

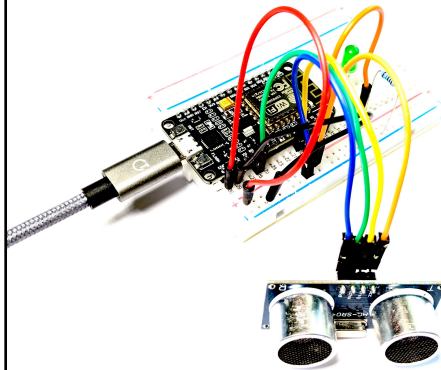
Smart Door

Part four: JSON data

Project description:

In this project, you will work with JSON data . You will build on the previous smart door exercise and use the web server to send JSON requests. JSON is a standard file format that allows you to store information about your circuit components and that you can serve it from your ESP web server for data exchange and retrieval. This is a continuation of the third part of the smart door exercise so please refer back to that part if you get lost.

```
1 // 20210722131648
2 // http://192.168.1.107/json
3
4 {
5   "Content-Type": "application/json",
6   "Status": 200,
7   "Sensor": {
8     "sensorName": "Distance Sensor",
9     "sensorPins": [
10      2,
11      15
12    ],
13     "sensorValue": 168
14   }
15 }
```



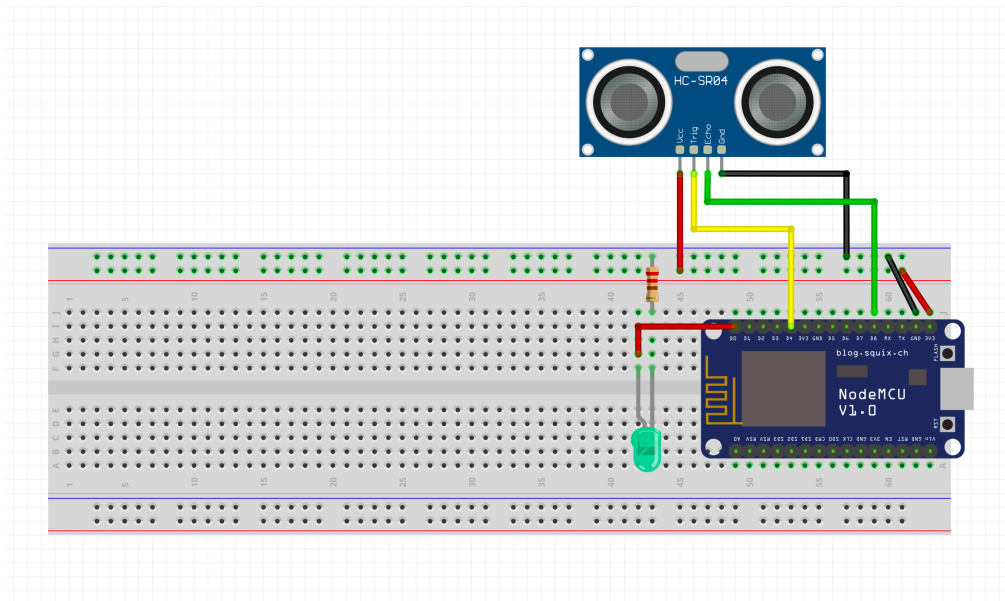
Project objectives:

- Explore the ArduinoJson library
- Create a JSON file and request it from the web server
- Access the JSON file from the web server

Step One

JSON and the circuit

You will work on the latest version of the smart door circuit:



You should have both the code and the circuit for the first few components of the smart door project. The door circuit, in fact, should use the distance sensor as a trigger to control the green LED using PWM. Furthermore, you added a web server on the previous smart door exercise and you should see the following web dashboard when you access the web server:

The Smart Door Dashboard

Welcome to the smart door dashboard

The distance from the door is: 14 cm.

The aim of this exercise is to add one more route to the web server and use it to request JSON data containing information about your circuit components.

JSON is a lightweight data-interchange format. Its popularity is given by the fact that it is easy for humans to read and write but also easy for machines to parse and generate. If you are familiar with JavaScript, JSON is based on a subset of such language. It is essentially a collection of key/value pairs and can be serialised as an object or other data structures in various programming languages.

Here an example of JSON data serialised as a Javascript object:

JSON DATA

```
{
  "name": "Andy",
  "age": 28,
  "hobbies": ["travelling", "sports", "cooking"],
  "bestFriend": {
    "name": "Cris",
    "age": 29
  }
}
```



JAVASCRIPT OBJECT

```
var object1 = {
  name: 'Andy',
  age: 28,
  hobbies: ['travelling', 'sports', 'cooking'],
  bestFriend: {
    name: 'Cris',
    age: 29,
  },
};
```

Step Two

Writing the Code

Let's see now how we can construct a JSON object containing information about the smart door circuit components and request it from the web server. Go ahead and create a new empty sketch from the Arduino IDE.

You should see an empty sketch like the following:



Feel free to save the sketch and rename it to something sensible: **smart_door_part4** for instance.

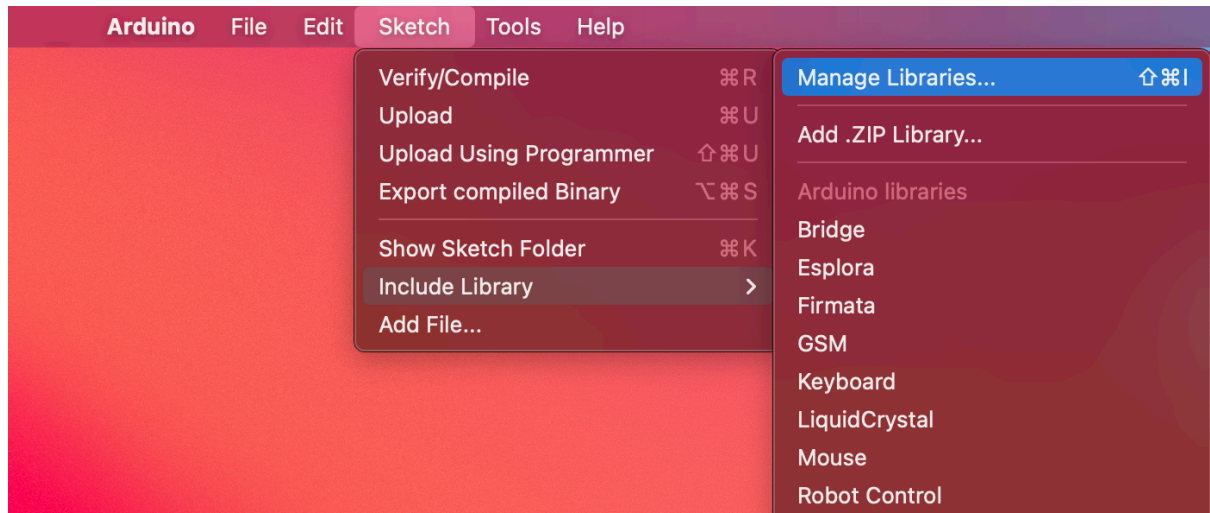
Now copy and paste the code that you wrote for the third part of the smart door exercise. You should have the code with the **ledControl()**, and the **distanceCentimeter()** utility functions. You should also have an additional function **get_index()** and the web server functionality for the smart door dashboard.

There are few functionalities that we want to add to the program here:

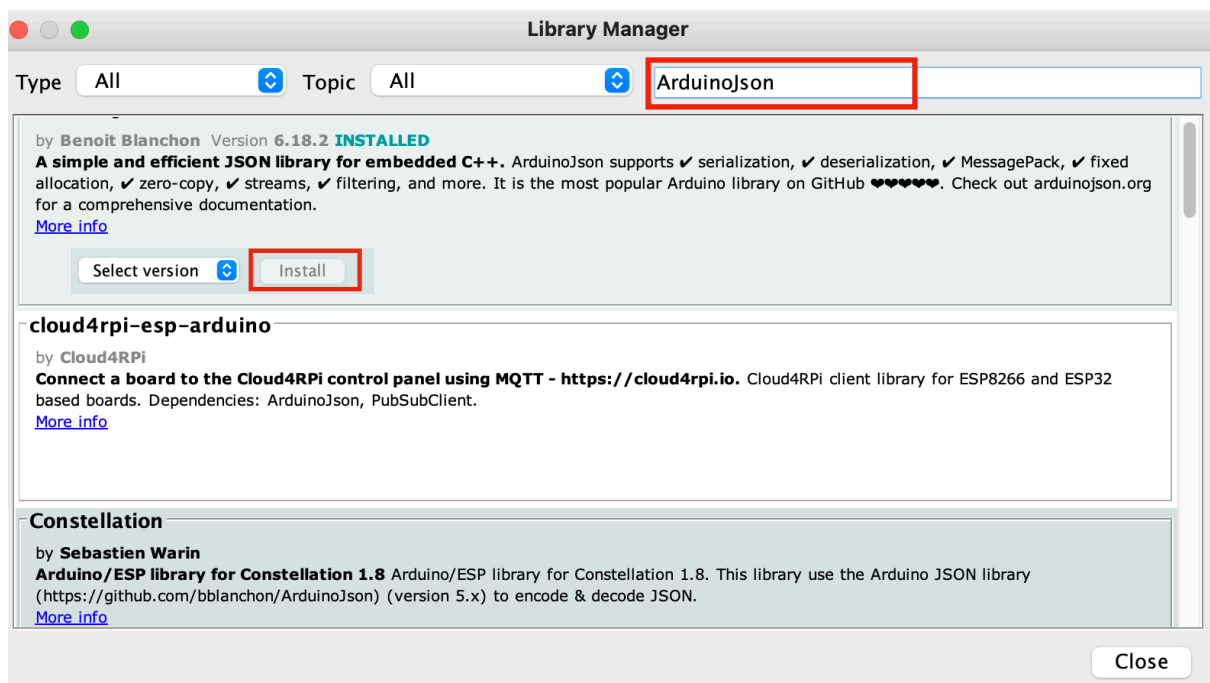
- Create another route on the web server
- Construct a JSON object with information about the http request and the distance sensor component
- Serve and access the JSON data using the web browser

Let's begin by first including the **ArduinoJson** library.

With your Arduino sketch opened, click on **Sketch -> Include Library -> Manage Libraries...** like shown below:



Next, search for “ArduinoJson” and install the latest version of the **ArduinoJson** library as shown below:



Now that the library has been installed, let's include it in the sketch. Add the following line of code just below the WebServer library:

```
#include <ESP8266WiFi.h>
#include <ESP8266WebServer.h>
#include <ArduinoJson.h>
```

Let's now initialise the json document just before the setup function:

```
// Allocate the JSON document
// Allows to allocated memory to the document dinamically.
DynamicJsonDocument doc(1024);
```

Type the above code just before the setup() function.

Here we initialise the json document using the ArduinoJson library. **DynamicJsonDocument** allocate memory to the document dynamically so that you do not need to worry about allocating in manually.

Next we want to add some code to the setup() function:

```
server.on("/", get_index); // Get the index page on root route
server.on("/json", get_json); // Get the json data on the '/json' route
```

Type the above code at the end, but inside, the setup function.

Here we create the web server route to access the JSON data. **get_json** is the callback function that will be executed when you visit the "<your esp board ip>/get_json" URL route on your web browser.

At this point, we have configured the ArduinoJson library and initialised a JSON document. Let's now add some utility functions to construct the JSON document and request it from the web server.

The **jsonDistanceSensor()** function:

```
void jsonDistanceSensor(){
    // Add JSON request data
    doc["Content-Type"] = "application/json";
    doc["Status"] = 200;

    // Add distance sensor JSON object data
    JsonObject distanceSensor = doc.createNestedObject("Sensor");
    distanceSensor["sensorName"] = "Distance Sensor";

    // Add distance sensor JSON data to the object
    JsonArray pins = distanceSensor.createNestedArray("sensorPins");
    pins.add(trigPin);
    pins.add(echoPin);
    distanceSensor["sensorValue"] = distance;
}
```

Type the above code at the end of the sketch, together with the already existing utility functions.

The **jsonDistanceSensor()** utility function creates a JSON document with information about the http request and the distance sensor. You can see how the object is structured with key/value pairs. Here we include the request type and status, the distance sensor name, the distance sensor pins, and the distance sensor value. Notice how the distance sensor data are structured in a nested object and the the sensor pins are stored in an array type since there are two pins associated with the sensor.

Finally, we need to write the code for the **get_json** function to complete the sketch program and access our data from the web server:

```
// Utility function to send JSON data
void get_json(){
    // Create JSON data
    jsonDistanceSensor();

    // Make JSON data ready for the http request
    String jsonStr;
    serializeJsonPretty(doc, jsonStr);

    // Send the JSON data
    server.send(200, "application/json", jsonStr);
}
```

Type the above code at the end of the sketch, together with the already existing utility functions.

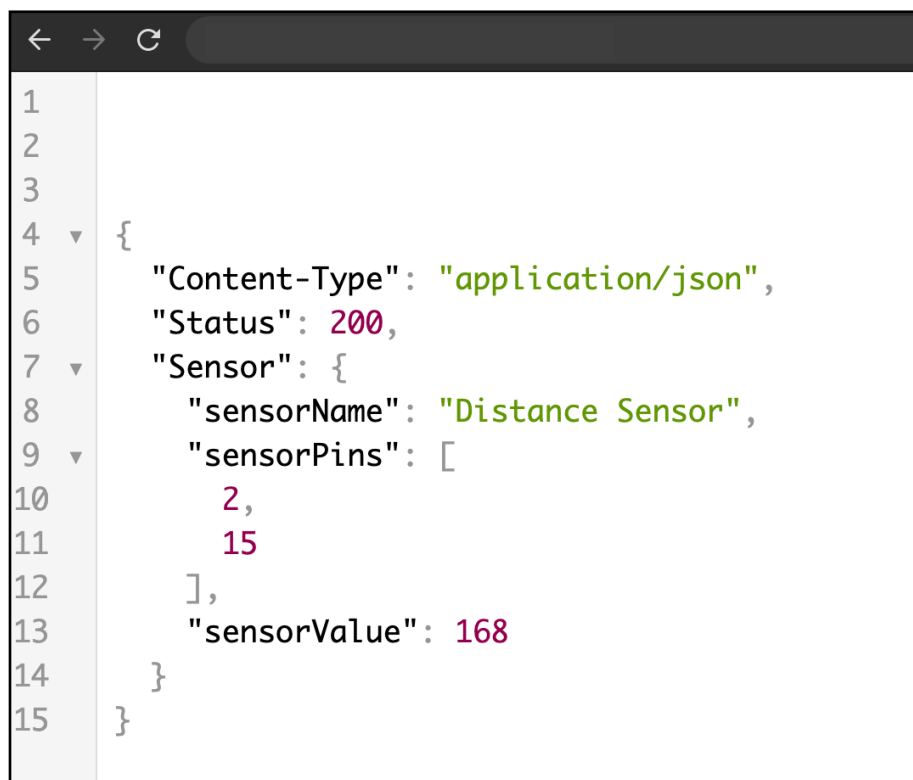
The **get_json()** utility function first calls the **jsonDistanceSensor()** function to construct the JSON object, then converts the object to a valid String format (serializeJsonPretty) so that it is valid for a http request, and finally sends such data to the web server. Notice here that the type of the request is no longer “text/html” but “application/json”.

Hurray, you have successfully completed the code section of the circuit. Go ahead, compile your code, and upload it to your ESP board.

Once the code has been uploaded, find the ip address of your board on the Serial Monitor and access the following route through a web browser:

<your ESP board IP>/get_json

You should see the following data:



The screenshot shows a web browser window with a dark header bar containing back, forward, and refresh icons. The address bar is empty. The main content area displays a JSON object with the following structure:

```
1 {
2
3
4   {
5     "Content-Type": "application/json",
6     "Status": 200,
7     "Sensor": {
8       "sensorName": "Distance Sensor",
9       "sensorPins": [
10        2,
11        15
12      ],
13       "sensorValue": 168
14     }
15   }
```


Step Three

Additional Tasks

In upcoming exercises, you will add more components to the smart door circuit.

Meantime, try the following:

- Can you add different routes to the web server to request different JSON data? What about adding the JSON data for the green led using the “/green_led” web route?
- Revisit both the smart chair and the smart fridge circuits and add JSON requests to the code. You can even add all the components of the circuit into a single JSON request.