Node-RED: A Comprehensive Tutorial

What is Node-RED?

Node-RED is an open-source, flow-based development tool that enables developers to wire together hardware devices, APIs, and online services. Created by IBM's Emerging Technology Services, it provides a browser-based editor that simplifies the process of creating applications with minimal programming effort. Node-RED is particularly popular in the Internet of Things (IoT) ecosystem for its ability to connect devices and services in real-time.

Key Features:

- **Flow-based Programming:** Create workflows by connecting pre-built nodes.
- **Browser-Based Interface:** An intuitive drag-and-drop interface.
- **Wide Ecosystem:** Support for various nodes and integrations.
- **Extensibility:** Add custom nodes and modules.
- **Open Source:** Free to use and backed by an active community.

Supported Hardware

Node-RED is platform-agnostic and can run on:

- **Raspberry Pi** (all models, including Raspberry Pi Zero and Raspberry Pi 4)
- **Desktop Systems:** Windows, macOS, Linux
- **Cloud Platforms:** AWS, Azure, IBM Cloud

- **Embedded Systems:** Arduino, ESP32, and other microcontrollers via communication protocols
- **Other Single Board Computers:** BeagleBone, NVIDIA Jetson Nano

How to Install Node-RED on Windows Operating System

Prerequisites:

- A Windows PC (e.g., Windows 10 or 11, 64-bit) with administrative access.
- Internet connectivity.
- Basic familiarity with the Command Prompt or PowerShell.

Step-by-Step Installation Guide:

1. **Install Node.js :**
   - **Download and install Node.js (LTS) from https://nodejs.org.**
   - **Verify with:**

```
node -v
npm -v
```

2. **Install Node-red :**

```
npm install -g --unsafe-perm node-red
```

3. **Start Node-red :**

```
node-red
```

4. Node-RED will start and display a URL (e.g., http://127.0.0.1:1880).

5. **Access the Editor:** Open a web browser on the Raspberry Pi or any device on the same network, and navigate to http://127.0.0.1:1880.

How to Make a Dashboard in Node-RED

Node-RED allows the creation of interactive dashboards using the @flowfuse/node-red-dashboard module.

Steps:

1. **Install Dashboard Nodes:** In the Node-RED editor, open the menu, select **Manage Palette**, and go to the **Install** tab. Search for @flowfuse/node-red-dashboard , node-red-node-sqlite and install them.
2. **Access Dashboard Nodes:** After installation, you'll see new nodes under the "dashboard 2" category in the palette.
3. **Add Dashboard UI Elements:** Drag and drop dashboard nodes such as switch, slider, or chart onto the workspace.
4. **Configure UI Groups and Tabs:**
   - Double-click on a dashboard node.
   - Assign it to a UI group (e.g., "Home") and tab (e.g., "Main").
5. **Deploy and Access Dashboard:** Deploy the flow and navigate to the dashboard by selecting the open Dashboard from the Dashboard 2.0 Tab or http://127.0.0.1:1880/dashboard

Sample Project: Smart IoT Dashboard with Role-Based Access using Node-Red

Objective:

The objective of this project is to develop a smart IoT dashboard using Node-RED and ESP32 for real-time environmental monitoring and device control. The system integrates weather, touch, and motion sensors, provides remote LED control, and incorporates user authentication with role-based access through an admin-managed dashboard.

Hardware Required:

- ESP32 Development Board
- LED
- BMP280 Sensor
- IR (Infrared) Sensor
- Capacitive Touch Sensor
- Breadboard and Resistor
- Jumper Wires
- Micro USB Cable

Software Required:

- Node-RED
- MQTT Broker – Mosquitto

- Arduino IDE
- SQLite – for storing user credentials and access levels
- *DB Browser for SQLite*

Steps:

1. **Software Setup:**
   - **Mosquitto Setup:**
     1. Go to https://mosquitto.org/download/ and Download the file for Windows(64 bit)
     2. Run the file as Administrator once it downloads
     3. Copy the path of the file where Mosquitto is stored. (Eg: C:\Program Files\mosquitto)
     4. Search for Environmental Variables in the search bar.
     5. Click on Environmental Variables ->System Variables->Path->New
     6. Paste the copied File path in the empty field and click Ok.

   - **Arduino IDE Setup:**

1 .Go to https://www.arduino.cc/en/software and Download the file for Windows(64 bit)

2. After Installation, Go to File -> Preferences
3. Paste https://dl.espressif.com/dl/package_esp32_index.json in the Additional Boards Manager URLs.
4. Go to Tools -> Board -> Boards Manager.
5. Search for "esp32" and install the latest library.

- **SQLite Setup:**

1. Go to https://sqlite.org/download.html and Download sqlite bundle for   Windows.
2. Extract the zip file to a separate folder and copy the folder path.
3. Search for Environmental Variables in the search bar.

4. Click on Environmental Variables ->System Variables->Path->New
5. Paste the copied File path in the empty field and click Ok.

- **DB Browser Setup :**
    1. Go to https://sqlitebrowser.org/dl/ and Download for Windows(64 bit)
    2. Install using the setup wizard and downloading it.
    3. Open it and Select “New Database”
    4. Create a folder for the database
    5. Name the database and Click Save.
    6. Edit Table window pops up.
    7. Name the table and create the table by creating columns with names and datatypes -> Select Ok.

2. **Hardware Setup:**
    - Connect the LED:
        - Long leg (anode) of the LED to 330-ohm resistor.
        - Short leg (cathode) of the LED to GND.
        - Another end of the 330-ohm resistor to GPIO 2.
    - Connect the BMP280 Sensor:
        - VCC of BMP280 to 3.3V of ESP32
        - GND of BMP280 to GND of ESP32
        - SCL of BMP280 to GPIO 22 pin of ESP32
        - SDA of BMP280 to GPIO 21 pin of ESP32
    - Connect the IR Sensor:
        - VCC of IR to 3.3V of ESP32
        - GND of IR to GND of ESP32
        - OUT of IR to GPIO 4 pin of ESP32
    - Connect the Touch Sensor:
        - VCC of Touch to 5V of ESP32
        - GND of Touch to GND of ESP32
        - I/O of Touch to GPIO 5 pin of ESP32

3. Create a Flow in Node-RED:

Nodes used in the flow:

MQTT Nodes:

- **mqtt in** - Subscribes to a specific topic from an MQTT broker. It receives messages published to that topic and outputs them to the flow as msg.payload. Used for receiving sensor data, commands, etc.

- **mqtt out** - Publishes messages to a specified MQTT topic. It takes `msg.payload` and sends it to connected devices or brokers (e.g., to control hardware like LEDs or motors).

Dashboard 2 (UI) Nodes:

- **ui_gauge -** Visual display component that shows numeric data in gauge form (e.g., temperature, humidity, speed). Takes `msg.payload` as input and updates accordingly.

- **ui_text -** Displays text on the dashboard. Used to show static or dynamic messages like status updates, sensor names, etc.
    -

- **ui_switch** - Toggle switch UI element. Outputs true/false or 1/0 when turned ON/OFF. Often used to control devices or trigger actions.

- **ui_form** - UI element that presents a form with input fields. Used for collecting data from users (e.g., login, sign-up, data entry). Outputs a structured msg.payload with field names and values.

- **ui_table** - Displays array or JSON data in a table format. Commonly used to list multiple records like users, logs, or sensor readings.

- **ui_control** - Controls the visibility and navigation of dashboard tabs or groups. Can be used to show/hide UI elements based on user role or system state.

- **ui_template** - Allows custom HTML, CSS, and JavaScript in the dashboard. Used for advanced UI customization and integration of external widgets or styles.

Logic and Function Nodes:

- **switch** - A logic node that routes messages based on the value of msg.payload or other properties. Used for conditional flows (e.g., if temperature > 30, do something).

- **function** - JavaScript code block node. You can manipulate msg, run custom logic, validate input, transform data, etc.

- **inject** - Manual trigger node that sends a predefined payload (timestamp, number, string, etc.) into the flow. Often used for testing or scheduling periodic events.
- **change** - used to modify, set, delete, or move properties of a message (`msg`) object without needing to write JavaScript like in a function node.

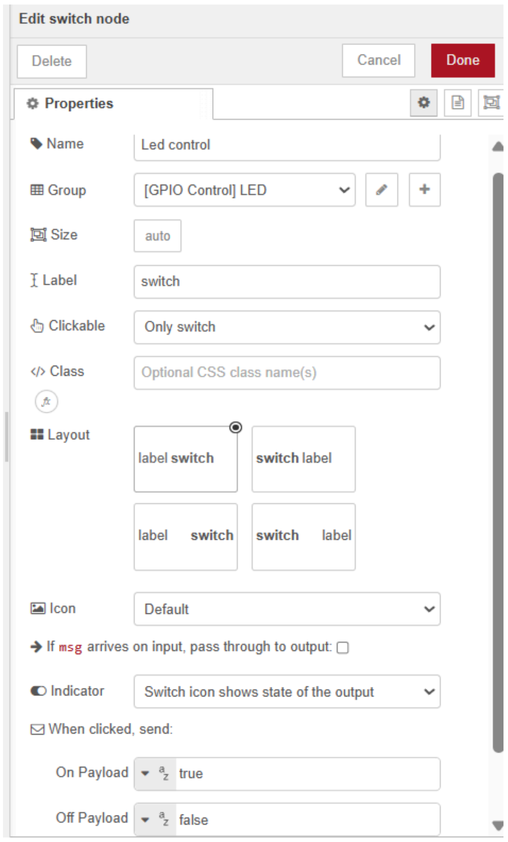
Database Node:

- **sqlite** - Executes SQL queries (SELECT, INSERT, DELETE, UPDATE) on a local SQLite database. Can store and retrieve user info, sensor logs, and more. Accepts queries via msg.topic and parameters via msg.payload.

4. Flow:

LED Flow:

- Drag "ui_switch" and "mqtt out" nodes to the flow.
- Click the nodes and edit the properties.

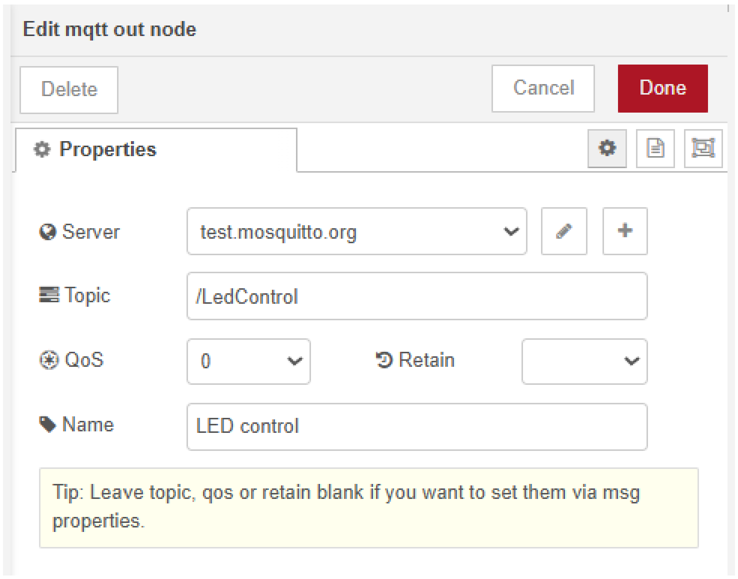


Edit switch node

Delete | Cancel | Done

Properties

Name: Led control
Group: [GPIO Control] LED
Size: auto
Label: switch
Clickable: Only switch
Class: Optional CSS class name(s)
Layout: label switch | switch label | label switch | switch label
Icon: Default
If msg arrives on input, pass through to output: ☐
Indicator: Switch icon shows state of the output
When clicked, send:
On Payload: true
Off Payload: false



Edit mqtt out node

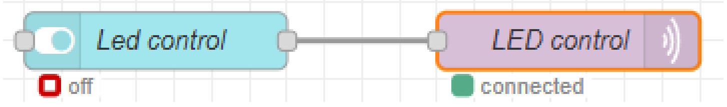
Delete | Cancel | Done

Properties

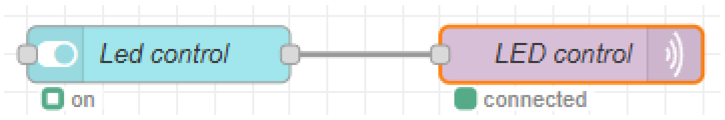
Server: test.mosquitto.org
Topic: /LedControl
QoS: 0
Retain:
Name: LED control

Tip: Leave topic, qos or retain blank if you want to set them via msg properties.

- Connect the "ui_switch" output to "mqtt out"
- Off state:

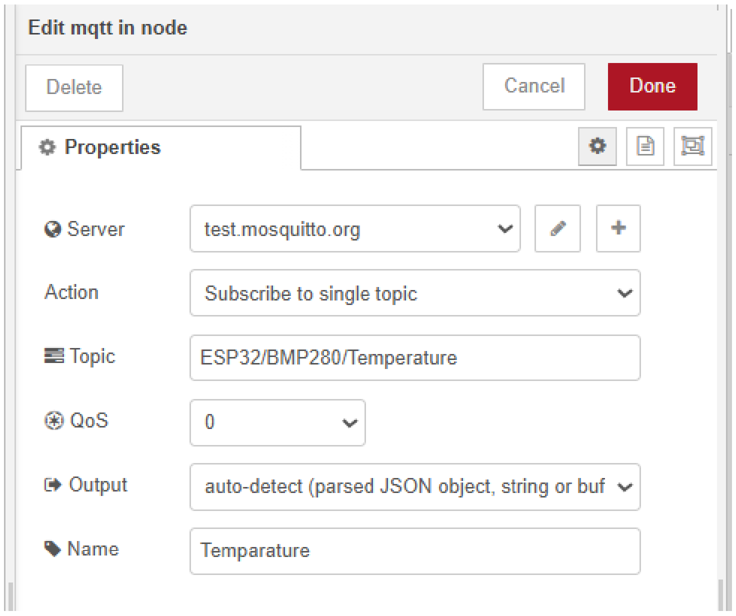


- On state:

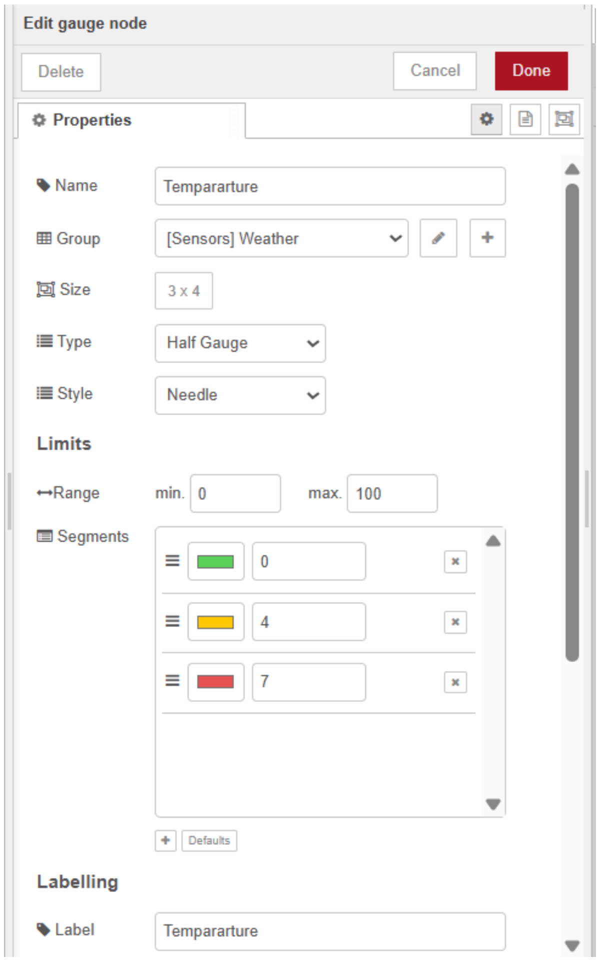


BMP280 Flow for temperature and pressure:

- Drag two “mqtt in” and two “ui_gauge” nodes
- Click the nodes and edit the properties.
- For temperature:



Server: test.mosquitto.org
Action: Subscribe to single topic
Topic: ESP32/BMP280/Temperature
QoS: 0
Output: auto-detect (parsed JSON object, string or buf
Name: Temparature



Name: Tempararture
Group: [Sensors] Weather
Size: 3 x 4
Type: Half Gauge
Style: Needle

Limits

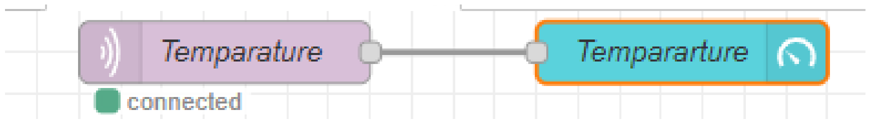
Range: min. 0 max. 100

Segments: 🟩 0, 🟨 4, 🟥 7

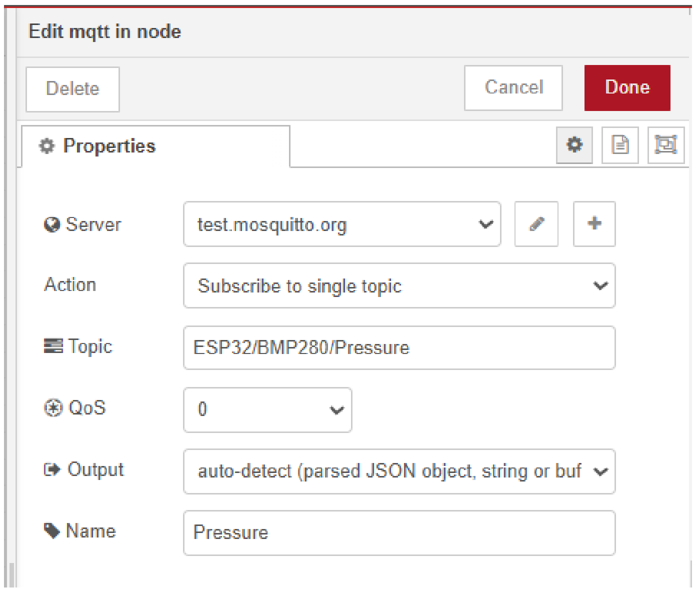
Labelling

Label: Tempararture

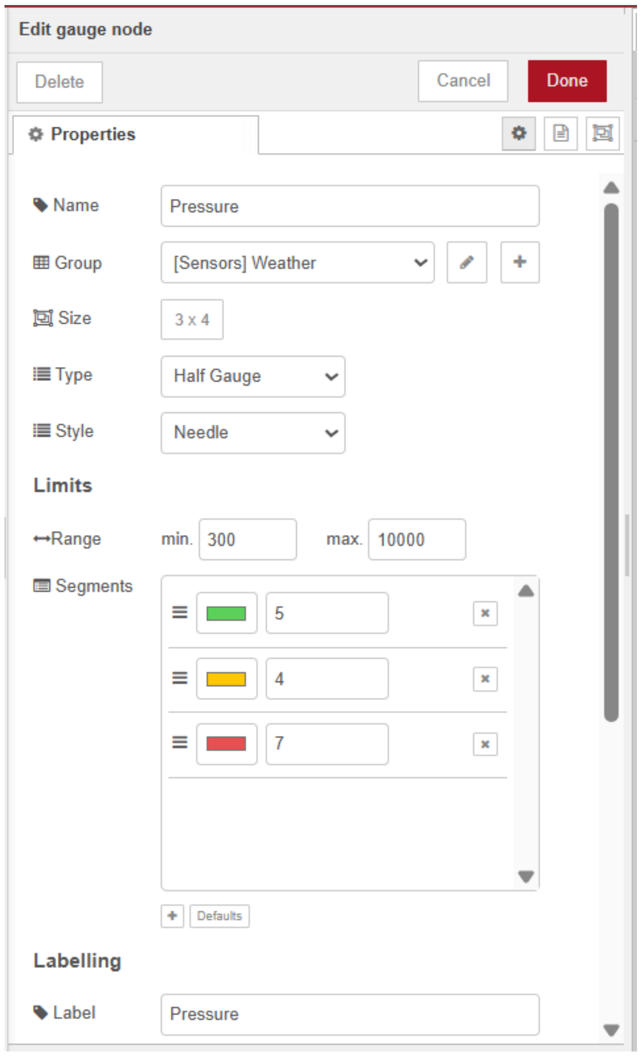
- Flow for temperature:



- For Pressure:


Edit mqtt in node

Server: test.mosquitto.org
Action: Subscribe to single topic
Topic: ESP32/BMP280/Pressure
QoS: 0
Output: auto-detect (parsed JSON object, string or buf
Name: Pressure


Edit gauge node

Name: Pressure
Group: [Sensors] Weather
Size: 3 x 4
Type: Half Gauge
Style: Needle

Limits

Range: min. 300 max. 10000

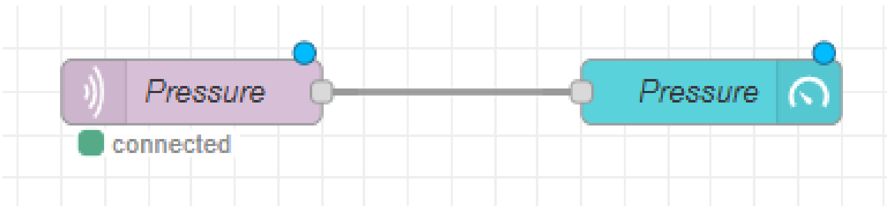
Segments:

🟩	5
🟨	4
🟥	7

Labelling

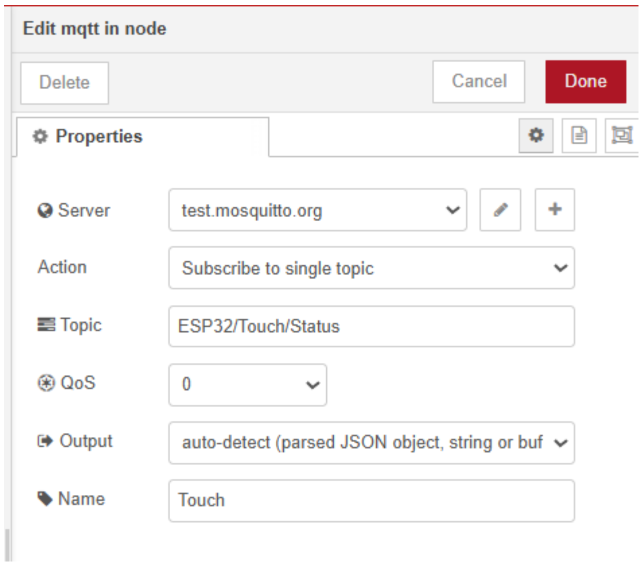
Label: Pressure

- Flow for Pressure:



Touch Sensor Flow:

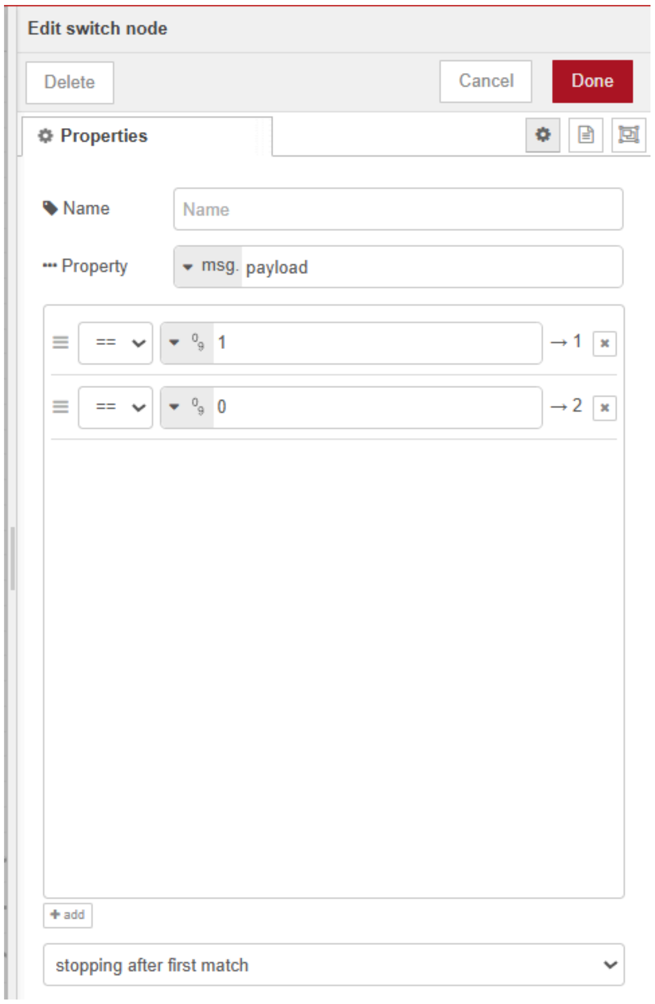
- Drag a “mqtt in” , “switch”, 2 “change” and a “ui_text” nodes.
- Click the nodes and edit the properties.


Edit mqtt in node

Delete | Cancel | Done

Properties

Server	test.mosquitto.org
Action	Subscribe to single topic
Topic	ESP32/Touch/Status
QoS	0
Output	auto-detect (parsed JSON object, string or buf
Name	Touch


Edit switch node

Delete | Cancel | Done

Properties

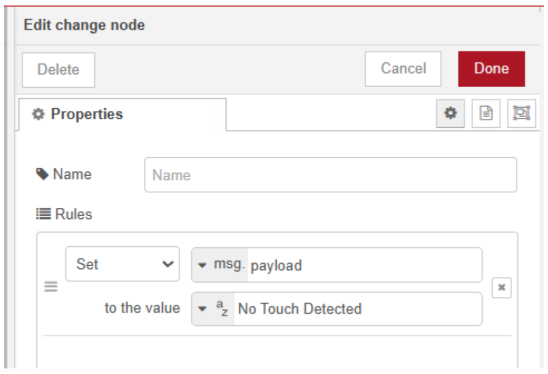
Name	Name
Property	msg. payload

==	1	→ 1
==	0	→ 2

+ add

stopping after first match

- Change nodes:


Edit change node

Delete | Cancel | Done

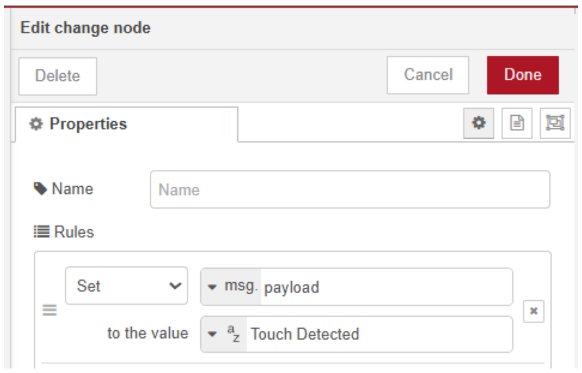
Properties

Name: Name

Rules

Set msg. payload

to the value No Touch Detected


Edit change node

Delete | Cancel | Done

Properties

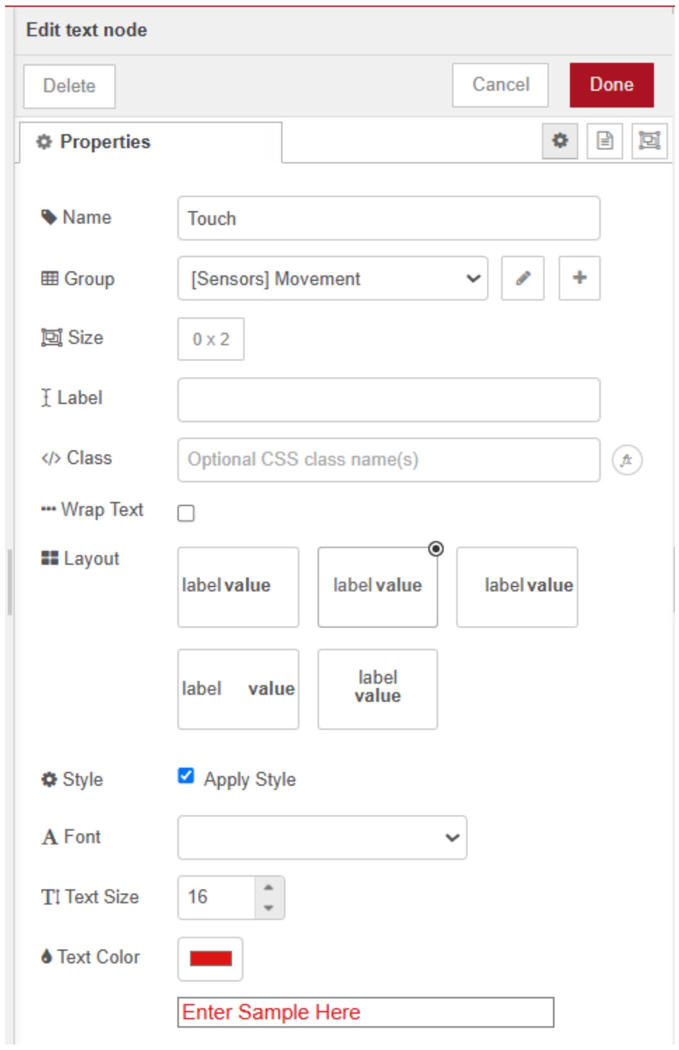
Name: Name

Rules

Set msg. payload

to the value Touch Detected

- ui_text node :



Edit text node

Delete | Cancel | Done

Properties

Name: Touch

Group: [Sensors] Movement

Size: 0 x 2

Label:

Class: Optional CSS class name(s)

Wrap Text: ☐

Layout: label value | label value [✓] | label value | label value | label value

Style: ☑ Apply Style

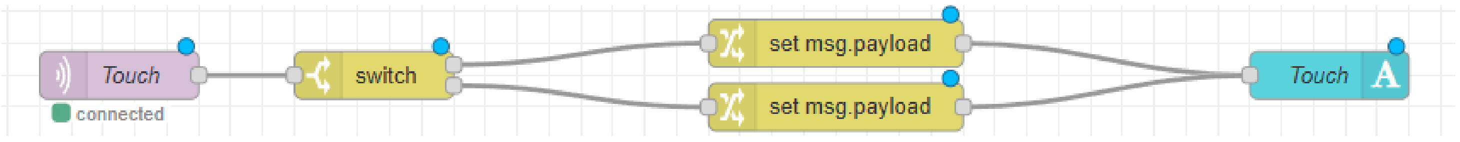
Font:

Text Size: 16

Text Color: 🟥

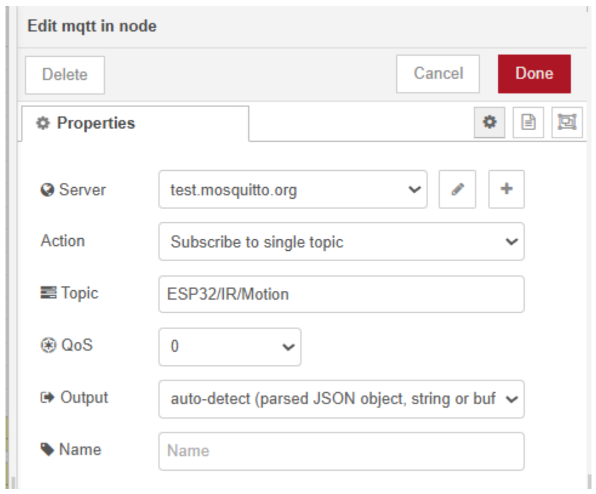
Enter Sample Here

- Connect the nodes according to the flow:

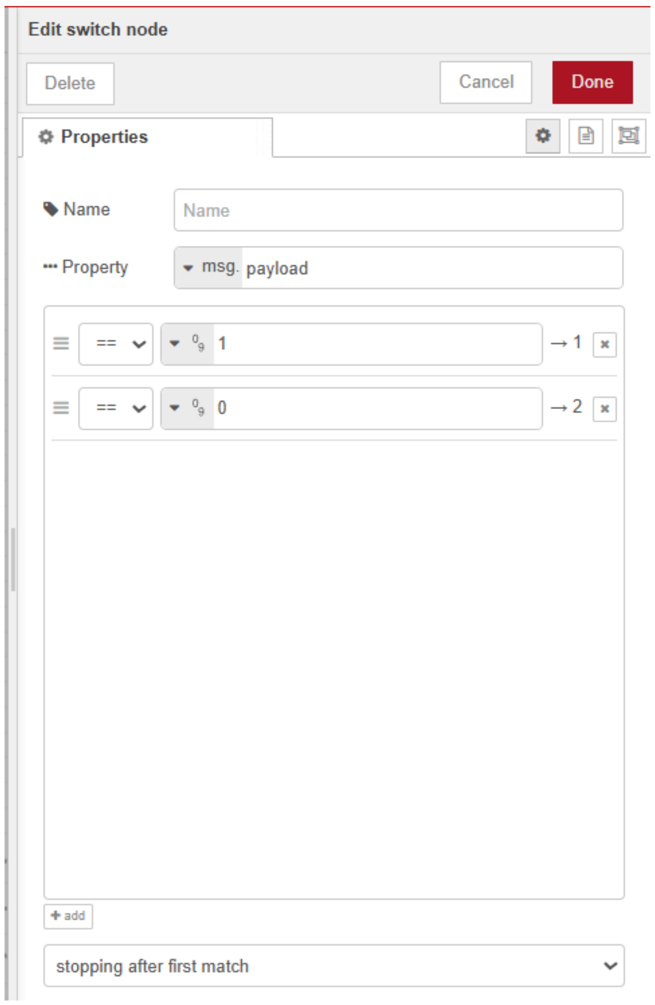


IR Sensor Flow:

- Drag a “mqtt in” , “switch”, 2 “change” and a “ui_text” nodes.
- Click the nodes and edit the properties.

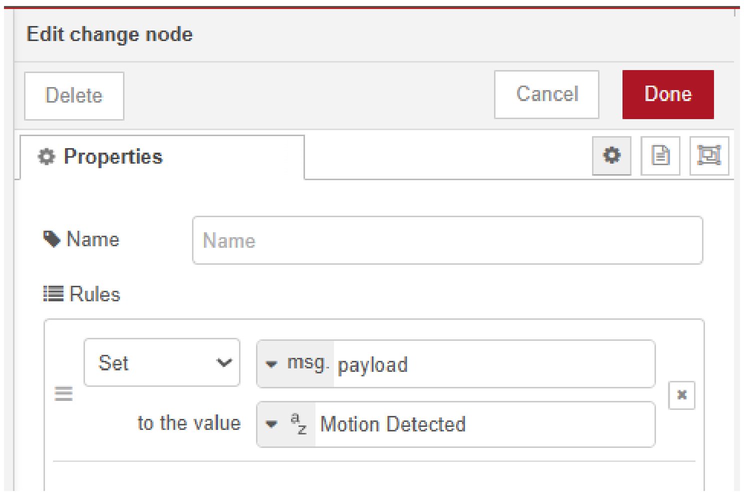


Server: test.mosquitto.org
Action: Subscribe to single topic
Topic: ESP32/IR/Motion
QoS: 0
Output: auto-detect (parsed JSON object, string or buf
Name: Name

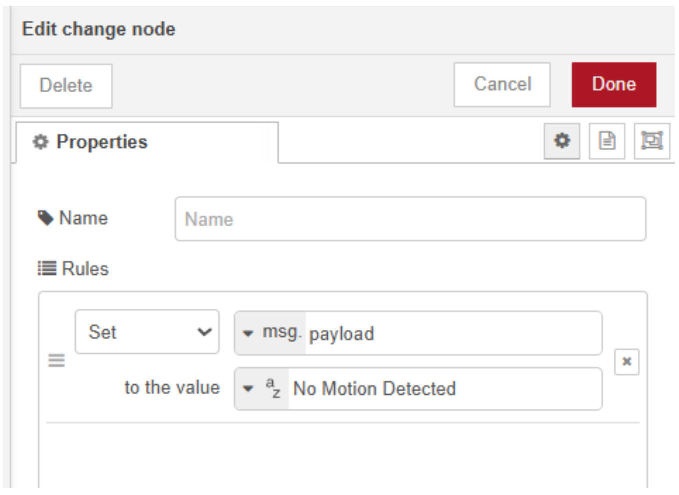


Name: Name
Property: msg. payload
== 1 → 1
== 0 → 2
stopping after first match

- change nodes:

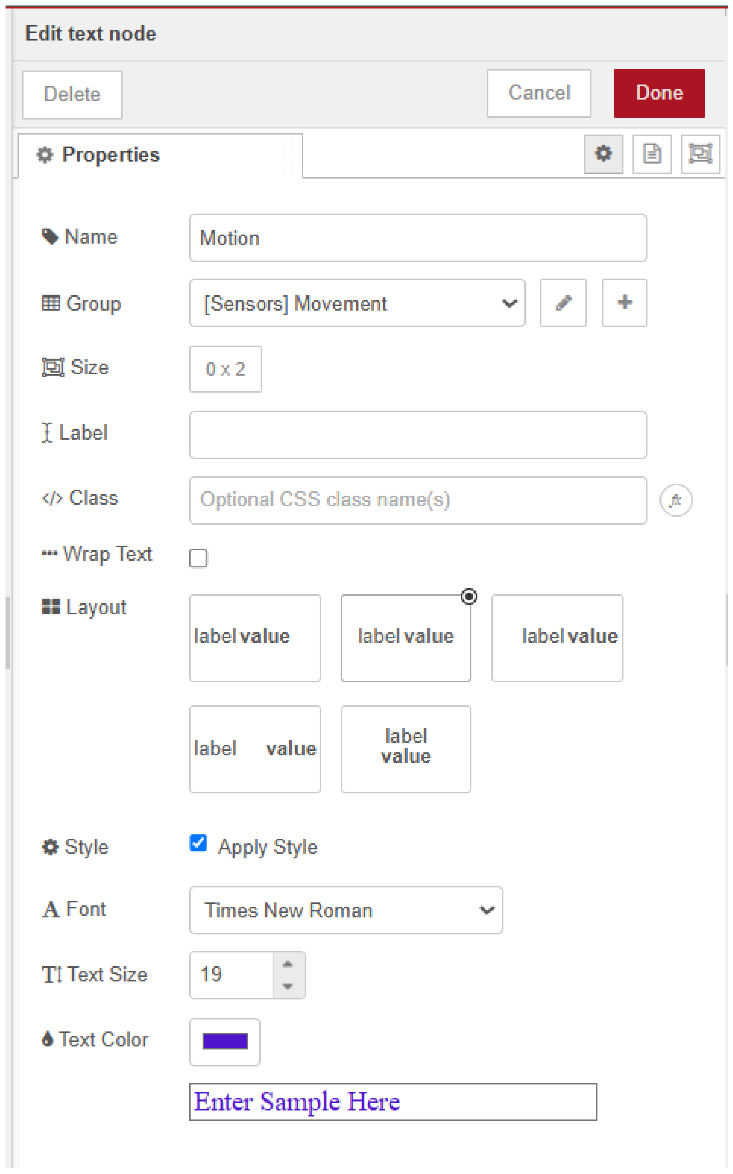


Name: Name
Rules
Set msg. payload
to the value Motion Detected



Name: Name
Rules
Set msg. payload
to the value No Motion Detected

- ui_text node



Edit text node

Delete | Cancel | Done

Properties

Name: Motion

Group: [Sensors] Movement

Size: 0 x 2

Label:

Class: Optional CSS class name(s)

Wrap Text: ☐

Layout: label value | [✓ label value] | label value | label value | label value

Style: ☑ Apply Style

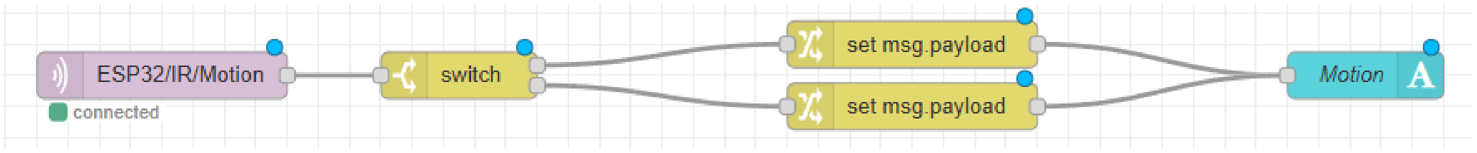
Font: Times New Roman

Text Size: 19

Text Color: 🟪

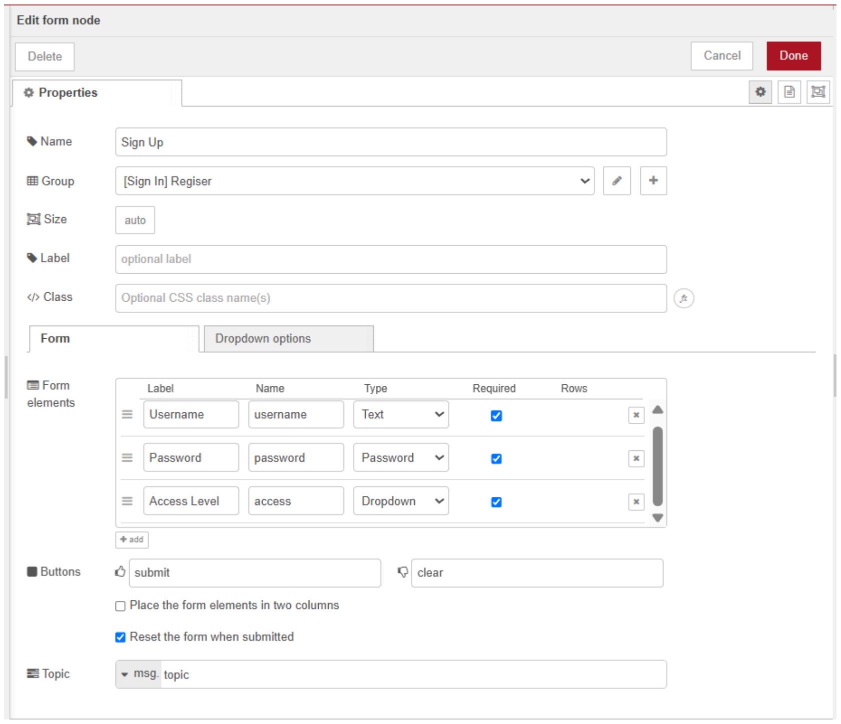
Enter Sample Here

- Connect the nodes according to the flow:



Sign Up page flow:

- Drag a “ui_form”, “function”, “sqlite”, “change”, “ui_text”.
- Click the nodes and edit the properties.
- ui_form:



Edit form node

Delete | Cancel | Done

Properties

Name: Sign Up
Group: [Sign In] Regiser
Size: auto
Label: optional label
Class: Optional CSS class name(s)

Form | Dropdown options

Form elements

Label	Name	Type	Required	Rows
Username	username	Text	✓	
Password	password	Password	✓	
Access Level	access	Dropdown	✓	

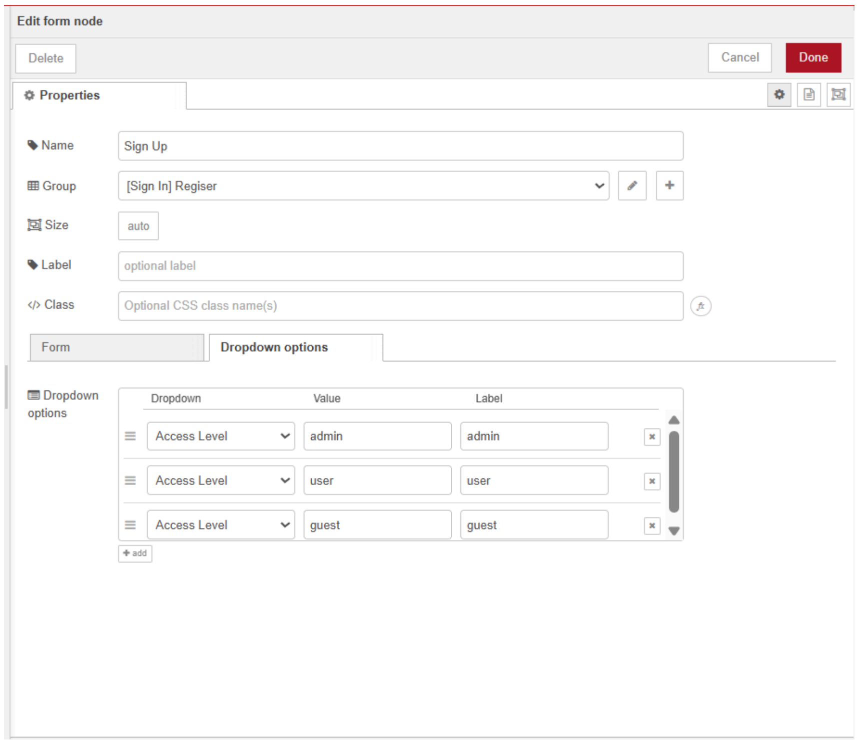
add

Buttons: submit | clear

☐ Place the form elements in two columns

☑ Reset the form when submitted

Topic: msg. topic



Edit form node

Delete | Cancel | Done

Properties

Name: Sign Up
Group: [Sign In] Regiser
Size: auto
Label: optional label
Class: Optional CSS class name(s)

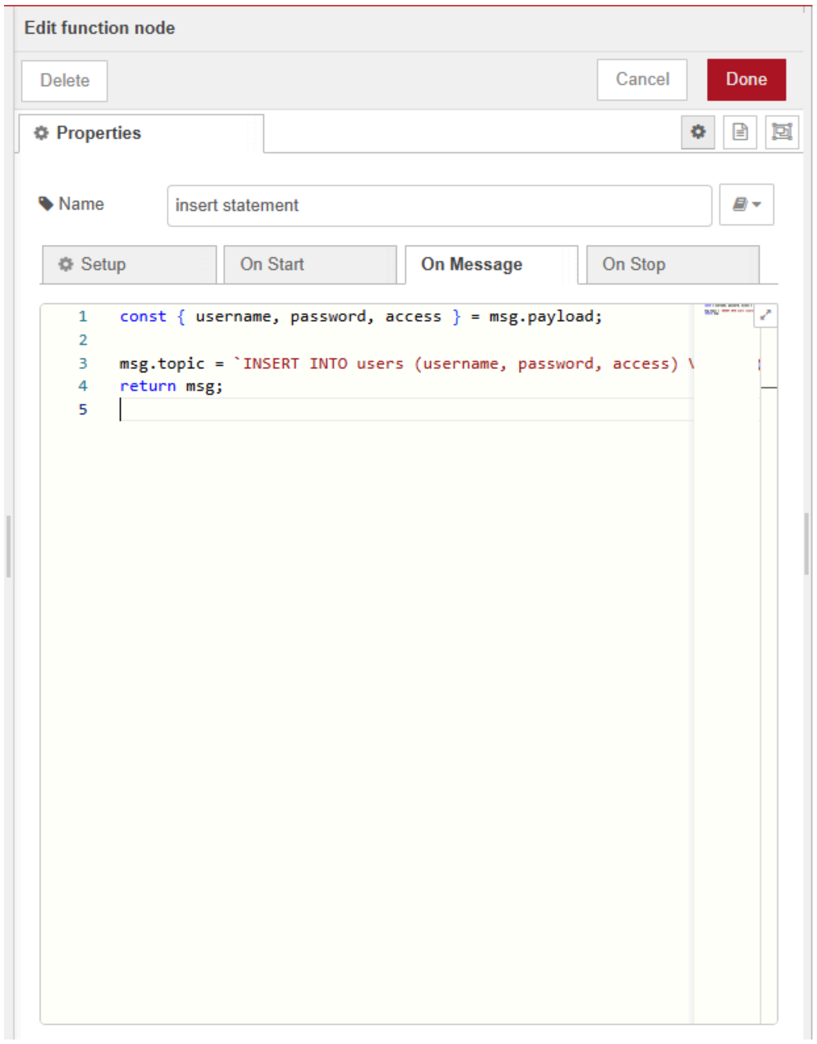
Form | Dropdown options

Dropdown options

Dropdown	Value	Label
Access Level	admin	admin
Access Level	user	user
Access Level	guest	guest

add

- function node:



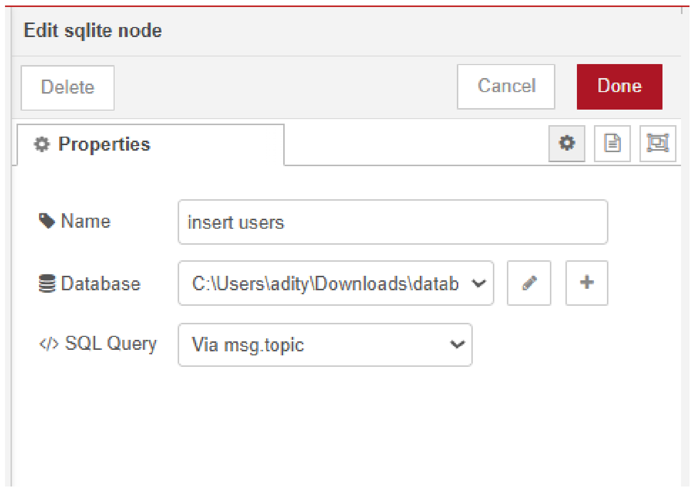
insert statement

```
const { username, password, access } = msg.payload;

msg.topic = `INSERT INTO users (username, password, access) \
return msg;
```

- function node code:

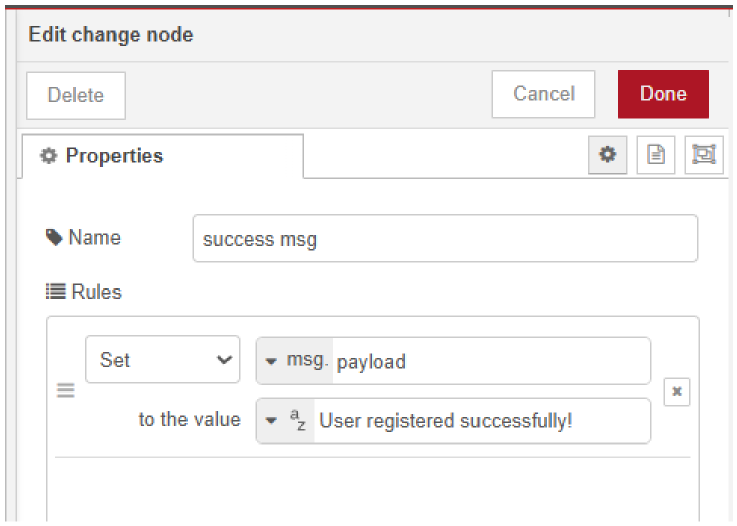
```
const { username, password, access } = msg.payload;

msg.topic = `INSERT INTO users (username, password, access) VALUES
('${username}', '${password}', '${access}')`;
return msg;
```

- sqlite node:



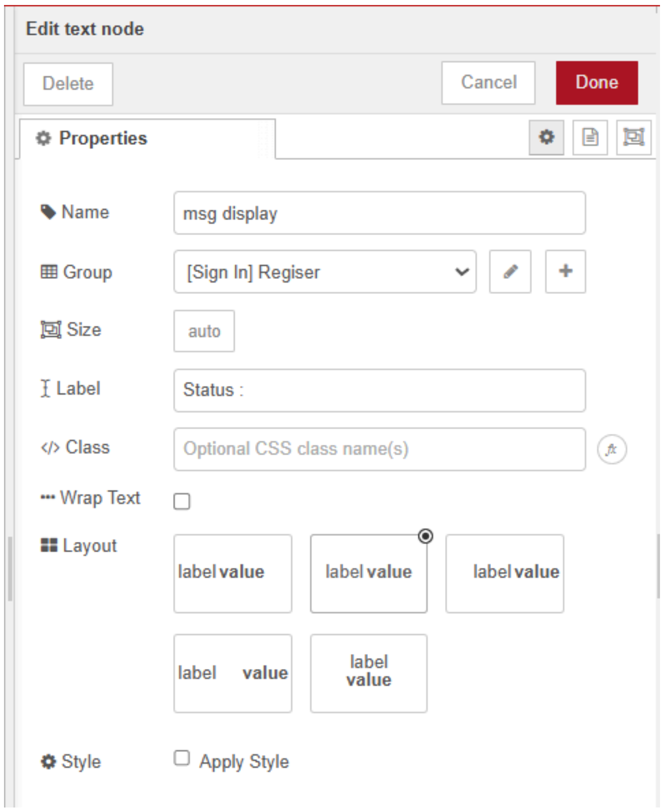
Name: insert users
Database: C:\Users\adity\Downloads\datab
SQL Query: Via msg.topic

- change node:



Name: success msg
Rules
Set msg. payload
to the value User registered successfully!

- ui_text node:



Edit text node

Name: msg display

Group: [Sign In] Regiser

Size: auto

Label: Status :

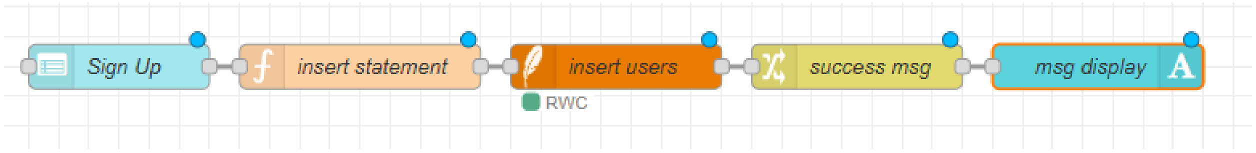
Class: Optional CSS class name(s)

Wrap Text: ☐

Layout: label value | [✓ label value] | label value | label value | label value

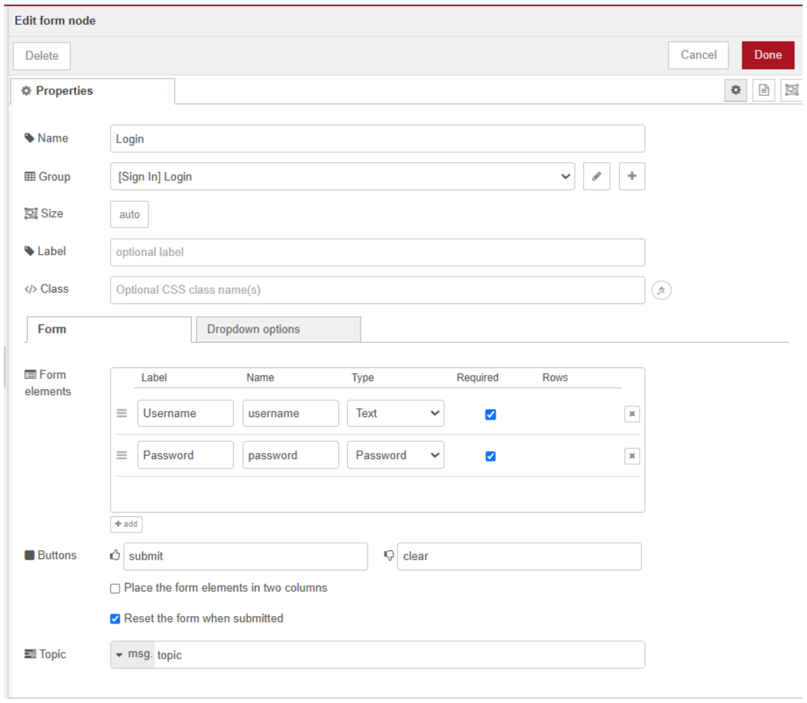
Style: ☐ Apply Style

- connect the nodes according to the flow :



Login page flow:

- Drag a “ui_form”, “function”, “sqlite”, “switch”, “ui_control”, “ui_template”, “inject”.
- Click the nodes and edit the properties.
- ui_form(Login):


Edit form node

Delete | Cancel | Done

Properties

Name	Login
Group	[Sign In] Login
Size	auto
Label	optional label
Class	Optional CSS class name(s)

Form | Dropdown options

Form elements

Label	Name	Type	Required	Rows
Username	username	Text	✓	
Password	password	Password	✓	

+ add

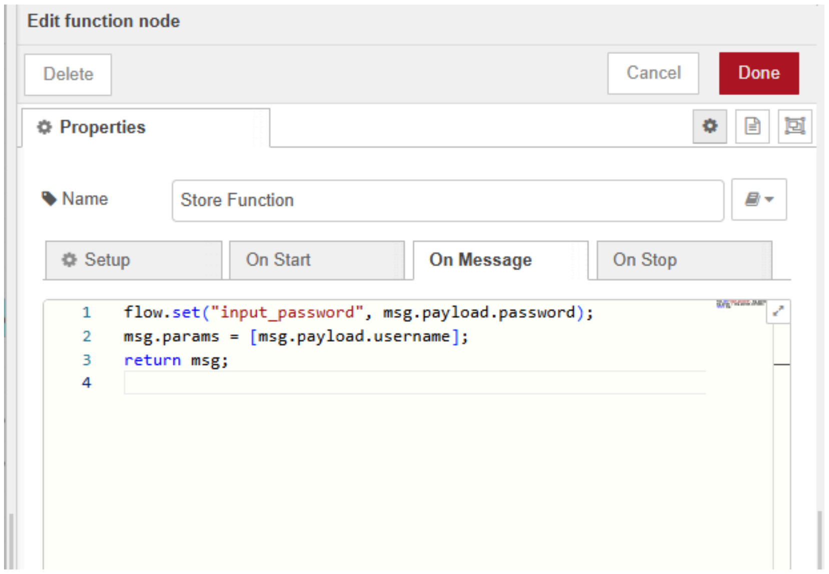
Buttons: submit | clear

☐ Place the form elements in two columns

☑ Reset the form when submitted

Topic: msg. topic

- Store function


Edit function node

Delete | Cancel | Done

Properties

Name: Store Function

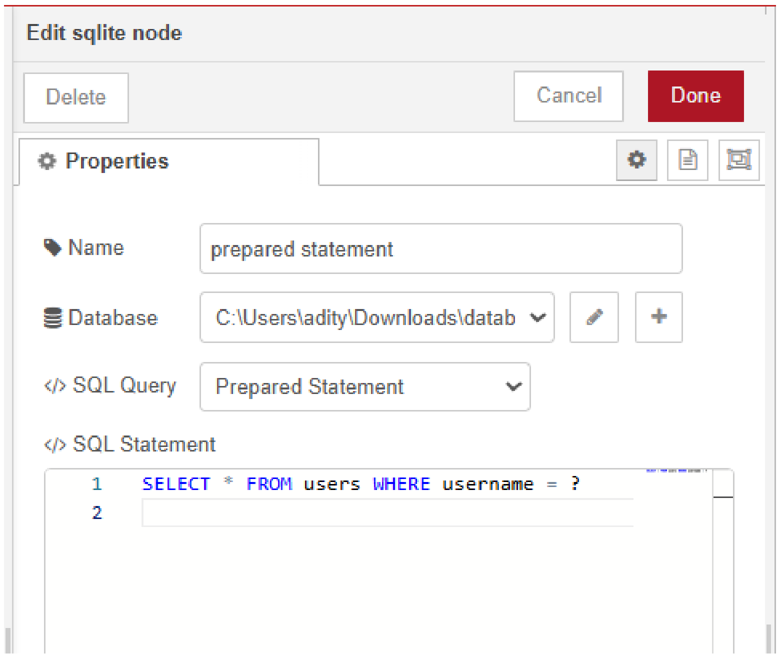
Setup | On Start | On Message | On Stop

```
flow.set("input_password", msg.payload.password);
msg.params = [msg.payload.username];
return msg;
```

- function code:

```
flow.set("input_password", msg.payload.password);
msg.params = [msg.payload.username];
return msg;
```

- sqlite(prepared statement):



Name: prepared statement

Database: C:\Users\adity\Downloads\datab

SQL Query: Prepared Statement

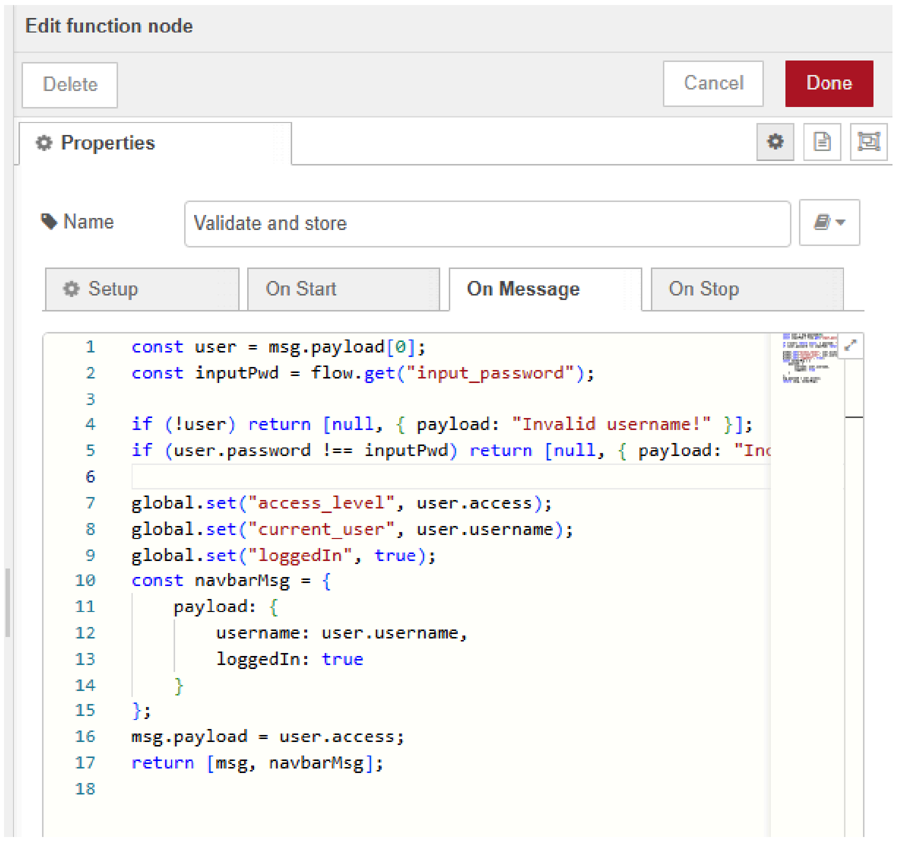
SQL Statement

```
SELECT * FROM users WHERE username = ?
```

- sqlite code:

```
SELECT * FROM users WHERE username = ?
```

- Validate and Store function:



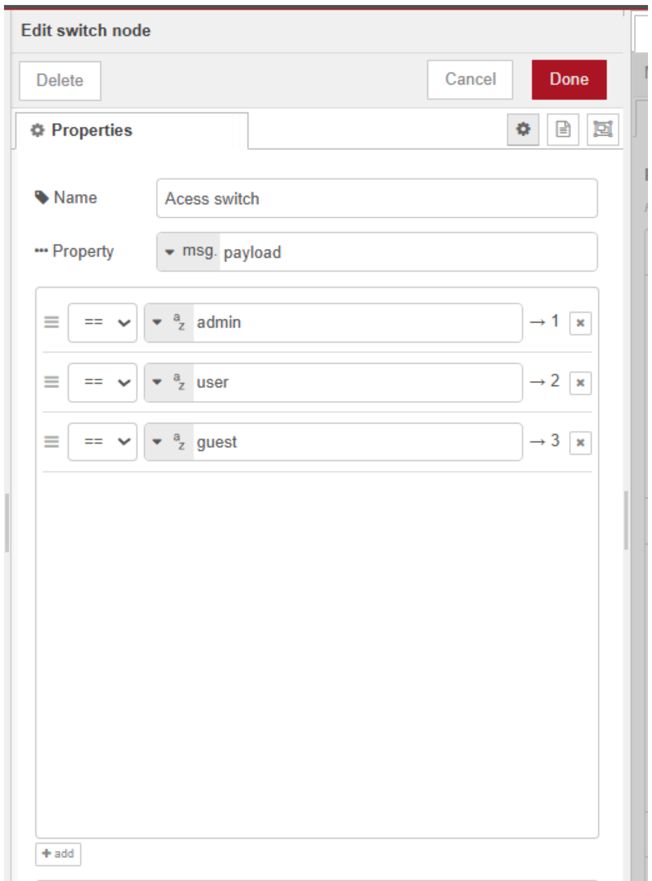
Name: Validate and store

```
const user = msg.payload[0];
const inputPwd = flow.get("input_password");

if (!user) return [null, { payload: "Invalid username!" }];
if (user.password !== inputPwd) return [null, { payload: "Inc

global.set("access_level", user.access);
global.set("current_user", user.username);
global.set("loggedIn", true);
const navbarMsg = {
    payload: {
        username: user.username,
        loggedIn: true
    }
};
msg.payload = user.access;
return [msg, navbarMsg];
```

- Validate and Store function code:

```
const user = msg.payload[0];
const inputPwd = flow.get("input_password");

if (!user) return [null, { payload: "Invalid username!" }];
if (user.password !== inputPwd) return [null, { payload: "Incorrect password!" }];

global.set("access_level", user.access);
global.set("current_user", user.username);
global.set("loggedIn", true);
const navbarMsg = {
   payload: {
      username: user.username,
      loggedIn: true
   }
};
msg.payload = user.access;
return [msg, navbarMsg];
```

- Access switch:

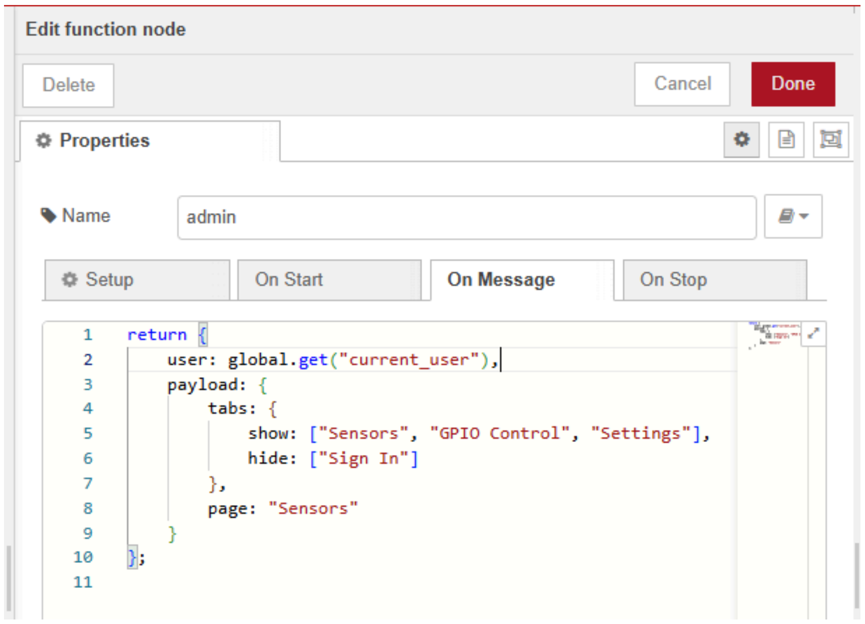

Edit switch node

Name: Acess switch

Property: msg. payload

==	admin	→ 1	
==	user	→ 2	
==	guest	→ 3	

- admin function:



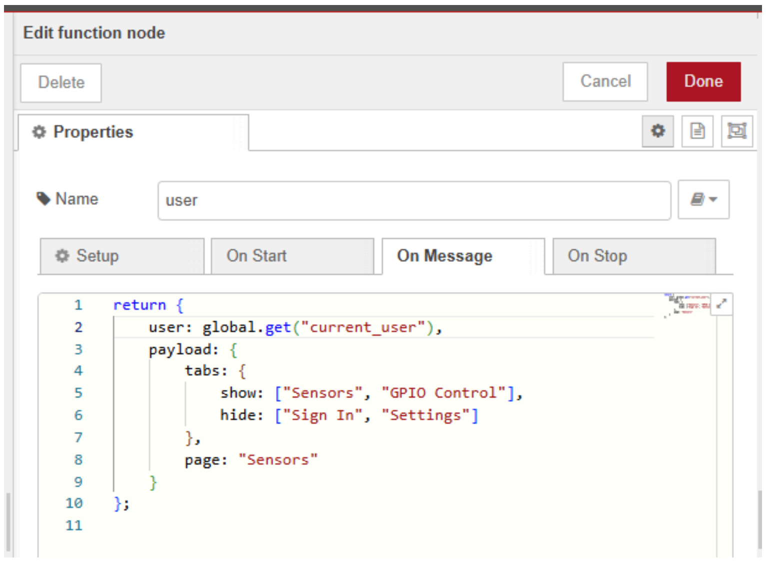
Name: admin

```
return {
    user: global.get("current_user"),
    payload: {
        tabs: {
            show: ["Sensors", "GPIO Control", "Settings"],
            hide: ["Sign In"]
        },
        page: "Sensors"
    }
};
```

- admin function code:

```
return {
   user: global.get("current_user"),
   payload: {
      tabs: {
         show: ["Sensors", "GPIO Control", "Settings"],
         hide: ["Sign In"]
      },
      page: "Sensors"
   }
};
```

- User function :



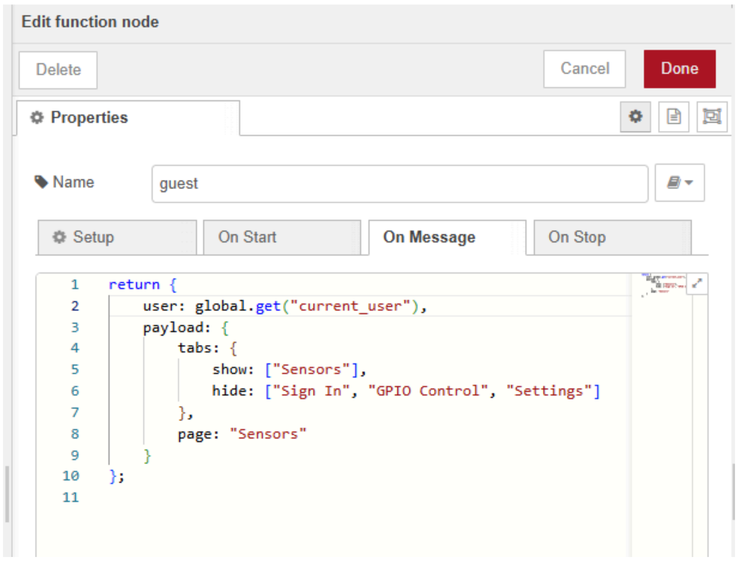
Name: user

```
return {
    user: global.get("current_user"),
    payload: {
        tabs: {
            show: ["Sensors", "GPIO Control"],
            hide: ["Sign In", "Settings"]
        },
        page: "Sensors"
    }
};
```

- User function code:

```
return {
   user: global.get("current_user"),
   payload: {
      tabs: {
         show: ["Sensors", "GPIO Control"],
         hide: ["Sign In", "Settings"]
      },
      page: "Sensors"
   }
};
```

- Guest function :



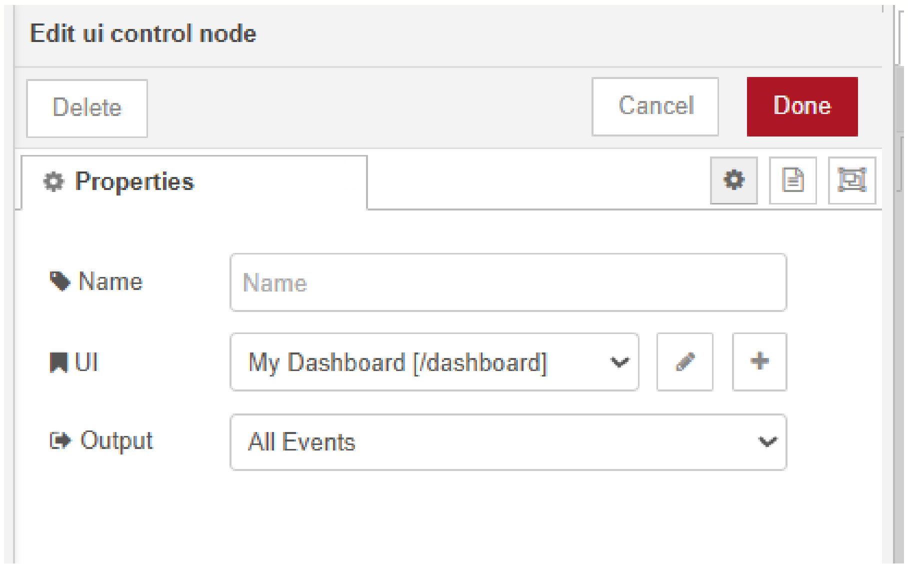
Name: guest

```
return {
    user: global.get("current_user"),
    payload: {
        tabs: {
            show: ["Sensors"],
            hide: ["Sign In", "GPIO Control", "Settings"]
        },
        page: "Sensors"
    }
};
```

- Guest function code :

```
return {
    user: global.get("current_user"),
    payload: {
        tabs: {
            show: ["Sensors"],
            hide: ["Sign In", "GPIO Control", "Settings"]
        },
        page: "Sensors"
    }
};
```

- `ui_control` :

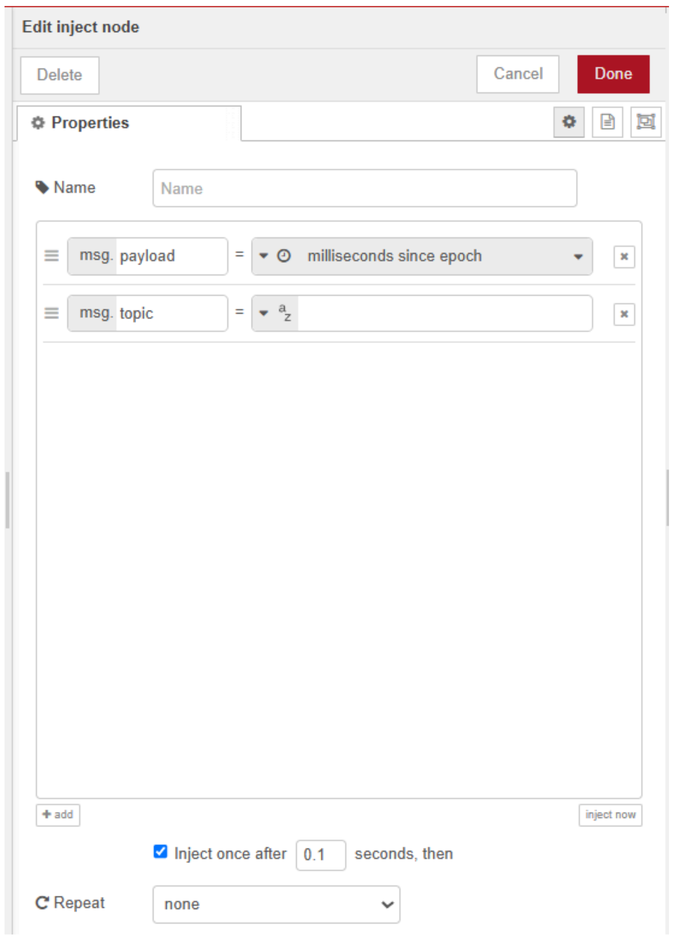


Name: Name

UI: My Dashboard [/dashboard]

Output: All Events

- Inject node :



Name: Name

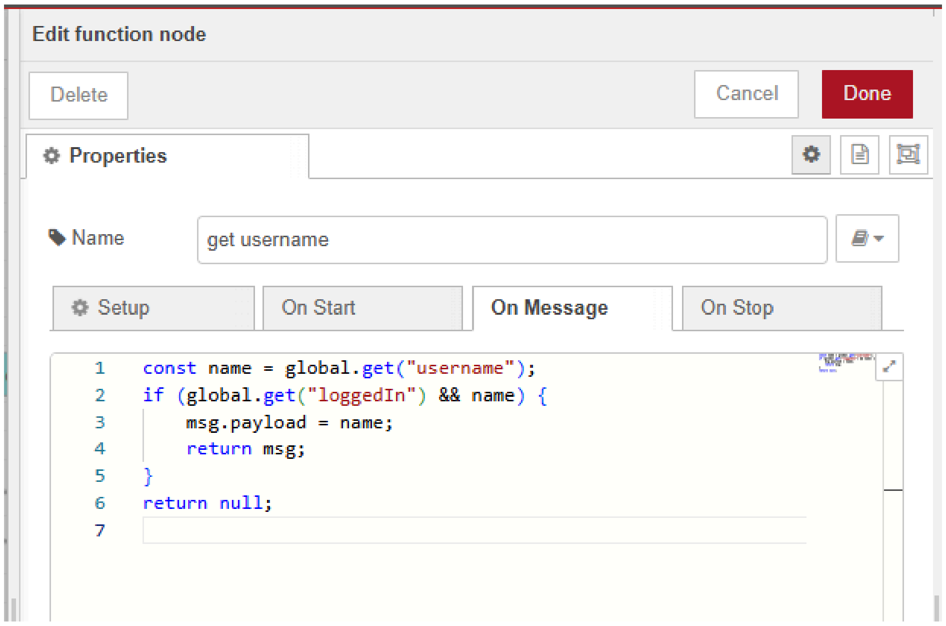
msg. payload = milliseconds since epoch

msg. topic =

☑ Inject once after 0.1 seconds, then

Repeat: none

- Get username function :



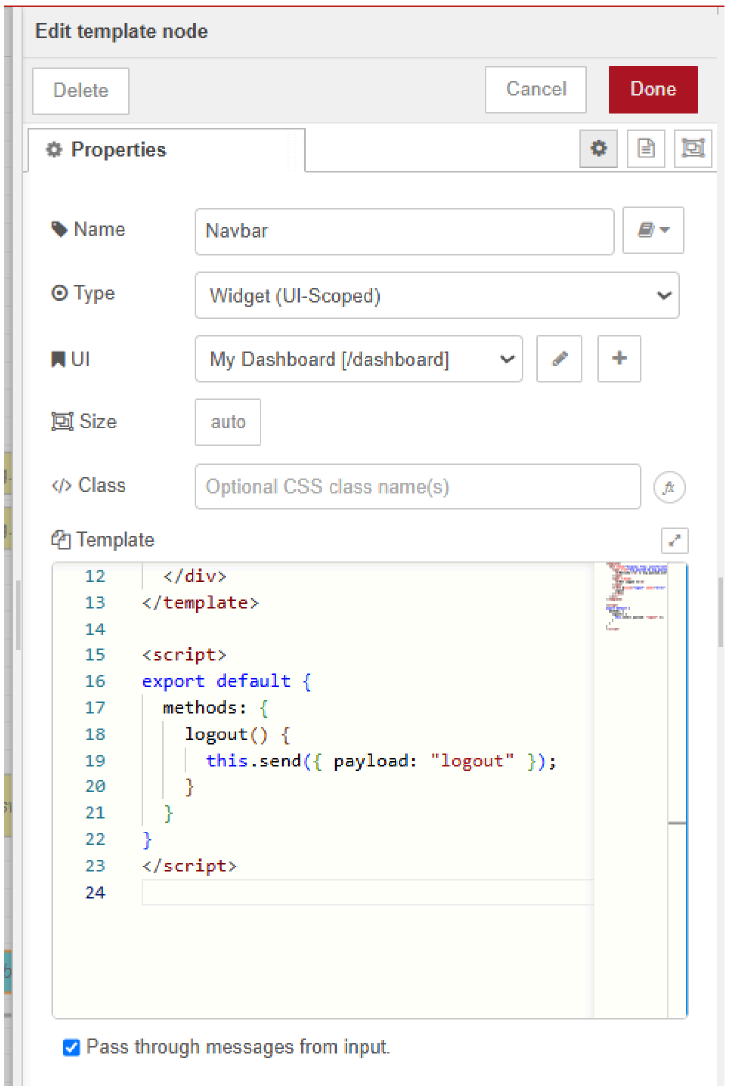
Name: get username

```
const name = global.get("username");
if (global.get("loggedIn") && name) {
    msg.payload = name;
    return msg;
}
return null;
```

- Get username function code :

```
const name = global.get("username");
if (global.get("loggedIn") && name) {
   msg.payload = name;
   return msg;
}
return null;
```

- ui_template :



Edit template node

Delete | Cancel | Done

Properties

Name: Navbar

Type: Widget (UI-Scoped)

UI: My Dashboard [/dashboard]

Size: auto

Class: Optional CSS class name(s)

Template

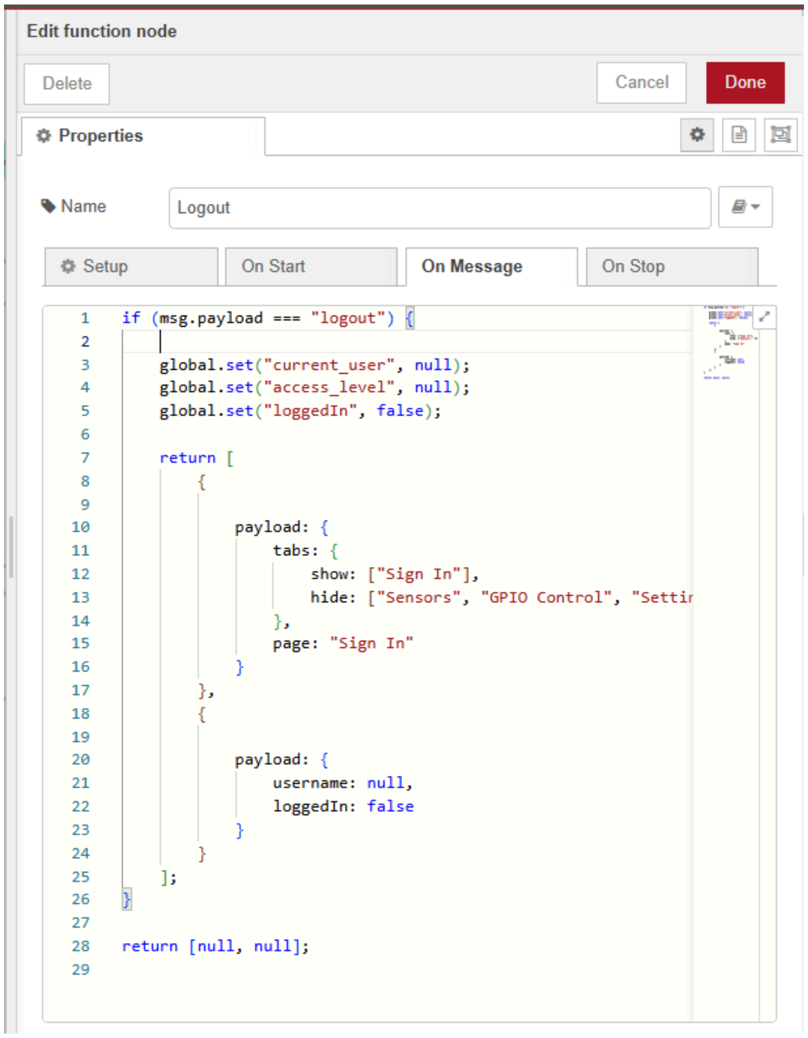
```
  </div>
</template>

<script>
export default {
  methods: {
    logout() {
      this.send({ payload: "logout" });
    }
  }
}
</script>

```

☑ Pass through messages from input.

- ui_template code :

```
<template>
  <div style="display: flex; justify-content: space-between; align-items: center;">
    <span v-if="msg.payload && msg.payload.loggedIn">
      <b>Welcome:</b> {{ msg.payload.username }}
    </span>
    <span v-else>
      <b>Not Logged In</b>
    </span>
    <v-btn @click="logout" color="error" dark small style="margin-left: auto;">
      Logout
    </v-btn>
  </div>
</template>

<script>
export default {
  methods: {
    logout() {
      this.send({ payload: "logout" });
    }
  }
}
</script>
```

- Logout function :



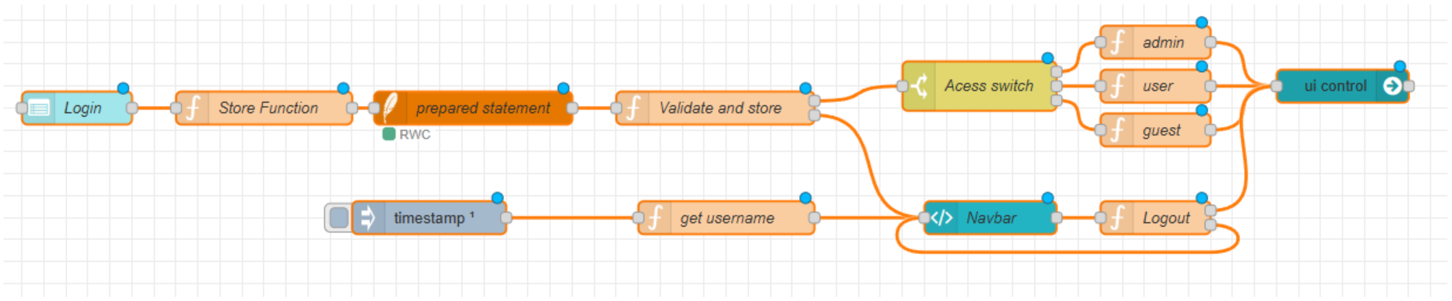
Name: Logout

```
if (msg.payload === "logout") {
    
    global.set("current_user", null);
    global.set("access_level", null);
    global.set("loggedIn", false);

    return [
        {

            payload: {
                tabs: {
                    show: ["Sign In"],
                    hide: ["Sensors", "GPIO Control", "Settir
                },
                page: "Sign In"
            }
        },
        {

            payload: {
                username: null,
                loggedIn: false
            }
        }
    ];
}

return [null, null];
```

Logout function code :

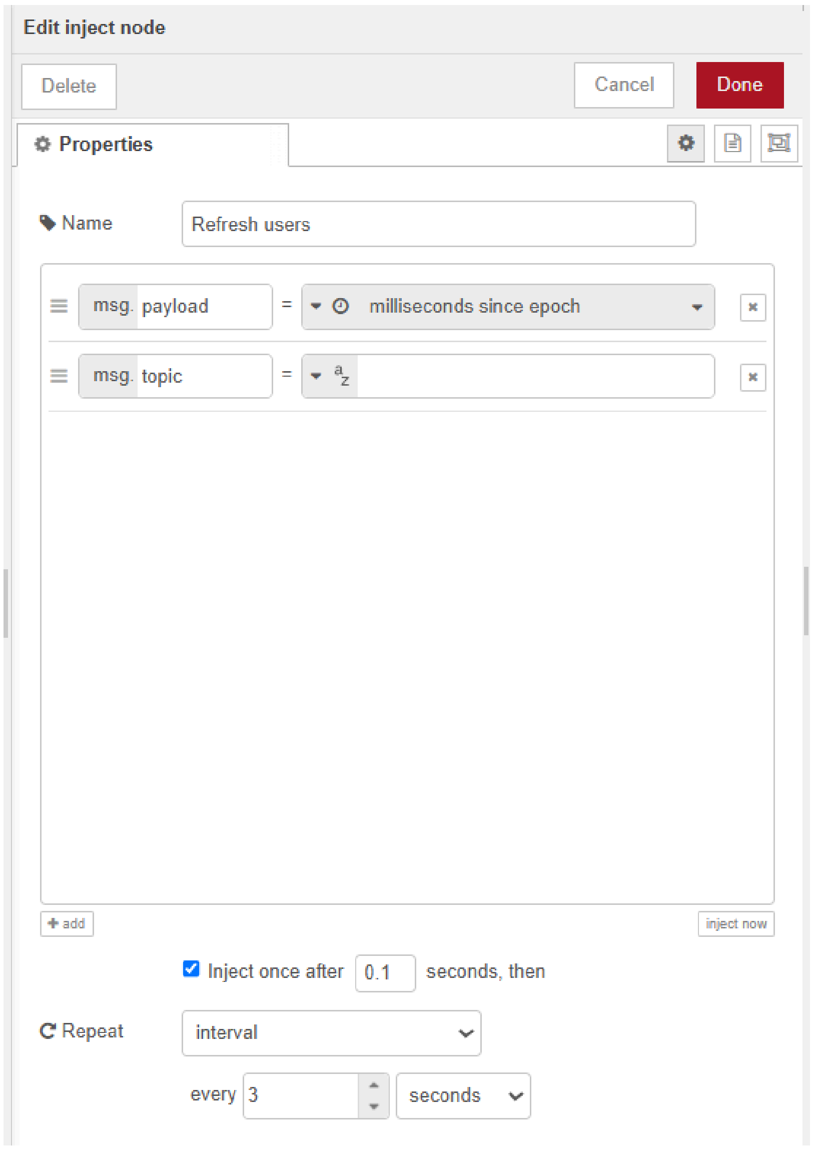
```
if (msg.payload === "logout") {

    global.set("current_user", null);
    global.set("access_level", null);
    global.set("loggedIn", false);

    return [
        {

            payload: {
                    tabs: {
                        show: ["Sign In"],
                        hide: ["Sensors", "GPIO Control", "Settings"]
                    },
                    page: "Sign In"
                }
            },
            {

                payload: {
                    username: null,
                    loggedIn: false
                }
            }
        ];
}

return [null, null];
```

- Connect the nodes according to the flow.



Settings page flow:

- Drag a “ui_form”, “function”, “sqlite”, “inject”, “ui_table” nodes.
- Click the nodes and edit the properties.
- Inject node :


Edit inject node

Delete Cancel Done

Properties

Name: Refresh users

msg. payload = milliseconds since epoch

msg. topic =

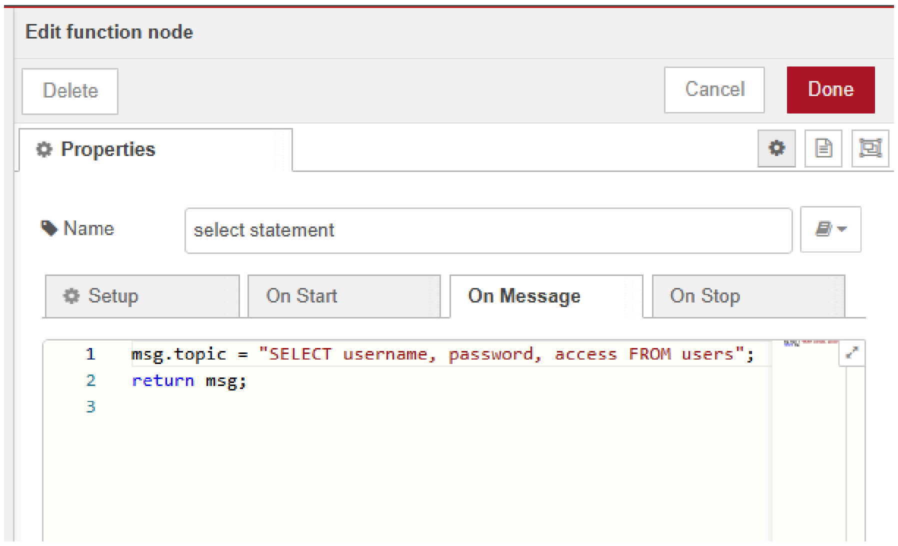
add inject now

☑ Inject once after 0.1 seconds, then

Repeat: interval

every 3 seconds

- Select statement function :


Edit function node

Delete Cancel Done

Properties

Name: select statement

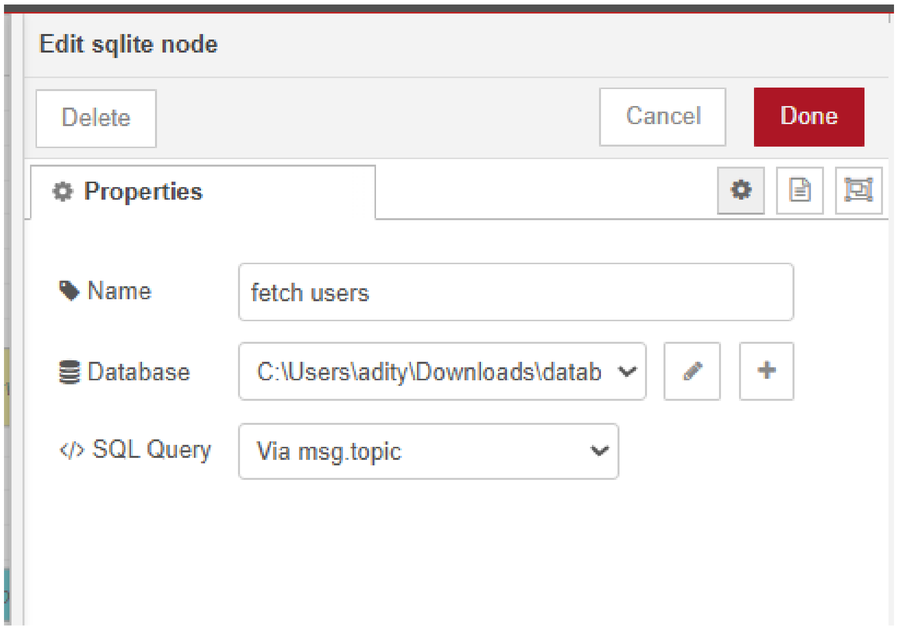
Setup | On Start | On Message | On Stop

```
msg.topic = "SELECT username, password, access FROM users";
return msg;

```

- Select statement function code :

```
msg.topic = "SELECT username, password, access FROM users";
return msg;
```

- Fetch users (sqlite) node :



Edit sqlite node

Delete Cancel Done

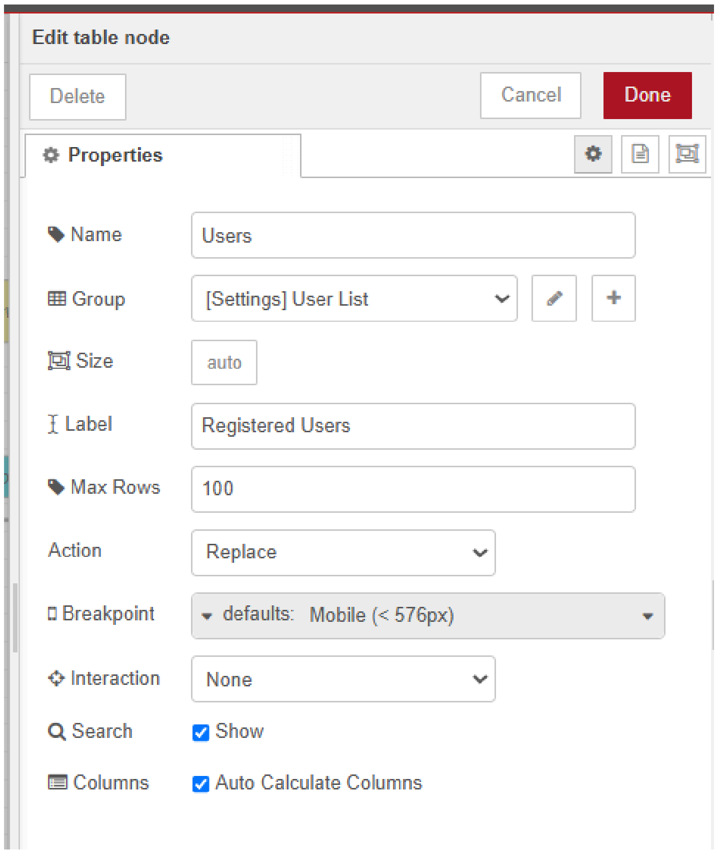
Properties

Name: fetch users

Database: C:\Users\adity\Downloads\datab

SQL Query: Via msg.topic

- Users table :



Edit table node

Delete Cancel Done

Properties

Name: Users

Group: [Settings] User List

Size: auto

Label: Registered Users

Max Rows: 100

Action: Replace

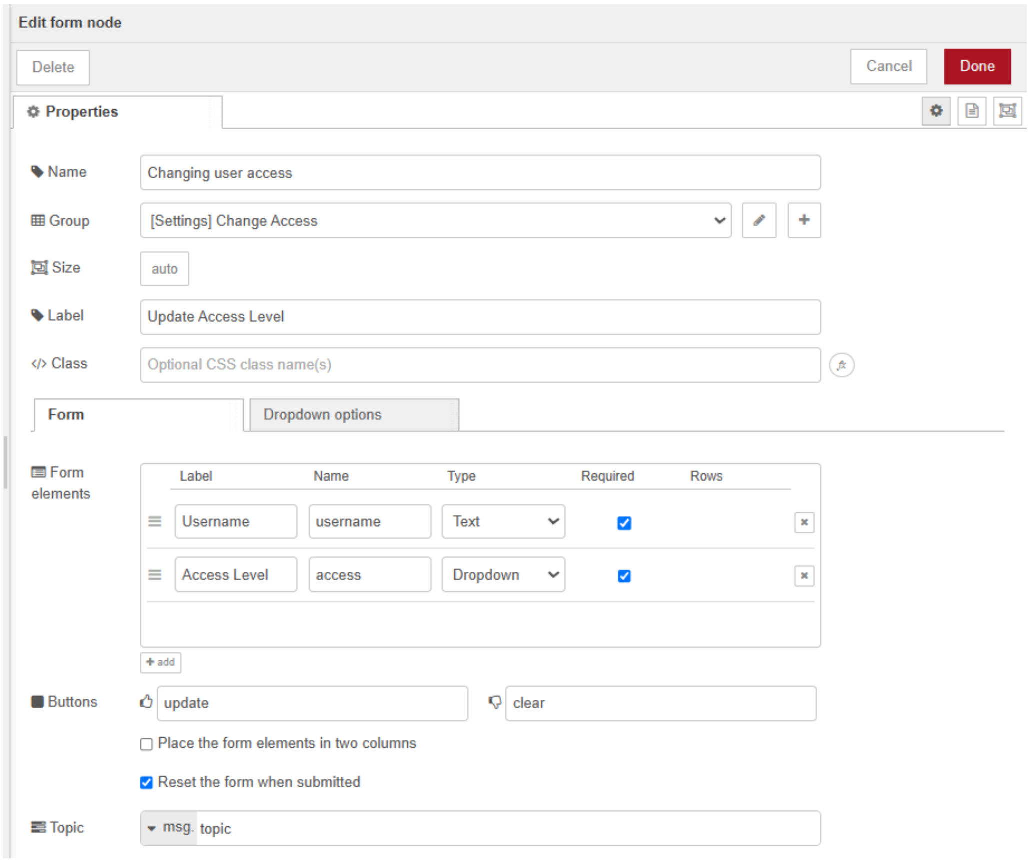
Breakpoint: defaults: Mobile (< 576px)

Interaction: None

Search: ✓ Show

Columns: ✓ Auto Calculate Columns

- Changing user access form



Edit form node

Delete | Cancel | Done

Properties

Name	Changing user access
Group	[Settings] Change Access
Size	auto
Label	Update Access Level
Class	Optional CSS class name(s)

Form | Dropdown options

Form elements

Label	Name	Type	Required	Rows
Username	username	Text	✓	
Access Level	access	Dropdown	✓	

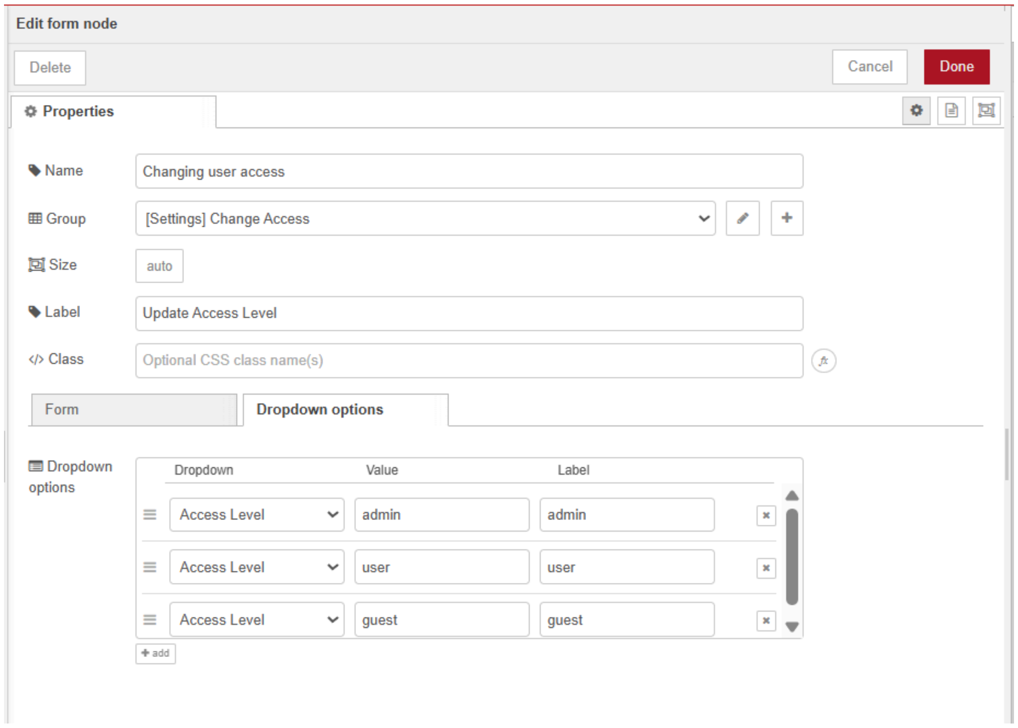
+ add

Buttons: update | clear

☐ Place the form elements in two columns

☑ Reset the form when submitted

Topic: msg. topic



Edit form node

Delete | Cancel | Done

Properties

Name	Changing user access
Group	[Settings] Change Access
Size	auto
Label	Update Access Level
Class	Optional CSS class name(s)

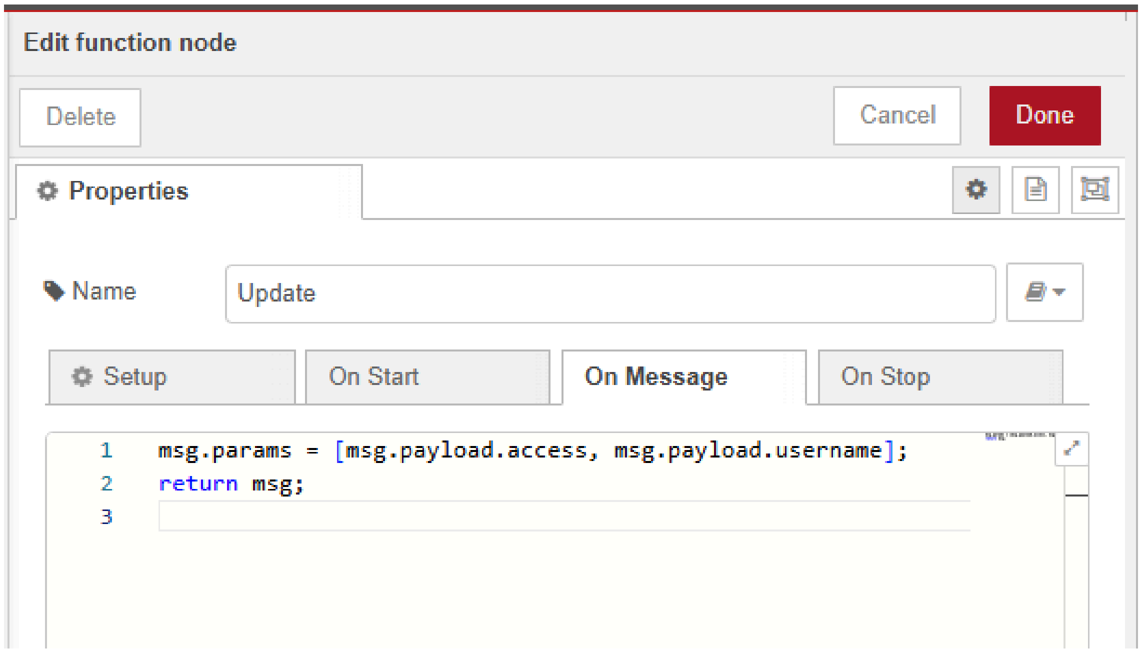
Form | Dropdown options

Dropdown options

Dropdown	Value	Label
Access Level	admin	admin
Access Level	user	user
Access Level	guest	guest

+ add

- Update function :



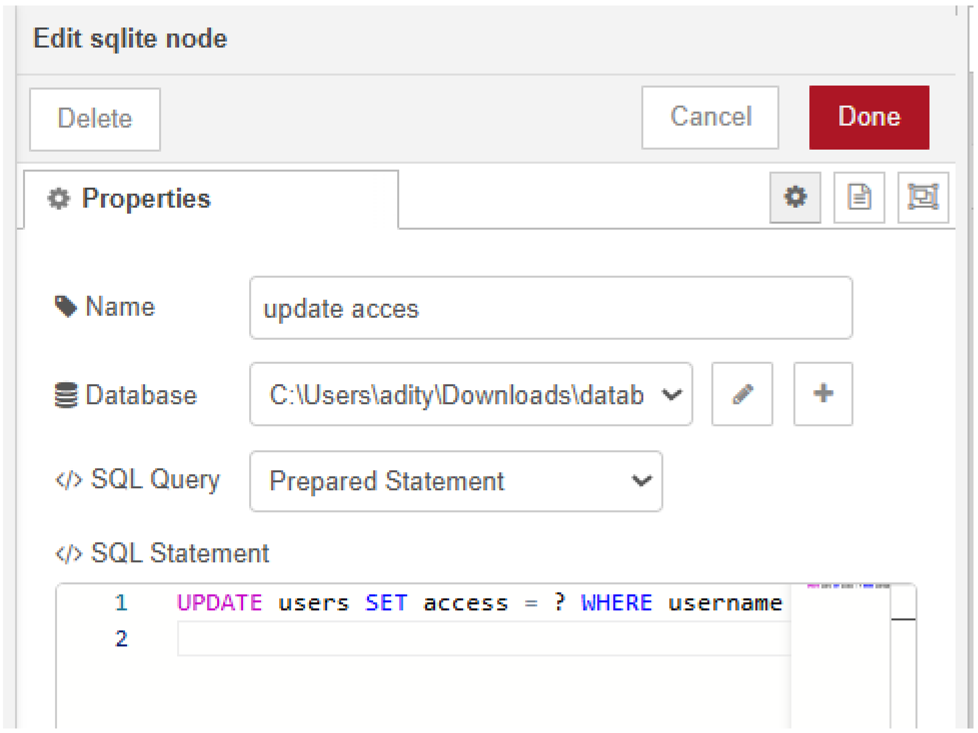
Name: Update

```
msg.params = [msg.payload.access, msg.payload.username];
return msg;
```

- Update function code :

```
msg.params = [msg.payload.access, msg.payload.username];
return msg;
```

- Update access (sqlite) :



Name: update acces

Database: C:\Users\adity\Downloads\datab

SQL Query: Prepared Statement

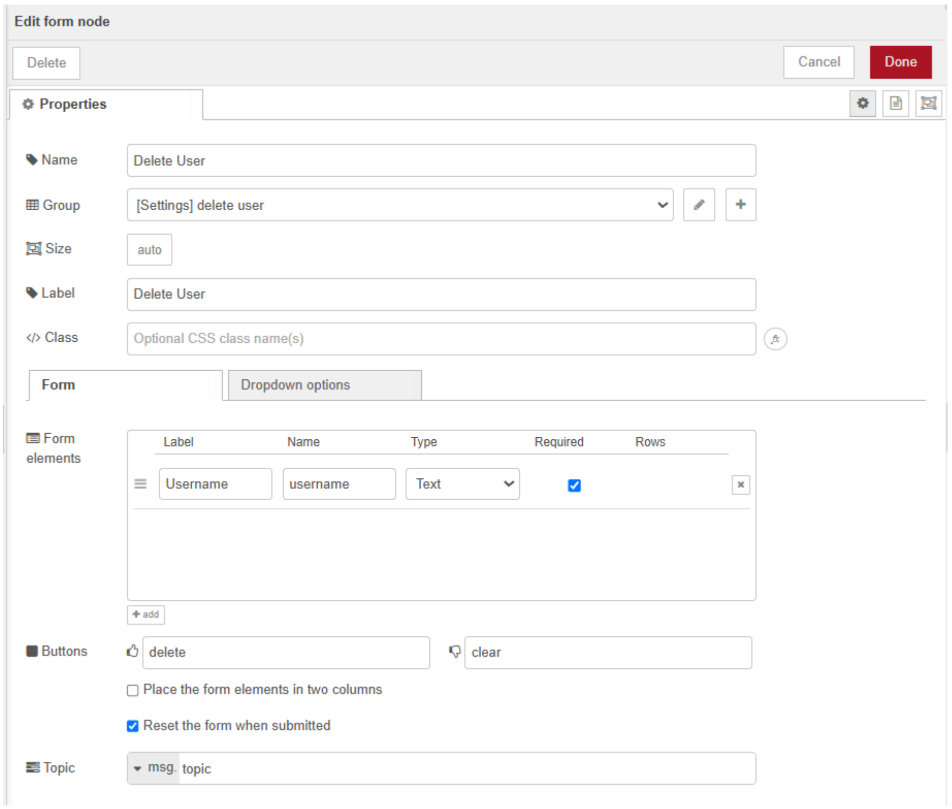
SQL Statement

```
UPDATE users SET access = ? WHERE username
```

- Update access code :

```
UPDATE users SET access = ? WHERE username = ?
```

- Delete user ui_form :



Edit form node

Delete | Cancel | Done

Properties

Name: Delete User

Group: [Settings] delete user

Size: auto

Label: Delete User

Class: Optional CSS class name(s)

Form | Dropdown options

Form elements

Label	Name	Type	Required	Rows
Username	username	Text	✓	

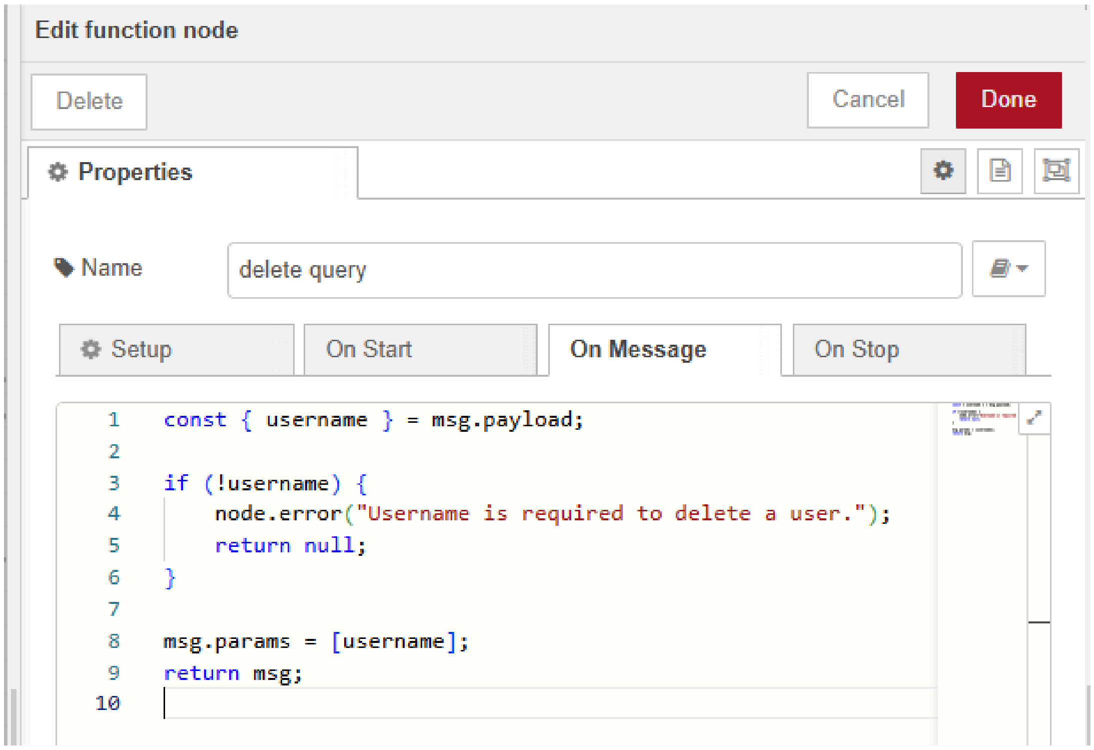
Buttons: delete | clear

☐ Place the form elements in two columns

☑ Reset the form when submitted

Topic: msg. topic

- Delete query function :



Edit function node

Delete | Cancel | Done

Properties

Name: delete query

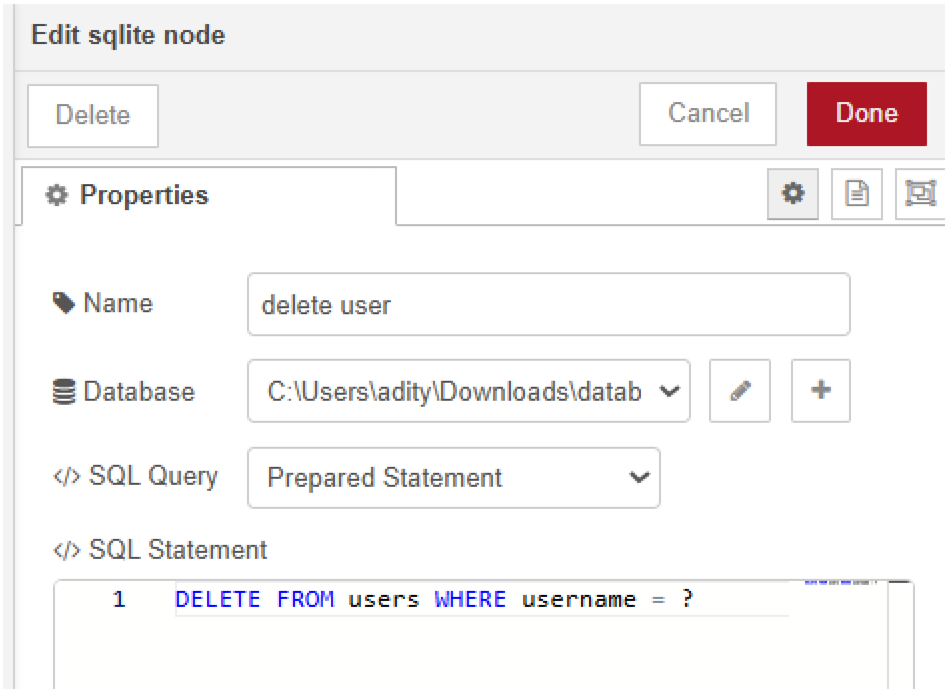
Setup | On Start | On Message | On Stop

```
const { username } = msg.payload;

if (!username) {
    node.error("Username is required to delete a user.");
    return null;
}

msg.params = [username];
return msg;
```

- Delete query function code :

```
const { username } = msg.payload;

if (!username) {
    node.error("Username is required to delete a user.");
    return null;
}

msg.params = [username];
return msg;
```

- Delete user (sqlite) :



Name: delete user

Database: C:\Users\adity\Downloads\datab

SQL Query: Prepared Statement

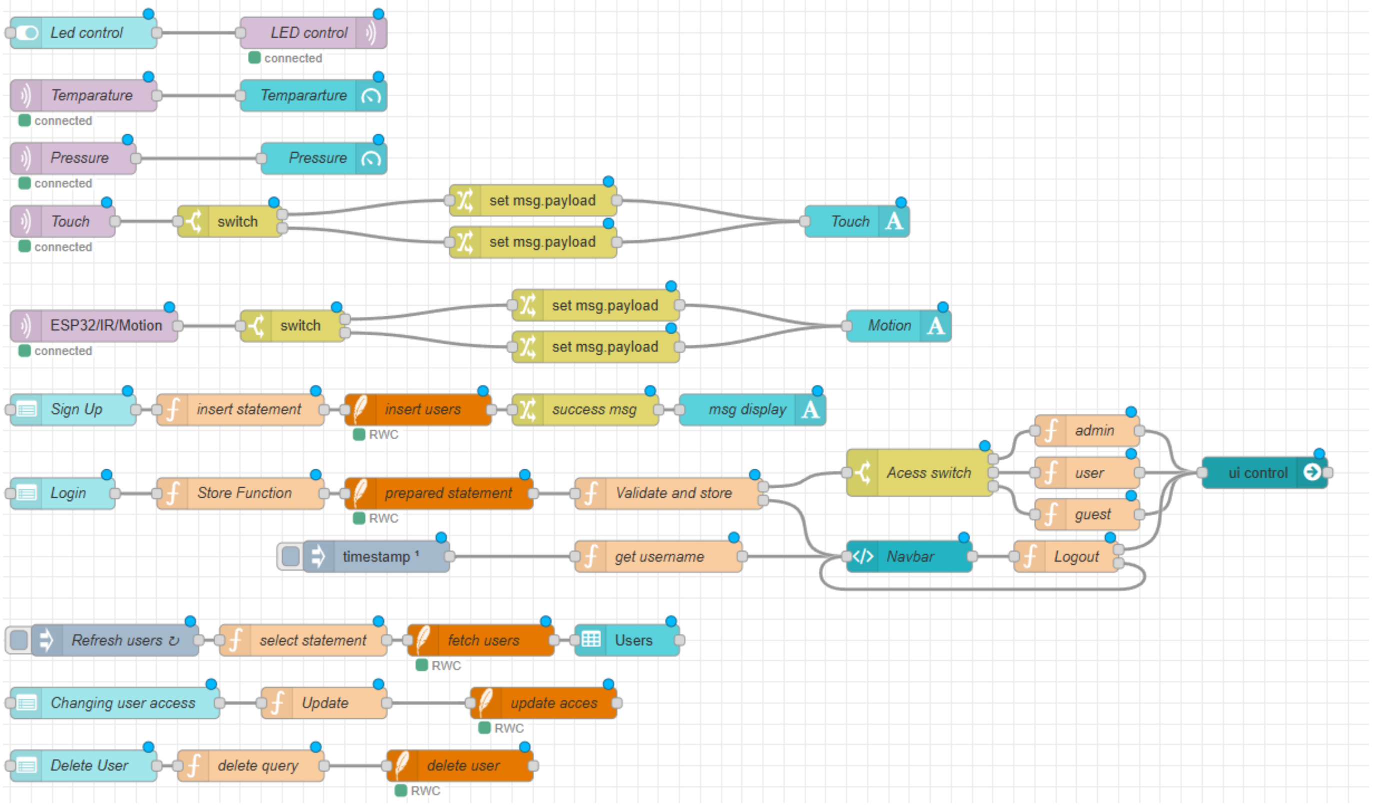
SQL Statement

```
DELETE FROM users WHERE username = ?
```

- Delete user code :

```
DELETE FROM users WHERE username = ?
```

5. Entire flow diagram :



6. Flow code :

Refer to flows-compact.json and flows-formatted.json for the flow code.

7. Flash the Arduino code into the ESP32 :

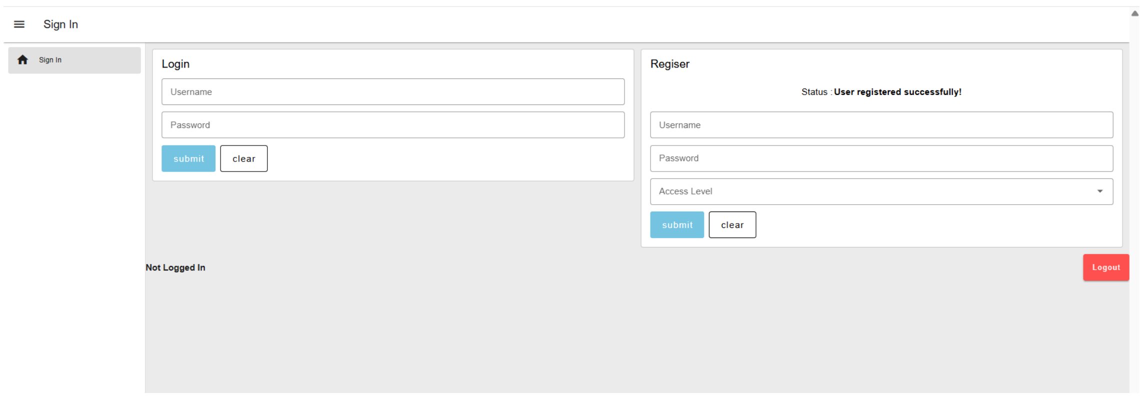
Refer to the esp32-code folder for the arduino code.

8. Deploy and Test :

- Deploy the flow.
- Access the dashboard (http://127.0.0.1:1880/dashboard).

9. OUTPUT Dashboards :

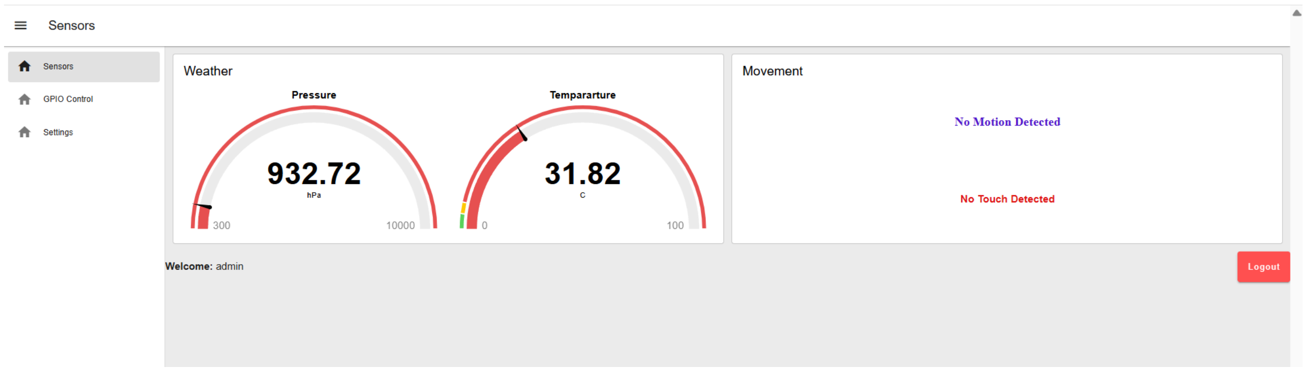
1. Dashboard before logging in:



Status : User registered successfully!

Not Logged In

2. Dashboard with admin login:

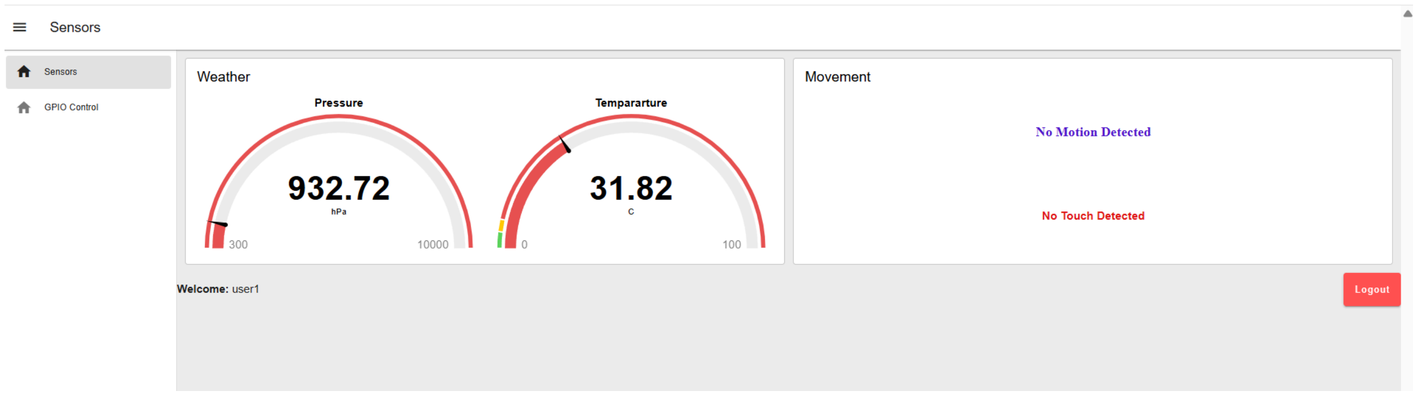


No Motion Detected

No Touch Detected

Welcome: admin

3. Dashboard with user login:

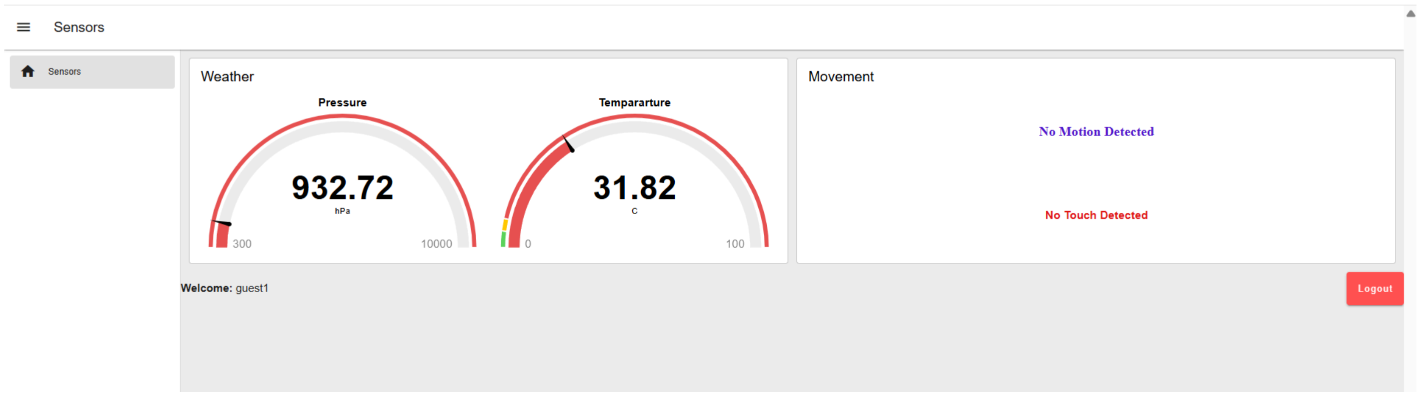


No Motion Detected

No Touch Detected

Welcome: user1

4. Dashboard with guest login:

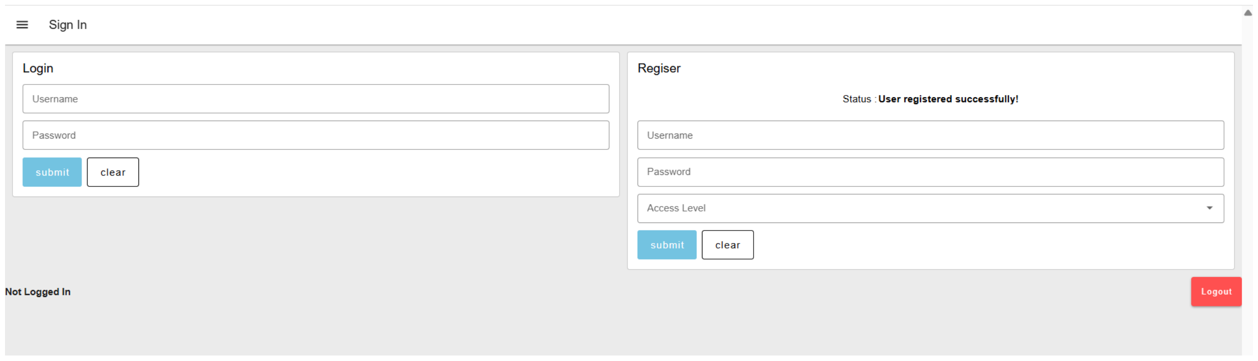


No Motion Detected

No Touch Detected

Welcome: guest1

Sign In



Sign In

Login

Username

Password

submit clear

Regiser

Status : **User registered successfully!**

Username

Password

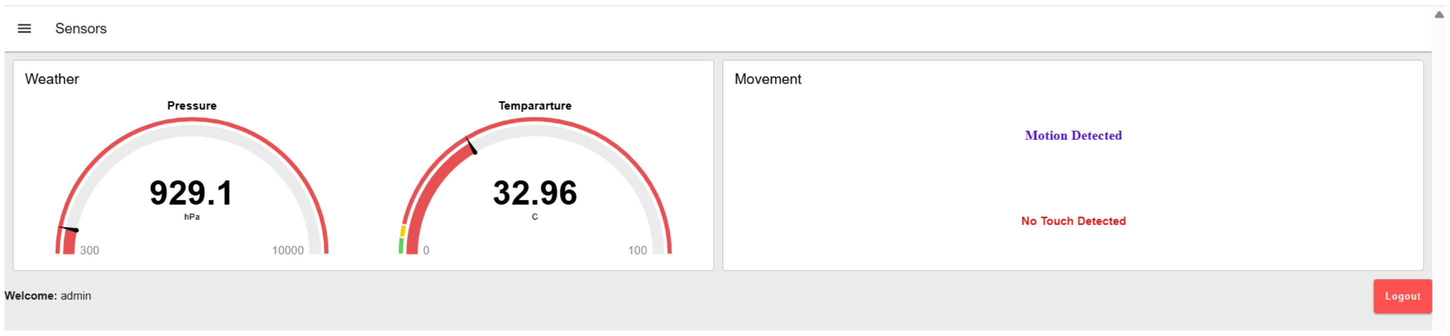
Access Level

submit clear

Not Logged In

Logout

Sensors:



Sensors

Weather

Pressure

929.1 hPa

300 10000

Temparature

32.96 C

0 100

Movement

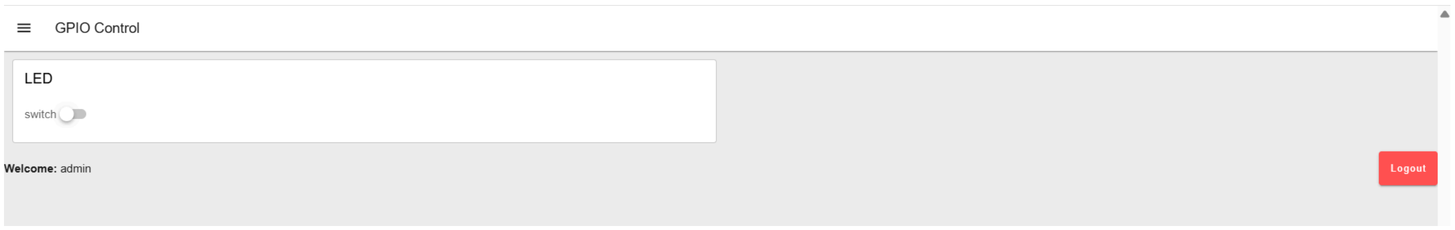
Motion Detected

No Touch Detected

Welcome: admin

Logout

GPIO Control:



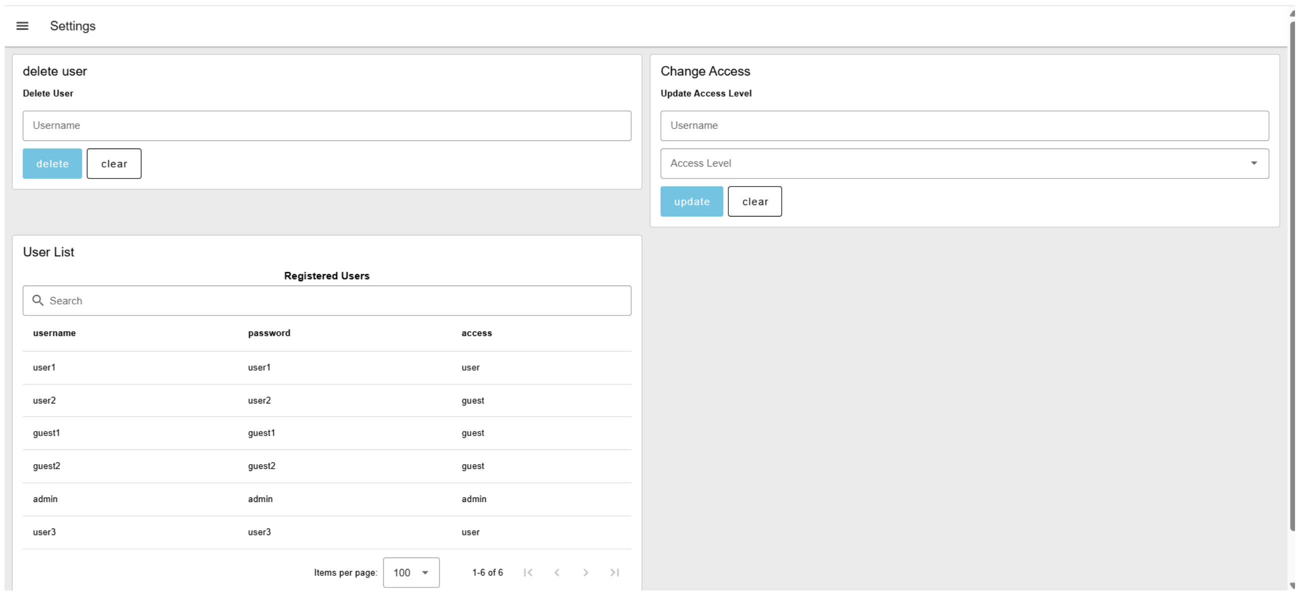
GPIO Control

LED

switch

Welcome: admin

Settings:



Settings

delete user

Delete User

Username

Change Access

Update Access Level

Username

Access Level

User List

Registered Users

Search

username	password	access
user1	user1	user
user2	user2	guest
guest1	guest1	guest
guest2	guest2	guest
admin	admin	admin
user3	user3	user

Items per page: 100 1-6 of 6

Learning Outcomes

1. Node-RED Flow Design

- Learn to visually design and deploy flows using Node-RED's drag-and-drop interface.
- Understand node types, wiring logic, and real-time data handling.

2. Sensor Data Integration

- Gain hands-on experience in connecting sensors (Temperature, Pressure, Touch, IR/Motion) using MQTT and displaying real-time values.

3. Dashboard Development

- Create a user-friendly dashboard with nodes like ui_gauge, ui_text, ui_switch, and ui_table.
- Display sensor data, system status, and user access dynamically using ui_control and ui_template.

4. User Management System

- Implement user registration, login, and access-level control using ui_form, function, sqlite, and ui_control.
- Learn to use SQL operations like insert, update, select, and delete within a Node-RED flow.

5. MQTT Communication

- Understand the working of mqtt in and mqtt out nodes for real-time communication between ESP32 and Node-RED.

6. Access-Based UI Rendering

- Learn to dynamically control what components are visible to different user types (admin, user, guest) using access logic.

7. Security & Validation

- Understand how to store credentials and validate user inputs before granting access or making database changes.

8. Automation & Scheduling

- Use inject nodes for timestamp generation or scheduled tasks.