Cereri AJAX cross-domain

Crearea de domenii virtuale multiple în aceeaşi instalare Apache (XAMPP)

Vom crea 2 servere virtuale în XAMPP[1] în Windows cu adrese de domeniu fictive: **site1.com** şi **site2.com**. Urmaţi paşii:

Pas1. Creaţi în htdocs câte un folder corespunzător fiecăruia dintre cele două site-uri: site1 şi site2

Pas2. Creaţi în fiecare din aceste foldere câte un script de test (cu numele script1.php şi script2.php), care să confirme vizitatorului printr-un mesaj simplu pe care site se află:

```
<?php
print "acesta e site-ul x";
?>
```

(x=1 sau 2 în funcție de folder)

Pas3. Asiguraţi-vă că toate componentele XAMPP-ului sunt oprite. Intraţi în folderul de instalare XAMPP până la fișierul **httpd-vhosts.conf** - la o instalare normală îl veţi găsi în folderul C:\xampp\apache\conf\extra

Pas4. În interiorul acestui fişier putem defini multiple adrese de domeniu fictive ("virtual hosts"). În mod implicit, tot conţinutul fişierului e dezactivat prin comentarii. Activaţi linia care începe cu **NameVirtualHost** şi adăugaţi la finalul fișierului domeniile fictive dorite. Fişierul final ar trebui să arate astfel (am înlocuit parte din liniile comentate cu puncte de suspensie):

```
# Virtual Hosts
#.............
# Use name-based virtual hosting.
#
NameVirtualHost *:80
#
# VirtualHost example:
# .....
##</VirtualHost>

<VirtualHost *:80>
   ServerAdmin webmaster@localhost
   DocumentRoot "C:/xampp/htdocs"
   ServerName localhost

        <Directory "C:/xampp/htdocs">
                Options Indexes FollowSymLinks Includes ExecCGI
                AllowOverride All
                Order allow,deny
                Allow from all
        </Directory>
</VirtualHost>

<VirtualHost *:80>
   ServerAdmin webmaster@site1.com
```

[1] Cine folosește alt mediu decât XAMPP va trebui să caute setări similare pentru propriul mediu, de exemplu pentru WAMPServer:

- https://www.youtube.com/watch?v=GQHmzCoqEHw
- https://www.youtube.com/watch?v=h-itXgXFli0

```
   DocumentRoot "C:/xampp/htdocs/site1"
   ServerName site1.com
   ErrorLog "logs/site1.com-error.log"
   CustomLog "logs/site1.com-access.log" common

        <Directory "C:/xampp/htdocs/site1">
                Options Indexes FollowSymLinks Includes ExecCGI
                AllowOverride All
                Order allow,deny
                Allow from all
        </Directory>
</VirtualHost>

<VirtualHost *:80>
   ServerAdmin webmaster@site2.com
   DocumentRoot "C:/xampp/htdocs/site2"
   ServerName site2.com
   ErrorLog "logs/site2.com-error.log"
   CustomLog "logs/site2.com-access.log" common

        <Directory "C:/xampp/htdocs/site2">
                Options Indexes FollowSymLinks Includes ExecCGI
                AllowOverride All
                Order allow,deny
                Allow from all
        </Directory>
</VirtualHost>
```

Observaţii:

- Fiecare domeniu fictiv se creează cu un tag <VirtualHost> ce conţine setările domeniului. Setările cheie sunt **DocumentRoot** (folderul rezervat site-ului fictiv), **ServerName** (adresa fictivă ce se va folosi în loc de localhost), **Directory** (din nou folderul, de data asta cu diverse drepturi și capabilități, le lăsăm pe cele implicite)
- În acest exemplu am definit 3 domenii:
    - **localhost** (cel implicit, cu fişierele păstrate în htdocs, să nu pierdem totuși posibilitatea lucrului cu localhost pentru alte exemple, alte proiecte),
    - **site1.com** (va folosi fişiere din htdocs/site1),
    - **site2.com** (va folosi fişiere din htdocs/site2)

Pas5. Aceste site-uri fictive trebuie definite şi la nivelul sistemului de operare, într-un fişier numit **hosts** - poate fi găsit în C:\Windows\System32\drivers\etc

Editarea fişierului necesită drepturi de administrator. De exemplu dacă îl editaţi cu Notepad++ porniţi mai întâi Notepad++ cu click dreapta - *Run as administrator* şi deschideţi fişierul din editor. Adăugaţi la finalul fişierului câte o mapare între adresa IP pentru localhost (127.0.0.1) şi cele două adrese de domeniu fictive:

```
# Copyright (c) 1993-2009 Microsoft Corp.
#
# This is a sample HOSTS file used by Microsoft TCP/IP for Windows.
#.....................................................
# localhost name resolution is handled within DNS itself.
#       127.0.0.1       localhost
#       ::1             localhost
127.0.0.1       site1.com
127.0.0.1       site2.com
```

Pas6. Porniţi Apache-ul din XAMPP. Dacă scripturile de la Pas2 au fost corect salvate, ar trebui să le puteţi accesa prin adresele:

http://site1.com/script1.php
http://site2.com/script2.php

În plus, și adresa http://localhost ar trebui să funcţioneze în continuare normal, permiţând acces la restul fişierelor din htdocs (alte eventuale proiecte care mai sunt acolo).

Verificarea constrângerii cross-origin în browsere

Începem cu un exemplu normal, cu componentele front-end și back-end în același site (folderul site1):

Componenta back-end returnează JSON (datejson.php în folderul site1):

```
<?php
header("Content-type:application/json");

$Produs1=["ID"=>"P1","Pret"=>100,"Denumire"=>"Televizor"];
$Produs2=["ID"=>"P2","Pret"=>30,"Denumire"=>"Ipod"];
$Produse=[$Produs1,$Produs2];
$DateRaspuns=["Comanda"=>["Client"=>"Poplon","Produse"=>$Produse]];

print json_encode($DateRaspuns);
?>
```

Componenta front-end va afișa denumirea primului produs (la apăsarea unui buton):

```
<html>
<head>
<script>
function solicitare()
        {
        cerere=new XMLHttpRequest()
        cerere.onreadystatechange=procesareRaspuns
        cerere.open("GET","datejson.php")
        cerere.send(null)
        }
function procesareRaspuns()
        {
        if (cerere.readyState==4)
                if (cerere.status==200)
                {
                raspuns=JSON.parse(cerere.responseText)
                produs=raspuns.Comanda.Produse[0].Denumire
// în loc de linia de deasupra, puteţi extrage date JSON şi cu sintaxa vectorială
// produs=raspuns["Comanda"]["Produse"][0]["Denumire"]
                tinta=document.getElementById("tinta")
                tinta.innerHTML+=produs
                }
        }
</script>
</head>
<body>
<input type="button" onclick="solicitare()" value="Declanseaza cerere"/>
<div id="tinta"></div>
</body>
</html>
```

După ce ne-am asigurat că exemplul funcționează, îl testăm și în condiții cross-domain:

- mutaţi componenta back-end (datejson.php) în folderul serverului fictiv site2 (verificați să fie vizibil în browser prin site2.com/datejson.php)
- în componenta front-end, redirecționați cererea spre noua locație a componentei back-end:

```
cerere.open("GET","http://site2.com/datejson.php")
```

Testaţi din nou pagina şi veţi remarca faptul că nu mai funcţionează (consola browserului va indica o eroare *cross-origin policy*). Un blocaj similar am întâlni și prin celelalte mecanisme de trimitere a cererii (JQuery, fetch)

În continuare vom depăși această limitare prin tehnici consacrate: **JSON-P, Proxy, CORS**.

Tehnica JSON-P

JSON-P în JavaScript nativ

Observați în front-endul de mai jos (salvat în site1) setarea dinamică a atributului SRC spre destinația PHP de pe site 2.

```
<html>
<head>
<script type="application/javascript" id="bloc"></script>
<script>

function cerere()
        {
        blocJS=document.getElementById("bloc")
        blocJS.src="http://site2.com/datejson.php"
        }

function procesareRaspuns(date)
        {
        produs=date.Comanda.Produse[0].Denumire
        tinta=document.getElementById("tinta")
        tinta.innerHTML+=produs
        }
</script>

</head>

<body>
        <input type="button" onclick="cerere()" value="Declanseaza cerere"/>
        <div id="tinta"></div>
</body>
</html>
```

Există însă o dilemă: *cum executăm funcția procesareRaspuns (cum o conectăm la cerere)?* Cel mai convenabil e ca **serverul să returneze chiar el linia de cod JS care să declanșeze, la sosire în client, funcția de procesare a răspunsului**:

```
<?php
header("Content-type:application/javascript");

$Produs1=["ID"=>"P1","Pret"=>100,"Denumire"=>"Televizor"];
$Produs2=["ID"=>"P2","Pret"=>30,"Denumire"=>"Ipod"];
$Produse=[$Produs1,$Produs2];
$DateRaspuns=["Comanda"=>["Client"=>"PopIon","Produse"=>$Produse]];

print "procesareRaspuns(".json_encode($DateRaspuns).")";
?>
```

Mai rămâne o dilemă: *cum știe back-endul (de pe site2) cum se numește funcția pe care front-endul și-a pregătit-o (pe site1) pentru a primi datele sosite*?

Cel mai simplu mod prin care site1 poate înștiința site2 asupra acelui detaliu e **să atașeze un parametru QueryString la adresa din SRC**. Pentru ca acest mecanism să poată fi generalizat, s-a stabilit convenția ca acel parametru să se numească callback[2], iar valoarea sa să fie numele funcției pe care site1 îl comunică lui site2 (și așteaptă să o primească înapoi cu argumente pentru execuție).

[2] Alternativ, e posibil ca serverul să ofere o documentație prin care să indice cum preferă să se numească acest parametru, dar s-a consacrat convenția ca el să se numească *callback*

În exemplul nostru asta aduce două modificări:

1. în front-end, linia care setează adresa din SRC va atașa și numele funcției, folosind convenția amintită (opțional poate adăuga și alți parametri cu date de trimis spre server):

```
blocJS.src="http://site2.com/datejson.php?callback=procesareRaspuns"
```

2. în back-end, ultima linie se va înlocui cu preluarea numelui funcției din *$_GET["callback"]*:

```
$NumeFunctieClient=$_GET["callback"];
print $NumeFunctieClient."(".json_encode($DateRaspuns).")";
```

Tehnica JSON-P suferă de neajunsul evidente că cererea trimisă prin SRC poate fi doar de tip GET, cu toate limitările care survin din asta.

JSON-P în JQuery

JQuery oferă un suport subtil pentru tehnica JSON-P. Activarea tehnicii JSON-P se poate face în JQuery în două moduri:

1. Varianta **getJSON + folosirea unui callback anonim**:

```
<html>
<html>
<head>
<script src="http://ajax.googleapis.com/ajax/libs/jquery/2.2.4/jquery.min.js"></script>
<script>

function cerere()
	{
	adresa="http://site2.com/datejson.php?callback=?"
	$.getJSON(adresa, function(date)
					{
					produs=date.Comanda.Produse[0].Denumire
					continutDeInserat="<div>"+produs+"</div>"
					$(continutDeInserat).appendTo(document.body)
					}
		)
	}

</script>
</head>

<body>
	<input type="button" onclick="cerere()" value="Declanseaza cerere"/>
</body>
</html>
```

2. Varianta **get + indicarea "jsonp" ca format așteptat**:

```
<html>
<head>
<script src="http://ajax.googleapis.com/ajax/libs/jquery/2.2.4/jquery.min.js"></script>
<script>

function trimiteCerere()
```

```
   {
   adresa="http://site2.com/datejson.php"
   $.get(adresa,procesareRaspuns,"jsonp")
   }

function procesareRaspuns(raspuns)
   {
   produs=raspuns.Comanda.Produse[0].Denumire
   continutDeInserat="<div>"+produs+"</div>"
   $(continutDeInserat).appendTo(document.body)
   }

</script>
</head>

<body>
   <input type="button" onclick="trimiteCerere()" value="Declanseaza cerere"/>
</body>
</html>
```

O mulţime de servicii API pun la dispoziţie tehnica JSON-P. Un exemplu de astfel de serviciu este geoNames, ce oferă acces la informaţii geografice:

```
function cerere()
        {
        adresa="http://www.geonames.org/postalCodeLookupJSON?postalcode=10504&country=US&callback=?"
        $.getJSON(adresa, function(date)
                                {
                                locatie=date.postalcodes[0].placeName
                                continutDeInserat="<div>"+locatie+"</div>"
                                $(continutDeInserat).appendTo(document.body)
                                }
                )
        }
```

Funcţia returnează numele locaţiei corespunzătoare unui cod poştal (10504) şi unei ţări (US) care se trimit ca parametri GET. Observaţi şi parametrul callback=? necesar tehnicii JSON-P în JQuery.

Mai multe detalii despre serviciile puse la dispoziţie de geoNames, inclusiv adresele şi parametri cu care acestea se pot apela găsiţi la:

http://www.geonames.org/export/web-services.html

Tehnica Proxy

Instrumente necesare pentru a trimite Cereri HTTP din PHP

Cum trimitem **cereri HTTP din PHP** (back-end la back-end)?

- Există câteva metode simple de a solicita conținut de la o adresă URL oarecare – funcțiile **fopen()** și **file_get_contents()**; acestea pot fi blocate de anumite setări de server (allow_url_fopen) și multă vreme au permis doar acces de tip GET; mai recent se pot folosi și pentru POST, PUT etc. însă în combinație cu programarea de fluxuri (*streams*), care nu face obiectul acestui curs (vom insista pe acele tehnici care seamănă cu ce am arătat deja în JavaScript);
- Există o extensie PHP numită **cURL**, disponibilă în instalarea de bază PHP, însă cu o sintaxă anevoioasă (vom oferi câteva exemple totuși);
- Există numeroase **librării externe** (trebuie instalate cu Composer) care încearcă să aducă modul de lucru cu cereri HTTP în PHP cât mai aproape de stilul consacrat de JavaScript (JQuery, Axios etc.); ne vom axa pe acestea, cu exemple folosind librăriile **GuzzleHttp** (în acest tutorial), iar mai târziu **EasyRdfHttpClient** (parte din librăria EasyRdf pentru grafuri, dar se poate folosi și pentru cereri HTTP oarecare)[3].

CURL este inclus în PHP din pachetul XAMPP[4]. Vom mai instala GuzzleHttp pentru a compara cele două instrumente:

Pentru instalare avem nevoie de programul **Composer**, care se ocupă de descărcarea şi instalarea a diverse extensii PHP (Composer este pentru PHP ceea ce este NPM pentru Node.js, sau Pip pentru Python).

Pas1. Căutaţi fişierul php.ini din XAMPP (la o instalare normală ar fi în C:\xampp\php). În acest fişier asiguraţi-vă că linia *extension=php_openssl.dll* nu e dezactivată prin comentariu (dacă e, ştergeţi semnul ; din faţa ei)

Pas2. Descărcaţi şi executaţi Composer-Setup.exe de la adresa de mai jos:

https://getcomposer.org/download/
(în timpul instalării veţi fi întrebaţi unde e instalat php, selectaţi calea la care a fost instalat de XAMPP)

Pas3. Folosiţi linia de comandă Windows pentru a naviga în site1:

```
cd c:\xampp\htdocs\site1
```

Aflaţi fiind în folderul site1, unde vom crea exemplul, instalaţi librăria GuzzleHttp cu comanda:

```
composer require guzzlehttp/guzzle
```

[3] Alte librării cu rol similar pot fi consultate la https://ourcodeworld.com/articles/read/674/top-7-best-php-http-client-libraries
[4] E disponibil și va program executabil pentru trimitere de cereri din linia de comandă:https://curl.haxx.se/

Documentaţia oficială GuzzleHttp poate fi consultată la adresa:

http://docs.guzzlephp.org/en/latest/

Cereri GET prin CURL şi GuzzleHTTP

Presupunem că avem în site2 același script ce returnează JSON, cu numele datejson.php:

```
<?php
header("Content-type:application/json");

$Produs1=["ID"=>"P1","Pret"=>100,"Denumire"=>"Televizor"];
$Produs2=["ID"=>"P2","Pret"=>30,"Denumire"=>"Ipod"];
$Produse=[$Produs1,$Produs2];
$DateRaspuns=["Comanda"=>["Client"=>"Poplon","Produse"=>$Produse]];

print json_encode($DateRaspuns);
?>
```

Construim scriptul proxy în site1 cu numele scriptproxy.php.

Versiunea CURL:

```
<?php
header("Content-type:application/json");

$cerere=curl_init();
$configurari=[CURLOPT_RETURNTRANSFER=>1,CURLOPT_URL=>"http://site2.com/datejson.php"];
curl_setopt_array($cerere,$configurari);
$rezultat=curl_exec($cerere);

print $rezultat;
curl_close($cerere);
?>
```

Versiunea GuzzleHttp:

```
<?php
require "vendor/autoload.php";
header("Content-type:application/json");

$client=new \GuzzleHttp\Client();
$cerere=$client->getAsync("http://site2.com/datejson.php");
$cerere->then("procesareRaspuns")->wait();

function procesareRaspuns($raspuns)
   {
   print $raspuns->getBody();
   }
?>
```

Mai departe construim pagina client care solicită datele de la scriptul proxy, din același folder (site1). Pagina nu va cunoaște sursa reală a datelor, căci comunică strict cu propriul back-end (scriptpoxy.php) fără a lansa vreodată cereri cross-domain:

```
<html>
<head>
<script src="http://ajax.googleapis.com/ajax/libs/jquery/2.2.4/jquery.min.js"></script>
<script>

function trimiteCerere()
```

```
		{
		$.get("scriptproxy.php", procesareRaspuns)
		}

function procesareRaspuns(raspuns)
		{
		produs=raspuns.Comanda.Produse[0].Denumire
		continutDeInserat="<div>"+produs+"</div>"
		$(continutDeInserat).appendTo(document.body)
		}

</script>
</head>

<body>
		<input type="button" onclick="trimiteCerere()" value="Declanseaza cerere"/>
</body>
</html>
```

Cereri POST prin CURL și GuzzleHTTP

În continuare refacem exemplul într-o nouă variantă: componentele back-end nu vor mai returna întreaga structură JSON, ci doar denumirea solicitată, pe baza ID-ului produsului dorit. ID-ul va trebui împins din front-end până la site2 – am putea face asta prin GET (alipindu-l la adresă) dar ne folosim de ocazie pentru a exemplifica și cereri POST, precum și transferul de date în format JSON (chiar dacă trimitem doar un ID).

Mai întâi construim pagina client, similar cazului precedent dar cu identificatorul P1 acum trimis în format JSON printr-o cerere de tip POST. Deoarece trimiterea de JSON trebuie declarată în antetul Content-Type, nu folosim $.post ci lucrăm cu $.ajax pentru o libertate mai mare de configurare a cererii ($.post trimite implicit în format QueryString). Configurările acoperă și setările de bază (adresa, tipul cererii, funcția rezervată) grupate toate într-un obiect cu câmpuri predefinite:

```
<html>
<head>
<script src="http://ajax.googleapis.com/ajax/libs/jquery/2.2.4/jquery.min.js"></script>
<script>

function trimiteCerere()
		{
		deTrimis=JSON.stringify({identificator:"P1"})
		configurari={url:"scriptproxy.php",
				type:"POST",
				data:deTrimis,
				contentType:"application/json",
				success:procesareRaspuns}
		$.ajax(configurari)
		}

function procesareRaspuns(raspuns)
		{
		continutDeInserat="<div>"+raspuns+"</div>"
		$(continutDeInserat).appendTo(document.body)
		}

</script>
</head>

<body>
		<input type="button" onclick="trimiteCerere()" value="Declanseaza cerere"/>
```

```
</body>
</html>
```

Mai departe, modificăm scriptul proxy (scriptproxy.php în site1) pentru a realiza următoarele:

- *preia datele JSON de la front-end și le forwardează fără a le deserializa* (în practică e posibil ca un script proxy să aplice și validări/filtrări suplimentare, să combine date din mai multe surse)
- *forwardează antetul Content-Type în mod similar*

Versiunea GuzzleHttp:

- recepționarea JSON se face prin citirea fluxului de intrare din **php://input**, nu din $_POST (care conține doar ce sosește în formate default aplicate de formulare HTML)[5]
- recepționarea declarației Content-Type se face din $_SERVER["CONTENT_TYPE"]
- pentru a atașa date la cererea Guzzle, datele se includ într-un array asociativ (**configurari**) sub eticheta **"body"** (alternativ, se pot folosi etichetele **"json"** sau **"forms_param"** pentru a aplica serializări în format JSON, respectiv QueryString, dar în acest caz le forwardăm așa cum au sosit, ele fiind deja în format JSON)
- pentru a seta antete la cererea Guzzle, se folosește în același array asociativ eticheta **"headers"**; ca și în cazul datelor, doar preluăm declarația din cererea sosită și o punem în cea care pleacă
- funcția se numește, evident, **postAsync**, în rest e aceeași logică; răspunsul primit de la site2 va fi împins spre front-end cu un simplu print/echo:

```
<?php
require "vendor/autoload.php";
$client=new \GuzzleHttp\Client();

$JSONsosit=file_get_contents("php://input");
$FormatSosit=$_SERVER["CONTENT_TYPE"];

$configurari=["body"=>$JSONsosit,"headers"=>["Content-Type"=>$FormatSosit]];

$cerere=$client->postAsync("http://site2.com/8datejsonpost.php",$configurari);
$cerere->then("procesareRaspuns")->wait();

function procesareRaspuns($raspuns)
        {
        print $raspuns->getBody();
        }
?>
```

Versiunea CURL: aplicăm același principiu de a forwarda ID-ul și antetul Content-Type fără a ne uita măcar ce valori au:

```
<?php

$JSONsosit=file_get_contents("php://input");
$FormatSosit=$_SERVER["CONTENT_TYPE"];

$cerere=curl_init();

$configurari=[CURLOPT_RETURNTRANSFER=>1,
        CURLOPT_URL=>"http://site2.com/datejsonpost.php",
        CURLOPT_POST=>1,
```

[5] Teoretic putem alipi un pachet JSON la un parametru QueryString, dar ar fi o dublă serializare fără sens

```
        CURLOPT_HTTPHEADER=>["Content-type"=>$FormatSosit],
        CURLOPT_POSTFIELDS=>$JSONsosit];

curl_setopt_array($cerere,$configurari);
$rezultat=curl_exec($cerere);

print $rezultat;
curl_close($cerere);
?>
```

Observați configurările cererii, unde am activat metoda POST (cu CURLOPT_POST), am specificat datele trimise (CURLOPT_POSTFIELDS), formatul trimis (CURLOPT_POST). Scriptul proxy e redus la rolul de a forwarda datele în ambele sensuri (ID-ul spre site 2, prin CURL, iar rezultatul spre front-end cu print).

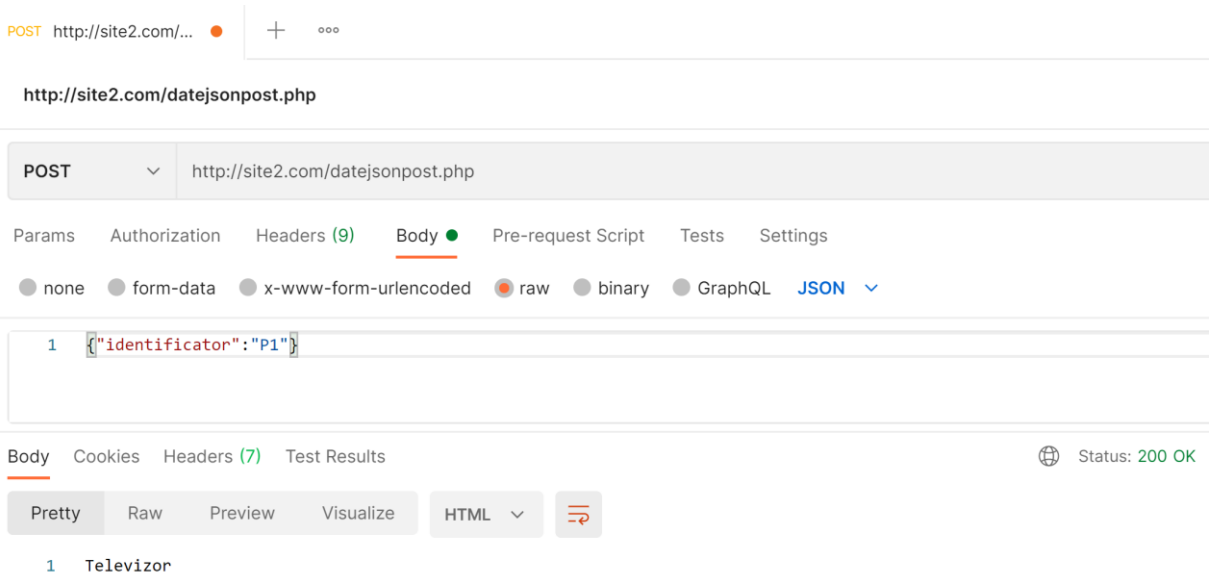
Pe site 2, scriptul back-end (datejsonpost.php) trebuie modificat pentru a prelua ID-ul din pachetul JSON și a face selecția denumirii produsului corespunzător. Evităm o abordare clasică cu căutări FOR/IF, folosindu-ne de facilități oferite de PHP pentru căutare în arrayuri (**array_search, array_column**, vezi comentariile):

```
<?php
$Produs1=["ID"=>"P1","Pret"=>100,"Denumire"=>"Televizor"];
$Produs2=["ID"=>"P2","Pret"=>30,"Denumire"=>"Ipod"];
$Produse=[$Produs1,$Produs2];
//presupunem ca avem datele in acest array

$JSONsosit=file_get_contents("php://input");
$dateSosite=json_decode($JSONsosit);
$IDsosit=$dateSosite->identificator;
//am receptionat ID-ul forwardat de proxy, cu deserializare din JSON

$pozitie=array_search($IDsosit, array_column($Produse,"ID"));
//array_column extrage campul ID din toate produsele și construiește un array numerotat din ele
//array_search cauta ID-ul in arrayul numerotat si returneaza pozitia sa

$denumire=$Produse[$pozitie]["Denumire"];

print $denumire;
?>
```

În sfârşit, reţineţi şi posibilitatea de a TESTA cereri HTTP cu ajutorul unor programe precum Postman. Adesea componentele back-end se creează înaintea celor front-end și buna lor funcționare trebuie testată înainte de a implementa cererile pe partea de client.

Postman se obţine gratuit de la https://www.getpostman.com/ şi permite să executăm cereri HTTP într-o interfaţă grafică. În imaginea de mai jos am construit:

- cerere POST
- spre site2.com/datejsonpost.php
- în secţiunea Body am indicat parametrul **identificator** cu valoarea **P1**
- tot acolo am setat formatul **raw**, apoi (în lista derulantă) **JSON**
- apăsarea butonului **Send** execută cererea, iar dedesubt se vede răspunsul.


POST http://site2.com/...
http://site2.com/datejsonpost.php
POST
http://site2.com/datejsonpost.php
Params
Authorization
Headers (9)
Body
Pre-request Script
Tests
Settings
none
form-data
x-www-form-urlencoded
raw
binary
GraphQL
JSON
1 {"identificator":"P1"}
Body
Cookies
Headers (7)
Test Results
Status: 200 OK
Pretty
Raw
Preview
Visualize
HTML
1 Televizor

Tehnica CORS

CORS presupune anumite configurări (*CORS policy*) pe serverul străin (site2) prin care acesta:

- declară permisiunea sau interdicția de a primi cereri de la browsere, din pagini de pe alte siteuri; în mod implicit e aplicată interdicția, permisiunea trebuie inclusă sub forma unor declarații de antet HTTP;
- restricționează browserul cu privire la ce are voie să trimită (ce tip de cereri are voie să trimită, de pe ce site, cu ce antete etc.); acest lucru e realizat cu ajutorul unor antete HTTP speciale ce vor determina browserul să blocheze trimiterea cererilor necorespunzătoare;
- deoarece browserul e cel care aplică blocarea, serverul trebuie la rândul său să verifice dacă acele restricții au fost respectate, cu ajutorul unor teste IF asupra cererilor preconizate; deși browserul nu ar trebui să lase să plece o cerere ce nu respectă restricțiile, se recomandă verificări asupra cererii și de către server (de ex. pentru a preveni cereri venite din alte surse decât browserul, vezi Postman, ori pentru a rafina restricțiile declarate inițial)[6].

Configurările CORS pot fi realizate în două moduri:

- *la nivel de server*, dacă dorim ca toate scripturile executate pe un server să se conformeze aceloraşi reguli, vezi https://enable-cors.org/server.html;
- individualizate *la nivel de script*, pe care le vom exemplifica în continuare.

Pe partea front-end, o cerere CORS nu diferă cu nimic de una obișnuită (doar browserul trebuie să fie o versiune recentă pentru a permite negocierea CORS). Exemplul de mai jos trimite un array în format JSON spre site2 printr-o cerere POST. Deoarece avem de anunțat și formatul în Content-Type, folosim funcția generală $.ajax:

```
<html>
<head>
<script src="http://ajax.googleapis.com/ajax/libs/jquery/2.2.4/jquery.min.js"></script>
<script>

function trimite()
        {
        studenti=[{"Nume":"Ana","Nota":10},{"Nume":"Petru","Nota":4}]
        datejson=JSON.stringify(studenti)
        configurari={url:"http://site2.com/scriptcors.php",
                type:"POST",
                data:datejson,
                contentType:"application/json",
                success:procesareRaspuns}
        $.ajax(configurari)
        }

function procesareRaspuns(raspuns)
        {
        $("#tinta").append(raspuns+"<br/>")
        }

</script>
</head>
<body>
<input type="button" onclick="trimite()" value="Declanseaza cerere"/>
```

[6] Blocarea de către browser are rolul să protejeze pachetele cookie, verificările pe server au rolul să protejeze resursele serverului, deci ambele părți colaborează la implementarea politicii CORS

```
<div id="tinta"></div>
</body>
</html>
```

Pentru tehnica CORS, cererile HTTP se împart în două categorii:

- **cereri cu risc scăzut de securitate** pentru server (site2); în această categorie intră cereri care s-ar putea trimite și de către un formular HTML (GET sau POST, cu datele în format QueryString/FormData); ca tip de cereri acestea sunt implicit lăsate să treacă de browser, dar ca origine sunt implicit respinse de un server ce nu le recunoaște originea;
- **cereri cu risc ridicat de securitate**, numite și *cereri pre-flighted*; aici intră orice alte cereri (altele decât GET/POST sau acele cereri POST care trimit formate non-default – JSON, XML etc.)

Cererile HTTP (pre-flighted) cu risc vor fi executate în 2 etape:

- înainte să execute cererea efectivă, **browserul va trimite o pre-cerere**, cu care verifică dacă serverul e compatibil CORS și ce restricții impune
    - pre-cererea e tot o cerere HTTP dar de tip OPTIONS (folosită pentru a solicita informări cu privire la setările serverului)
    - pre-cererea nu trebuie programată de noi, ea e trimisă automat de browsere (recente) ca pas premergător execuției unei cereri din JavaScript[7], practic e un mod de a întreba serverul dacă cererea pe care am programat-o e acceptabilă
- serverul trebuie să se aștepte la sosirea unei cereri OPTIONS și **să răspundă la ea cu o serie de antete HTTP din familia Access-Control** prin care va informa browserul privind:
    - de la cine acceptă cereri (*Access-Control-Allow-Origin*)
    - ce tipuri de cereri acceptă (*Access-Control-Allow-Methods*)
    - ce antete HTTP acceptă (*Access-Control-Allow-Headers*)
    - cât timp să țină minte browserul aceste restricții (*Access-Control-Max-Age*), în caz că s-ar putea schimba în timp
    - eventual și altele, cu rol restrictiv similar[8]
- browserul verifică aceste antete HTTP și poate decide să blocheze cererea principală, afișând eroarea same-origin în consolă; dacă nu are motive să o blocheze, **va trimite cererea propriu-zisă**
- suplimentar, pe server ar trebui programate verificări asupra cererii principale, filtrări suplimentare etc.

Așadar pe server (site2), scriptcors.php va fi pregătit atât pentru pre-cererea implicită (de tip OPTIONS) cât și pentru cererea programată de noi – într-o combinație de instrucțiuni header() și verificări cu IF:

```
<?php
header("Access-Control-Allow-Origin: *");
// valoarea * permite cereri din orice sursă, asigurând o deblocare de principiu a barierei CORS ce va permite cereri de risc scăzut
// cererile de risc crescut mai sunt trecute prin filtrele următoare, ținând cont că ele sunt precedate de o pre-cerere de tip OPTIONS

if ($_SERVER["REQUEST_METHOD"]=="OPTIONS")
        {
        header("Access-Control-Allow-Methods: OPTIONS,POST,GET");
```

[7] Chrome nu o afișează în *Developer Tools-Network*, decât dacă o facem vizibilă cu setarea *chrome://flags/#out-of-blink-cors*

[8] https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS#the_http_response_headers

```
        header("Access-Control-Allow-Headers: Content-Type");
        header('Access-Control-Max-Age: 0');
        }
// până aici am construit răspunsul la pre-cerere, format exclusiv din antete Access-Control

// mai jos, receptăm cererea reală și verificăm respectarea restricțiilor
elseif ($_SERVER["REQUEST_METHOD"]=="POST")
        {
        if (($_SERVER["HTTP_ORIGIN"]=="http://site1.com")&&($_SERVER["CONTENT_TYPE"]=="application/json"))
                {
                $dateClient=file_get_contents("php://input");
                $datePHP=json_decode($dateClient);
                print "Serverul confirma primirea de note pentru ".$datePHP[0]->Nume." si ".$datePHP[1]->Nume; //mesaj1
                }
        elseif ($_SERVER["CONTENT_TYPE"]!="application/json")
                {
                print "Nu accept cereri care nu-mi trimit JSON!"; //mesaj2
                }
        elseif ($_SERVER["HTTP_ORIGIN"]!="http://site1.com")
                {
                print "Nu accept cereri decat de la site1.com!"; //mesaj3
                }
        }
else print "Nu accept cereri GET!"; //mesaj4
?>
```

Pas1. antetul *Access-Control-Allow-Origin* ar trebui setat primul, indicând sursele din care se acceptă cereri indiferent dacă serverul se așteaptă la cereri de risc scăzut sau crescut (pre-flighted).

Valoarea ***** ridică orice limitări de origine – setare periculoasă, însă scriptul poate interveni mai târziu să restricționeze suplimentar originea cu ajutorul unui IF;

- în practică, dacă acest back-end e un API va putea ține evidența unor clienți abonați, incluzându-i în acest antet cu o construcție de genul:

```
$sursaCererii = $_SERVER['HTTP_ORIGIN'];
//se verifică de unde a venit cererea

$surseAbonate = ['http://siteX.com','http://siteY.com'];
//se caută într-o bază de date clienții agreați

if (in_array($sursaCererii, $surseAbonate))
{
   header("Access-Control-Allow-Origin: $sursaCererii");
}
//se ridică restricția de origine dacă s-a găsit clientul în lista celor agreați
```

Pas2. prima parte a scriptului va pregăti un răspuns pentru pre-cererea OPTIONS: pe lângă restricția de origine, ar trebui să informeze browserul măcar cu privire la:

- *tipuri de cerere acceptate* (în mod normal ar trebui ca cererile GET, POST, OPTIONS să fie permise implicit[9], aici am lăsat o permisiune explicită dar ulterior cererea GET o vom respinge - prin verificări cu IF, permițând strict doar cererea POST pe care am programat-o)
- *antete acceptate* (Content-Type, dacă vrem să permitem alte formate decât QueryString)

[9] Primele două sunt folosite și în cererile de risc scăzut, unde dacă nu e restricționată sursa cererii vor fi lăsate să treacă

- *cât timp să țină minte browserul aceste limitări* (putem evita astfel repetarea pre-cererii, dar în teste dezactivăm opțiunea să ne asigurăm că protecția funcționează)

Pas3. apoi scriptul va verifica toate tipurile de cereri pe care le așteaptă și va genera răspunsuri conforme în caz de încălcare a regulilor; în cazul nostru, avem acele teste IF care verifică în $_SERVER diverse aspecte ale cererii:

- verificăm să primim cereri POST
- deși am lăsat inițial originea relaxată, aici impunem ca cererea să poată veni doar din site1.com
- verificăm dacă am primit date JSON

Putem testa aceste restricții pe rând:

- accesați scriptul back-end direct din browser prin simpla tastare a adresei sale; aceasta va fi considerată o cerere GET - va fi lăsată să treacă de browser[10] dar va fi respinsă de ultimul IF ducând la mesajul de eroare 4 (vezi comentariile numerotate)
- în scriptul front-end, setați tipul cererii la PUT în loc de POST și încercați să-l executați; browserul va bloca cererea cu eroare în consolă, văzând că PUT nu se află pe lista agreată de server în antetul *Access-Control-Allow-Methods*
- în scriptul front-end (cu POST setat la loc), setați *contentType: "text/plain"*, serverul va returna mesajul de eroare 2, pentru nerespectarea formatului
- accesați scriptul back-end din Postman, construind o cerere POST cu ceva date JSON; se va returna mesajul de eroare 3, căci originea cererii nu e site1.com (nici măcar nu e browserul, iar Postman nu aplică niciunul din blocajele solicitate de antetele Access-Control-...!)
- în scriptul back-end, comentați linia *Access-Control-Allow-Headers* iar în consecință browserul nu va permite să plece cererea POST cu opțiunea contentType setată din scriptul front-end

În fine, dacă front-endul e executat în forma lui inițială, va apărea mesajul 1, singurul care indică o funcționare normală. Iată așadar că mecanismul CORS permite scripturilor server să ridice blocajul cross-origin într-o manieră controlată:

- recomandând browserului ce să blocheze (prin antetele *Access-Control*),
- apoi aplicând propriile verificări asupra elementelor cererii (asupra $_SERVER pentru a preveni ocolirea blocajelor din browser, respectiv a cererilor pe care browserul le lasă implicit să treacă precum GET).

Acest exemplu a fost unul pentru cereri cu risc de securitate crescut.

Am amintit deja că pentru cererile de risc scăzut (GET sau POST cu date în format QueryString/default) nu se aplică regim la fel restrictiv:

- nu se mai trimite pre-cererea OPTIONS

[10] Ar fi lăsată să treacă și dacă n-am fi trecut-o în lista Access-Control-Allow-Methods, fiind un tip de cerere fără risc

- browserul e interesat doar de antetul *Access-Control-Allow-Origin*[11], pentru a bloca sau nu cererea în funcție de siteul/scriptul care a trimis-o și nu în funcție de tipul cererii

Așadar în varianta cea mai simplă, pentru a permite cererilor GET și POST clasice să treacă de bariera cross-domain, e suficient ca scriptul PHP de pe site 2 să înceapă cu:

```
header("Access-Control-Allow-Origin: *");
```

Și invers, e suficient ca acesta să lipsească astfel încât să nu putem trimite nici cea mai simplă cerere spre un alt site (de ex. să nu putem face Web scraping direct din JavaScript)

*(într-un caz real se vor folosi adrese de surse agreate, căci cu * anulăm practic rațiunea de securitate pentru care există bariera cross-domain; însă în exerciții e cel mai rapid mod de a depăși restricția fără alte verificări suplimentare)*

[11] Eventual și de *Access-Control-Allow-Credentials* dacă serverul condiționează cererile cu parolă