we can browse to 127.0.01:1880/viewdata to see our table.

Graphical user interface

Description automatically generated with low confidence