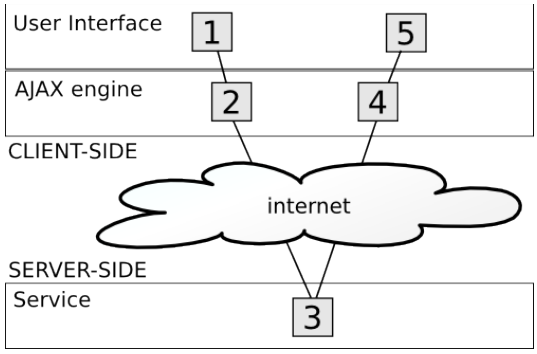
MASRAD :: AJAX – Examen final

Auteur : Olivier Ertz - Date : 10/04/2014

Question 1 :



➔ Quels sont les ingrédients qui composent le modèle d'application web dit AJAX ?

➔ Selon le schéma ci-contre, décrivez les différentes étapes d'un dialogue AJAX caractéristique en précisant à chaque étape les ingrédients qui interviennent ainsi que leur rôle.

Question 2 :

Votre collègue souhaite mettre en place une application web exploitant l'API Panoramio (voir documentation en **Annexe B**). Il vous appelle comme expert AJAX car ça ne fonctionne pas.

Il vous invite alors à consulter son code source (voir **Annexe A**) en vous expliquant qu'il souhaite invoquer le service http://www.panoramio.com/map/get_panoramas.php de Panoramio pour ajouter des photographies dans son application web. Pour l'instant il souhaite simplement afficher le flux JSON résultat. Or son application indique l'erreur « The request has failed » !

Pourtant, vous constatez que l'exécution de l'application génère une requête AJAX tout à fait conforme à l'API. La voici :

http://www.panoramio.com/map/get_panoramas.php?set=public&from=0&to=20&minx=6&miny=46&maxx=7&maxy=47

➔ Expliquez à votre collègue pourquoi cela ne fonctionne pas. Que faudrait-il pour que cela fonctionne sans rien modifier à son code ?

➔ Expliquez deux possibles adaptations de son code pour que cela fonctionne.

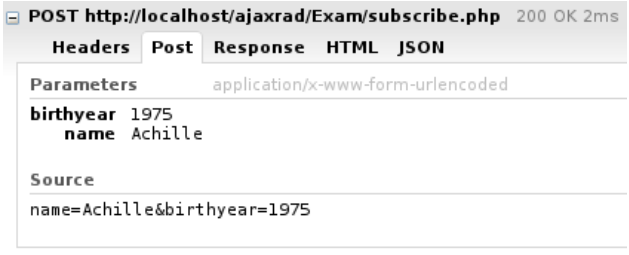
➔ Reprogrammez son application avec une de ces deux solutions.

Question 3 :

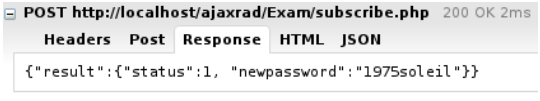
Ci-dessous une capture d'écran de requête HTTP asynchrone vue depuis Firebug :

➔ Reprogrammez l'application avec les instructions XMLHTTPRequest qui correspondent à un tel dialogue AJAX, de l'appel à l'exploitation du résultat avec une boîte d'alerte.

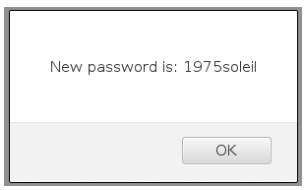
NB : ne créez pas de formulaire, utilisez des valeurs `name` et `birthyear` « en dur »



La requête d'appel



La réponse

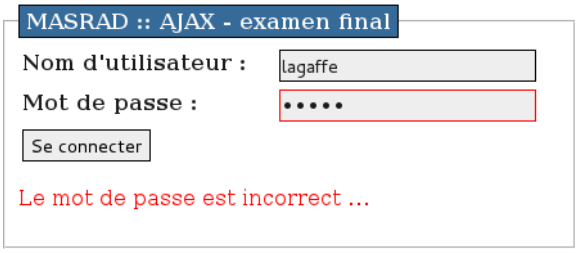


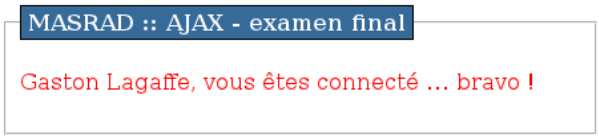
L'affichage

Question 4 :

Il s'agit de créer une application web complète sur le modèle d'application AJAX. C'est une application simplifiée de connexion avec vérification de couple nom d'utilisateur / mot de passe.

Votre application doit présenter une interface homme-machine proche de la maquette ci-dessous et interagir avec le service d'authentification ci-fourni (loginService.php).





A chaque tentative de connexion, le service d'authentification est appelé, celui-ci étant chargé de vérifier la validité du couple nom d'utilisateur / mot de passe.

Il fournit un résultat sous la forme d'un flux XML :

```
<?xml version="1.0" encoding="UTF-8"?>
<result status="1">
   <user>
      <lastname>Lagaffe</lastname>
      <firstname>Gaston</firstname>
   </user>
</result>
```

Le noeud `result` contient tout d'abord une information sur l'authentification (`status`) :

- status = 0 : les paramètres fournis au service sont mauvais
- status = 1 : le couple nom d'utilisateur / mot de passe est correct
- status = 2 : le nom d'utilisateur est inconnu
- status = 3 : le mot de passe est incorrect

Aussi, lorsque l'authentfication est correcte (status = 1), le flux contient en plus les informations sur l'utilisateur authentifié, c'est-à-dire son nom et son prénom (**`user`**).

Il est demandé de gérer tous ces cas au travers de l'interface homme-machine comme illustré dans la démonstration :

- pour chaque cas amenant à un échec d'authentication, un message spécifique est affiché en rouge sous le formulaire (cf. capture ci-dessus)
- de plus, si le nom d'utilisateur est inconnu ou si le mot de passe est incorrect, le champ de saisi correspondant est cadré de rouge (cf. capture ci-dessus)
- pour le cas d'une authentification avec succès, le formulaire disparait et un message approprié est affiché en indiquant le nom et prénom de l'utilisateur (cf. capture ci-dessus)

Notez bien, seul le couple nom d'utilisateur = lagaffe et mot de passe = menfin est valide.

Annexe A (« ça ne marche pas ! ») :

```
<!DOCTYPE html>
<html>
    <head>
        <title>Panoramio API</title>
        <meta charset="UTF-8">
        <meta name="viewport" content="width=device-width">

        <script type="text/javascript">

            window.onload = function() {

                var params = {
                    set: "public",
                    from: 0,
                    to: 20,
                    minx: 6,     // Canton de Vaud et sud Léman
                    miny: 46,
                    maxx: 7,
                    maxy: 47
                };

                var url = "http://www.panoramio.com/map/get_panoramas.php" + serialize(params);

                xhr = new XMLHttpRequest();
                xhr.open("GET", url, true);

                xhr.onload = function(e) {
                    displayResult(this.responseText);
                };

                xhr.onerror = function(e) {
                    console.log("The request has failed");
                };

                xhr.send(null);
            };

            function displayResult(data) {
                console.log(data);
            }

            /***
             * serialize : fonction utilitaire qui formatte une liste de clé:valeur
             * en URL query string (de la forme ?param1=val1&param2=val2&param3=val3)
             *
             * @param {Object} p
             * @returns {String}
             */
            function serialize(p) {
                var qs = "?";
                for (key in p) qs += key + "=" + p[key] + "&";
                return qs.substring(0, qs.length - 1);
            }

        </script>

    </head>
    <body>
        <h1>Working with Panoramio API</h1>
        <div id="container"></div>
    </body>
</html>
```

Notez bien :

- `window.onload = function() {...}` est équivalent à `<body onload="uneFonction() ;">`
- voir le code ici : http://ogo.heig-vd.ch/ajaxrad/Exam/

Annexe B (Panoramio API)

Display photos on your own website

It's a very simple REST api, you only have to do a GET on:

http://www.panoramio.com/map/get_panoramas.php?set=public&from=0&to=20&minx=-180&miny=-90&maxx=180&maxy=90&size=medium

- for **`set`** you can use: `public` (popular photos) | `full` (all photos) | user ID number
- for **`size`** you can use: `original` | `medium` (default value) | `small` | `thumbnail` | `square` | `mini_square`
- the **`minx`**, **`miny`**, **`maxx`**, **`maxy`** define the area to show photos from (minimum longitude, latitude, maximum longitude and latitude, respectively).

You can define the number of photos to be displayed using "from=X" and "to=Y", where Y-X is the number of photos included. The value 0 represents the latest photo uploaded to Panoramio. For example, "from=0 to=20" will extract a set of the last 20 photos uploaded to Panoramio, "from=20 to=40" the previous set of 20 photos and so on. The maximum number of photos in one query is 100.

The result data is formatted using JSON. An example:


```
{ "count" : 7478,
  "has_more" : true,
  "map_location" : { "lat" : 46.490555499999999,
      "lon" : 6.5948194569297796,
      "panoramio_zoom" : 9
    },
  "photos" : [ { "height" : 240,
        "latitude" : 46.216628999999998,
        "longitude" : 6.5613950000000001,
        "owner_id" : 4055012,
        "owner_name" : "arno18",
        "owner_url" : "http://www.panoramio.com/user/4055012",
        "photo_file_url" : "http://mw2.google.com/mw-panoramio/photos/small/66817984.jpg",
        "photo_id" : 66817984,
        "photo_title" : "9 / 10 / 2011 ",
        "photo_url" : "http://www.panoramio.com/photo/66817984",
        "upload_date" : "15 February 2012",
        "width" : 209
      },
      { "height" : 177,
        "latitude" : 46.225043999999997,
        "longitude" : 6.1471689999999999,
        "owner_id" : 4055012,
        "owner_name" : "arno18",
        "owner_url" : "http://www.panoramio.com/user/4055012",
        "photo_file_url" : "http://mw2.google.com/mw-panoramio/photos/small/79570101.jpg",
        "photo_id" : 79570101,
        "photo_title" : "La magie de l'automne",
        "photo_url" : "http://www.panoramio.com/photo/79570101",
        "upload_date" : "26 September 2012",
        "width" : 240
      }
    ]
}
```


The **count** property is the total number of photos available on that set of photos on that area. The **photos** property is an array with the requested photos. The variables of each **photo** object should be trivial to interpret.

It's also available as JSONP, just add an extra `callback=your_function_name` to the GET, and you will get: `your_function_name({ "count": 773840, "photos": [...] })`

If you have any doubt about the code, don't hesitate to contact us at questions@panoramio.com