To call an API from JS, there are 4 ways to do that

* XML HTTP Request
* Fetch API
* AXIO
* JQuery

XML HTTP Request is the most old one, that has been replaced by Fetch APi. But we gonna learn how to do XML http request cause AXIOS is built on this. It has been built in in browser as an object. We can console.log(window) and can see that.

So to hTTprquest could be GET, PUT, DELETE, PUT, PATCH.

const getData =()=> {

  const xhr = **new** *XMLHttpRequest*();

  console.log(xhr);  //we can see the properties of this object.

}

getData();

this is just simple calling the XMLHttpRequest() and saving it in a variable and we can log out the properties of this object.

Now, we gonna use Open method/function to start requesting or initializing and put the method we want to work and give URL, then,

We have to use Send method/function to send the request.

const getData =()=> {

  const xhr = **new** *XMLHttpRequest*();

  xhr.open('GET','https://jsonplaceholder.typicode.com/posts')

  xhr.send()

}

getData();

now if we got to browser and see on devtool and go to network on fetch tab we can see there is a status of 200 and we can see response and request header with info. But not the DATA that we have fetched. To see the data, we have to go response tab. But not in console.

so to bring it on console we have to use an EVENT. Which called onLoad event. This event will print out the data. But we gonna save the data in a variable call ‘data’. And the data we will have from the response property.

const getData =()=> {

  const xhr = **new** *XMLHttpRequest*();

  xhr.open('GET','https://jsonplaceholder.typicode.com/posts')

  xhr.send()

  xhr.onload=()=>{

    let data= xhr.response;

    console.log(data);

  }

}

getData();

so now we can see the data’s on the console. But this are all in json format. So to use in JS we have to convert it to js object.

  xhr.onload=()=>{

    let data= xhr.response;

    console.log(JSON.parse(data));

  }

Simplye this above code change the json object into js object

Now lets have the error event. Since above url working find it does not show any error. but suppose this will not work as fine as it shows then onError event will work. We can check it but changing url it a falty character.

const getData =()=> {

  const xhr = **new** *XMLHttpRequest*();

  xhr.open('GET','https://jsonplaceholder.typicode.com/posts')

  xhr.send()

  xhr.onload=()=>{

    let data= xhr.response;

    console.log(JSON.parse(data));

  }

  xhr.onerror=()=>{

    console.log('error is here buddy !')

  }

}

getData();

now if we can see that we are initializing and using the url and getting the data. But if we want to use it we have to do these above process every time. So to avoid that we can just save it in a memory and use it by passing method of our demand request and url

const makeRequest =(*method*, *url*)=>{

    const xhr = **new** *XMLHttpRequest*();

  xhr.open(*method*,*url*)

  xhr.send()

  xhr.onload=()=>{

    let data= xhr.response;

    console.log(JSON.parse(data));

  }

  xhr.onerror=()=>{

    console.log('error is here buddy !')

  }

}

const getData =()=> {

  makeRequest('GET','https://jsonplaceholder.typicode.com/posts')

}

getData();

so we have created a separate function to create request by the name of makeRequest and we are passing method type and url link as a parameter in that function from a different function. So that we can use it multiple times easily.

We have learned about **GET** data, now lets learn about **POST** data

As when we want to use the data we have to convert it to JS object from JSON

But to post it. We have to change it to JSON from JS. Simple

Now lets see the code.

const makeRequest =(*method*, *url*, *data* )=>{

  const xhr = **new** *XMLHttpRequest*();

  xhr.open(*method*,*url*)

  xhr.setRequestHeader('Content-type','application/json')

  xhr.onload=()=>{

    let data= xhr.response;

    console.log(JSON.parse(data));

  }

  xhr.onerror=()=>{

    console.log('error is here buddy !')

  }

  xhr.send(JSON.stringify(*data*));

}

const getData =()=> {

  makeRequest('GET','https://jsonplaceholder.typicode.com/posts')

}

const sendData =()=> {

  makeRequest('POST','https://jsonplaceholder.typicode.com/posts',{

    title: 'foo',

    body: 'bar',

    userId: 1,

  })

}

sendData();

from above code we can see some things.

1. We have created another function by the name sendData where we calling the makeRequest function and passing the method type which is POST and the url. But this time we are sending data so we have to give the data. So in the same line, separating with coma, in a curly brace we are giving data.
2. In the makeRequest function we can see that there is another new parameter which is by the name data, and sending the the data to the server url by using built in function xhr.send() and ofcourse changing it to JSON object.
3. And there is another built in method we have use to let the api know which type of content we are giving, so we are using setRequestHeader built in method to give the content type.
4. REMEMBER one thing, this setRequestHeader method has to be in in the middle of OPEN and SEND method. ☺

We have learn **GET and POST** method now learn how to update by using **PUT** method.

We can just add the function just below the above total code.

const updateData =()=> {

  makeRequest('PUT','https://jsonplaceholder.typicode.com/posts/1',{

    id: 1,

    title: 'fooMango',

    body: 'barMango',

    userId: 1,

  })

}

updateData();

we can see that we are updating the whole body. And if we see that url we can see that there is 1 by slashing at the very end.

Now we have learn how to update the whole body of that particular ID. But what about we want to just update a single data. We can use PATCH method

So lets have a function

const updateSingleData =()=> {

  makeRequest('PATCH','https://jsonplaceholder.typicode.com/posts/1',{

    title: 'Changing the Title by PUT method',

  })

}

We are giving the url along with its id we want to update by patching. And just changing the title.

We have learn about **GET, POST, PUT, PATCH** and now we will learn **DELETE.**

Remember delete method does not send any data. Just give the method type and url along with the id we want to delete.

const deleteData =()=> {

  makeRequest('DELETE','https://jsonplaceholder.typicode.com/posts/1')

}

deleteData();

now if we see the total code.

const makeRequest =(*method*, *url*,*data* )=>{

  const xhr = **new** *XMLHttpRequest*();

  xhr.open(*method*,*url*)

  xhr.setRequestHeader('Content-type','application/json');

  xhr.onload=()=>{

    let data= xhr.response;

    console.log(JSON.parse(data));

  }

  xhr.onerror=()=>{

    console.log('error is here buddy !')

  }

  xhr.send(JSON.stringify(*data*));

}

const getData =()=> {

  makeRequest('GET','https://jsonplaceholder.typicode.com/posts')

}

const sendData =()=> {

  makeRequest('POST','https://jsonplaceholder.typicode.com/posts',{

    title: 'foo',

    body: 'bar',

    userId: 1,

  })

}

const updateData =()=> {

  makeRequest('PUT','https://jsonplaceholder.typicode.com/posts/1',{

    id: 1,

    title: 'fooMango',

    body: 'barMango',

    userId: 1,

  })

}

const updateSingleData =()=> {

  makeRequest('PATCH','https://jsonplaceholder.typicode.com/posts/1',{

    title: 'Changing the Title by PUT method',

  })

}

const deleteData =()=> {

  makeRequest('DELETE','https://jsonplaceholder.typicode.com/posts/1')

}

deleteData();

NOW,

We could use promise to do the stuff we did. How ? If we call the makeReuest function and if we want to return a promise then we have return a promise which takes two parameter resolve and reject and then in the function, just the thing we did withtout promise we can cut paste there.

const makeRequest =(*method*, *url*,*data* )=>{

  return **new** *Promise*((*resolve*,*reject*)=>{

  const xhr = **new** *XMLHttpRequest*();

  xhr.open(*method*,*url*)

  xhr.setRequestHeader('Content-type','application/json');

  xhr.onload=()=>{

    let data= xhr.response;

    console.log(JSON.parse(data));

  }

  xhr.onerror=()=>{

    console.log('error is here buddy !')

  }

  xhr.send(JSON.stringify(*data*));

  })

}

So now if we call the makeRequest we can recive what its sending by promise by using then method.

const getData =()=> {

  makeRequest('GET','https://jsonplaceholder.typicode.com/posts')

  .then((*resbal*)=>{

    console.log(*resbal*);

  })

}

getData();

simple.

So one thing we can see that, XMLHTTPRequest we have to make promise and then work. But in fetch or Axios is already prmisify. Which will return promise. So in fetch and axios we don’t need to code the whole thing inside the promise function as we did above. Like open /send method.

We can just directly can access using .then

**THAT ALL ABOUT XMLHTTPREQUEST**

NOW lets learn about Fetch API.

//fetch() has replaced XMLHTTPRequest

//fetch() global method for making http request

// 2 ways to call = then and Async/Await

//fetch() is easy to use compare to XMLHTTR and return  a promise.

//return promise can only handle network error

//does not support all the old BORWSER.

If we console log (window) we can see that there is a property call fetch.

fetch('https://jsonplaceholder.typicode.com/posts/1')

.then((*resbal*)=>{

  console.log(*resbal*);

})

We are just loging out the response and we can see body header status text and a lot of things.

Now we are having the response. Which is a ReadbleSTREAM, But we cannot use the data. So we have to convert to json and the we can use the data

fetch('https://jsonplaceholder.typicode.com/posts/1')

.then((*resbal*)=>*resbal*.json())

.then((*resbal*)=>console.log(*resbal*))

Now the return of the promise could be an error and we have to take care of it. So lets change the url by 101 at the and as data is from 1-100 and we are giving 101. So we gonna get error of 404 if we do following code.

fetch('https://jsonplaceholder.typicode.com/posts/101')

.then((*resbal*)=>*resbal*.json())

.then((*resbal*)=>console.log(*resbal*))

.catch((*err*)=>console.log(*err*))

How to handle it ? as error handler could not catch the 404 eror.

fetch('https://jsonplaceholder.typicode.com/posts/101')

.then((*resbal*)=>{

  if(!*resbal*.ok){

    const errorMesssage= `Your Error:${*resbal*.status}`;

    throw **new** *Error*(errorMesssage);

  }

  return *resbal*.json();

})

.then((*resbal*)=>console.log(*resbal*))

.catch((*err*)=>console.log(*err*))

We are checking if the response is oke or not. If not oke then we have to throw and error. and we are saving a a message by the name errorMessage and throwing the variable. And if we do this then the .catch will catch the error. so now we can catch the 404 eror.

NOW lets learn about POST method

fetch('https://jsonplaceholder.typicode.com/posts',{

  method: 'POST',

  headers: {

    'Content-type': 'application/json; charset=UTF-8',

  },

  body: JSON.stringify({

    title: 'fooMango',

    body: 'barMango',

    userId: 1,

  }),

})

Remember .then part is there after the code. To minimize the space we just pasting the post method part in the doc.

We can see that we have to type method which is post and then set the header. And then the body which has to be converted to stringify as we are posting JS object to json object.

If we look closely. First the URL then coma and inside the curly brace all the property as an object. So method, header, body everyting is property and then closing the curly brace then close the fetch method first brace. SO DON’T GET CONFUSE WITH THE BRACES.

Now lets PUT data

Simple as that just in the url add the id at the very and and just change the body of the data if you want and then just same as the previous code.

fetch('https://jsonplaceholder.typicode.com/posts/1',{

  method: 'PUT',

  headers: {

    'Content-type': 'application/json; charset=UTF-8',

  },

  body: JSON.stringify({

    id: 1,

    title: 'fooMangoMango',

    body: 'barMangoMango',

    userId: 1,

  }),

})

Now lets learn about update byt PATCH data

fetch('https://jsonplaceholder.typicode.com/posts/1',{

  method: 'PATCH',

  headers: {

    'Content-type': 'application/json; charset=UTF-8',

  },

  body: JSON.stringify({

    title: 'title has been changed',

  }),

})

Just changing the data that we want to update. In this case which is ttile.

Lets learn DELETE method

fetch('https://jsonplaceholder.typicode.com/posts/1',{

  method: 'DELETE'

})

Just passing the delete url and id at the end which we want to delete and that’s it.

So we learn how to do all the stuff with different method

NOW lets learn with ASYNCH and AWAIT.

const makeRequest = async ()=>{

  const resbal= await fetch('https://jsonplaceholder.typicode.com/posts');

  const databal = await resbal.json();

  return databal

}

const getData= ()=>{

  makeRequest()

  .then((*res2ndbal*)=>console.log(*res2ndbal*))

}

getData();

we are creating a function which is asynch by the name makeRequest and saving the fetched data in a variable call resbal. Before fetching we are giving await as this is an async fucntin. Then we are saving the data converting json data by the name as databal. And returning databal

another function to getData. Which is going to call the makeRequest which will deliver a promise which will be caught in getData method using .then

so now we can just use this makeRequest anytime with different function.

Now Error handling

const makeRequest = async (*url*)=>{

  const resbal= await fetch(*url*);

  if(!resbal.ok){

    const messageError= `Error: ${resbal.status}`;

    throw **new** *Error*(messageError);

  }

  const databal = await resbal.json();

  return databal

}

const getData= ()=>{

  makeRequest('https://jsonplaceholder.typicode.com/posts')

  .then((*res2ndbal*)=>console.log(*res2ndbal*))

  .catch((*err*)=>{console.log(*err*)})

}

getData();

slightly we changed the code as we are now passing the url from the getData function as the url of making request could be changed so we should not hard code it. So simply taking as a parameter;.

NOW lets learn how to POST data

const makeRequest = async (*url*,*config*)=>{

  const resbal= await fetch(*url*,*config*);

  if(!resbal.ok){

    const messageError= `Error: ${resbal.status}`;

    throw **new** *Error*(messageError);

  }

  const databal = await resbal.json();

  return databal

}

const sendData= ()=>{

  makeRequest('https://jsonplaceholder.typicode.com/posts',{

    method: 'POST',

    body: JSON.stringify({

    title: 'foo',

    body: 'bar',

    userId: 1,

  }),

  headers: {

    'Content-type': 'application/json; charset=UTF-8',

  }})

  .then((*res2ndbal*)=>console.log(*res2ndbal*))

  .catch((*err*)=>{console.log(*err*)})

}

sendData();

we are passing the config of our selve which is going to be posted and passing it as a parameter along with the URL. Just we learned in XHTTPRequest.

PUT method

const updateData= ()=>{

  makeRequest('https://jsonplaceholder.typicode.com/posts/1',{

    method: 'PUT',

    body: JSON.stringify({

    title: 'fooMango',

    body: 'barMango',

    userId: 1,

  }),

  headers: {

    'Content-type': 'application/json; charset=UTF-8',

  }})

  .then((*res2ndbal*)=>console.log(*res2ndbal*))

  .catch((*err*)=>{console.log(*err*)})

}

updateData();

PATCH method

const patchData= ()=>{

  makeRequest('https://jsonplaceholder.typicode.com/posts/1',{

    method: 'PATCH',

    body: JSON.stringify({

    title: 'fooMangoMango',

  }),

  headers: {

    'Content-type': 'application/json; charset=UTF-8',

  }})

  .then((*res2ndbal*)=>console.log(*res2ndbal*))

  .catch((*err*)=>{console.log(*err*)})

}

patchData();

Delete method

const deleteData= ()=>{

  makeRequest('https://jsonplaceholder.typicode.com/posts/1',{

    method: 'DELETE'

  })

  .then((*res2ndbal*)=>console.log(*res2ndbal*))

  .catch((*err*)=>{console.log(*err*)})

}

deleteData();

SIMPLE.

AXIOS

XMLHTTPREQUESt and fetch is by default built in. but axios is a library and we have to install it.

//axio is a library

//it helps to make request from browser. (plain js/vue/react/), node.js

//its very easy to use

//it returns promise

//throws erro brilliantly

//no need to set header cause axios is intelligent

//axios(config)

//axios [url,config]

//axios.get(url, [,config])

//axios.post(url, [,config])

//axios.put(url, [,config])

//axios.patch(url, [,config])

//axios.delete(url, [,config]

//axios returns a response object = data, status, statustext, header, config

If we wanted to use it by node.js we could have used, npm install axio –save but as we are going to use it from html we can just install cdn. Lets go to axios cdn library and copy paste the script url and paste it in html header and save.

Now if we console.log(window) we can see that axios is there.

So now lets get the data by using url from fake api Json placeholder

axios.get('https://jsonplaceholder.typicode.com/posts')

.then((*res*)=>console.log(*res*);

As we know axios returns a promise this above code is going to give us config, data, status, header, stadtutext, request.

So now to print data. We just have to type

axios.get('https://jsonplaceholder.typicode.com/posts')

.then((*res*)=>console.log(*res*.data));

Its going to return 100 data.

Now lets have error handling

axios.get('https://jsonplaceholder.typicode.com/posts/101')

.then((*res*)=>console.log(*res*.data))

.catch((*err*)=>console.log(*err*))

We are changing the url and catching the error. simple

Now POST method

axios.post('https://jsonplaceholder.typicode.com/posts', {

  method: 'POST',

  body: JSON.stringify({

  title: 'foo',

  body: 'bar',

  userId: 1,

  }),

})

.then((*res*)=>console.log(*res*.data))

.catch((*err*)=>console.log(*err*))

As you can see that now axios.post method and inside the method we are declaring properties and config. But one thing we don’t need to tell header body now. That’s the update of AXIOS. If we go to dev tool and go to network and we can see in the response header that there is content type is written. Simple

Now Put Method

axios.put('https://jsonplaceholder.typicode.com/posts/1', {

  method: 'PUT',

  body: JSON.stringify({

  title: 'fooMango',

  body: 'barMango',

  userId: 1,

  }),

})

.then((*res*)=>console.log(*res*.data))

.catch((*err*)=>console.log(*err*))

Now Patch metod

axios.patch('https://jsonplaceholder.typicode.com/posts/1', {

  method: 'PATCH',

  body: JSON.stringify({

  body: 'barMangoMango',

  }),

})

.then((*res*)=>console.log(*res*.data))

.catch((*err*)=>console.log(*err*))

Now delete method

axios.delete('https://jsonplaceholder.typicode.com/posts/1')

.then((*res*)=>console.log(*res*.data))

.catch((*err*)=>console.log(*err*))

NOW axios using ASYNC and AWAIT.

const makeRequest = async(*config*)=>{

  return await axios(*config*)

}

const getData= ()=>{

  makeRequest("https://jsonplaceholder.typicode.com/posts/")

  .then((*resbal*)=>console.log(*resbal*))

  .catch((*err*)=>console.log(*err*))

}

getData();

we can see that we are using async and await and just declaring a function by the name makeRequest which is going to receive a parameter name config as the request could be any method. It could be post , put , patch so we don’t explicityly say that. We just do as it is shown. We are passing the url from the getData function by calling makeRequest function and getting the promise and catching the error if any.

Now if we want to Create data with POST

const createData= ()=>{

  makeRequest({

    url: 'https://jsonplaceholder.typicode.com/posts',

    method: 'POST',

    data: JSON.stringify({

    title: 'foo',

    body: 'bar',

    userId: 1,

    }),

  })

  .then((*resbal*)=>console.log(*resbal*))

  .catch((*err*)=>console.log(*err*))

}

createData();

we can see that now in the makeRequest function where its getting called we have slight change. In a curly brace we are giving method, url, data as a json obj.

update with PUT

const updateData= ()=>{

  makeRequest({

    url: 'https://jsonplaceholder.typicode.com/posts/1',

    method: 'PUT',

    data: JSON.stringify({

    id: 1,

    title: 'fooMango',

    body: 'barMango',

    userId: 1,

    }),

  })

  .then((*resbal*)=>console.log(*resbal*.data))

  .catch((*err*)=>console.log(*err*))

}

updateData();

Delete method

const deleteData= ()=>{

  makeRequest({

    url: 'https://jsonplaceholder.typicode.com/posts/1',

    method: 'DELETE',

  })

  .then((*resbal*)=>console.log(*resbal*.data))

  .catch((*err*)=>console.log(*err*))

}

deleteData();