**Exp No.: 2**

**Date: 11/02/2021**

# Title- Creating Login and Registration

**AIM: To create login and registration form using node red**

**Software Used:**

|  |  |
| --- | --- |
| **Si .No** | **Components and Tools Required** |
| 1 | Node-Red |
|  |  |

**THEORY:**

**(Brief theory about the implementation)**

Node-RED is a flow-based development tool for visual programming developed originally by IBM for wiring together hardware devices, APIs and online services as part of the Internet of Things. Node-RED provides a web browser-based flow editor, which can be used to create JavaScript functions

We'll need two routes to make any signup form function. The first is a route that displays the HTML for the login form, and the second captures the data sent from the form. Start by dragging two HTTP-IN nodes in Node-RED.

**ALGORITHM:**

(Explain the sequence of flow in execution- purpose of the nodes used)

**HTTP IN – NODE**

You want to create an HTTP endpoint that responds to GET requests with some static content, such as an HTML page or CSS stylesheet.

**TEMPLATE -NODE**

The node provides a convenient way to embed a body of content into a flow. It may be desirable to maintain such static content outside of the flow.

**HTTP RESONPSE - NODE**

The node uses the payload property of messages it receives as the body of the response. Other properties can be used to further customize the response - they are covered in other recipes.

**FUNCTION – NODE**

The function node is used to run JavaScript code against the msg object. The function node accepts a msg object as input and can return 0 or more message objects as output. This message object must have a payload property (msg. payload), and usually has other properties depending on the proceeding nodes.

**JSON-NODE**

Node-RED's json node is a sort of convenience function, in that it parses the incoming message and tries to convert it to/from JSON. So if you send it a JSON **string**, it will convert it to a JavaScript object, and vice versa

**DEBUG-NODE**

Open the starting file (typically index. js ), activate the Run and Debug pane, and click the Run and Debug Node. js (F5) button.

**FLOW DIAGRAM DESIGN :**

(Configuration parameters for the flow and node along with the node properties)

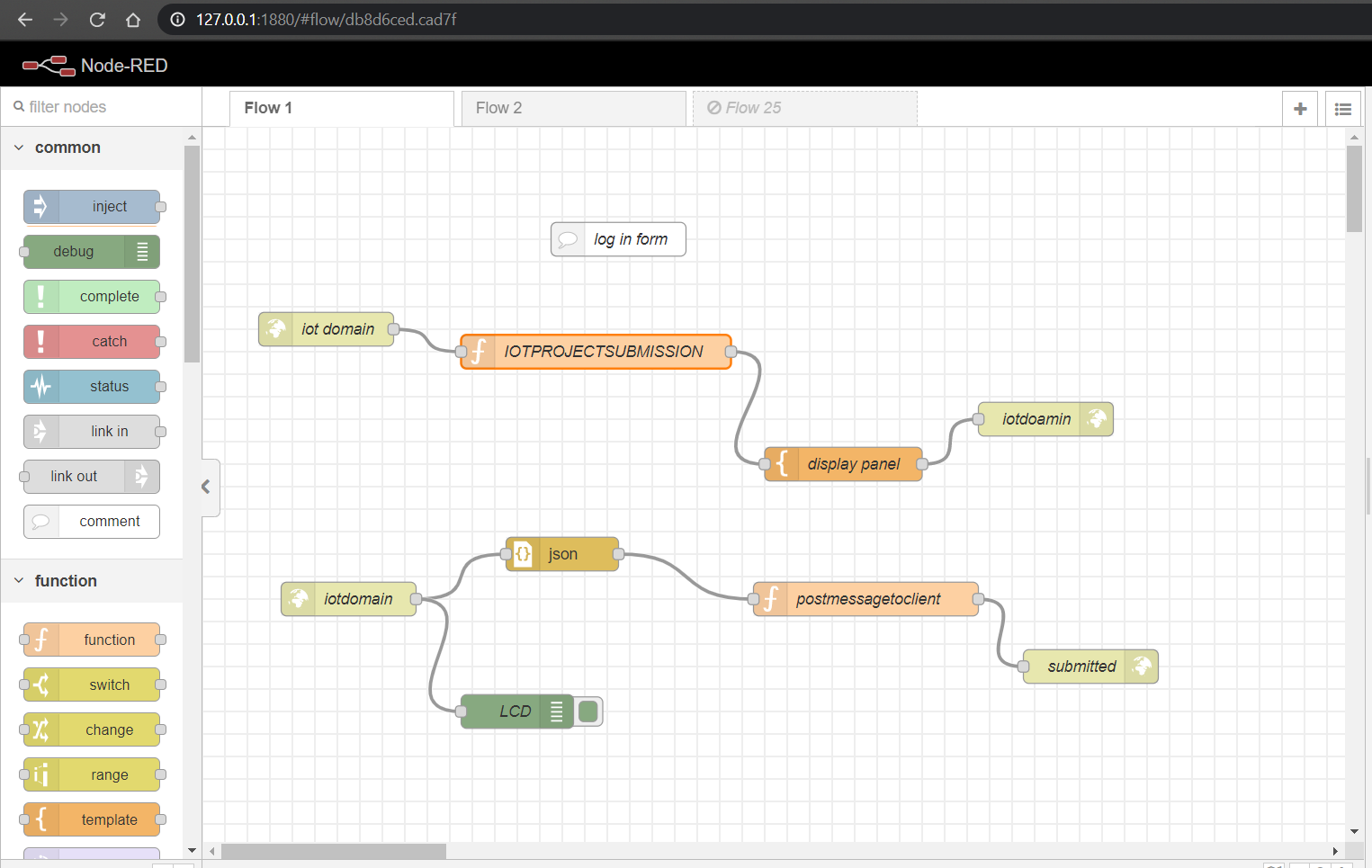
**PROCEDURE:**

(Explanation of the properties and procedures involved in the implementation)

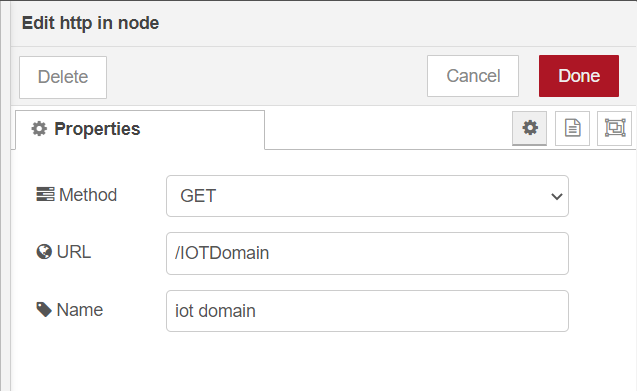
1. We'll need two routes to make any signup form function. The first is a route that displays the HTML for the login form, and the second captures the data sent from the form. Start by dragging two HTTP-IN nodes in Node-RED. Configure the first node with a method of GET and a URL of /signup. You can change this URL at a later time, but you should keep the method the same.
2. This is the URL that will display our signup form. Now, we'll grab a template node from the palette and drag it onto the canvas. Double-click it, and add the following code to the template area.
3. This will create a simple form that contains a single field, *email*. When the user clicks the submit button, the data in the form will be sent to the /signup url, but using the POST method. This is a pretty common pattern, and is easy enough for us to remember.
4. Let's finish out this flow with an *HTTP Response* to complete the response.
5. Now, we can move on to processing the form. In the second *HTTP IN* node you added, double-click it to configure. In the URL section type */signup* and select POST from the Method dropdown. This will be the receiver of the data we sent from our form. We can also add a Debug node to this receiver so we can try out our form and see what kind of data is coming back to us.
6. Notice that the data is coming back in the msg.payload object. The msg represents the message that we're sending from one node to another, and by convention the payload field will contain any data from that node. The Debug node automatically outputs the msg.payload object since, and shortly we'll look at how we can manipulate other fields in the msg object.
7. Now, go ahead and fill out the form and click Submit. The first thing you'll notice is that our browser seems to "hang", which means that it's waiting for a response. Since we haven't added an *HTTP Response* node to the canvas yet, there's no response being sent back to our browser. However, we can still pop on over to Node-RED and check out what our receiver got in the *debug* panel.

**PROPERTIES SET FOR EACH NODE WITH EXPLANATION:**

(SNAPSHOTS)

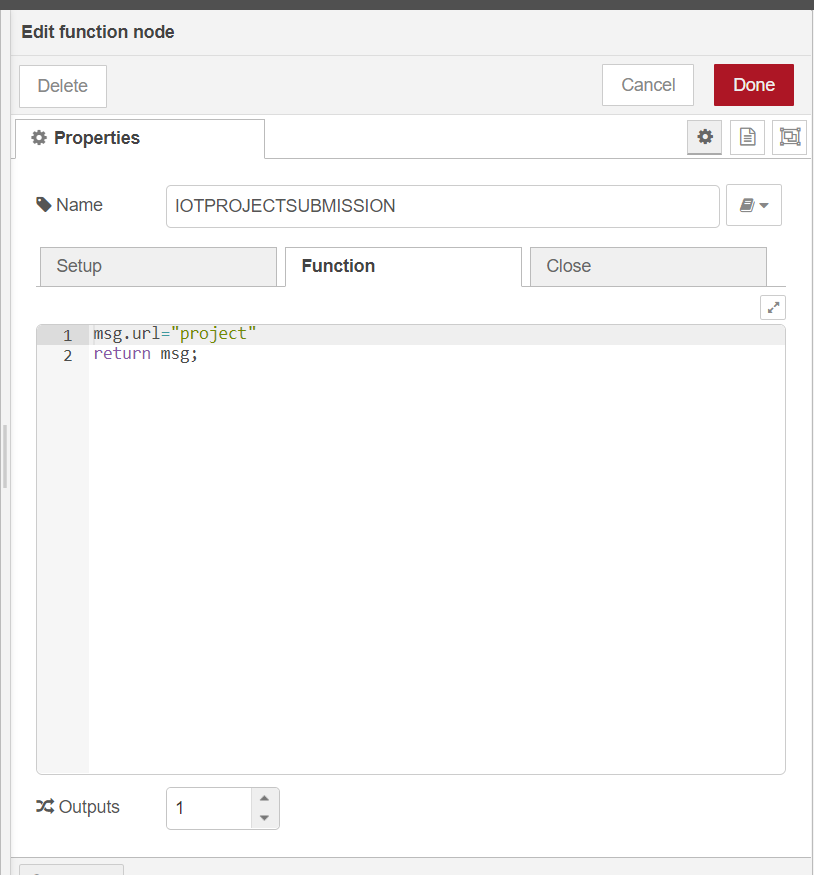


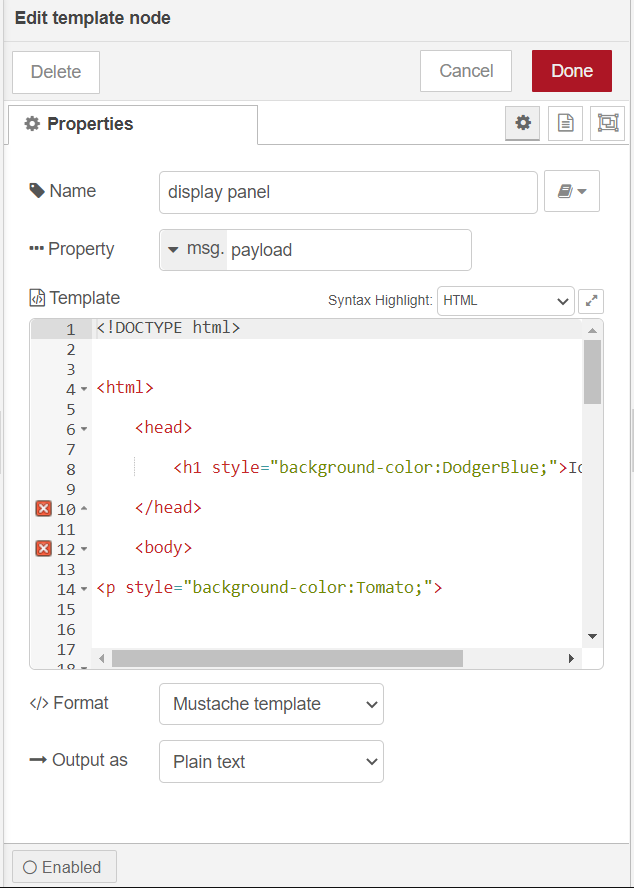
**HTTP IN - NODE**



**Set method to get and create URL.**

**FUNCTION – NODE**

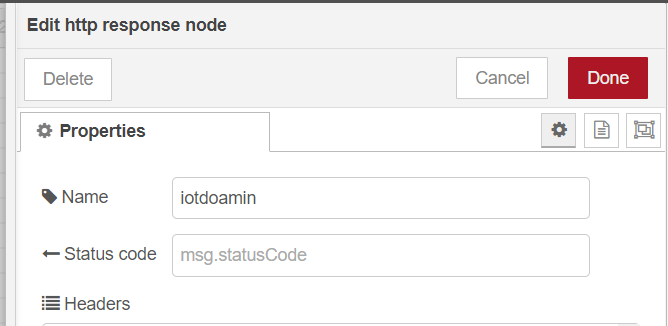
**TEMPLATE -NODE**



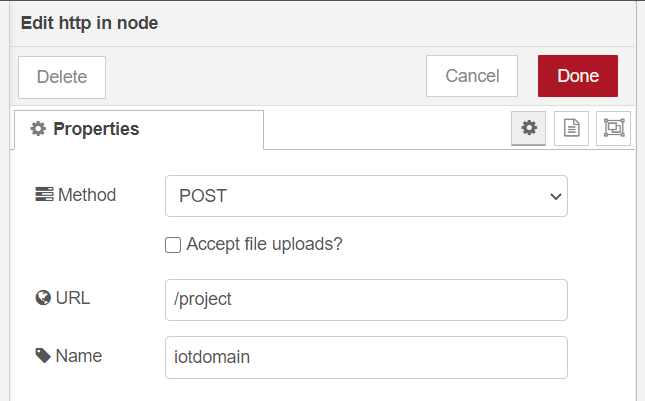
**Where syntax is set to html because its Skelton to create a website**

HTML (Hypertext Markup Language) is the code that is used to structure a web page and its content.

**HTTP RESONPSE - NODE**

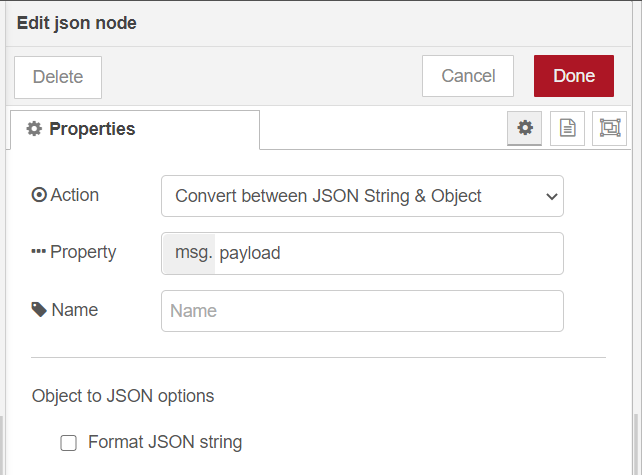


**HTTP IN-NODE**

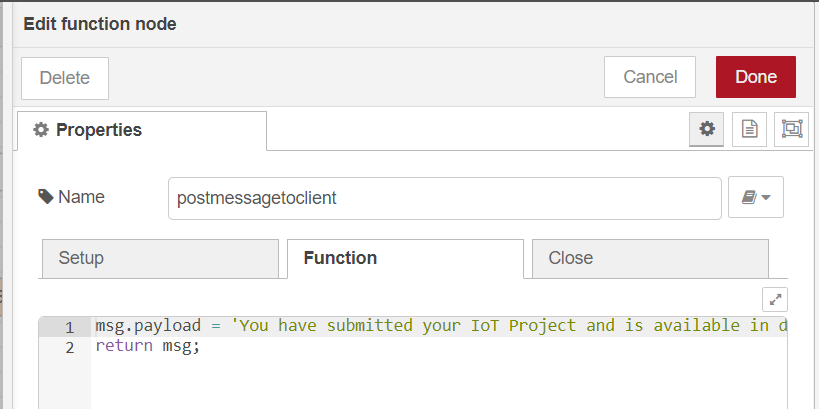


**Set method to post ,to display a output**

**JSON-NODE**

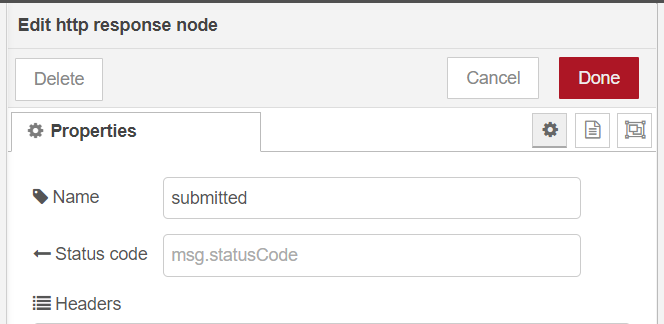


**FUNCTION-NODE**

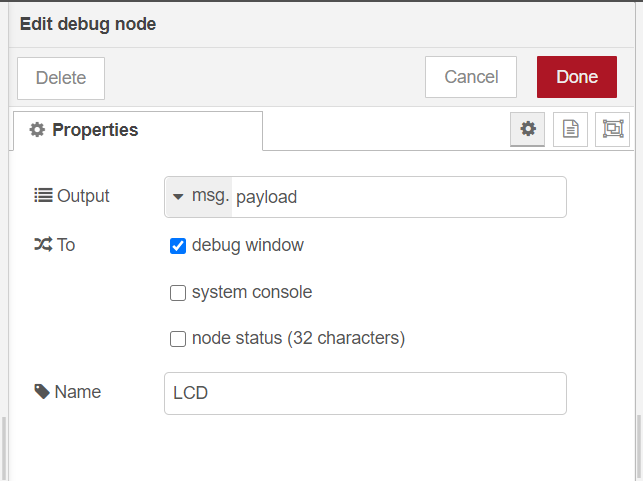


**JavaScript is used in function workspace. this code displays the output.**

**HTTP RESPONSE NODE**



**DEBUG-NODE**

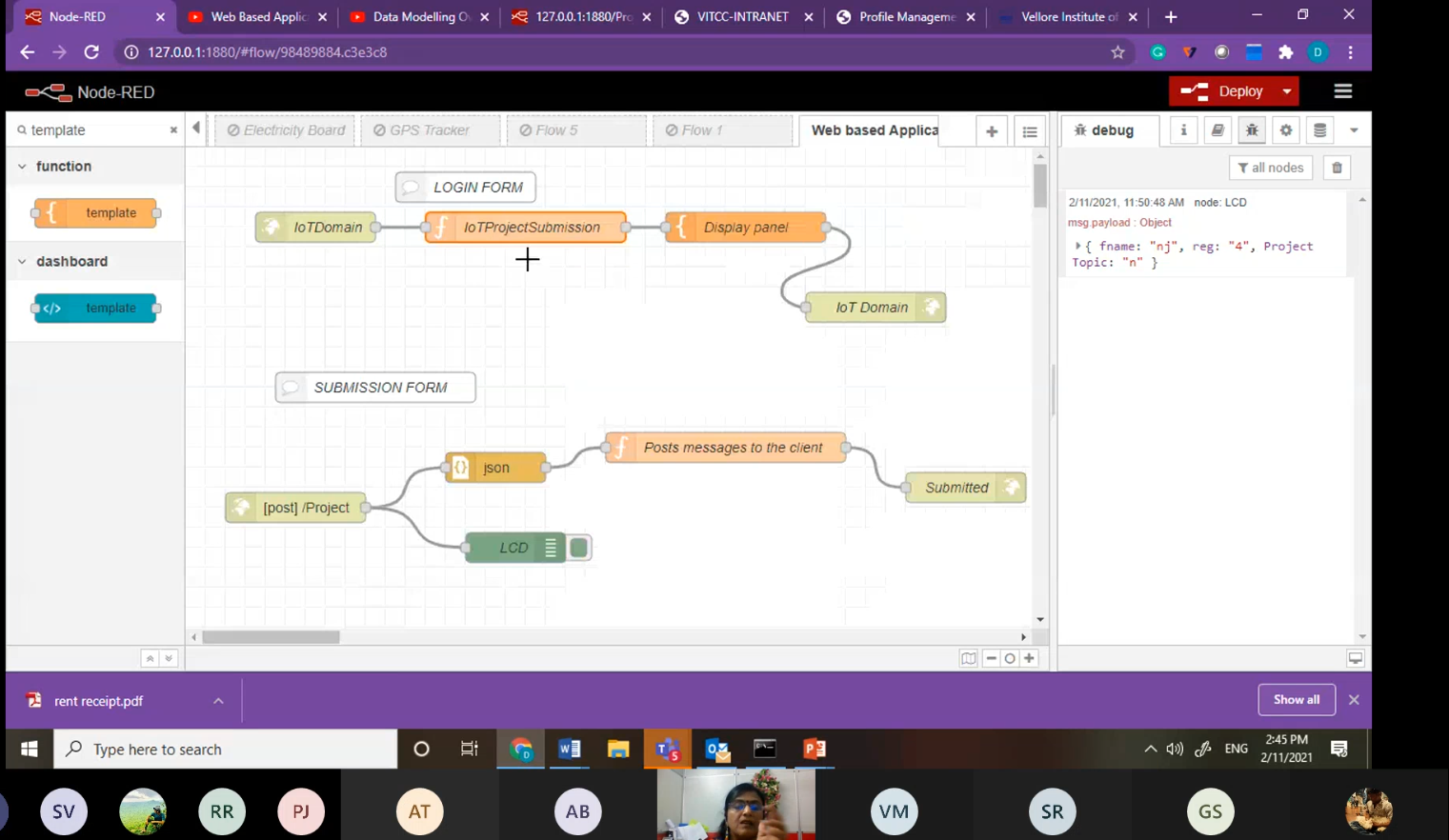


**This node used to display a output in debug window.**

**OUTPUT:**

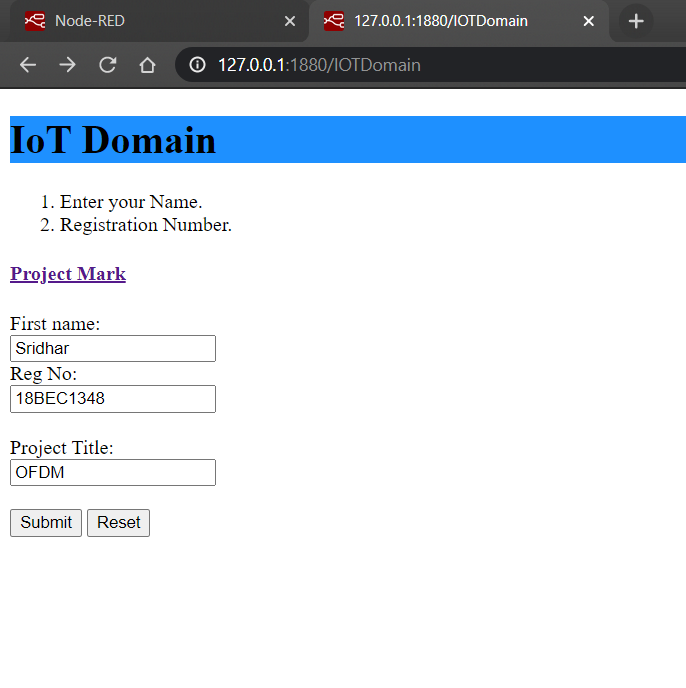
**SNAPSHOTS:**

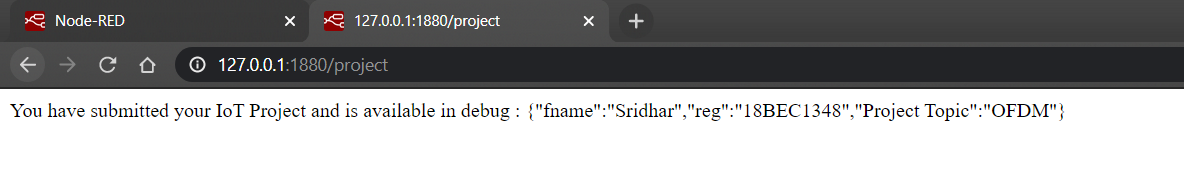
(Snapshot of the class attended)

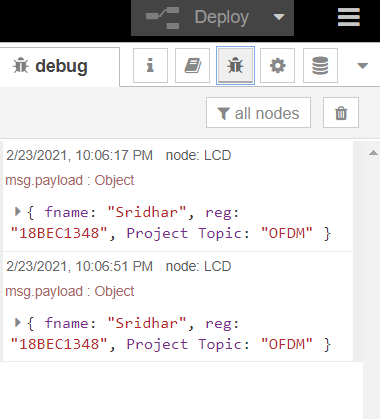


(SCREENSHOTS OF THE OUTPUT WITH REGISTER NUMBER)

(Snapshots of the browser /debug window/dashboard)





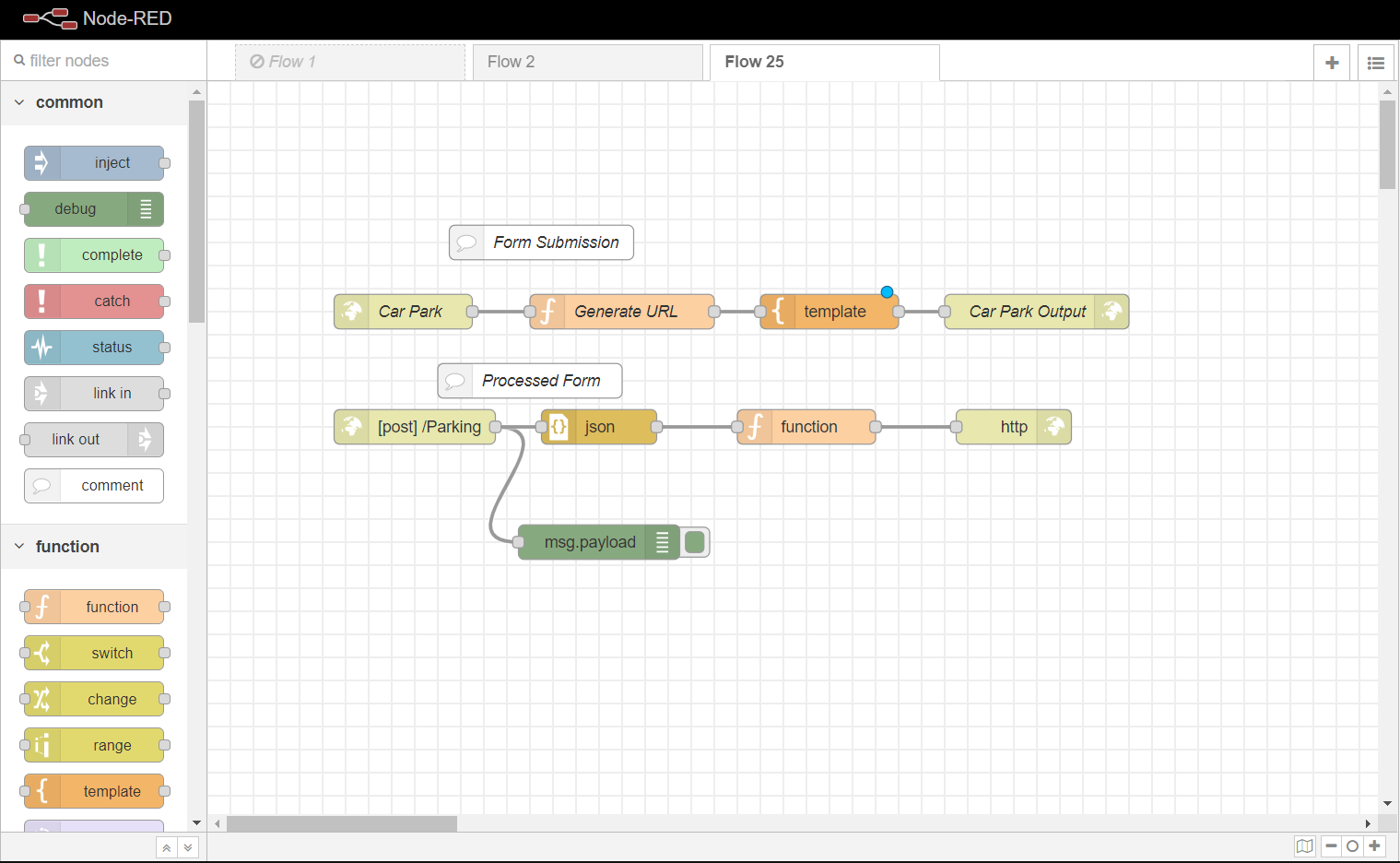


**RESULT:**

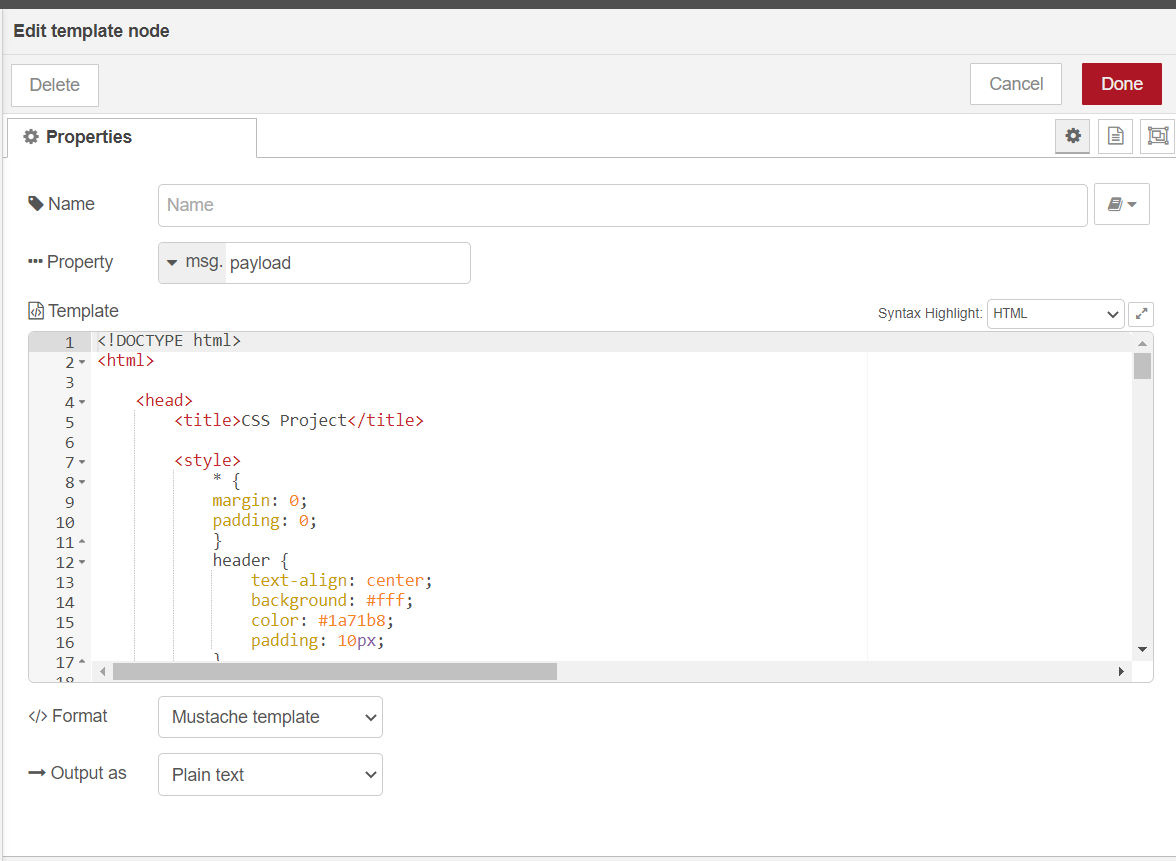
Inferences obtained from the experiment along with the values.

Task -2

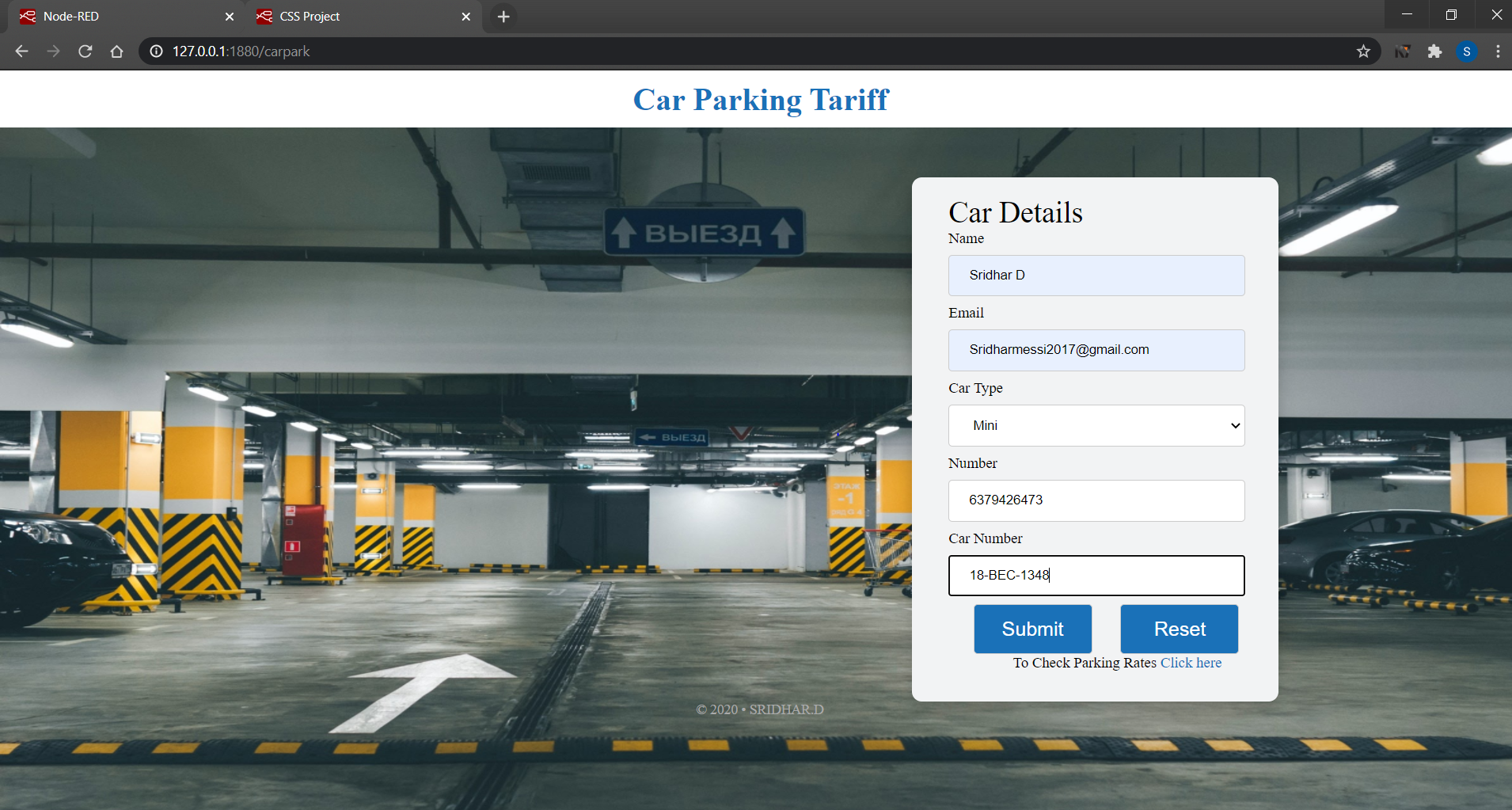
Implantation

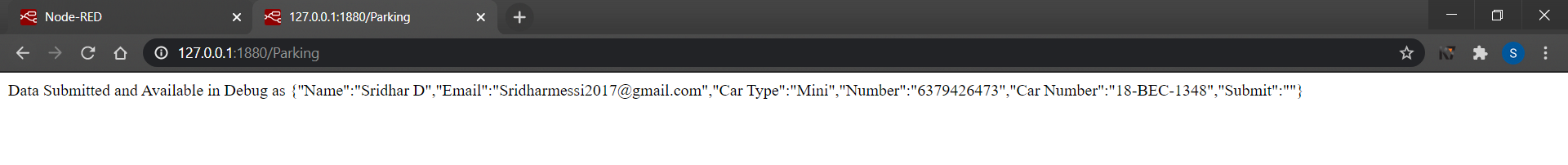


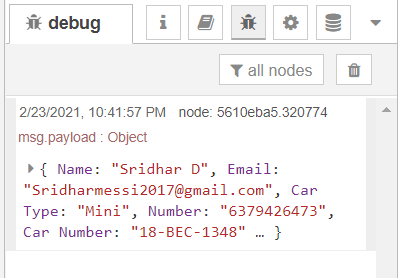
Html code



Output







Html code

<!DOCTYPE html>

<html>

<head>

<title>CSS Project</title>

<style>

\* {

margin: 0;

padding: 0;

}

header {

text-align: center;

background: #fff;

color: #1a71b8;

padding: 10px;

}

body {

background-image: url("https://lh3.googleusercontent.com/s\_bXoge9wYeh63Hvp8TG0DyML4fhiDuSZdffc5Nq5ltKqLTk6B\_wf-p7z5RnpJPF\_mzYEsVh0bx8FIR4Q8yi3GcQp0991ALIx6C5\_n7Va4uC4Byzy5gNdaN9jvAO0TR5YSHweMyK8Nw=w2400?source=screenshot.guru");

background-repeat: no-repeat;

background-attachment: fixed;

background-size: 100% 100%;

}

footer {

text-align: center;

color: #aaa;

font-size: 14px;

}

.SignUp {

margin-top: 50px;

margin-left: 60%;

width: 370px;

height: 530px;

background-color: #F2F3F4 ;

box-shadow: 0px 8px 16px 0px rgba(0,0,0,0.2);

border-radius: 10px;

}

.SignUp a {

text-decoration: none;

color: #1a71b8;

}

.head {

font-size: 30px;

margin-left: 10%;

font-family: font-family: 'IBM Plex Serif', serif;

}

.foot {

margin-left: 65px;

}

.signUpForm {

margin-left: 10%;

font-size: 15px;

}

button {

width: 120px;

height: 50px;

margin-left: 25px;

padding: 12px 20px;

border: 1px solid #ccc;

border-radius: 4px;

box-sizing: border-box;

background-color:#1a71b8;

color: white;

font-size: 20px;

}

button:hover {

background-color: #117A65;

color: white;

}

input[type=text], input[type=email ], input[type=password] {

width: 90%;

padding: 12px 20px;

margin: 8px 0;

display: inline-block;

border: 1px solid #ccc;

border-radius: 4px;

box-sizing: border-box;

}

select {

width: 90%;

padding: 12px 20px;

margin: 8px 0;

display: inline-block;

border: 1px solid #ccc;

border-radius: 4px;

box-sizing: border-box;

}

.error {

color: white;

font-size: 8px;

font-weight: bolder;

margin-left: 5px;

background: crimson;

padding: 3px 5px;

border-radius: 100%;

font-family: monospace;

display: none;

}

</style>

</head>

<body>

<header>

<h1>Car Parking Tariff</h1>

</header>

<div class="SignUp">

<br>

<p class="head">Car Details</p>

<form class="signUpForm" action="/{{url}}" onsubmit="return validate()" method="post">

<label for="Name"> Name</label><br>

<input type="text" name="Name" id="name" value="" placeholder="Name" required ><span id="n\_error" class="error">x</span><br>

<label for="Email"> Email</label><br>

<input type="email" name="Email" id="email" value="" placeholder="Email" required > <span id="e\_error" class="error">x</span><br>

<label for = "Car Type">Car Type</label>

<select name = "Car Type">

<option value = "Mini">Mini</option>

<option value = "Sedan">Sedan</option>

<option value = "SUV">SUV</option>

<option value = "MPV">MPV</option>

</select><br>

<label for="Number"> Number</label><br>

<input type="text" name="Number" value="" id="number" placeholder="Mobile Number" required><span id="m\_error" class="error">x</span><br>

<label for="Car Number"> Car Number</label><br>

<input type="text" name="Car Number" id="name" value="" placeholder="Car Number XX - XX - XXXX" required ><span id="n\_error" class="error">x</span><br>

<button type="submit" name="Submit"> Submit</button>

<button type="reset" name="Reset"> Reset</button>

<p class ="foot">To Check Parking Rates <a href="https://chennaimetrorail.org/wp-content/uploads/2018/05/Parking-Tariff-stations.pdf"> Click here </a></p>

</form>

</div>

<footer>

<p> &copy; 2020 &bull; SRIDHAR.D

</footer>

<script type="text/javascript">

function validate() {

var name = document.getElementById('name');

var email = document.getElementById('email');

var pwd = document.getElementById('pwd');

var num = document.getElementById('number');

var test = 0;

alert(email.value);

if (email.value == "") {

document.getElementById('email').style.border = "solid 1px red";

document.getElementById('email').placeholder ="Enter a valid email address";

document.getElementById('e\_error').style.display = "inline-block";

test++;

}

// alert(num.value.length);

if (num.value.length != 10) {

document.getElementById('number').style.border = "solid 1px red";

document.getElementById('number').placeholder ="Enter a valid number";

document.getElementById('m\_error').style.display = "inline-block";

test++;

}

if(test > 0) {

return false;

}

return true;

}

</script>

</body>

</html>