.

├── app.js

├── node\_modules

├── package-lock.json

├── package.json

├── public

└── routes

3 directories, 3 files

Mels-MBP:express\_apps meldejesus$ tree -L 2

.

├── app.js

├── node\_modules

├── package.json

├── public

│ ├── index.html.

│ └── main.js

└── routes

└── api.js.

INITIAL DATA: in your server file, an array of objects is present.

var data = [{

name: "Jack", age: 26

}]

LISTENERS: In your main JS file, on click will call submit. If values are submitted, these are grabbed. Values are typically grabbed for PUT and POST methods.

let y = document.getElementById("submit");

y.addEventListener("click", submit)

function submit(e) {

e.preventDefault();

var name=document.getElementsByName("username")[0].value;

var rank=document.getElementById("rank").value;

PREP DATA: if PUT or POST, encode your payload either as JSON or URL. You will have to note your choice in the requestHeader.

In the RequestHeader, JSON would have a “Content-Type” of “application/json”.

var newEntry = {

name: name,

rank: rank

}

newEntry=JSON.stringify(newEntry)

Hwever, URL encoding would have a “Content-Type” of “application/x-www-form-urlencoded”. It uses ASCII character-set, replacing “unsafe” ASCII with a “%” followed by two hexadecimals. For example, a space is encoded as %20.

var newEntry = "name="+name+"&rank="+rank;

MAKE CALL: setup and call ajax function

loadDoc(“data/api”, later);

loadDoc(“url-2”, myFunc2);

function loadDoc(url, callback) {

STEP ONE: create an instance of the XHR object, which allows us to make HTTP requests

var xhr = new XMLHttpRequest();

STEP TWO: the open parameters define the METHOD and URL and ASYNC boolean:

xhr.open( “GET”, “data/api”, true);

STEP THREE: This is used to define the payload body, which is typically only present for POST and PUT requests. These parameters will be headerName and headerValue:

JSON:

xhr.setRequestHeader("Content-Type", "application/json")

URL-Encoding:

xhr.setRequestHeader('Content-Type', 'application/x-www-form-urlencoded');

url-encoded data looks like this:

key0=value0&key1=value1

And every key and every value are, of course, URI encoded.

'name=John&your%20age=50’

STEP FOUR: onreadystatechange defines the function that will be called only when readyState changes. readyState is the property holding the status of the XMLHttpRequest. In short, the response will not be available to the user until the readyState = 4, so it’s best to have it not process the response until this readyState has been achieved.

xhr.onreadystatechange = function () {

if (xhr.status === 200 && xhr.readyState === 4 ){

//with JSON, you have to parse before accessing object

let response = JSON.parse(xhr.responseText)[0].name)

callback(response)

} else if (xhr.status !== 200) {

alert('Request failed. Returned status of ' + xhr.status);

}

};

STEP FIVE: Send data

xhr.send()

If a PUT or POST, insert your payload as an argument:

xhr.send(newEntry)

HEADS UP NOTES

XHR has no PUT or DELETE method, so you’d have to create a function in the server that handles this.

The call would look exactly like the POSt except for:

xhr.open('PUT', 'api/data', true);

and the router side might capture this, iterate through the data in search of a foreign key (in this case, “age”), delete this item, and replace it with the new item using the ‘splice’ method:

router.put('/data', function(req, res) {

data.map(function(e, i) {

if (req.body.age == e.age) {

data.splice(i, 1, req.body )

}

})

res.send(data)

})

STEP FOUR NOTES: READY STATES

0 – request not initialized; open() hasn’t been called yet

1 – requrest has been set up; send() has not been called yet

2 – request has been sent; send() has been called, and headers and status available

3 – request is in process; downloading – responseText holds partial data

4 – request is complete

SOME TERMS

Xhr.onreadystatechange - Defines a function to be called when the readyState property changes

Xhr.readyState - Holds the status of the XMLHttpRequest. A readyState prior to 4 will yield an empty responseText

Onload -> called when readyState==4, but not if there’s an error

Onerror?

Onloadend –> this fires on readyState === 4

xhr.status & statusText hods status of XHR object

xhr.responseText includes the raw response

xhr.responseXML is a DOM object if you requested an XML doc