I - Appels de WS REST en javascript , CORS

1. Appels de WS REST (HTTP) depuis js/ajax

Avec ajax , ça va briller!

***AJAX** est l'acronyme d'**A**synchronous **J**avaScript **A**nd **X**ML, mais ça peut être utilisé avec Json .*

1.1. Cadre des appels

Lorsqu'une requête http est initiée depuis du code javascript s'exécutant dans le contexte d'une page html , on dit que l'on effectue un appel "ajax" .

Le sigle Ajax correspondant à peu près à "asynchronous javascript activation framework" indique :

- le déclenchement non bloquant d'une requête http

- l'enregistrement d'une fonction "callback" qui sera automatiquement appelée en différé lorsque la réponse reviendra

NB : l'appel non bloquant peut être considéré comme asynchrone mais le protocole HTTP est un protocole de transport synchrone (avec timeout si la réponse ne revient pas dans un délai raisonnable).

Techniquement, un appel ajax s'effectue en s'appuyant sur un objet technique "**X**ml**H**ttp**R**equest" fourni par tous les navigateurs pas trop anciens .

La partie Xml de XmlHttpRequest tient au fait qu'historiquement les premiers webServices normalisés (SOAP) étaient au format Xml. Bien que le terme "XmlHttpRequest" n'ai pas été changé pour des raisons de compatibilité ascendante du code javascript, il est possible de déclencher n'importe quelle requête HTTP depuis XHR , y compris des requêtes au format "JSON" .

Pour simplifier la syntaxe d'un appel ajax on peut éventuellement s'appuyer sur des librairies "javascript" complémentaires ("jquery" , "fetch" , "RxJs" , ...) .

Le principe des appels "ajax" sert essentiellement à déclencher une requête HTTP suite à un événement utilisateur en vue d'obtenir des données permettant de réactualiser une partie de la page HTML courante . La page courante n'est pas entièrement remplacée par une autre. Seule une sous partie de celle ci est ajustée par code js/DOM (Document Object Model).

Certaines applications , dites "SPA: Single Page Application" sont entièrement bâties sur ce principe . Au lieu de switcher de pages html on switch de sous pages (<div ...>....) .

1.2. XHR (XmlHttpRequest) dans tout navigateur récent

Exemple basique :

```
function makeAjaxRequest(callback) {
	var xhr = new XMLHttpRequest();
	xhr.onreadystatechange = function() {
		if (xhr.readyState == 4 && (xhr.status == 200 || xhr.status == 0)) {
			callback(xhr.responseText);
		}
	};
	xhr.open("GET", "handlingData.php", true);
	xhr.send(null);
}

function readData(sData) {
	if (sData!=null) {
		alert("good response");
		// + display data with DOM
	} else {
		alert("empty response");
	}
}
makeAjaxRequest(readData);
```

variations en mode "POST" :

```
xhr.open("POST", "handlingData.php", true);
xhr.setRequestHeader("Content-Type", "application/x-www-form-urlencoded");

xhr.send("param1=xx&param2=yy");
```

```
xhr.open("POST", "/json-handler");
xhr.setRequestHeader("Content-Type", "application/json");
xhr.send(JSON.stringify({prenom:"Jean", nom:"Bon"}));
```

Exemple plus élaboré :

my_ajax_util.js

```
//subfunction with errCallback as optional callback :
function registerCallbacks(xhr,callback,errCallback) {
	xhr.onreadystatechange = function() {
		if (xhr.readyState == 4){
			if((xhr.status == 200 || xhr.status == 0)) {
				callback(xhr.responseText);
			}
			else {
				if(errCallback)
				    errCallback(xhr.responseText);
			}
		}
	};
}

function makeAjaxGetRequest(url,callback,errCallback) {
	var xhr = new XMLHttpRequest();
	registerCallbacks(xhr,callback,errCallback);
	xhr.open("GET", url, true);    xhr.send(null);
}

function makeAjaxDeleteRequest(url,callback,errCallback) {
	var xhr = new XMLHttpRequest();
	registerCallbacks(xhr,callback,errCallback);
	xhr.open("DELETE", url, true);        xhr.send(null);
}

function makeAjaxPostRequest(url,jsonData,callback,errCallback) {
	var xhr = new XMLHttpRequest();
	registerCallbacks(xhr,callback,errCallback);
	xhr.open("POST", url, true);
	xhr.setRequestHeader("Content-Type", "application/json");
	xhr.send(jsonData);
}

function makeAjaxPutRequest(url,jsonData,callback,errCallback) {
	var xhr = new XMLHttpRequest();
	registerCallbacks(xhr,callback,errCallback);
	xhr.open("PUT", url, true);
	xhr.setRequestHeader("Content-Type", "application/json");
	xhr.send(jsonData);
}
```

appelAjax.js

```
var traiterReponse = function (response){
   //response ici au format "json string"
   var zoneResultat = document.getElementById("spanRes");
   var jsDevise = JSON.parse(response);
   zoneResultat.innerHTML=jsDevise.change; //ou .rate
}

function onSearchDevise(){
   var zoneSaisieCode =  document.getElementById("txtCodeDevise");
   var codeDevise = zoneSaisieCode.value;
   console.log("codeDevise="+codeDevise);
   var urlWsGet="./devise-api/public/devise/"+codeDevise;
   makeAjaxGetRequest(urlWsGet,traiterReponse); //non bloquant (asynchrone)
   //....
}
```

appelAjax.html

```
<html>
<head>
   <meta charset="UTF-8">
   <title>appelAjax</title>
   <script src="js/my_ajax_util.js"></script>
   <script src="js/appelAjax.js"></script>
</head>
<body>
   codeDevise :<input type="text" id="txtCodeDevise"/> <br/>
   <input type="button" id="btnSearch" onclick="onSearchDevise()"
         value="rechercher devise"/> <br/>
   Devise : <span id="spanRes"></span>
</body>
</html>
```

1.3. Appel ajax via jQuery

La syntaxe des appels "ajax" est un peu plus explicite en s'appuyant sur la bibliothèque "jquery".

Exemple :

```
<html>
<head>
        <meta charset="ISO-8859-1">
        <title>browse-spectacles</title>
        <script src="lib/jquery-3.3.1.min.js"></script>
        <script src="js/my-jq-ajax-util.js"></script>
<script>
$(function() {

 //appel ajax pour récupérer la liste des catégories et remplir le <select>
$.ajax({
          type: "GET",
         url: "spectacle-api/public/spectacle/allCategories",
         contentType : "application/json",
         success: function (data,status,xhr) {
          if (data) {
               var categoryList = data;
               for(categoryIndex in categoryList){
                  var category=categoryList[categoryIndex];
                 $('#selectCategory').append('<option value="'+ category.id +'">'+
                                  category.id + ' (' + category.title + ')</option>');
                           }
                 //$("#spanMsg").html(JSON.stringify(data));
                   }
                },
          error: function( jqXHR, textStatus, errorThrown ){
                    $("#spanMsg").html( xhrStatusToErrorMessage(jqXHR)  ) ;
                }
        });//end $.ajax
 });

</script>
</head>
<body>
  <h3> BROWSE Spectacles </h3>
  categorie : <select id="selectCategory"> </select><br/>
  ...   <span id="spanMsg"></span> <br/>...
</body>
</html>
```

fonctions utilitaires dans *js/my-jq-ajax-util.js*

```
function setSecurityTokenForAjax(){

        var authToken = sessionStorage.getItem("authToken");
                //localStorage.getItem("authToken");

        $(document).ajaxSend(function(e, xhr, options) {
```

```
                //retransmission du jeton d'authentification dans l'entête http de la requete ajax
                xhr.setRequestHeader('Authorization',"Bearer "+ authToken);
            });

}

function xhrStatusToErrorMessage(jqXHR){
        var errMsg = "ajax error";//by default
        var detailsMsg=""; //by default
        console.log("jqXHR.status="+jqXHR.status);
        switch(jqXHR.status){
                case 400 :
                        errMsg = "Server understood the request, but request content was invalid.";
                        if(jqXHR.responseText!=null)
                                detailsMsg = jqXHR.responseText;
                        break;
                case 401 :
                        errMsg = "Unauthorized access (401)"; break;
                case 403 :
                        errMsg = "Forbidden resource can't be accessed (403)"; break;
                case 404 :
                        errMsg = "resource not found (404)"; break;
                case 500 :
                        errMsg = "Internal server error (500)"; break;
                case 503 :
                        errMsg = "Service unavailable (503)"; break;
        }
        return errMsg+" "+detailsMsg;
}
```

Variation en mode "POST" et "application/x-www-form-urlencoded"

```
var dataJsObject = { prenom : "jean",   nom : "Bon" ,     taille: 175     } ;
$.ajax({
        type: "POST",
        url: "./my-api/person",
        contentType : "application/x-www-form-urlencoded; charset=utf-8",
        data: $.param(dataJsObject),
        dataType : 'json_or_text_or_...',
        beforeSend: function (xhr) {
                xhr.setRequestHeader ("Authorization",
                                        "Basic " + btoa("usernameXy" + ":" + "passwordXy"));
                        },
        success: ... ,  error: ....
});//end $.ajax
```

Variation en mode "POST" et "JSON" in/out :

```
//setSecurityTokenForAjax();//js/my-jq-ajax-util.js
$.ajax({
        type: "POST",
        url: "spectacle-api/spectacle",
        data : JSON.stringify(spectacleAdditionJsObject),
        dataType : "json",
        contentType : "application/json",
        success: function (data,status,xhr) {
                if (data) {
                    $("#spanMsg").html(JSON.stringify(data));
                        }
        },
        error: function( jqXHR, textStatus, errorThrown ){
                $("#spanMsg").html( xhrStatusToErrorMessage(jqXHR)  ) ;
        }
});//end $.ajax
```

1.4. Api fetch

Api récente (syntaxe concise basé sur enchaînement asynchrone et "**Promise**") mais pas encore supporté par tous les navigateurs.

Exemple :

```
fetch('./api/some.json')
  .then(
    function(response) {
      if (response.status !== 200) {
        console.log('Problem. Status Code: ' + response.status);
        return;
      }
      // Examine the text in the response :
      response.json().then(function(data) {
        console.log(data);
      });
    }
  )
  .catch(function(err) {
    console.log('Fetch Error :-S', err);
  });
```

1.5. Appel ajax via RxJs (api réactive)

Le framework "RxJs" lié au concept de "programmation asynchrone réactive" est assez sophistiqué et permet de déclencher une série de traitements d'une façon assez indépendante de la source de données (ex : données statiques , réponse ajax , push web-socket , ...) .

RxJs peut soit être directement utilisé en tant que bibliothèque javascript dans une page HTML , soit être utilisé via "typescript et le framework Angular 2,4,5,6 ou autre) .

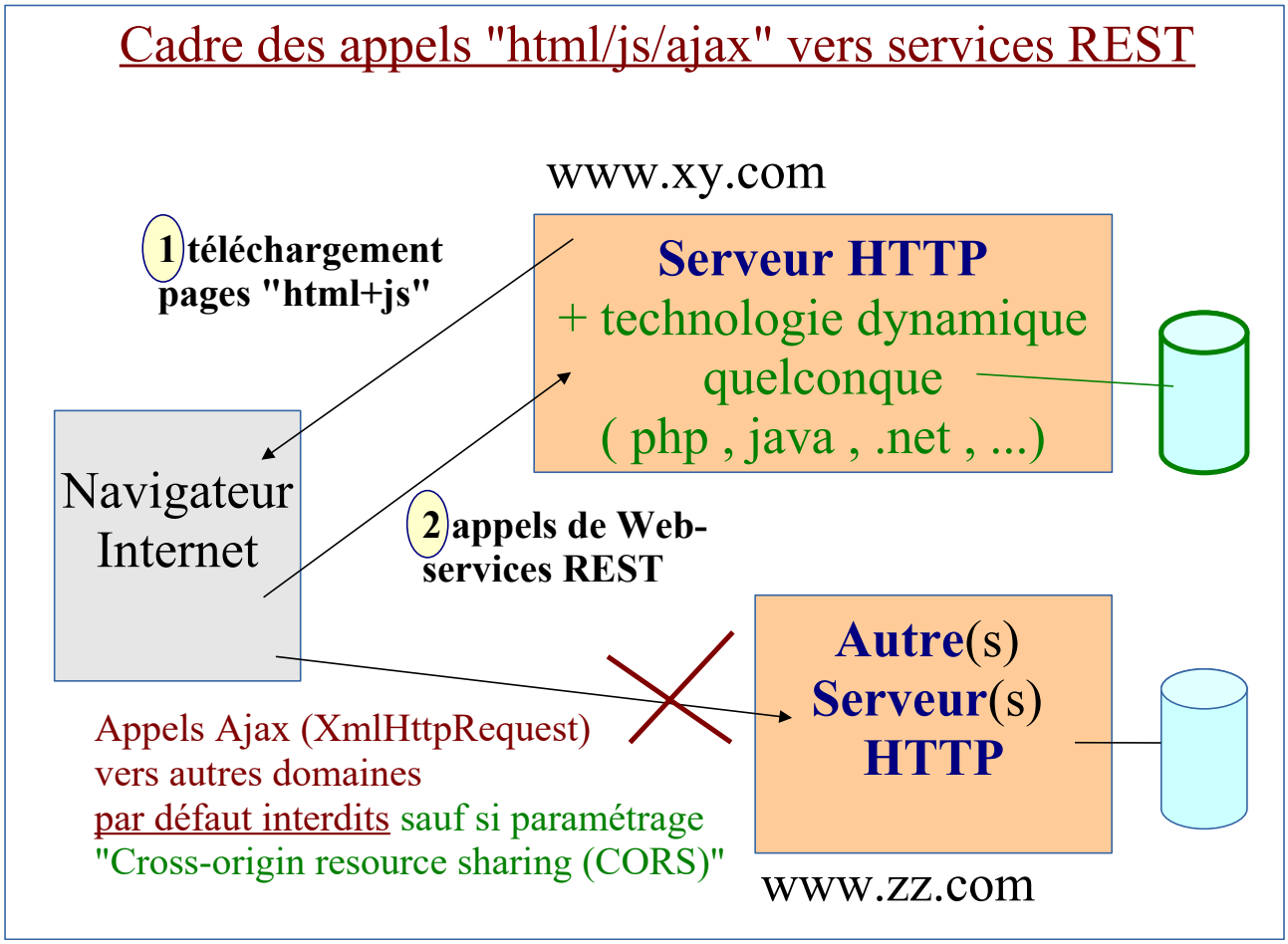
Attention à la version utilisée (différences significatives dans la version récente de RxJs accompagnant Angular 6) .

--> une bonne présentation de RxJS est accessible au bout de l'URL suivante

https://www.julienpradet.fr/tutoriels/introduction-a-rxjs/

--> exemples d'appel ajax via RxJs et de config proxy dans le support de cours "Angular 4,5,6"

2. Limitations Ajax sans CORS



3. CORS (Cross Origin Resource Sharing)

CORS=Cross Origin Resource Sharing

CORS est une **norme du W3C** qui précise certains **champs** à placer dans une **entête HTTP** qui serviront à échanger entre le navigateur et le serveur des informations qui serviront à décider si une requête sera ou pas acceptée.

(utile si domaines différents) , dans requête simple ou bien dans pré-échange préliminaire quelquefois déclenché en plus :

Au sein d'une requête "demande autorisation" envoyée du client vers le serveur :

Origin: http://www.xy.com

Dans la "réponse à demande d'autorisation" renvoyée par le serveur :

Access-Control-Allow-Origin: http://www.xy.com
Ou bien
Access-Control-Allow-Origin: * *(si public)*
→ *requête acceptée*

Si absence de "Access-Control-Allow-Origin :" ou bien valeur différente
---> requête refusée

CORS=Cross Origin Resource Sharing (2)

NB1: toute requête "CORS" valide doit absolument comporter le champ "***Origin :***" dans l'entête http. Ce champ est toujours construit automatiquement par le navigateur et jamais renseigné par programmation javascript.
Ceci ne protège que partiellement l'accès à certains serveurs car un "méchant hacker" utilise un "navigateur trafiqué".
Les mécanismes "CORS" protège un peu le client ordinaire (utilisant un vrai navigateur) que dans la mesure ou la page d'origine n'a pas été interceptée ni trafiquée (l'utilisation conjointe de "https" est primordiale) .

NB2 : Dans le cas (très classique/fréquent) , où la requête comporte "***Content-Type: application/json***" (ou ***application/xml*** ou …) , la norme "CORS" (considérant la requête comme étant "pas si simple") impose un **pré-échange préliminaire** appelé "**Preflighted request/response**" .

Paramétrages CORS à effectuer coté serveur

L'application qui coté serveur, fourni quelques Web Services REST , peut (et généralement doit) autoriser les requêtes "Ajax / CORS" issues d'autres domaines ("*" ou "www.xy.com").
Attention: ce n'est pas une "sécurité coté serveur" mais juste **un paramétrage autorisant ou pas à rendre service à d'autres domaines et en devant gérer la charge induite** *(taille du cluster, consommation électrique, ...)* .

```
// Exemple : CORS enabled with express/node-js :
app.use(function(req, res, next) {
  res.header("Access-Control-Allow-Origin", "*"); //"*" ou "xy.com , ..."
  res.header("Access-Control-Allow-Methods",
            "POST, GET, PUT, DELETE, OPTIONS"); //default: GET, ...
  res.header("Access-Control-Allow-Headers",
     "Origin, X-Requested-With, Content-Type, Accept , Authorization");
  next();
});
```

Paramétrage "CORS" avec Spring-mvc

```
import org.springframework.web.bind.annotation.CrossOrigin;
...
@RestController
@CrossOrigin(origins = "*")
//@CrossOrigin(origins = { "http://localhost:4200" ,
//                         "http://www.partenaire-particulier.com" })
@RequestMapping(value="/rest/products" , headers="Accept=application/json")
public class ProductCtrl {...
}
```

et

```
@Configuration
public class WebSecurityConfig extends WebSecurityConfigurerAdapter {
...
  @Override
  protected void configure(final HttpSecurity http) throws Exception {
            http.authorizeRequests()
                  .antMatchers("/",    "/favicon.ico",      "/**/*.png",
                              "/**/*.gif",      "/**/*.svg",  "/**/*.jpg",
                                          "/**/*.html",      "/**/*.css",     "/**/*.js").permitAll()
                  .antMatchers("/devise-api/public/**").permitAll()
                  .antMatchers("/devise-api/private/**").authenticated()
                   .and().cors() //enable CORS (avec @CrossOrigin sur class @RestController)
                  .and().csrf().disable()
                  .exceptionHandling().authenticationEntryPoint(unauthorizedHandler)
                  .and()
                  .sessionManagement().sessionCreationPolicy(SessionCreationPolicy.STATELESS)
                  .and()
                  .addFilterBefore(jwtAuthenticationFilter,
                                          UsernamePasswordAuthenticationFilter.class);
            }
...
}
```