COMMUNICATION WITH SERVER. AJAX. Fetch API

Ivaniuk Oleh

Upd. 12.2020

softserve

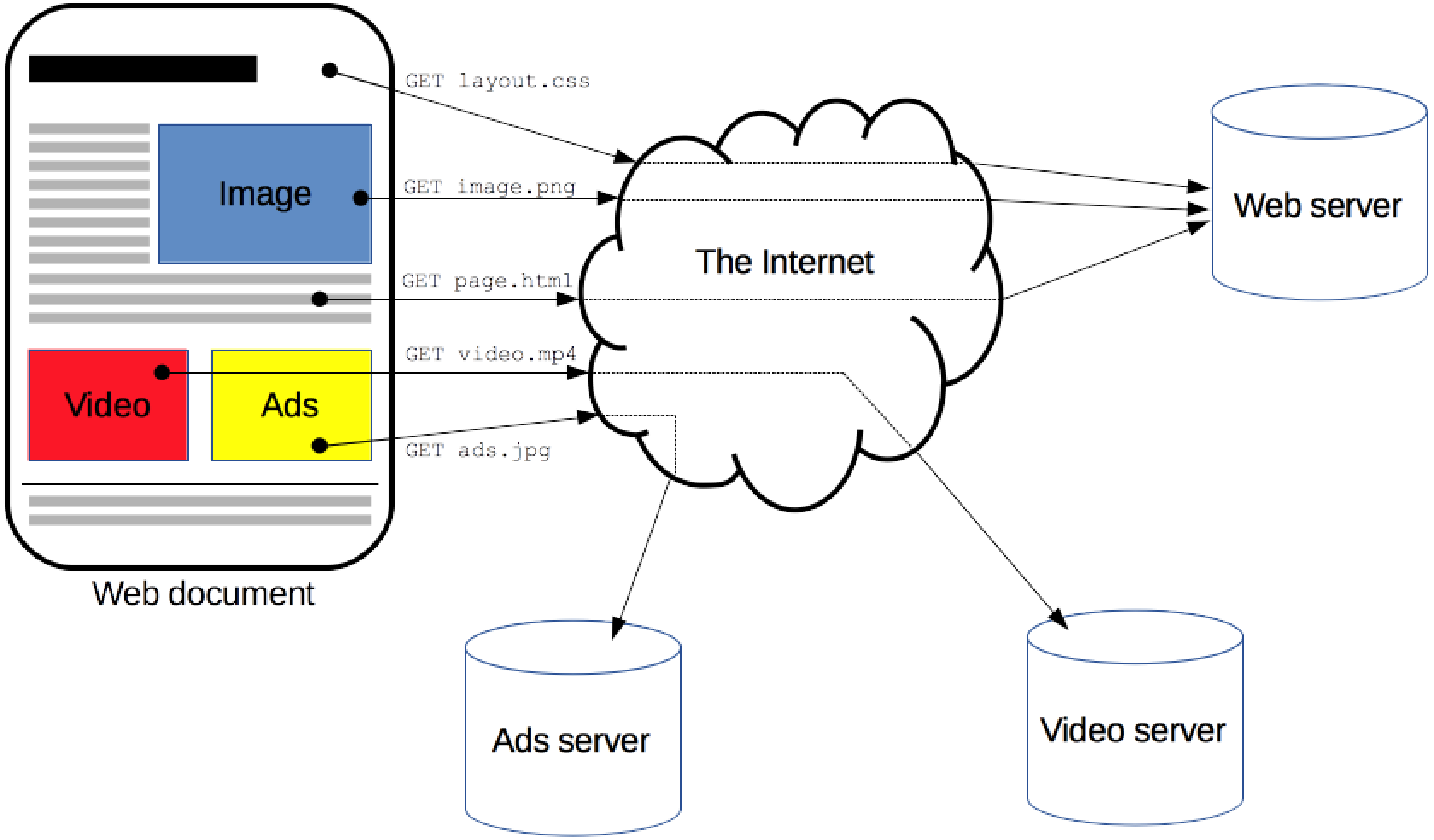
AGENDA

- Communication with server
  - HTTP
  - HTTP request structure
  - HTTP response structure
  - HTTP request methods
  - Idempotence methods
  - Stateless protocol
- AJAX
  - AJAX web application model
  - Create AJAX-request to server
  - Handling server response
  - Fetch API

COMMUNICATION WITH SERVER

softserve

COMMUNICATION OF WEB BROWSER WITH SERVER



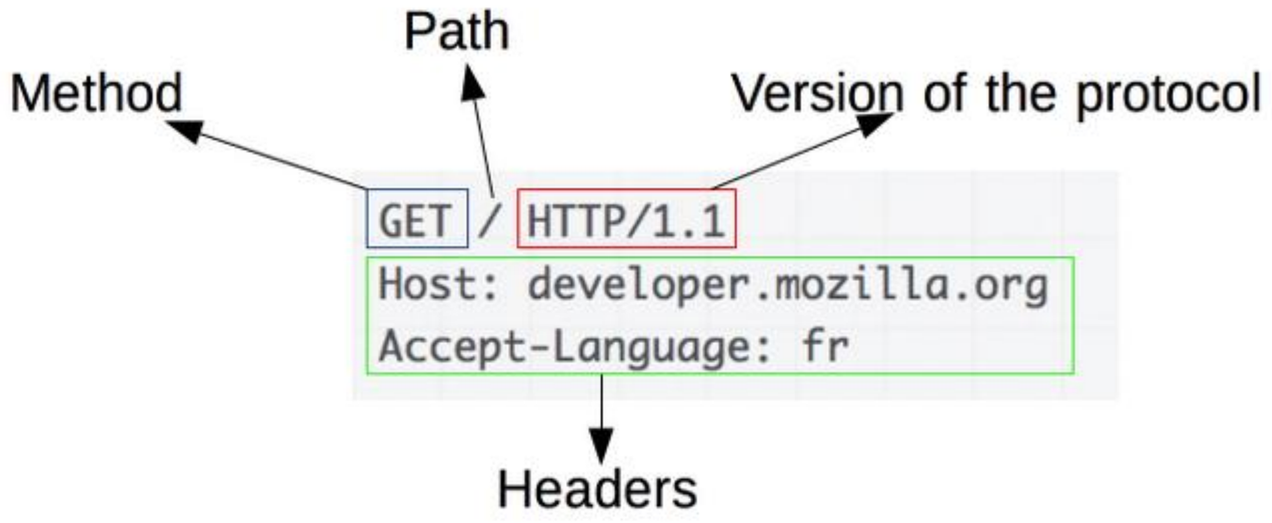
softserve

HTTP

- **Hypertext Transfer Protocol** (**HTTP**) is an application protocol for the transmission of hypertext documents such as HTML.
- It is designed to communicate between web browsers and web servers, although in principle HTTP can be used for other purposes.
- The protocol follows the classic **client-server model** when the client opens a connection to create a request and then waits for a response.
- HTTP is a **stateless protocol**, that is, the server does not store any data (status) between two request-response pairs.
- Although HTTP is TCP/IP based, it can also use any other transport layer with guaranteed delivery.

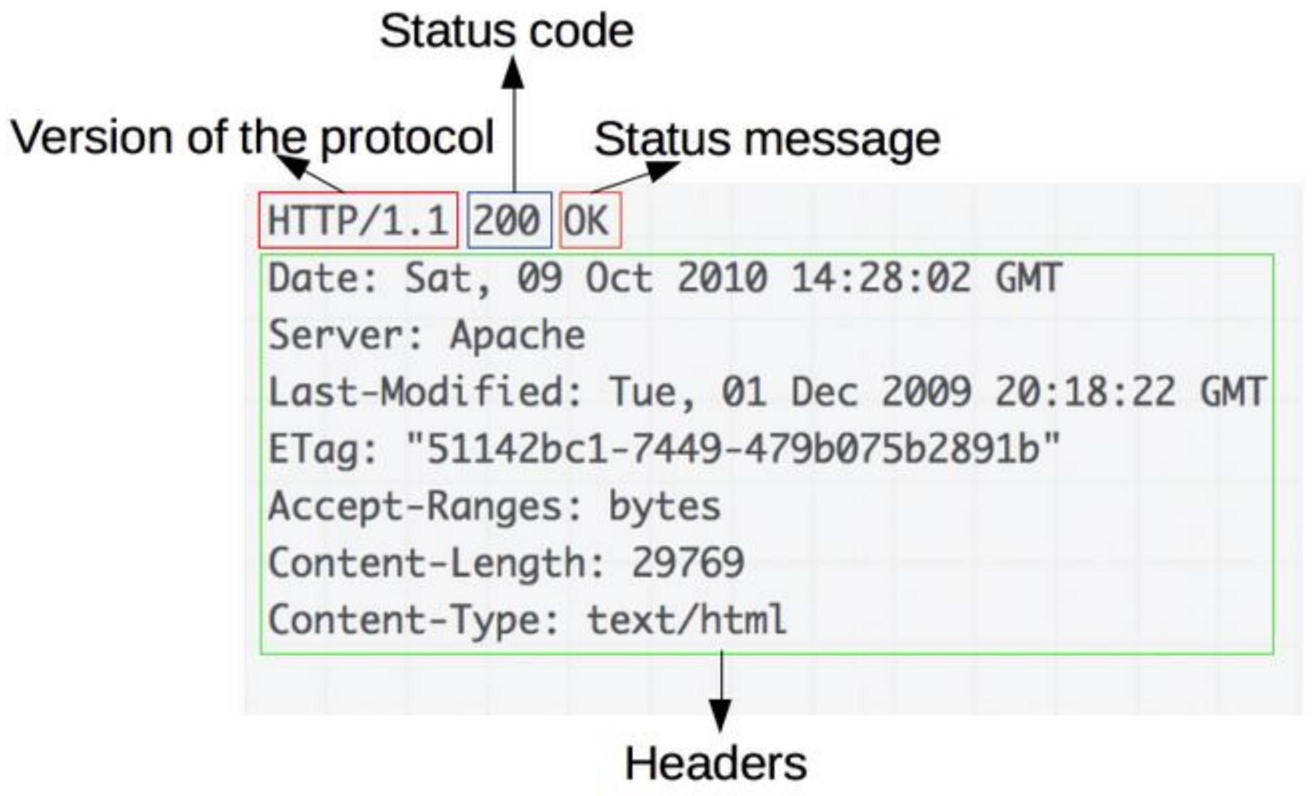
softserve

HTTP request structure



```
GET / HTTP/1.1
Host: developer.mozilla.org
Accept-Language: fr
```

HTTP response structure



```
HTTP/1.1 200 OK
Date: Sat, 09 Oct 2010 14:28:02 GMT
Server: Apache
Last-Modified: Tue, 01 Dec 2009 20:18:22 GMT
ETag: "51142bc1-7449-479b075b2891b"
Accept-Ranges: bytes
Content-Length: 29769
Content-Type: text/html
```

HTTP request methods

GET - the GET method requests the representation of a resource. Queries using this method can only receive data.

POST - used to send data to a specific resource. It often causes a change in status or some side effects on the server.

PUT - replaces all current views of the resource with query data.

PATCH - used to partially modify a resource.

DELETE - deletes the specified resource.

OPTIONS - is used to describe the connection options of a resource.

HTTP status codes

1xx Informational responses - request received, process continues

2xx Success - the request was successfully received, understood and processed

3xx Redirection - further steps are required to complete the request

4xx Client errors - the query contains bad syntax or cannot be executed

5xx Server errors - the server did not complete the request

Full list HTTP Status Messages

softserve

IDEMPOTENCE METHODS

The HTTP method is **idempotent** if the repeated identical request, made one or more times in a row, has the same effect, which does not change the status of the server.

In other words, the idempotent method should have no side-effects other than collecting statistics or similar operations.

The GET, HEAD, PUT and DELETE methods are idempotent, but not the POST method. Also, all safe methods are idempotent.

The GET, HEAD, OPTIONS, and TRACE methods are defined as **secure**, meaning that they are for information only and should **not change the status of the server**.

softserve

STATELESS PROTOCOL

The **stateless protocol** is a data protocol that assigns each request to an independent transaction that is not associated with a previous request, that is, communicating with the server consists of independent request-response pairs.

The stateless protocol does not require storing session information on the server or status for each client during multiple requests. In contrast, a protocol that requires an account of the server's internal state is called a **stateful protocol**.

Sample protocol without saving status - HTTP means that every request message can be understood in isolation from other requests.

AJAX

softserve

AJAX

AJAX (**Asynchronous JavaScript and XML**) - server access technology without reloading the page. This reduces response time and makes the web application more like a desktop.

AJAX allows web pages to be updated asynchronously by exchanging data with a web server behind the scenes. This means that it is possible to update parts of a web page, without reloading the whole page.

AJAX is a web development technique for creating interactive and fast web applications.

softserve

AJAX. A COMBINATION OF TECHNOLOGIES

JavaScript

- Loosely typed scripting language.
- JavaScript function is called when an event occurs in a page.
- Glue for the whole AJAX operation.

DOM

- API for accessing and manipulating structured documents.
- Represents the structure of XML and HTML documents.

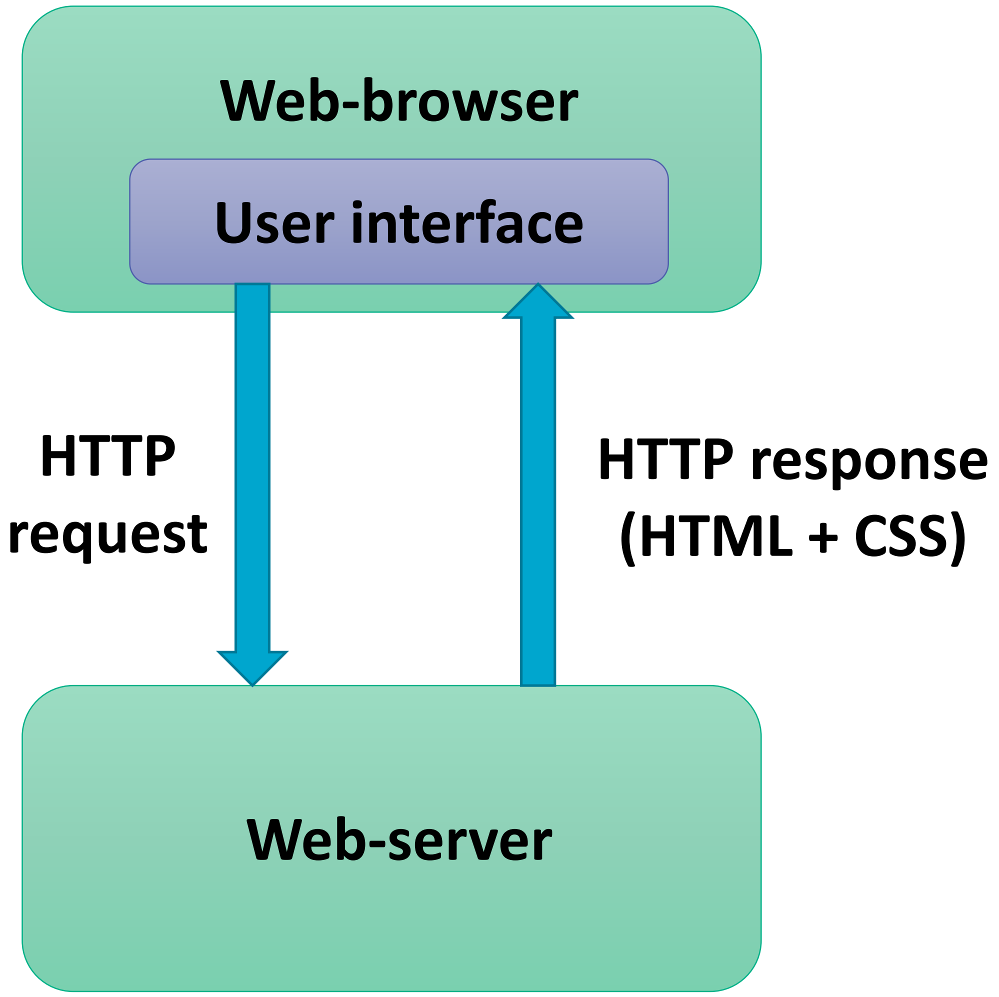
CSS

- Allows for a clear separation of the presentation style from the content and may be changed programmatically by JavaScript

XMLHttpRequest

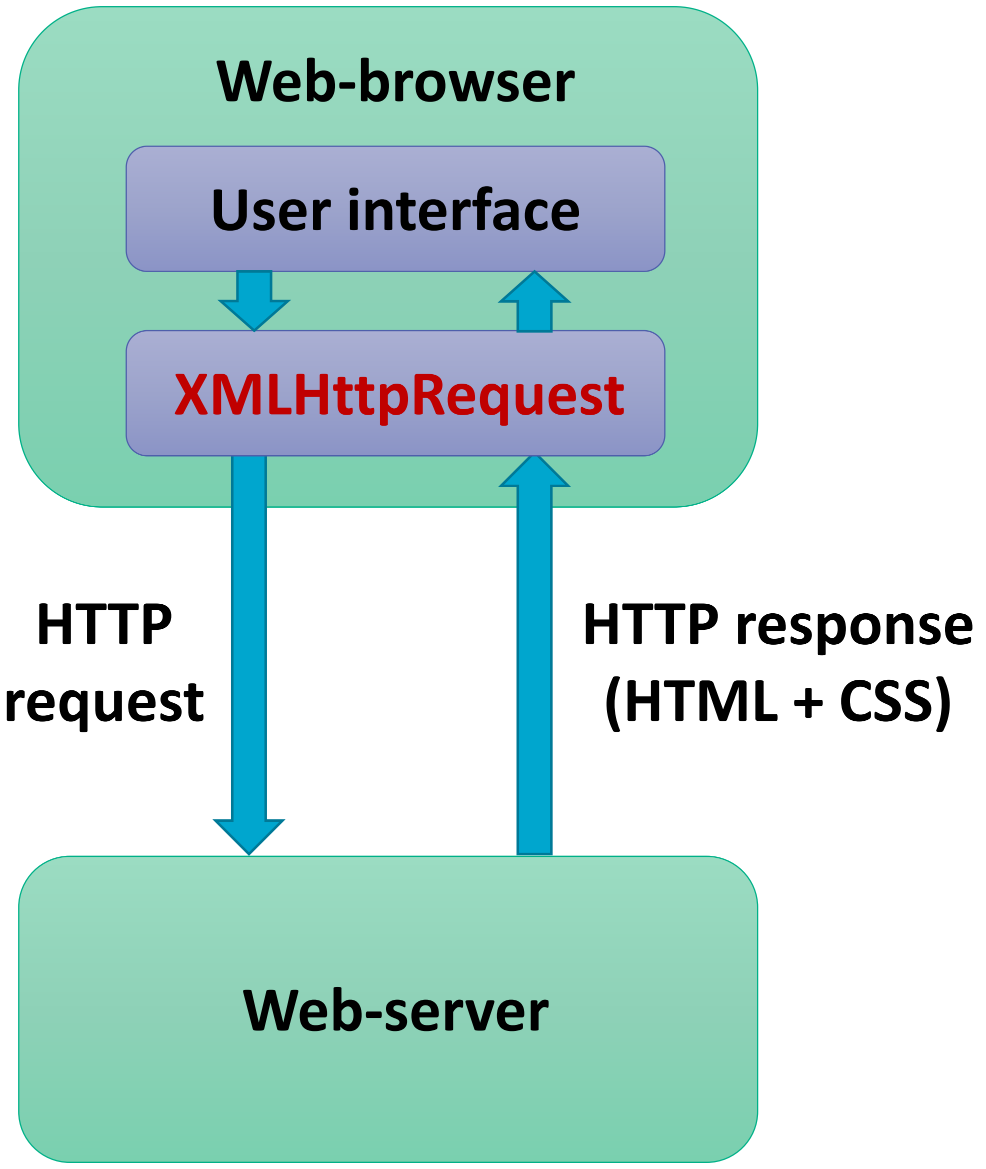
- JavaScript object that performs asynchronous interaction with the server.

CLASSIC WEB APPLICATION MODEL



softserve

AJAX WEB APPLICATION MODEL



softserve

DATA FORMATS FOR EXCHANGE WITH SERVER

Data formats for exchange with server

XML

- XML (*e**X**tensible **M**arkup **L**anguage*) - extensible markup language. XML solves the problem of storing and transporting data, focusing on what that data is, HTML solves the problem of displaying data, focusing on how that data looks.

HTML

- Simple HTML code. This data is often called HTML snippets (code snippets).

JSON

- JSON (JavaScript Object Notation). The general idea is to deliver some text (string) that can be interpreted as a JavaScript object

STEP 1. CREATE AN XMLHTTPREQUEST OBJECT

Before you perform Ajax communication between client and server, the first thing you must do is to instantiate an **XMLHttpRequest object**, as shown below:

```
1)  const ajaxRequest = new XMLHttpRequest();
```

The XMLHttpRequest object allows JavaScript to make HTTP requests to the server without reloading the page from JavaScript.

Despite the word "XML" in the title, XMLHttpRequest can **work with any data**, and not just with XML.

All modern browsers (Chrome, Firefox, IE7+, Edge, Safari, Opera) have a built-in XMLHttpRequest object.

softserve

STEP 2. CONFIGURING THE XMLHTTPREQUEST OBJECT

Next step in sending the request to the server is to instantiating the newly-created request object using the ***open()*** method of the XMLHttpRequest object:

```
2) ajaxRequest.open(method, URL, [async, user, password] );
```

- ***method***: HTTP-method. As a rule, GET or POST is used, although more exotic ones are available, like TRACE / DELETE / PUT, etc.
- ***URL***: request address - the server (file) location. You can use not only http/https, but other protocols, such as ftp:// and file://.
- ***async***: true (asynchronous) or false (synchronous)
- ***user, password*** - login and password for HTTP authorization, if needed.

Example:

```
ajaxRequest.open("GET", "info.txt", true);
```

STEP 3. HANDLING SERVER RESPONSE

The ***readyState*** property holds the status of the XMLHttpRequest.

The ***onreadystatechange*** property defines a function to be executed when the *readyState* changes.

The ***status*** property and the ***statusText*** property holds the status of the XMLHttpRequest object.

3)

```
ajaxRequest.onreadystatechange = function () {
        if (ajaxRequest.readyState < 4)
        // while waiting response from server
        document.getElementById('div1').innerHTML = "Loading...";
} else if  (ajaxRequest.readyState === 4) {
         // 4 = Response from server has been completely loaded.
        if (ajaxRequest.status === 200)  {
        // http status is 200 - successful
                document.getElementById('div1').innerHTML =
                ajaxRequest.responseText;
        }
}
```

softserve

READYSTATE PROPERTIES

0 – UNSENT – initial state.

1 – OPENED – called open

2 - HEADERS_RECEIVED – received headers

3 – LOADING – body being loaded (regular data packet received)

4 – DONE – request completed

The request passes the states in the order 0 → 1 → 2 → 3 →… → 3 → 4, state 3 is repeated every time the next data packet is received over the network.

softserve

EVENTS DURING REQUEST PROCESSING

Loadstart – request started

Progress – the browser received another data packet

Abort – request canceled

Error – an error occurred

Load – the request was successfully (no error) completed

Timeout – request completed by timeout

Loadend - request completed (successful or unsuccessful)

XMLHTTPREQUEST METHODS

Method	Description
new XMLHttpRequest()	Creates a new XMLHttpRequest object
abort()	Cancels the current request
getAllResponseHeaders()	Returns header information
getResponseHeader()	Returns specific header information
open(method, url, async, user, psw)	Specifies the type of request
send()	Sends the request to the server. Used for GET requests
send(string)	Sends the request to the server. Used for POST requests
setRequestHeader()	Adds a label/value pair to the header to be sent

rve

XMLHTTPREQUEST PROPERTIES

Property	Description
onreadystatechange	Defines a function to be called when the readyState property changes
readyState	Holds the status of the XMLHttpRequest (possible values 0, 1, 2, 3, 4)
responseText	Returns the response data as a string
responseXML	Returns the response data as XML data
status	Returns the status-number of a request (from 1xx to 5xx)
statusText	Returns the status-text (e.g. "OK" or "Not Found")

softserve

STEP 4. SENDING A REQUEST TO THE SERVER

To send a request to a server, we use the **send()** method of the XMLHttpRequest object:

```
4) ajaxRequest.send( [body] );
```

This method opens the connection and sends a request to the server.

The ***body*** is the request body. Not every request has a body, for example, GET-requests do not have a body, while POST has basic data just transmitted through the body.

Post request with body example:

```
ajaxRequest.open("POST", "demo_post2.asp", true);
ajaxRequest.setRequestHeader("Content-type", "application/x-www-form-urlencoded");
ajaxRequest.send("fname=Henry&lname=Ford");
```

GET VS POST

- GET requests can be cached
- GET requests remain in the browser history
- GET requests can be bookmarked
- GET requests should never be used when dealing with sensitive data
- GET requests have length restrictions
- GET requests should be used only to retrieve data

vs

- POST requests are never cached
- POST requests do not remain in the browser history
- POST requests cannot be bookmarked
- POST requests have no restrictions on data length

softserve

SYNCHRONOUS AND ASYNCHRONOUS REQUESTS

If the *open* method sets the ***async*** parameter to ***false***, the request will be **synchronous**.

Synchronous calls are used extremely rarely, as they block the interaction with the page until the download is completed.

```
// Synchronous request
ajaxRequest.open('GET', ‘data.json', false);

// Send him
ajaxRequest.send();
// ... all JavaScript will "hang" until the request is completed
```

To make the request **asynchronous**, we set the ***async*** parameter to ***true***.

! Synchronous XMLHttpRequest (async = false) is not recommended because the JavaScript will stop executing until the server response is ready.

AN EXAMPLE OF USING XMLHTTPREQUEST

```
// 1. Create a new object XMLHttpRequest
let ajaxRequest = new XMLHttpRequest();

// 2. Configuring it: GET request to URL 'data.json'
ajaxRequest.open('GET', ‘data.json', true);     // asynchronous requests

// 3. If the server response code is not 200, then this is an error
if (ajaxRequest.status != 200) {
  // handle error
  alert(ajaxRequest.status + ': ' + ajaxRequest.statusText); // sample output:
                                                                 404: Not Found
} else {
  // display the result
  alert(ajaxRequest.responseText); // responseText -- response text
}
// 4. Sending request
ajaxRequest.send();
```

softserve

FULL EXAMPLE: AJAX WITH HTML & JAVASCRIPT

```
<!DOCTYPE HTML>
<html>
  <head>
    <meta charset="utf-8">
  </head>
  <body>
    <button onclick="downloadBooks()">Download books</button>
    <script>
      function downloadBooks() {
        const ajaxRequest = new XMLHttpRequest();
       ajaxRequest.open('GET', 'books.json', true);
       ajaxRequest.onreadystatechange = function() {          // for asynchronous requests
          if (ajaxRequest.readyState != 4) return;             // for asynchronous requests
          if (ajaxRequest.status != 200) {
             alert(‘Error ' + ajaxRequest.status + ': ' + ajaxRequest.statusText);
          } else { alert(ajaxRequest.responseText); }
        }
       ajaxRequest.send();
      }
    </script>
  </body>
</html>
```

softserve

SERVER ON NODEJS. SETTING UP THE ENVIRONMENT

1. First install the Node.JS server itself (http://nodejs.org)
2. Select the directory where you will solve problems. Run in it:

```
npm install node-static
```

This will install the *node-static* module in the current directory, which will become automatically available for scripts from subdirectories.

Node-static simple flexible module for reading and writing files for Node. Node-static understands and supports conditional GET and HEAD requests.

! If you have Windows and the command did not work, then most likely the fact is that new paths “jumped”. Restart the file manager or console.

softserve

SERVER ON NODEJS. VERIFICATION

```
const http = require('http');
var url = require('url');
var static = require('node-static');
var file = new static.Server('.');

function accept(req, res) {
     if (req.url == '/files_on_server.json') {
        file.serve(req, res);              // can set delay
     } else {
        file.serve(req, res);
     }
   }
// ------ run server -------
http.createServer(accept).listen(8080);
console.log("Server running on port 8080");
```

softserve

SAME ORIGIN POLICY & AJAX

The **same-origin policy** is a critical security mechanism that restricts how a document or script loaded from one origin can interact with a resource from another origin.

For security reasons, browsers do not allow you to make cross-domain AJAX requests. This means you can only make AJAX requests to URLs from the same domain as the original page, for example, if your application is running on the domain "mysite.com", you cannot make AJAX request to "othersite.com" or any other domain.

To overcome this restriction it is possible to use **JSONP** and **CORS.**

softserve

Fetch API

softserve

Fetch API

- The Fetch API provides a JavaScript interface for working with HTTP requests and responses. It also provides a global **fetch()** method that makes it easy and logical to fetch resources over the network asynchronously.
- The main difference from XMLHttpRequest is that the Fetch API uses **Promises**, which allows for a simpler and cleaner API,

```
const result = fetch(url, [options])
```

- url - a string containing the URL to which you want to send a request.
- options - not required. An object that contains additional query parameters:
  - method - HTTP-method: "GET", "HEAD", "POST".
  - headers - the object of the request header.
  - body - the body of the request.
  - mode - cross-domain mode: "same-origin", "no-cors", "cors".
  - credentials - whether to send cookies: "same-origin", "include", "omit".
  - cache - caching mode: "default", "no-store", "reload", "no-cache", "force-cache", "only-if-cached"
  - redirect - redirect mode: "follow" or "error".

softserve

Fetch API

- Without options, it's a simple GET request that downloads content to a url.

```
fetch('https://api.github.com/users/chriscoyier/repos')
  .then(response => response.json())
  .then(data => {
    // list of repositories
    console.log(data);
  });
```

- We can of course use async / await

```
const url = 'https://api.github.com/users/chriscoyier/repos';
const response = await fetch(url);

const data = await response.json(); // read the response in JSON format

console.log(data);
```

softserve

Fetch API.
Response properties and methods

Response parameters:

- response.status - HTTP response code,
- response.ok - true if the response status is in the range 200-299.
- response.headers is a Map-like object with HTTP headers.

Methods for getting the response body:

- response.text() - returns the response as plain text,
- response.json() - converts the response to a JSON object,
- response.formData() - returns the response as a FormData object
- response.blob() - returns an object as Blob (binary data with type),
- response.arrayBuffer() - returns response as ArrayBuffer (low-level binary data),

Fetch API. POST request

- In practice, a POST request looks like this:

```
const data = {
    title: 'name',
    body: 'content',
    userId: 1,
  }

fetch('https://jsonplaceholder.typicode.com/posts', {
  method: 'POST',
  body: JSON.stringify(data),
  headers: {
    'Content-type': 'application/json; charset=UTF-8',
  },
})
  .then((response) => response.json())
  .then((json) => console.log(json));
```

softserve

Useful links

https://developer.mozilla.org/en-US/docs/Web/HTTP/Overview

https://www.tutorialspoint.com/ajax/index.htm

http://learn.javascript.ru/ajax

https://www.w3schools.com/js/js_ajax_intro.asp

https://javascript.info/fetch

softserve

THANKS

softserve