# Data visualization

## Handle Click Events with JavaScript using the onclick property

An **onclick** event handler triggers when a user clicks on an element with the ID that you assigned. You get that using the following code. An addEventListener() is a method that attaches an event to an element. It is possible to add multiple events to an element. You can add this method to HTML elements and windows.

The **DOMContentLoaded** does not wait for stylesheets to load.

**Example**

document.addEventListener('DOMContentLoaded', function() {

document.getElementById('getMessage').onclick = function() {

document.getElementsByClassName('message')[0].textContent="Here is the message";

}

});

You can change or update an HTML element using JavaScript. In the code below, the text element is changed when the user clicks on ‘getMessage’. It adds content to the class ‘message’.

## Get JSON with the JavaScript XMLHttpRequest Method

You can request data from external sources. API’s are tools that computers use to communicate with one another and most web API’s transfer data in a format called **JSON.** It looks like Javascript. JSON has object properties and their current values, sandwiched between a { and a }. These properties and their values are often referred to as "**key-value pairs**".

**Example**

document.addEventListener('DOMContentLoaded', function() {

document.getElementById('getMessage').onclick = function() {

const req = new XMLHttpRequest();

req.open("GET",'/json/cats.json',true);

req.send();

req.onload = function(){

const json = JSON.parse(req.responseText);

document.getElementsByClassName('message')[0].innerHTML = JSON.stringify(json);

};

}

});

1. First, an instance of the **XMLHttpRequest** object is created and saved in the **req** variable. It creates a new XMLHttpRequest.

const req = new XMLHttpRequest();

1. The open method then initializes a request, it’s requesting data from an API, see ‘**GET’** request. The second argument is the URL of the API you’re trying to get data from and **true** makes it an asynchronous request, which means the client continues execution after initiating the request and processes the result whenever the AppServer makes it available.

req.open("GET",'/json/cats.json',true);

1. It then sends the request using req.send();

req.send();

1. The **onload** event parses\* the returned data and applies the JSON.stringify method to convert the object into a string and is inserted as the text message.   
    const json = JSON.parse(req.responseText);

document.getElementsByClassName('message')[0].innerHTML = JSON.stringify(json);

};

*To parse:* To parse is to break up a sentence or group of words into separate components, including the definition of each part's function or form.

## Get JSON with the JavaScript fetch method

fetch('/json/cats.json')

.then(response => response.json())

.then(data => {

document.getElementById('message').innerHTML = JSON.stringify(data);

})

This initially does the same as the first method, however now the **fetch()** creates the GET request. If the requests succeeded, the **then** method is executed. The **then** method converts the response to JSON. It is handled by the next **then** method is going to select the element that will receive the data and convert the string.

## Access the JSON Data from an API

Understanding structure is important. It influences how you retrieve the values that you need.

[ ] -> Square brackets represent an array.  
{ } -> Curly brackets represent an object.  
" " -> Double quotes represent a string. They are also used for key names in JSON.

**Key-value-pairs** are separated by comma’s. To select the right values, you need to use the right dot notation and brackets in order to display the right thing.

[{"id":0,"imageLink":"https://s3.amazonaws.com/freecodecamp/funny-cat.jpg","altText":"A white cat wearing a green, helmet shaped melon on its head. ","codeNames":["Juggernaut","Mrs. Wallace","Buttercup"]},{"id":1,"imageLink":"https://s3.amazonaws.com/freecodecamp/grumpy-cat.jpg","altText":"A white cat with blue eyes, looking very grumpy. ","codeNames":["Oscar","Scrooge","Tyrion"]},{"id":2,"imageLink":"https://s3.amazonaws.com/freecodecamp/mischievous-cat.jpg","altText":"A ginger cat with one eye closed and mouth in a grin-like expression. Looking very mischievous. ","codeNames":["The Doctor","Loki","Joker"]}]

console.log(json[0].altText); = console.log(json[2].codeNames[1]); displaying Loki.

It selects the second id [2] and second [1] name in the string.

## Convert JSON Data to HTML

  document.addEventListener('DOMContentLoaded', function(){

    document.getElementById('getMessage').onclick = function(){

      const req = new XMLHttpRequest();

      req.open("GET",'/json/cats.json',true);

      req.send();

      req.onload = function(){

        const json = JSON.parse(req.responseText);

          let html = "";

          json.forEach(function(val) {

            const keys = Object.keys(val);

            html += "<div class = 'cat'>";

            keys.forEach(function(key) {

              html += "<strong>" + key + "</strong>: " + val[key] + "<br>";

            });

            html += "</div><br>";

          });

        document.getElementsByClassName('message')[0].innerHTML = html;

      };

    };

  });

When you're looping through objects, you can use this imageLink property to display this image in an img element.

html += "<img src = '" + val.imageLink + "' " + "alt='" + val.altText + "'>";

## Pre-filter data

If you don’t want to render everything, you should add a filter to your code. The **val.id !== 1** removes the id value of 1.

<script>

  document.addEventListener('DOMContentLoaded', function(){

    document.getElementById('getMessage').onclick = function(){

      const req = new XMLHttpRequest();

      req.open("GET",'/json/cats.json', true);

      req.send();

      req.onload=function(){

        let json = JSON.parse(req.responseText);

        let html = "";

        // Add your code below this line

          json = json.filter(function(val) {

          return (val.id !== 1);

        });

        // Add your code above this line

         json.forEach(function(val) {

           html += "<div class = 'cat'>"

           html += "<img src = '" + val.imageLink + "' " + "alt='" + val.altText + "'>"

           html += "</div>"

         });

         document.getElementsByClassName('message')[0].innerHTML = html;

       };

     };

  });

</script>

## Check Geolocation Data

if (navigator.geolocation){

navigator.geolocation.getCurrentPosition(function(position) {

document.getElementById('data').innerHTML="latitude: " + position.coords.latitude + "<br>longitude: " + position.coords.longitude;

});

}

First, it checks if the navigator.geolocation object exists. If it does, the getCurrentPosition method on that object is called, which initiates an asynchronous request for the user's position. If the request is successful, the callback function in the method runs. This function accesses the position object's values for latitude and longitude using dot notation and updates the HTML.

## Post Data with the JavaScript XMLHttpRequest Method

<script>

  document.addEventListener('DOMContentLoaded', function(){

    document.getElementById('sendMessage').onclick = function(){

      const userName = document.getElementById('name').value;

      const url = 'https://jsonplaceholder.typicode.com/posts';

      // Add your code below this line

const xhr = new XMLHttpRequest();

xhr.open('POST', url, true);

xhr.setRequestHeader('Content-Type', 'application/json; charset=UTF-8');

xhr.onreadystatechange = function () {

  if (xhr.readyState === 4 && xhr.status === 201){

    const serverResponse = JSON.parse(xhr.response);

    document.getElementsByClassName('message')[0].textContent = serverResponse.userName + serverResponse.suffix;

  }

};

const body = JSON.stringify({ userName: userName, suffix: ' loves cats!' });

xhr.send(body);

      // Add your code above this line

    };

  });

</script>

You've seen several of these methods before. Here the open method initializes the request as a POST to the given URL of the external resource, and uses the true Boolean to make it asynchronous. The setRequestHeader method sets the value of an HTTP request header, which contains information about the sender and the request. It must be called after the open method, but before the send method. The two parameters are the name of the header and the value to set as the body of that header. Next, the onreadystatechange event listener handles a change in the state of the request. A readyState of 4 means the operation is complete, and a status of 201 means it was a successful request. The document's HTML can be updated. Finally, the send method sends the request with the body value, which the userName key was given by the user in the input field.