**Lesson 4: Fetch API, HTTP web module**

**Fetch API:**

For years, [XMLHttpRequest](https://developer.mozilla.org/en-US/docs/Web/API/XMLHttpRequest) has been web developers’ trusted sidekick. Whether directly or under the hood, XMLHttpRequest has enabled Ajax and a whole new type of interactive experience, from Gmail to Facebook.

However, XMLHttpRequest is slowly being superseded by the [Fetch API](https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API). Both can be used to make network requests, but the Fetch API is Promise-based, which enables a cleaner, more concise syntax

The Fetch API provides a fetch() method defined on the window object, which you can use to perform requests. This method returns a [Promise](https://www.sitepoint.com/overview-javascript-promises/) that you can use to retrieve the response of the request.

The fetch method only has one mandatory argument, which is the URL of the resource you wish to fetch.

We can’t block the user interface waiting until the request finishes. That’s why fetch() returns a Promise, an object which represents a future result. In the above example, we’re using the then method to wait for the server’s response and log it to the console.

**Class code Demonstration:**

<html>

    <script>

function fetchdata()

{

 myimage = document.querySelector('.imag');

 fetch('https://upload.wikimedia.org/wikipedia/commons/7/77/Delete\_key1.jpg')

 .then(res => res.blob())

 .then(res => {

    const objectURL =  URL.createObjectURL(res);

    myimage.src = objectURL;

 });

}

</script>

<body>

<button onclick = "fetchdata()"> Fetch the Data</button> <br>

<img class = "imag" src = "https://wikipedia.org/static/images/project-logos/frwiki-1.5x.png"/>

</body>

</html>

**HTTP Module: -** [**https://nodejs.org/api/http.html**](https://nodejs.org/api/http.html)

To make HTTP requests in Node.js, there is a built-in module **HTTP** in Node.js to transfer data over the HTTP. To use the HTTP server in node, we need to require the HTTP module. The HTTP module creates an HTTP server that listens to server ports and gives a response back to the client.

It can be included using

const http = require('http')

The module provides some properties and methods, and some classes.

## Properties

### http.METHODS

This property lists all the HTTP methods supported:

> require('http').METHODS

### http.STATUS\_CODES

This property lists all the HTTP status codes and their description:

> require('http').STATUS\_CODES

### http.globalAgent

Points to the global instance of the Agent object, which is an instance of the http.Agent class.

It's used to manage connections persistence and reuse for HTTP clients, and it's a key component of Node.js HTTP networking.

## Methods

### http.createServer()

Return a new instance of the http.Server class.

## Usage:

const server = http.createServer((req, res) => {

//handle every single request with this callback

})

**http.request()**

Makes an HTTP request to a server, creating an instance of the http.ClientRequest class.

**http.get()**

Similar to http.request(), but automatically sets the HTTP method to GET, and calls req.end() automatically.

## Classes

The HTTP module provides 5 classes:

* http.Agent
* http.ClientRequest
* http.Server
* http.ServerResponse
* http.IncomingMessage

**http.Agent**

Node.js creates a global instance of the http.Agent class to manage connections persistence and reuse for HTTP clients, a key component of Node.js HTTP networking.

This object makes sure that every request made to a server is queued and a single socket is reused.

It also maintains a pool of sockets. This is key for performance reasons.

**http.ClientRequest**

An http.ClientRequest object is created when http.request() or http.get() is called.

When a response is received, the response event is called with the response, with an http.IncomingMessage instance as argument.

The returned data of a response can be read in 2 ways:

* you can call the response.read() method
* in the response event handler you can setup an event listener for the data event, so you can listen for the data streamed into.

**http.Server**

This class is commonly instantiated and returned when creating a new server using http.createServer().

Once you have a server object, you have access to its methods:

* close() stops the server from accepting new connections
* listen() starts the HTTP server and listens for connections

**http.ServerResponse**

Created by an http.Server and passed as the second parameter to the request event it fires.

Commonly known and used in code as res:

const server = http.createServer((req, res) => {

//res is an http.ServerResponse object

})

The method you'll always call in the handler is end(), which closes the response, the message is complete and the server can send it to the client. It must be called on each response.

These methods are used to interact with HTTP headers:

* getHeaderNames() get the list of the names of the HTTP headers already set
* getHeaders() get a copy of the HTTP headers already set
* setHeader('headername', value) sets an HTTP header value
* getHeader('headername') gets an HTTP header already set
* removeHeader('headername') removes an HTTP header already set
* hasHeader('headername') return true if the response has that header set
* headersSent() return true if the headers have already been sent to the client

After processing the headers you can send them to the client by calling response.writeHead(), which accepts the statusCode as the first parameter, the optional status message, and the headers object.

To send data to the client in the response body, you use write(). It will send buffered data to the HTTP response stream.

**http.IncomingMessage**

An http.IncomingMessage object is created by:

* http.Server when listening to the request event
* http.ClientRequest when listening to the response event

It can be used to access the response:

* status using its statusCode and statusMessage methods
* headers using its headers method or rawHeaders
* HTTP method using its method method
* HTTP version using the httpVersion method
* URL using the url method
* underlying socket using the socket method

The data is accessed using streams, since http.IncomingMessage implements the Readable Stream interface.

**Class code Demonstration:**

**Client.js**

var http = require('http');

var fetch = require('node-fetch');

//options to be used by request

/\*

var options =

{

host: 'localhost',

port: '8081',

path : '/pes.html',

};

//callback function is used to deal with the response

var callback = function(response)

{

    var body = '';

    response.on('data',function(data)

    {

        body+= data;

    });

}

//make a request to the server

var req = http.request(options,callback);

req.end();  \*/

/\*fetch('http://localhost:8081/sample.json', {

        method: 'GET',

        headers: { 'Content-Type': 'application/json' },

    })

    .then(res => res.json())

    .then(res => console.log(res));

\*/

    fetch('http://localhost:8081/sample.json', {

        method: 'POST',

        body:    JSON.stringify({"name":"Aruna modified","col":"MIT"}),

        headers: { 'Content-Type': 'application/json' },

    })

     .then(res => console.log(res));

**Server.js**

// http module ccreating a web server

var url = require('url');

var http = require('http');

var fs = require('fs');

var MongoClient = require('mongodb').MongoClient;

//create a server

http.createServer(function (request, response) {

    //parse the request containing the filename

    var pathname = url.parse(request.url).pathname;

    //print the name of the file for which request is made

    console.log("Request for" + pathname + "Received.");

    if (request.method == "GET") {

        //Read the requested file content from filesystem

        // fs.readFile(pathname.substr(1), function (err, data) {

        //     if (err) {

        //         console.log(err);

        //         //HTTP Status 404 not found

        //         response.writeHead(404, { 'Content-Type': 'text/html' });

        //     }

        //     else {

        //         //page found and status of 200 has to be returned

        //         response.writeHead(200, { 'Content-Type': 'text/html' });

        //         //write the content of the file to response body

        //         response.write(data.toString());

        //     }

        //     //send the responseBody

        //     response.end();

        // });

        //Read from the mongodb

        console.log('executing mongo');

        MongoClient.connect("mongodb://localhost:27017", {

            useUnifiedTopology:

                true

        }, function (err, client) {

            console.log("Connected successfully to server");

            const db = client.db("pes");

            db.collection("student").find({}).toArray(function (err, docs) {

                response.writeHead(200, { 'Content-Type': 'application/json' });

                //write the content of the file to response body

                response.write(JSON.stringify(docs));

                client.close();

                response.end();

            });

        });

    }

    else {

        let body = [];

        request.on('data', (chunk) => {

            body.push(chunk);

        }).on('end', () => {

            body = Buffer.concat(body).toString();

            // at this point, `body` has the entire request body stored in it as a string

        });

        // write to file

        // fs.writeFile(pathname.substr(1), body, (err, res) => {

        //     response.writeHead(200, { 'Content-Type': 'application/json' });

        //     response.end();

        // });

        //write to mongodb

        console.log('executing mongo');

        MongoClient.connect("mongodb://localhost:27017", {

            useUnifiedTopology:

                true

        }, function (err, client) {

            console.log("Connected successfully to server");

            const db = client.db("pes");

            db.collection("student").insertOne(JSON.parse(body)).then(r => {

                response.writeHead(200, { 'Content-Type': 'application/json' });

                //write the content of the file to response body

                client.close();

                response.end();

            })

        });

    }

}).listen(8081);

console.log('server running at the link http://localhost/8081');