

# JSON, AJAX and NodeJS

In my example, we have a JSON object called **myCustomer**:

```
var myCustomer = {  
  name: "Awad",  
  address: "123 easy street",  
  phone: "720-555-5555",  
  email: "test@email.com",  
  request: "be cool"  
}
```

... it is in curly braces.

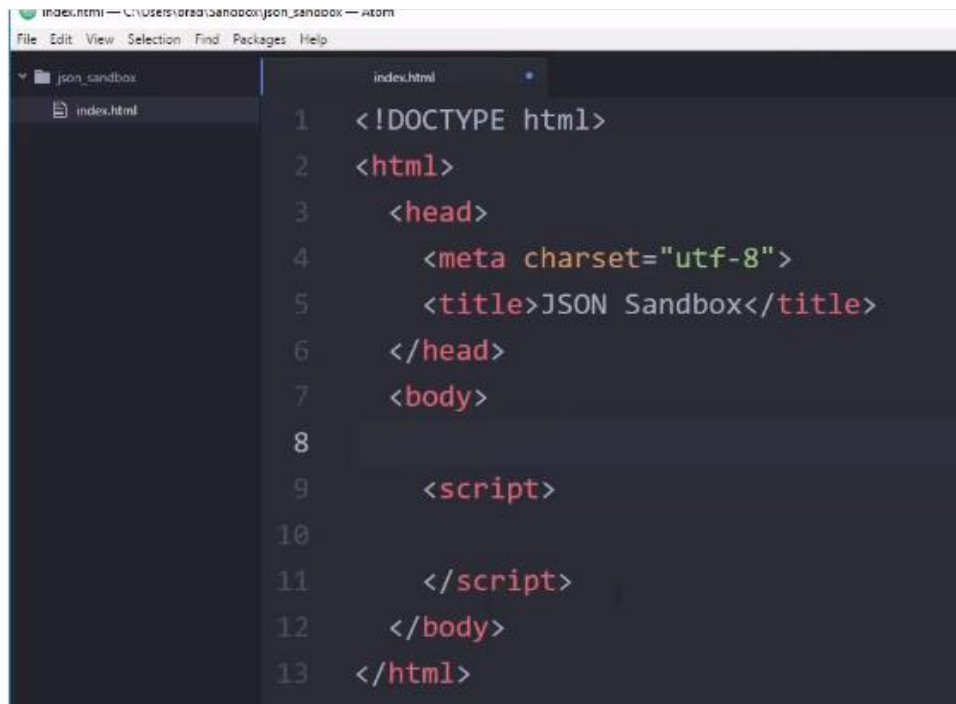
... it uses key-value pairs.

... the values are strings.

... notice the keys don't have double quotes, so it technically isn't valid JSON.

To Start:

We have the index.html inside a folder called json\_sandbox:



The image shows a screenshot of the Atom text editor. The title bar at the top reads "index.html - C:\Users\pradi\Documents\json\_sandbox - Atom". Below the title bar is a menu bar with "File", "Edit", "View", "Selection", "Find", "Packages", and "Help". On the left side, there is a file explorer pane showing a folder named "json\_sandbox" which contains a file named "index.html". The main editor area displays the content of "index.html" with line numbers 1 through 13 on the left. The code is as follows:

```
1 <!DOCTYPE html>
2 <html>
3   <head>
4     <meta charset="utf-8">
5     <title>JSON Sandbox</title>
6   </head>
7   <body>
8
9     <script>
10
11   </script>
12 </body>
13 </html>
```

```

<!DOCTYPE html>
<html>
<head>
  <meta charset = "utf-8">
</head>
<body>

  <h1>Welcome to the ever-growing repository of patrons (God Willing).</h1>

  <p id="demo"></p>

  <button onclick="myFunction()">Refresh</button>

  <script>
    var myCustomer = {
      name: "Awad"
    }

    function myFunction() {
      document.getElementById("demo").innerHTML = myCustomer;
    }
  </script>
</body>
</html>

```

We have a button that, onclick, invokes the function myFunction.

We will create a variable called myCustomer and set it to an object stored in the curly braces. The object is Awad.

The function myFunction uses the getElementById method to display the myCustomer object on the page...

... the element displaying the object myCustomer will be p... how do we know this?... it has the id "demo"

We will see we get object object as key value pairs ... why?... we did not specify which property of myCustomer to access:



... so if we specify the property **name** of the object **myCustomer**:

```
function myFunction() {  
  document.getElementById("demo").innerHTML = myCustomer.name;  
}
```

... this will return us:



Note: for better UI and UX, the button was moved between the header and p element.

Let's prepare this as a valid JSON file and send it via AJAX to a server:

Let's change the myCustomer object to a string:

```
//convert the object myCustomer into a string.  
myCustomer = JSON.stringify(myCustomer);  
  
//display the properties of the object myCustomer in p element.  
function myFunction() {  
  document.getElementById("demo").innerHTML = myCustomer;  
}
```

...which will give us the object with all of its properties as a string:



If we try to add name property to myCustomer for display...

```
//convert the object myCustomer into a string.
myCustomer = JSON.stringify(myCustomer);

//display the properties of the object myCustomer in p element.
function myFunction() {
  document.getElementById("demo").innerHTML = myCustomer.name;
}
```

we will get “undefined,” because it’s no longer an object:



... to overcome this, we may parse it:





Now, let's set street to an object with its own set of properties of street and city:

```
var myCustomer = {  
  name: "Awad",  
  address: {  
    street: "123 main st",  
    city: "Aurora"  
  },  
  phone: "720-555-5555",  
  email: "test@email.com",  
  request: "be cool"  
}
```

... and comment out the stringify and parse methods:

```
//convert the object myCustomer into a string.  
//myCustomer = JSON.stringify(myCustomer);  
  
//parse the string myCustomer to allow access of properties.  
//myCustomer = JSON.parse(myCustomer);
```



If we want to access the street, we would enter **myCustomer.address.street** because it's another level of object within an object:

```
//display the properties of the object myCustomer in p element.  
function myFunction() {  
  document.getElementById("demo").innerHTML = myCustomer.address.street;  
}
```

← → ↻ ⓘ File | C:/Users/Omer/Desktop/json/index.html

**Welcome to the ever-growing repository of patrons (God Willing).**

Refresh

123 main st

Now let's create an array of payment methods called payment:

```
var myCustomer = {  
  name: "Awad",  
  address: {  
    street: "123 main st",  
    city: "Aurora"  
  },  
  phone: "720-555-5555",  
  email: "test@email.com",  
  request: "be cool",  
  payment: ["card", "cash"]  
}
```

... the array contains strings called card and cash.

Let's say we want to access card in the payment method. We would specify **myCustomer.payment** with an index of zero:

```
//display the properties of the object myCustomer in p element.  
function myFunction() {  
  document.getElementById("demo").innerHTML = myCustomer.payment[0];  
}
```

← → ↺ ⓘ File | C:/Users/Omer/Desktop/json/index.html

**Welcome to the ever-growing repository of patrons (God Willing).**

Refresh

card

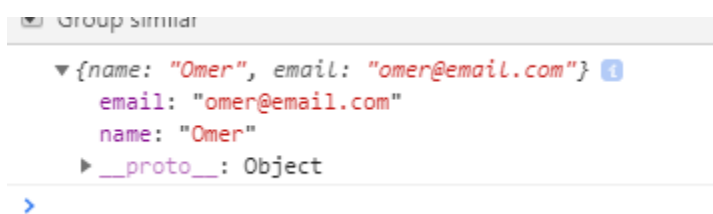
If we were to create an array of objects called customers like so...

```
var customers = [  
  {  
    name: "Omer",  
    email: "omer@email.com"  
  },  
  {  
    name: "Musab",  
    email: "musab@email.com"  
  }  
];
```

... and if we wish to access Omer, we can specify index zero with stringify and parse...

```
console.log(customers[0]);  
  
//convert the object myCustomer into a string.  
customers = JSON.stringify(customers);  
  
//parse the string myCustomer to allow access of properties.  
customers = JSON.parse(customers);
```

... but it is only visible in the console:



```
> Group similar  
▼ {name: "Omer", email: "omer@email.com"} ⓘ  
  email: "omer@email.com"  
  name: "Omer"  
  ▶ __proto__: Object  
>
```

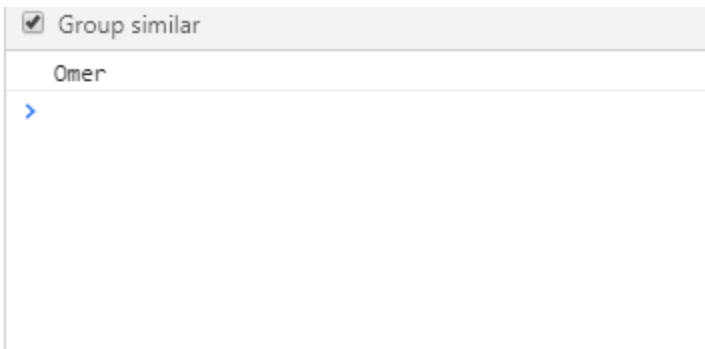
If we wish to access Omer's name...

```
console.log(customers[0].name);

//convert the object myCustomer into a string.
customers = JSON.stringify(customers);

//parse the string myCustomer to allow access of properties.
customers = JSON.parse(customers);
```

... this would display in the console:



Let's comment out the console display of customers:

```
//console.log(customers[0].name);
```

And update the p element from "demo" to "customers":

```
<!--display of myCustomer properties-->  
<p id="customers"></p>
```

... and we create a for loop of customer names:

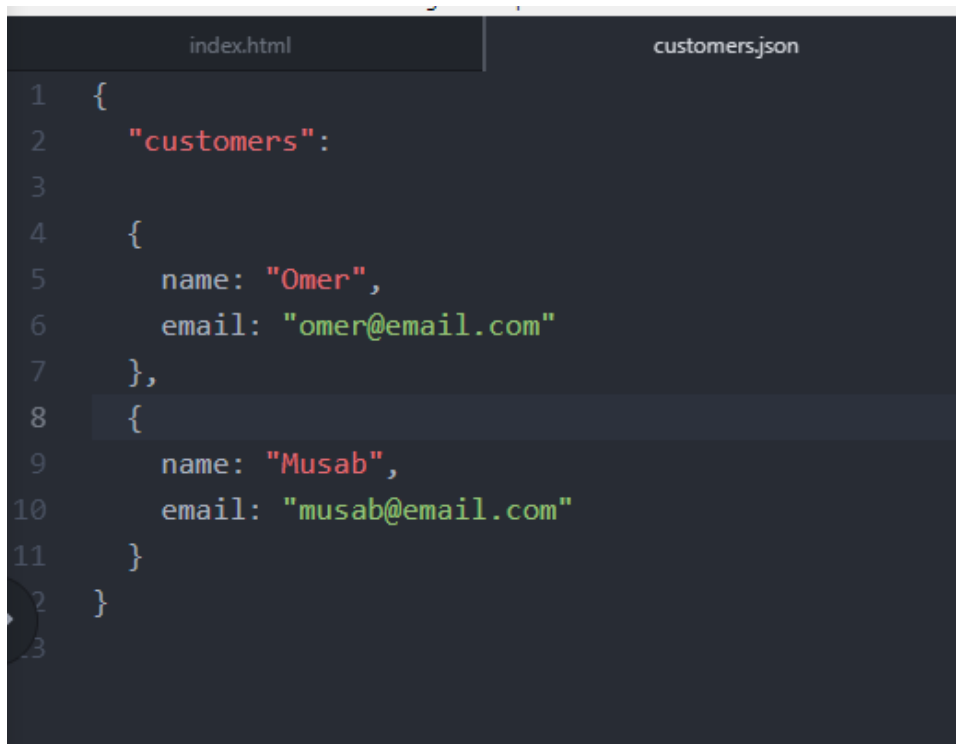
```
var output = '';  
for(var i = 0; i < customers.length; i++){  
    output += '<li>'+customers[i].name+'</li>';  
}  
document.getElementById('customers').innerHTML = output;
```

... this will be displayed on our page:



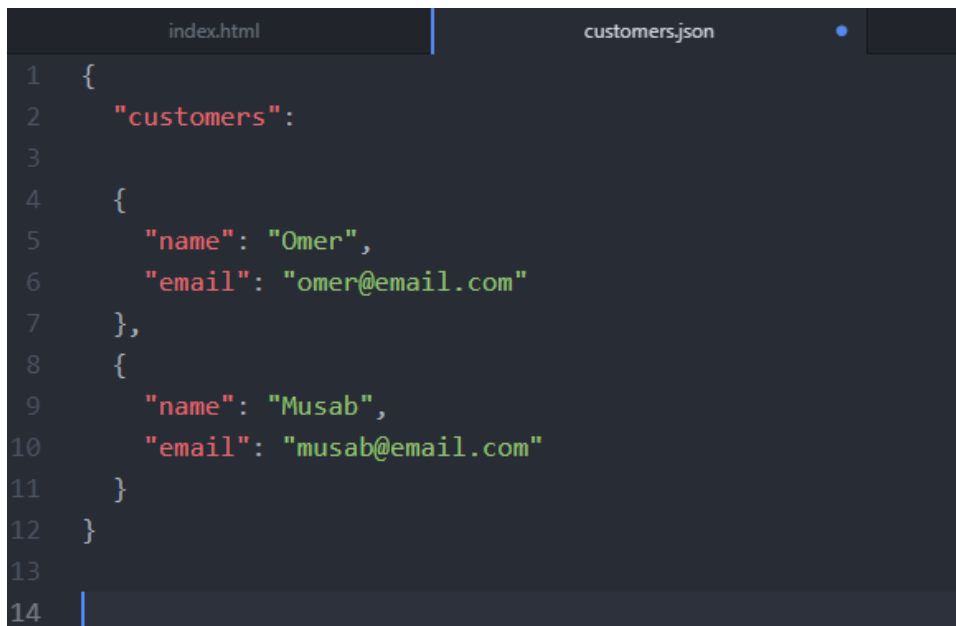
Note: the Refresh button in the image is not utilized, since it invokes function myFunction from earlier.

Now let's create a JSON file, in which we will store the customers array:



```
1  {
2    "customers":
3
4    {
5      name: "Omer",
6      email: "omer@email.com"
7    },
8    {
9      name: "Musab",
10     email: "musab@email.com"
11   }
12 }
13
```

... however, this is not valid JSON, because the keys are missing double quotes:



```
1  {
2    "customers":
3
4    {
5      "name": "Omer",
6      "email": "omer@email.com"
7    },
8    {
9      "name": "Musab",
10     "email": "musab@email.com"
11   }
12 }
13
14
```

... now it's valid JSON.

Now let's comment everything out in the script opening and closing tags.

Next, let's add an XML HTTP Request:

Next, let's set up our Node.js server:

After installing Node.js, open the terminal and enter the following commands to access the directory of the index.html file:

**cd Desktop**

**cd json**

**npm install -g install live-server**

```
Command Prompt
Microsoft Windows [Version 10.0.18362.778]
(c) 2019 Microsoft Corporation. All rights reserved.

C:\Users\Omer>cd Desktop

C:\Users\Omer\Desktop>cd json

C:\Users\Omer\Desktop\json>npm install -g live-server
npm WARN deprecated chokidar@2.1.8: Chokidar 2 will break on node v14+. Upgrade to chokidar 3 with
npm WARN deprecated opn@6.0.0: The package has been renamed to `open`
npm WARN deprecated fsevents@1.2.13: fsevents 1 will break on node v14+ and could be using insecure
C:\Users\Omer\AppData\Roaming\npm\live-server -> C:\Users\Omer\AppData\Roaming\npm\node_modules\liv
npm WARN optional SKIPPING OPTIONAL DEPENDENCY: fsevents@^1.2.7 (node_modules\live-server\node_modu
npm WARN notsup SKIPPING OPTIONAL DEPENDENCY: Unsupported platform for fsevents@1.2.13: wanted {"os
+ live-server@1.2.1
added 194 packages from 149 contributors in 61.302s

C:\Users\Omer\Desktop\json>
```

... this will install the live server module.

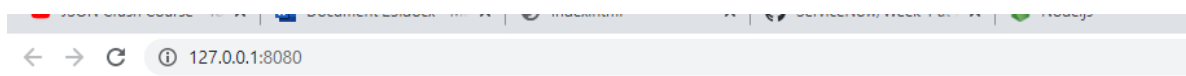
... this live server module is installed globally, so we may access it from anywhere.

Next, we run the index.html on our server by entering the following command:

**live-server**

```
C:\Users\Omer\Desktop\json>live-server
Serving "C:\Users\Omer\Desktop\json" at http://127.0.0.1:8080
Ready for changes
GET /favicon.ico 404 13.124 ms - 150
```

... so our application has opened up on port 8080:



**Welcome to the ever-growing repository of patrons (God Willing).**