Développer un calendrier dynamique avec AJAX et PHP

par Raideman http://j-seignalet.developpez.com

Date de publication : 20/11/2007

Dernière mise à jour :

L'avènement du "web 2.0" avec notamment ses interfaces riches a placé l'AJAX au centre des applications web. Pour faciliter le développement, de nombreux framework ont vu le jour comme les projets prototype et openrico. Au travers d'une application "calendrier", nous allons tenter ici de les mettre en oeuvre.

1 - Présentation du projet "calendrier"

Tout au long de cet article, nous allons développer un calendrier qui s'affichera mois par mois à la manière d'un "google calendar" ou de ce que l'on peut faire avec l'api Yahoo. Nous verrons alors comment peupler ce calendrier avec les dates à l'aide d'AJAX et de la librarie prototype.

Nous ne nous pencherons pas sur le fonctionnement détaillée de la technologie AJAX et je vous conseille de lire l'article de **Gaël Donnat sur AJAX** pour plus d'informations.

Ensuite, pour chaque date, nous pourrons afficher des évènements stockés dans une base de données.

Enfin, nous mettrons en forme notre calendrier à l'aide du **framework** javascript openrico.

Les langages utilisés seront donc javascript et PHP.

Afin de disposer d'un support, vous pouvez télécharger le source complet de l'application : **Télécharger le code source complet de l'application (archive .zip)**

Et aussi voir un exemple du script sur **cette page de démonstration.**

Utiliser un framework est un choix que je vous encourage à faire si vous le pouvez ! Vous bénéficierez généralement de plus de souplesse, de qualité et éviterez de nombreux casse-tête quant à la compatibilité du code avec les navigateurs.

1.1 - Pré-requis

Vous trouverez dans les paragraphes suivants la liste des outils nécessaires à la création de cette application.

1.1.1 - PHP MYSQL

Pour notre projet, nous avons besoin de PHP pour les traitements et d'une base de données MYSQL pour stocker les évènements liés à chaque date.

Pour se faire nous allons donc déployer **WAMP** sur une station windows.

Télécharger WAMP

Une fois les serveurs en route, vous pouvez créer la base de données qui contiendra les évènements associés à chaque jour. Voici un jeu d'essai que vous pouvez utiliser pour tester l'application :

```
CREATE DATABASE `calendrier` ;
```

```
CREATE TABLE `evenements` ( `evenement_id` INT NOT NULL AUTO_INCREMENT PRIMARY KEY
 ,`evenement_date` DATE NOT NULL ,`evenement_comment` TEXT NOT NULL ) TYPE = innodb;
INSERT INTO `evenements` ( `evenement_id` , `evenement_date` , `evenement_comment` )
VALUES (
 '1', '2007-11-04', 'Réunion projet "calendrier"'
 ), (
 '2', '2007-11-04', 'Départ à la retraite de thierry'
);
INSERT INTO `evenements` ( `evenement_id` , `evenement_date` , `evenement_comment` )
VALUES (
 '3', '2007-11-08', 'Démarrage du projet "SEO"'
 ), (
 '4', '2007-12-24', 'Préparation des cadeaux :)'
);
```

1.1.2 - Le framework prototype

Le framework prototype propose un ensemble de fonctions javascript facilitant le développement AJAX.

Pour appréhender plus en détails les particularités et le fonctionnement de ce framework, vous pouvez consulter **l'article sur "prototype" par Aurélien MILLET**

Ce framework est directement intégré dans la bibliothèque openrico, vous n'avez pas donc pas besoin de le télécharger. Une documentation de ce framework est disponible en français :

Documentation prototype (v.1.4.0)

1.1.3 - La librairie openrico

Openrico est une bibliothèque de fonctions javascript permettant de créer de nombreux effets graphiques (comme des menus en accordéon, des blocs arrondis ou encore la gestion du drag and drop).

Ce projet intègre le framework prototype pour la gestion des requêtes AJAX. Télécharger la bibliothèque et décompresser l'archive par exemple dans le répertoire "js" de votre application web

Télécharger OPENRICO (v 2.0, PROTOTYPE inclus)

1.2 - Fonctionnement de l'application

Note application se décomposera en trois fichiers principaux :

- **calendrier.php** : ce fichier va contenir notre calendrier, c'est à dire un tableau de 7 colonnes (pour chaque jour de la semaine) et de 6 lignes.
- **ajax/ajax_calendrier.php** : il aura la charge de peupler le calendrier avec les dates pour un mois de l'année donné en paramètre.
- **ajax/ajax_commentaires.php** : ce fichier aura pour but d'afficher la liste des évènements pour le jour du calendrier que l'on aura choisi.

Comme parfois un schéma vaut mieux qu'un long discours, voici une petite illustration graphique du comportement de notre application :

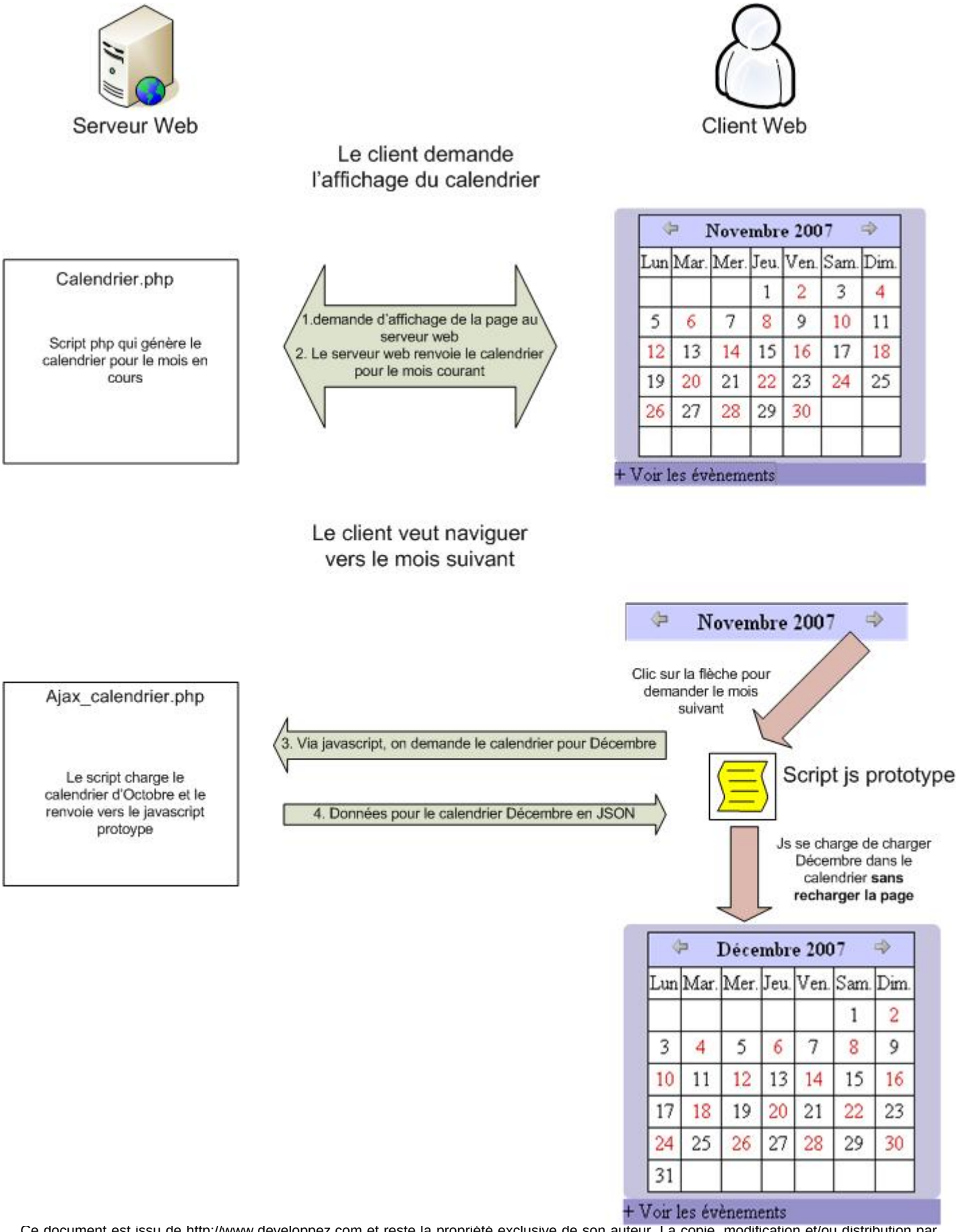


Schéma fonctionnel de l'application calendrier

Dans la pratique, nous allons donc afficher notre calendrier avec la possibilité de naviguer de mois en mois. Pour chaque mois choisi, une requête ajax va être transmise vers le fichier ajax_calendrier.php qui va retourner un tableau contenant tous les jours à afficher dans le calendrier pour le mois demandé.

Cette réponse sous forme de tableau sera en fait au format **JSON** ! (qui est un format directement lisible par prototype , on en verra l'intérêt un peu plus tard).

Lorsque l'utilisateur cliquera sur un jour du calendrier, une nouvelle requête ajax va être envoyé via prototype au fichier ajax_commentaires.php. Ce fichier se chargera d'aller chercher dans la base de données les évènements liés au jour choisi et de les renvoyer à prototype pour les afficher à l'écran.

Passons maintenant à la pratique et rentrons dans le code !

Vous noterez que l'on ne fait plus vraiment de l'AJAX , puisque comme son nom l'indique, AJAX implique une réponse au format XML et qu'on va plutôt opter pour un format JSON. Ceci dit, pour des raisons pratiques, je continuerai à parler d'AJAX.

2 - Développement de notre calendrier

2.1 - Création du calendrier

calendrier.php (entête)

```
<?php
//connexion à la base de données mysql
$ress_mysql = mysql_connect('servername', 'username', '');
$db = mysql_select_db('dbname', $ress_mysql) or die ("Connexion impossible");
?>
<!-- On inclut la librairie openrico / prototype -->
<script src="./js/rico/src/prototype.js" type="text/javascript"></script>
```

On effectue ici deux opérations. La première c'est la connexion à la base de données. Et ensuite, ce qui est déjà plus intéressant, on importe la bibliothèque javascript openrico.

On remarque au passage que la librairie prototype fait partie des fichiers "openrico" à inclure.

Maintenant, on peut se charger d'afficher le tableau html qui va contenir notre calendrier.

calendrier.php (corps calendrier)

```
<div id="calendrier" class="conteneur calendrier" style="width:260px;background-color:#c6c3de;">
    <table class="tab_calendrier" align="center" style="margin:auto;background:#ffffff;">
       <tr>
     <td class="titre_calendrier" colspan="7" width="100%">
     <a id="link_precedent" href="#">Precedent</a>
     <span id="titre"></span>
     <a id="link_suivant" href="#">Suivant</a>
  </td>
   </tr>
      <tr>
        <td  class="cell_calendrier" >
        Lun
        </td>
        <td  class="cell_calendrier" >
        Mar.
     </td>
     <td  class="cell_calendrier">
  Mer.
  </td>
  <td  class="cell_calendrier">
  Jeu.
  </td>
  <td  class="cell_calendrier" >
   Ven.
  </td>
  <td  class="cell_calendrier">
  Sam.
  </td>
  <td  class="cell_calendrier">
  Dim.
  </td>
      </tr>
      <?php
             $compteur_lignes=0;
             $total=1;
             while($compteur_lignes<6){
                echo '<tr>';
                $compteur_colonnes=0;
```

calendrier.php (corps calendrier)

```
                    while($compteur_colonnes<7){
                        echo '<td id="'.$total.'" class="cell_calendrier" >';
                        echo '</td>';
                        $compteur_colonnes++;
                        $total++;
                    }
                    echo '</tr>';
                    $compteur_lignes++;
                }
        ?>
    </table>
</div>
```

On génère avec ce code, un tableau qui va contenir les emplacements des différents jours du mois.

On veille aussi à attribuer un attribut "id" à tous les éléments HTML dans lesquels on va afficher des informations dynamiques (les liens mois suivant et mois précédent, les cellules pour les jours du mois, etc...).

Notre page est prête, nous pouvons maintenant peupler le calendrier.

2.2 - AJAX en action avec prototype

Notre calendrier est sympathique mais désespérément vide !!

Nous allons le remplir à l'aide de prototype via une requête AJAX. Pour se faire, nous allons donc créer un fichier que j'appellerai /js/function.js qui va se charger de générer la requête d'affichage complet du calendrier.

/js/function.js

```
tableau = function(mois,annee)
{
   var url = './ajax/ajax_calendrier.php';
   var parametres = 'mois=' + mois + '&annee=' + annee;

  var myAjax = new Ajax.Request(
   url,
   {
    method: 'get',
    parameters: parametres,
    onComplete: remplirCalendrier
   }
  );
}
```

Vous ne remarquez rien ?

Fini le pavé de code où l'on déclarait un objet XMLHttpRequest en fonction des navigateurs !!

Grâce à l'objet **Ajax.Request**, on peut très facilement envoyer une requête à notre serveur. Faisons un petit tour des variables utilisées dans cette fonction :

- **mois et annee** : on donne ici le mois et l'année pour pouvoir générer les jours de ce mois.
- **url** : c'est l'adresse du fichier distant qui va se charger d'effectuer le traitement de calcul des jours à afficher pour le mois et l'année choisis.
- **parametres** : ce sont la liste des variables que l'on passe au script distant ajax_calendrier.php

- **myAjax** : c'est une instance d'un objet "prototype".
- **Attribut method** : la méthode d'envoi des paramètres , POST ou GET.
- **Attribut parameters** : la chaine des variables à envoyer.
- **onComplete** : fonction javascript à appeler une fois la réponse du serveur envoyée. (remplirCalendrier)

Les données de notre calendrier vont être générées grâce au fichier distant "ajax_calendrier.php" sollicité par notre fonction tableau.

Allons donc jeter un coup d'oeil sur ce fameux fichier distant.

2.3 - Traitement distant et réponse JSON

2.3.1 - Format du JSON

Avant toute chose, quelques mots sur JSON *(JavaScript Object Notation)* .

JSON est, à l'instar d'XML, un format de données. Son principal avantage est qu'il permet d'être directement traité par javascript, puisqu'il a le même format qu'un objet javascript.

De plus, prototype permet de lire ce format avec une facilité relativement déconcertante. Simple et léger, il peut convenir à de nombreuses applications.

Revenons à nos moutons. Notre fichier ajax_calendrier.php va construire un fichier JSON qui sera envoyé comme réponse à notre javascript.

A partir des paramètres mois et année, on va donc afficher le nom du mois, les liens vers les mois suivant et précédent, les jours du mois et enfin spécifier si un évènement a bien lieu pour le jour concerné.

structure de la réponse JSON

```
{
     "mois_en_cours" : "Janvier 2008"
     "lien_precedent" : "tableau('12','2007') " ,
     "lien_suivant" : "tableau('02','2008') " ,
     "calendrier": [
               {
                 "fill": "1" //(jour du mois)

               },
               {
                 "fill": "2" //(jour du mois)

               }//etc...
                  ]
}
```

La variable fill contiendra soit :

***"vide"** si la case ne doit rien afficher (cas des lignes de début et de fin de mois).*

***Le numéro du mois** (1,2, ... 31) pour la case concernée.*

Le numéro du mois sous forme de lien *si des évènements sont prévus pour le jour donné.*

2.3.2 - Calcul des dates et des évènements

Pour peupler notre calendrier, nous allons simplement construire notre réponse JSON.

Tout d'abord, on va chercher le nom du mois en fonction de son numéro, et la valeur de l'attribut onclick qu'auront les liens "mois suivant" et "mois précédent".

/ajax/ajax_calendrier.php (en tête)

```
header('Content-Type: text/x-json; charset: UTF-8');
////////////////////////////////////////////////////////////////////////////////
//
//                    On prépare le début du retour au format JSON
////////////////////////////////////////////////////////////////////////////////
$retour_json='';
////////////////////////////////////////////////////////////////////////////////
//On détermine d'abord les liens "suivant" "precedent" et le "mois en cours" du calendrier
////////////////////////////////////////////////////////////////////////////////
$mois=$_GET['mois'];
$annee=$_GET['annee'];
$retour_json.='{';
//mois en cours
$mois_fr=getMois($mois);
$titre=htmlentities($mois_fr." ".$annee,ENT_QUOTES);
$retour_json.='"mois_en_cours" : "'.$titre.'" , ';
//lien suivant
$date_suivant=getSuivant($mois,$annee,1);
$lien_suivant="tableau('".$date_suivant[mois]."','".$date_suivant[annee]."')";
$retour_json.='"lien_suivant" : "'.$lien_suivant.'" , ';
//lien précédent
$date_precedent=getSuivant($mois,$annee,-1);
$lien_precedent="tableau('".$date_precedent[mois]."','".$date_precedent[annee]."')";
$retour_json.='"lien_precedent" : "'.$lien_precedent.'" , ';
```

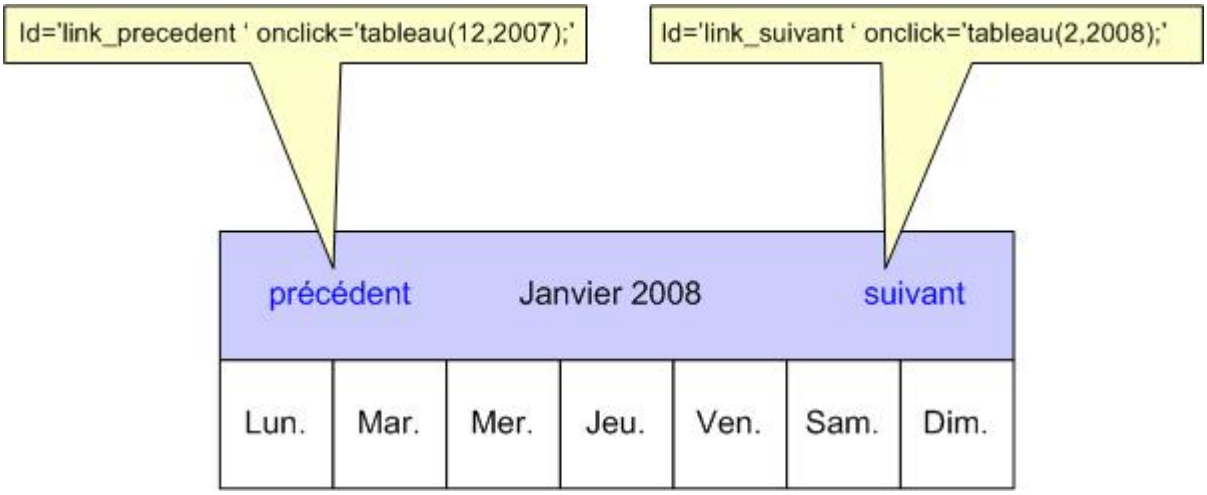
La réponse JSON est stockée dans la variable $retour_json.

la fonction **getMois** est une fonction permettant d'obtenir le nom du mois en français à partir de son numéro (1=>Janvier par exemple, ces fonctions sont définies dans le fichier /conf/func_calendrier.php disponible dans les sources).

la fonction **getSuivant** est une fonction permettant d'obtenir l'année et le mois suivants ou précédents une date donnée (pour Janvier 2008 par exemple , getSuivant(1,2008,1) retournera le mois 2 et l'année 2008 et getSuivat(1,2008,-1) retournera 12 et 2007).

Pour comprendre le rôle variables $lien_suivant et $lien_precedent, il faut se rappeller que ses variables serviront à renseigner l'attribut onclick des liens permettant de naviguer de mois en mois.

Toujours en prenant l'exemple de Janvier 2008, on aura la page html suivante :



Structure HTML de la page calendrier

Nous avons notre en-tête de calendrier, déterminons maintenant les dates à afficher et les évènements associés.

/ajax/ajax_calendrier.php (corps)

```
//Maintenant, on peut peupler le calendrier sous forme d'un tableau de 6 lignes * 7 colonnes
//On crée notre tableau de 6semaines*7jours soit 42 éléments.
//On récupère le jour qui démmarre le mois
//ON va stocker tous les jours du mois dans un tableau tab_jours php
$tab_jours=array();
$num_jour=getFirstDay($mois,$annee);
$compteur=1;
$num_jour_courant=1;
while($compteur<43){
     if($compteur<$num_jour){
          //On gère les "blancs" du calendrier car un mois ne commence pas toujours un Lundi.
    $tab_jours[$compteur]='';
     }else
     {
         //si la date existe, on affiche alors le jour dans la cellule du tableau
         if(checkdate($mois,$num_jour_courant,$annee)){
              //On vérifie si un évènement a lieu ce jour ci
              $date=$annee."/".$mois."/".$num_jour_courant;
              $contenu='';
              $requete="select * from evenements where evenement_date='".$date."'";
              $ress=mysql_query($requete);
              if($ress){
                  $nbre=mysql_num_rows($ress);

                  if($nbre>0){
                     //lien vers le script qui va déclencher l'affichage des évènement pour le jour
donné
                     $lien='<a href=\'#\'
onclick=\'showEvent(\\"'.$date.'\\");\'>'.$num_jour_courant.'</a>';
                     $tab_jours[$compteur]=$lien;
                  }else
                  {

                    $tab_jours[$compteur]=$num_jour_courant;
                  }
                  mysql_free_result($ress);
              }
              $num_jour_courant++;

         }else
         {
```

/ajax/ajax_calendrier.php (corps)

```
            //On gère ici les "blanc" de fin de ligne car un mois ne se termine pas toujours un
    dimanche.
        $tab_jours[$compteur]='';
            }

        }
          $compteur++;
    }
```

On stocke dans le tableau $tab_jours pour les 42 cases du tableau calendrier (7 jours * 6 semaines), le contenu qui sera affiché dans la case du calendrier.

Si on est positionné sur un jour du calendrier, on regarde dans la base de données si un ou plusieurs évènements existent pour ce jour. Si c'est le cas, on stocke un lien qui permettra d'afficher les évènements (nous verrons cela un peu plus tard).

Vous remarquerez peut être l'utilisation d'une fonction getFirstDay($mois,$annee). Cette fonction se trouve dans le fichier /conf/func_calendrier.php et permet de renvoyer le numéro du premier jour d'un mois en notation française (1=>Lundi, 2=>Mardi ... 7=>dimanche).

Maintenant que l'on a toutes nos données, on peut finir le fichier JSON en parcourant le tableau $tab_jours, et on retourne cette réponse .

/ajax/ajax_calendrier.php (renvoi du fichier JSON)

```
    ////////////////////////////////////////////////////////////////////////////////
    // Maintenant que l'on a notre tableau d'évènements pour chaque jour du mois
    // On finit de construire la réponse JSON
    ////////////////////////////////////////////////////////////////////////////////
    if(!empty($tab_jours)){
       $retour_json.=' "calendrier" : [ ';
       $compteur=1;
       while ( $compteur < 43){
          if($compteur==42){
             $retour_json.=' { "fill" : "'.$tab_jours[$compteur].'" } ';
          }else
          {
             $retour_json.=' { "fill" : "'.$tab_jours[$compteur].'" } , ';
          }
       $compteur++;
       }
       $retour_json.=' ] ';
   }
   $retour_json.=' } ';
    echo $retour_json;
```

Pour chaque case de notre calendrier, on a donc défini ce qu'il faut afficher :

- **rien** pour les cases vides.
- **un numéro de jour** pour les dates qui existent.
- **un numéro de jour sous forme de lien** pour voir les évènements prévus ce jour-ci.

Notre réponse JSON est correctement formée et peut maintenant être traitée par javascript. C'est ce que nous allons voir maintenant.

2.3.3 - Traitement de la réponse JSON.

Le flux JSON est renvoyé sous forme de variables vers notre javascript. Rappelez vous, nous avions ce code :

/js/function.js

```
var myAjax = new Ajax.Request(
 url,
 {
  method: 'get',
  parameters: parametres,
  onComplete: remplirCalendrier
 }
);
```

L'évènement onComplete est déclenché lorsque la réponse est envoyée. Si c'est le cas, la réponse est transmise vers la fonction remplirCalendrier.

Regardons donc comment nous allons traiter ce flux JSON dans notre fonction remplirCalendrier.

/js/function.js

```
function remplirCalendrier(reponsejson) {
        //on utilise la fonction evalJSON de prototype pour récupérer la réponse JSON
        var data=reponsejson.responseText.evalJSON();
        //On place les liens suivants,précédents et le mois en cours
        $('link_suivant').onclick=function(){eval(data.lien_suivant) ;};
        $('link_precedent').onclick=function(){eval(data.lien_precedent) ;};
        $('titre').innerHTML=data.mois_en_cours;
        //Maintenant, on affiche tous les jours du calendrier
        var compteur=1;
        var id='';
        while(compteur<43){
           id=compteur.toString();
           $(id).innerHTML=data.calendrier[(compteur-1)].fill;

           compteur++;
        }
}
```

Dans un premier temps, on place nos données JSON dans une variable **data**, qui, grâce à prototype sera un objet directement utilisable contenant toutes les données de notre calendrier ! (grâce à **evalJSON**

On accède ensuite aux valeurs via la syntaxe data.nom_de_la_variable . Dans la structure de notre JSON, nous avions spécifié que les jours du calendrier seraient stockés dans un tableau nommé "calendrier".

Pour récupérer les valeurs de ce tableau, il nous suffit alors de faire une boucle pour parcourir ce tableau et l'afficher dans la case du calendrier correspondante.

*Vous remarquerez aussi la syntaxe **$(element)** . C'est une fonctionnalité apportée par prototype et qui équivaut au traditionnel document.getElementById(element).*

2.4 - Affichage des évènements journaliers

Notre calendrier est désormais fonctionnel et permet une navigation de mois en mois sans rechargement visible de la page.

Nous allons afficher maintenant les évènements liés aux jours de notre calendrier.

2.4.1 - Création de la requête AJAX

Lors de la génération du flux JSON dans le fichier ajax_calendrier.php, nous avions, lorsqu'un évènement était détecté, crée le lien suivant :

/ajax/ajax_calendrier.php

```
$lien='<a href=\'#\' onclick=\'showEvent(\\"'.$date.'\\");\'>'.$num_jour_courant.'</a>';
```

L'idée est relativement simple, on va créer une fonction javascript showEvent qui va envoyer une requête AJAX vers un fichier distant php qui va aller chercher, en fonction de la date, les évènements du jour en question.

/js/function.js

```
function showEvent(date){

   var url = './ajax/ajax_commentaires.php';
   var parametres = 'date=' + date ;

  var myAjax = new Ajax.Request(
   url,
   {
    method: 'get',
    parameters: parametres,
    onComplete: remplirCommentaires
   }
  );
}
```

Comme nous l'avions vu lors de notre première requête AJAX, on va utiliser l'objet prototype **Ajax.request** qui nous simplifie grandement la vie.

2.4.3 - Génération de la réponse

Rien de sorcier ici. On effectue une simple requête SQL dont on extrait les données.

ajax/ajax_commentaires.php

```
<?php
     //Connexion à la base de données
  include("../conf/mysql.php");
     $date=$_GET['date'];
     $requete="select * from evenements where evenement_date='".$date."'";
     $ress=mysql_query($requete);
     $retour='';
     if($ress){
               while($liste_evenements=mysql_fetch_assoc($ress)){
                  $retour.=htmlentities($liste_evenements[evenement_comment],ENT_QUOTES).'<br>';
               }
               //on renvoie la réponse
               echo $retour;
     }
?>
```

2.4.4 - Traitement et affichage de la réponse

Dans notre appel AJAX, nous avions défini que la réponse serait traitée via une fonction js remplirCommentaires, la voici :

/js/function.js

```
function remplirCommentaires(reponse){
        var commentaires=reponse.responseText;
        $('Evenements').innerHTML=commentaires;
        PullDown.panel = Rico.SlidingPanel.top( $('outer_panel'), $('inner_panel'));
        PullDown.panel.toggle();
}
```

Comme précédemment, on récupère la réponse avec **reponse.responseText**.

On réinitialise le panel afin qu'il soit en position fermée, puis on le déroule.

On affiche ensuite les évènements via la syntaxe "raccourcie" prototype : **$('Evenements').innerHTML=commentaires;**

On crée alors simplement dans calendrier.php une balise html div ayant pour id la valeur "Evenements".

calendrier.php

```
<div id="Evenements">
</div>
```

Nous avons donc notre calendrier, fonctionnel et dynamique. Cependant, il serait plus sympathique avec quelques améliorations visuelles. C'est ce que nous allons voir maintenant avec l'aide notamment de la librairie openrico.

3 - Aller plus loin avec OPENRICO

Jusqu'à présent, nous avons utilisé le framework prototype intégré à openrico mais pas réellement openrico lui même.

En effet, cette librairie apporte de nombreux effets graphiques javascript que l'on peut plus ou moins facilement ajouter à notre application.

Dans le cadre de notre calendrier, nous allons l'utiliser pour styliser deux affichages :

- **Arrondir les bords du calendrier**
- **Créer un panel déroulant pour afficher les évènements**

3.1 - Arrondir les bords du calendrier

Depuis quelques temps maintenant, et inscrit dans la mouvance "2.0", on aperçoit de plus en plus de blocs html avec les bords arrondis.

Si vous êtes rompu au langage HTML, vous savez qu'il est souvent délicat de créer ce genre d'effets. OPENRICO va vous permettre d'y parvenir avec plus de facilité.

Tout d'abord, nous allons inclure les fichiers javascript nécessaires dans notre fichier calendrier.php

calendrier.php (inclusion des js openrico)

```
<script src="./js/rico/src/rico.js" type="text/javascript"></script>
<script src="./js/rico/src/ricoStyles.js" type="text/javascript"></script>
<script src="./js/rico/src/ricoEffects.js" type="text/javascript"></script>
<!-- ... -->
<div id="calendrier" class="conteneur calendrier" style="width:260px;background-color:#c6c3de;">
<!-- code du calendrier -->
</div>
</div>
<div id="Evenements">
</div>
```

Maintenant, on va appliquer le bord arrondi à la balise conteneur "conteneur calendrier" de notre application. Ce qui aura pour effet d'arrondir les bords du calendrier.

Sous l'affichage du calendrier, on va insérer le code suivant :

calendrier.php (appel de la fonction arrondir)

```
<script>
                $$('div.conteneur').each(function(e){Rico.Corner.round(e)});
</script>
```

Pour arrondir les angles, on va simplement appliquer la fonction **Rico.Corner.round** qui va se charger d'arrondir les bords de **tous** les éléments html dont l'attribut class commence par le mot "conteneur". En gardant cette syntaxe, on observera donc les comportements suivants :

- **attribtut class="conteneur calendrier"** : les bords vont être arrondis.
- **attribtut class="conteneur test"** : les bords seront là aussi arrondis.

- **attribtut class="test"** : les bords ne seront pas arrondis.

Le calendrier arrondi, nous allons maintenant nous pencher sur l'élément HTML "Evenements" pour styliser l'affichage des évènements pour un jour donné.

3.2 - Construire un panel déroulant

Pour utiliser les fonctions de panel déroulant dans openrico, il faut tour d'abord importer de nouvelles librairies.

calendrier.php (inclusion des js openrico)

```
<script src="./js/rico/src/ricoComponents.js" type="text/javascript"></script>
```

Il ne nous reste plus qu'à préparer le conteneur "Evenements" pour qu'il se présente sous forme déroulante.

calendrier.php (div evements)

```
<div style="position: relative; width: 260px;">
      <div id="top-panel" style="background-color : #9791cb;position: relative;width:
 260px;z-index: 1500;">
          <a href="javascript:void(0);" id="code-button" onclick="PullDown.panel.toggle(); return
 false;">
               + Voir les évènements
          </a>
     </div>
     <div id="main-part">
      <div id="outer_panel" class="panel-top" style="overflow: hidden; position: absolute;
 z-index: 1600;top: 19px; width: 260px;height: 132px;">
        <div style="position: relative;top: 1px;background-color: #c6c3de;margin:0px;border: 1px
 solid #9791cb;" id="inner_panel">
            <div id="Evenements" style="height:150px">
                  </div>
           </div>
       </div>
     </div>
</div>
<script>
             Event.observe(window, 'load', function(){
                  PullDown.panel = Rico.SlidingPanel.top( $('outer_panel'), $('inner_panel'));
             })
             var PullDown = {};
</script>
```

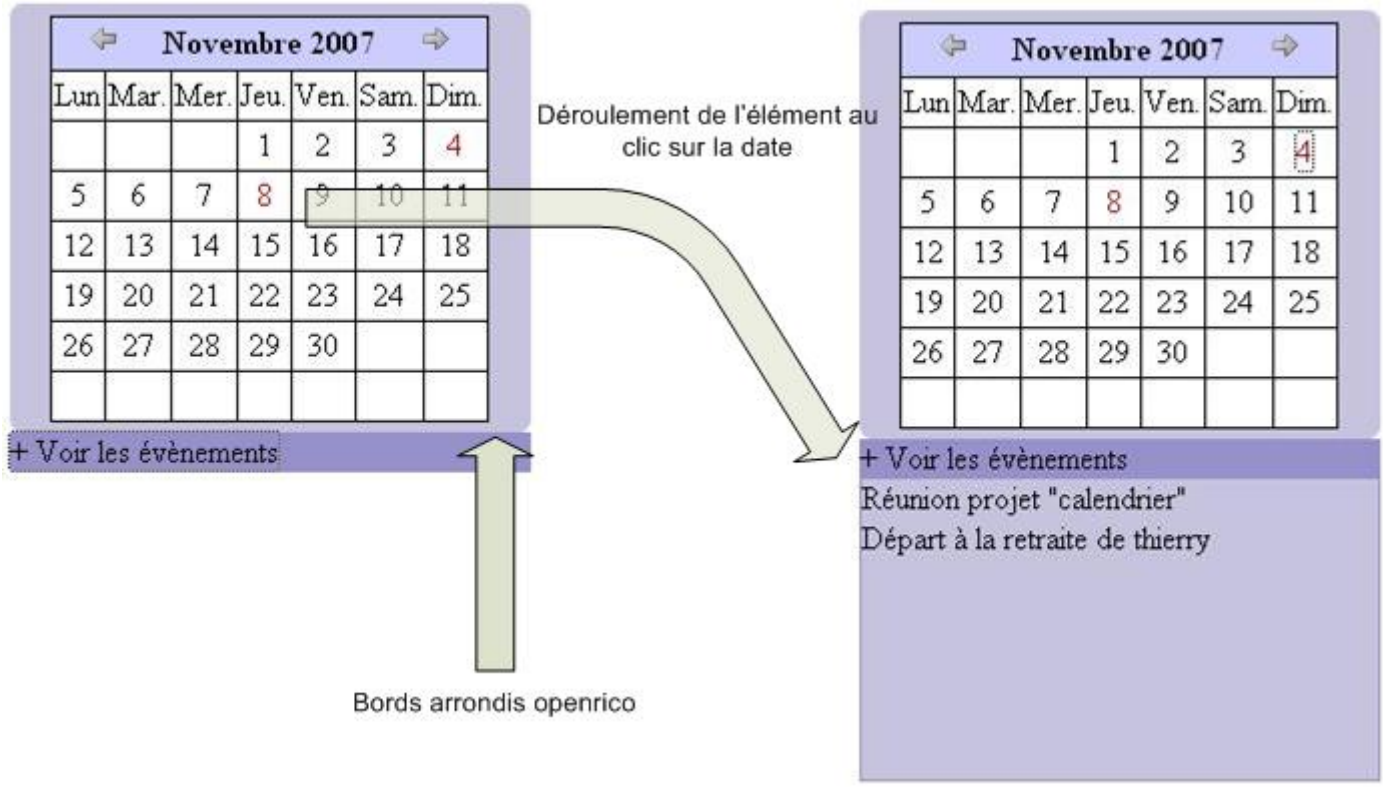
On remarque tout d'abord le lien **onclick="PullDown.panel.toggle(); return false;"** . C'est en cliquant sur ce lien que l'élément va dérouler.

La fonction PullDown.panel.toggle() représente en fait l'objet openrico Rico.SlidingPanel.top(element1,element2) déclaré un peu plus bas. C'est cette fonction toggle() qui va s'occuper d'enrouler / dérouler le panel selon nos besoins.

Ensuite les éléments inner_panel et outer_panel sont les éléments qui vont "bouger" lorsque l'on va cliquer sur le lien "voir les evenements".

Enfin, la div "Evenements" est la même que celle que l'on a créee tout à l'heure et servira à stocker les évènements prévus pour un jour donné.

Jetons maintenant un coup d'oeil au résultat final :



OpenRico en action

4 - Un petit bilan

4.1 - Les avantages des framework

Le développement web est une pratique difficile car les contraintes sont nombreuses: multiplicité des technologies existantes, multiplicité des environnement clients (compatibilité des navigateurs ?), besoins du client ...

L'utilisation d'un framework prend alors toute son importance ! Ces outils, si ils sont performants, vous permettent de gagner du temps en évitant de réinventer la roue, et vous garantissent généralement une réponse fiable à vos besoins.

Même si leurs documentations ne sont pas toujours claires et complètes, il est enrichissant de les pratiquer et de les mettre en oeuvre.

4.2 - Les améliorations possibles

La mise en oeuvre de ce calendrier a avant tout un but pédagogique, et il existe bien sûr d'autres méthodologies pour obtenir un résultat similaire !

Selon vos contraintes, vous pouvez tout à fait privilégier d'effectuer bon nombre de traitements dans vos scripts javascript au lieu des scripts php distants (notamment pour l'affichage du calendrier).

De plus, vous pouvez tout à fait améliorer encore l'esthétisme de l'application en utilisant des liens images (avec des flèches par exemple) pour afficher les liens "mois suivant" et "mois précédent".

En couplant cette pratique avec une feuille de style à votre goût, vous pouvez construire une application encore plus ergonomique (en colorant les cases du calendrier qui ont des évènements par exemple)

4.3 - Les limites

Utiliser des technologies reposant sur le client est un choix. En effet, vous devez prendre en considération le fait que des environnements utilisateurs ne vont pas prendre en charge javascript par exemple.

Essayez donc toujours de proposer une solution alternative à une fonctionnalité javascript.

De plus, vous permettrez ainsi aux moteurs de recherche de pouvoir indexer ce contenu car le javascript n'est que peu ou pas pris en compte par les "crawlers".

Toujours à propos des moteurs de recherche, pensez à interdire aux robots l'accès à vos fichiers distants php que vous utilisez avec AJAX. Cela vous évitera de voir des pages "inutiles" pollués les pages déjà indéxées.

Dans un souci de lisibilité, nous avons volontairement omis de nombreuses règles de sécurité de la programmation web, faites attention à ne pas les oublier lors de vos déploiements.

5 - Netographie

Wamp :

Tutoriel sur l'utilisation de WAMP.

Site web du projet WAMP

Prototype :

Site officiel du projet prototype

Openrico :

Site officiel du projet openrico

JSON :

Le format JSON

Gestionnaire de calendrier

La bibliothèque CALENDAR de yahoo

Le calendrier partagé GOOGLE

Un autre exemple de calendrier full php