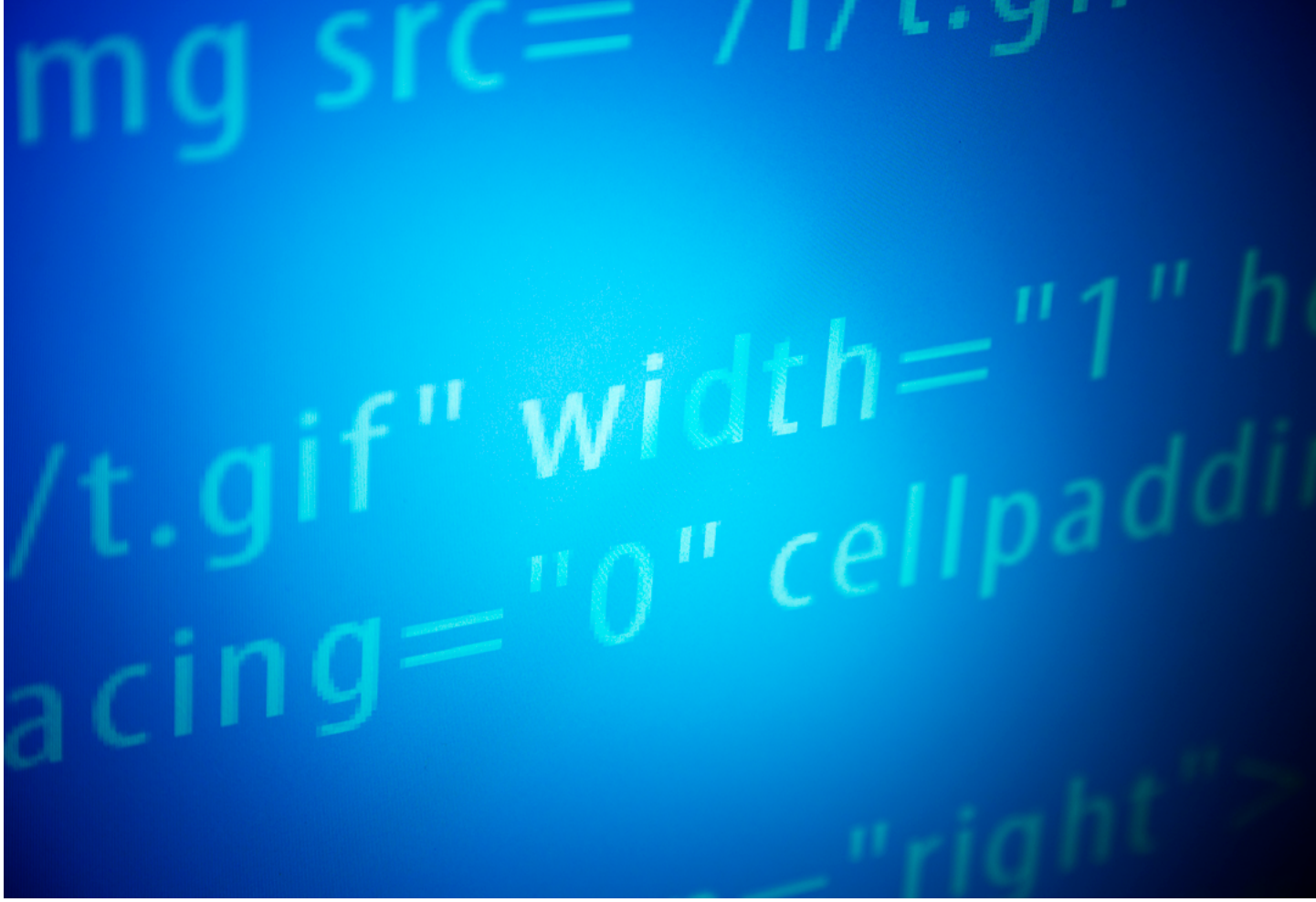
Création de vignettes

Sommaire

Les repères de lecture

Retour au chapitre
Définition
Objectif(s)
Espace Élèves
Vidéo / Audio
Point important / À retenir
Remarque(s)
Pour aller plus loin
Normes et lois
Quiz



Introduction

Les vignettes sont un redimensionnement des images chargées. Elles ont les mêmes proportions mais sont plus petites, ce qui permet de composer une galerie d'image présentée sur une page et de cliquer sur chaque vignette pour consulter l'image au grand format. Les pages web étant limitées par la taille de l'écran de consultation, il est naturellement plus ergonomique de présenter sur une même page dix images possédant une largeur de 100 pixels que de 500 pixels.

Lorsque nous évoquons les dimensions d'une image, nous commençons toujours par définir sa largeur puis sa hauteur. Une image dont nous disons qu'elle possède « une taille de 200 sur 300 » a donc une largeur de 200 pixels et une hauteur de 300 pixels. La dimension étant évoquée en pixel qui est l'unité de mesure du Web.

I. Fonction *imagecopyresampled*

Nous définissons donc que, pour l'application en cours, nos vignettes auront une largeur de 150 pixels. Nous avons besoin de la fonction PHP *imagecopyresampled* qui permet de réduire les images en préservant la qualité et la proportion.

La fonction *imagecopyresampled* « copie, redimensionne, rééchantillonne une image » comme l'indique la page du manuel PHP : http://php.net/manual/fr/function.imagecopyresampled.php

Cette fonction possède un grand nombre d'arguments qu'il est nécessaire de bien étudier :

- **dst_image** : lien vers la **ressource cible de l'image**, c'est-à-dire le chemin de la vignette ;
- **src_image** : lien vers la **ressource source de l'image**, c'est-à-dire le chemin de l'image d'origine ;
- **dst_x** : coordonnées du point de destination, c'est-à-dire le **point d'abscisse (axe horizontal) de l'image de destination** où commencera la recopie de l'image d'origine ;
- **dst_y** : coordonnées du point de destination, c'est-à-dire le **point d'ordonnée (axe vertical)** ;
- **de l'image de destination** où commencera la recopie de l'image d'origine ;
- **src_x** : coordonnées du point source, c'est-à-dire le **point d'abscisse de l'image d'origine** où commencera la copie de l'image d'origine ;
- **src_y** : coordonnées du point source, c'est-à-dire le **point d'ordonnée de l'image d'origine** où commencera la copie de l'image d'origine ;
- **dst_w : largeur de la destination**, c'est-à-dire la largeur de la vignette ;
- **dst_h : hauteur de la destination**, c'est-à-dire la hauteur de la vignette ;
- **src_w : largeur de la source**, c'est-à-dire la largeur de l'image d'origine ;
- **src_h : hauteur de la source**, c'est-à-dire la hauteur de l'image d'origine.

L'exemple donné par la page du manuel PHP est suffisamment explicite pour être adapté dans notre application. Mais comme à l'accoutumée, il est recommandé lorsque nous abordons une nouveauté de commencer par la tester. Nous allons donc au préalable effectuer des exemples de développement avec les fonctions PHP orientées image. Puis nous utiliserons *imagecopyresampled*.

II. Extension GD, flux d'image et fonctions image

L'extension GD ou librairie GD est un composant de PHP, généralement préinstallé sur WAMP ainsi que sur les environnements offerts par les hébergeurs qui permet à PHP de créer et de manipuler des images dans les principaux formats rencontrés sur le Web notamment JPG, GIF et PNG.

La liste de toutes les fonctions de la librairie GD dédiées au traitement d'images est disponible sur cette page du manuel PHP : http://fr2.php.net/manual/fr/book.image.php

Nous allons étudier et tester les quatre fonctions suivantes, souvent utilisées :

- **imagecreatefromjpeg** et **imagejpeg**, qui sont associées. La fonction *imagejpeg* permet de créer un flux d'image, c'est-à-dire un affichage d'image créée à la volée par la fonction *imagecreatefromjpeg* ;
- **imagecreatetruecolor** qui permet de créer des images (et qui est préférable à *imagecreate*) ;
- **imagecopyresampled**.

1. Les fonctions *imagejpeg* et *imagecreatefromjpeg*

http://fr2.php.net/manual/fr/function.imagejpeg.php

La fonction *imagejpeg* permet d'afficher une image directement dans la page, à la volée, ou dans un fichier.

Le premier argument obligatoire de *imagejpeg* étant la ressource image fournie par *imagecreatefromjpeg*. La fonction *imagecreatefromjpeg*, dont le nom est explicite, permet de créer une nouvelle image depuis un fichier ou une URL (pour ce dernier point, des restrictions concernant Windows sont expliquées dans le manuel PHP).

Par exemple, si nous souhaitons afficher à la volée une image **test.jpg** avec *imagejpeg*, nous commençons par **créer la ressource image** avec *imagecreatefromjpeg*.

```
$img = imagecreatefromjpeg('test.jpg'); // test.jpg étant le chemin
```

Fig. 1

La variable *$img* à laquelle nous affectons le résultat retournée par *imagecreatefromjpeg* contient cette ressource image puisque la fonction *imagecreatefromjpeg* retourne une ressource image.

Ayons le réflexe du *var_dump* et regardons ce qui est retourné :

```
var_dump ($img);
```

Fig. 2

Ceci affiche :

```
resource(3, gd)
```

Fig. 3

Si le chemin est erroné, nous obtenons :

```
boolean false
```

Fig. 4

En cas d'erreur, la fonction retourne donc bien soit *FALSE* (et PHP génère également dans ce cas une erreur de type *Warning*), sinon la fonction retourne une ressource. Une ressource est une information non intelligible mais que PHP comprend et qui peut être passée à une autre fonction pour être exploitée. La ressource retournée par *imagecreatefromjpeg* peut être exploitée par la fonction *imagejpeg*.

Afin d'afficher directement l'image nous spécifions que l'en-tête (header) est du type MIME *image/jpeg* avec la fonction *header*.

```
header('Content-Type: image/jpeg');
```

Fig. 5

Et enfin nous utilisons la ressource *image* (la variable *$img*) à la fonction *imagejpeg* :

```
imagejpeg($img);
```

Fig. 6

Pour récapituler, le code complet est donc le suivant :

Fichier **flux-image.php** (à placer dans un répertoire destiné à tester les fonctions PHP) :

```
$img = imagecreatefromjpeg('test.jpg');
header('Content-Type: image/jpeg');
imagejpeg($img);
```

Fig. 7

Si nous ne spécifions pas le header avec le bon MIME, alors le navigateur tente d'afficher l'image comme une page web, c'est-à-dire un fichier de type texte, ce qui a pour résultat d'afficher à l'écran non pas l'image mais le code constitutif de l'image qui est, comme la plupart des fichiers, un élément de type binaire.

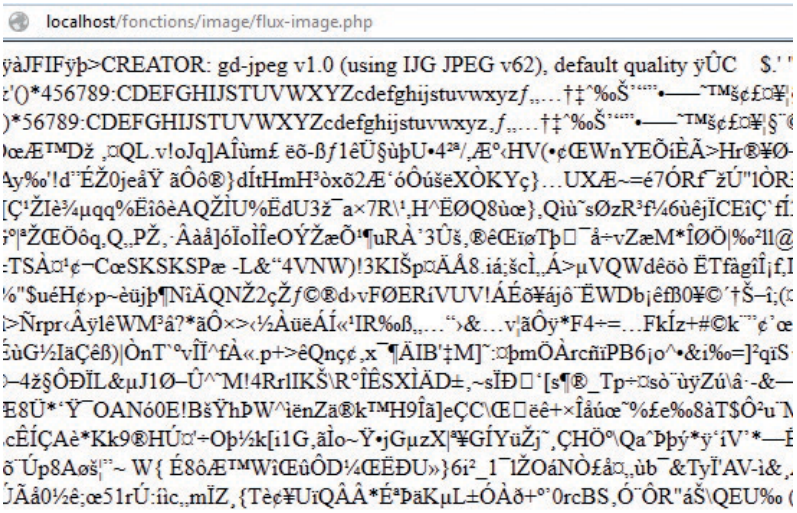


Fig. 8

Si nous spécifions le bon *header*, alors l'image sera correctement affichée dans la page, comme le montre le résultat suivant :

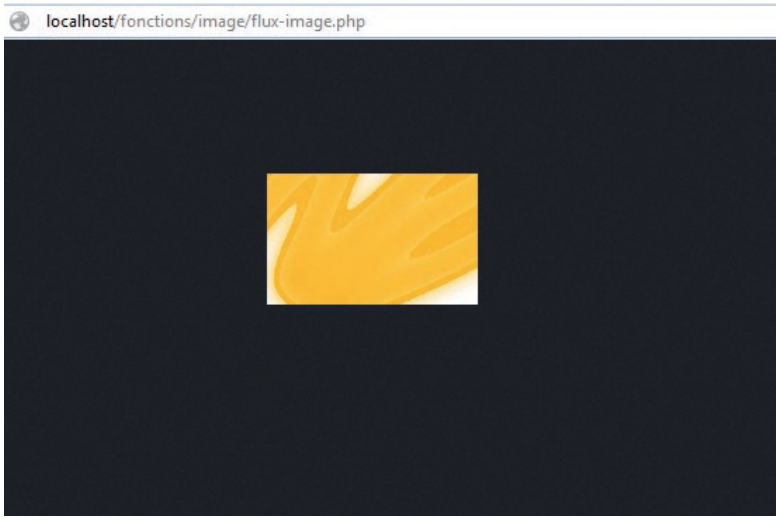


Fig. 9

L'intérêt de cette fonction peut paraître limité puisque nous savons déjà afficher une image très simplement en HTML avec l'élément IMG, mais il faut bien voir que nous n'avons pas affiché une page, nous avons affiché une image, de manière dynamique, ce qui constitue un flux d'image géré en PHP.

Pour s'en convaincre il suffit d'effectuer deux manipulations :

– **un clic droit sur la page** : dans ce cas, celui-ci ne permet pas de consulter le code source de la page (puisqu'il ne s'agit pas d'une page) :

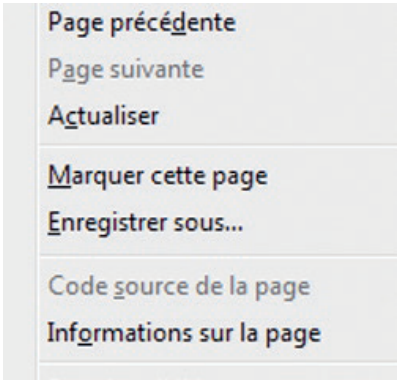


Fig. 10

– d'enregistrer l'image sur son disque dur : par défaut le nom de fichier proposé est celui du fichier PHP (« **flux-image.php**» ou « **flux-image.php.jpg** » en fonction des navigateurs) et non pas le nom du fichier image.

L'intérêt des flux d'images est que nous pouvons effectuer des calculs et passer des paramètres aux images affichées. Il existe d'autres fonctions GD qui permettent par exemple d'écrire à la volée des informations dans les images affichées. Il est également possible de les redimensionner comme nous le verrons avec *imagecopyresampled*.

Il est également possible de ne pas se limiter au flux dans le navigateur, mais **de créer des fichiers sur notre disque dur avec imagejpeg en spécifiant l'argument filename**, qui est un chemin d'accès vers un nom de fichier. (Dans l'exemple ci-dessous il faudra que le répertoire *dir_test* existe).

```
imagejpeg($img, 'dir_test/foo.jpg');
```

Fig. 11

Dans ce cas en revanche, il sera nécessaire de ne pas spécifier de MIME dans le header, puisque nous ne cherchons pas à afficher un flux d'image mais à générer un fichier. Les usages de ces fonctions sont fréquents et nombreux et c'est d'ailleurs la fonction *imagejpeg* qui permettra dans notre application de générer les vignettes.

2. La fonction *imagecreatetruecolor*

La fonction *imagecreatetruecolor* permet de créer une nouvelle image supportant les couleurs vraies, c'est-à-dire plusieurs millions de couleurs.

Concrètement, elle crée une image noire avec des dimensions passées en argument, par exemple 200 x 200 pixels.

```
$img = imagecreatetruecolor(200, 200);
```

Fig. 12

Résultat : cf. Figure ci-dessous

Fig. 13

L'intérêt est que cette image noire va pouvoir être utilisée par d'autres fonctions, comme par exemple la fonction *imagestring* qui permet d'écrire du texte. Cette image noire peut également être « remplie » par d'autres couleurs avec la fonction *imagecolorallocate* qui permet donc concrètement de dessiner sur ce fond noir.

Page du manuel sur la fonction *imagestring* :
http://php.net/manual/fr/function.imagestring.php

Page du manuel sur la fonction *imagecolorallocate* :
http://php.net/manual/fr/function.imagecolorallocate.php

Cette image peut également **récupérer une copie d'une autre image** à des fins de redimensionnement avec la fonction *imagecopyresampled*.

Comme pour toutes les fonctions il est intéressant de consulter la section intitulée « Valeurs de retour » de la page du manuel dédiée à la fonction *imagecreatetruecolor* et de constater qu'elle « retourne un identifiant de ressource d'image en cas de succès, ou *FALSE* si une erreur survient » de la même manière que *imagecreatejpg*.

A. Redimensionnement des images d'origine avec *imagecopyresampled*

Ayant approché la façon de gérer les images en PHP, nous reprenons le développement de notre application.

Nous décidons de redimensionner les images originales afin qu'elles aient une taille maximale. Certaines photos par exemple peuvent atteindre des tailles de 4 000 pixels ce qui peut être trop grand pour nos besoins. Si les images d'origine sont trop grandes, par exemple si elles sont supérieures à 1 200 pixels de largeur ou 2 000 pixels de hauteur, nous choisissons de les redimensionner pour qu'elles ne dépassent pas cette largeur ou cette hauteur.

Le principe est le suivant :

– **création de l'image source** avec la fonction *imagecreatefromjpeg* qui va créer une image JPG depuis l'image uploadée ;

– **création d'une image noire** « destinaire » dont les dimensions seront celles des vignettes avec *imagecreatetruecolor* ;

– **copie de l'image source** vers la vignette avec *imagecopyresampled*. Cf. Fig. 21 et 22 ;

– **enregistrement final** (ou affichage à l'écran) avec *imagejpg*.

Nous intégrons le code dans une méthode *createThumbnail* que nous déclarons et à laquelle nous passons un argument *filename* qui sera le nom de fichier initial de l'image.

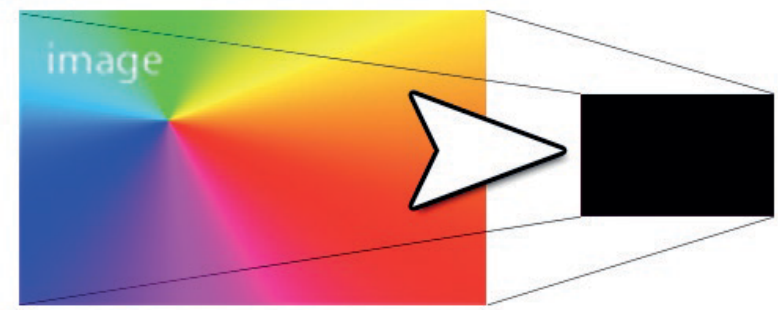


Fig. 14

Fig. 15

Pour récupérer les dimensions de l'image uploadée nous utilisons la fonction *getimagesize*.

http://fr2.php.net/manual/fr/function.getimagesize.php

Cette fonction est très souvent utilisée car elle retourne sous la forme d'un tableau les informations essentielles d'une image accessibles via les index suivants :

Array

(

[0] => 250 : largeur en pixels

[1] => 156 : hauteur en pixels

[2] => 2 : type de l'image (2 = JPG)

[3] => width="250" height="156" : cet index est très pratique pour afficher les attributs

width et height sous la forme d'une chaîne directement exploitable dans un élément <img> et qui sont obligatoires en HTML.

[bits] => 8 : nombre d'octets pour chaque couleur.

[channels] => 3 : code pour les images RGB, c'est-à-dire créées pour de la diffusion vidéo.

[mime] => image/jpeg : le MIME de l'image

)

Dans la méthode *createThumbnail* nous utiliserons les index 0 et 1.

Voici le code de la méthode *createThumbnail* :

```
public function createThumbnail ($filename)
{
     // 1. définition des chemins des images et des vignettes
     $image            = IMAGE_DIR_PATH . $filename ;
     $vignette   = THUMB_DIR_PATH . $filename ;

     // 2. récupération des dimensions de l'image source
     $size = getimagesize($image);
     $largeur = $size[0];
     $hauteur = $size[1];

     // 3. définition des valeurs souhaitées pour les vignettes
     // ce sont des valeurs maximales
     $largeur_max = 200;
     $hauteur_max = 200;

     // 4. création de l'image source avec imagecreatefromjpeg
     $image_src = imagecreatefromjpeg($image);

     // 3. On crée un ratio (une proportion)
     // et on verifie que l'image source ne soit pas
     //plus petite que l'image de destination

     if ($largeur > $largeur_max OR $hauteur > $hauteur_max)
     {
       if ($hauteur <= $largeur) // si largeur plus grande que hauteur
       {
           $ratio = $largeur_max / $largeur;
       }
       else {
           $ratio = $hauteur_max / $hauteur;
       }
     }
     else
     {
          $ratio = 1; // l'image créee sera identique à l'originale
     }

// 4. création de l'image noire de destination avec imagecreatetruecolor
     $image_destination = imagecreatetruecolor(round($largeur * $ratio),
                          round($hauteur * $ratio));

// 5. fabrication de la la vignette avec dimensions souhaitées
     imagecopyresampled($image_destination, $image_src, 0, 0, 0, 0,
round($largeur * $ratio), round($hauteur * $ratio), $largeur, $hauteur);

     // 6. Envoi de la nouvelle image JPEG dans le fichier
     if(!imagejpeg($image_destination, $vignette))
     {
       $error_msg = 'la création de la vignette a echouée pour l\'image '.
                    $image;
       return $error_msg;
          }
     else
     {
          return true;
     }
}
```

Fig. 16

Nous intégrons ensuite la méthode *createThumbnail* au script de l'*upload* dans la partie relative à *move_uploaded_file* :

```
if ($error == UPLOAD_ERR_OK)
{
     $tmp_name = $files['upload']['tmp_name'][$key];
     $filename = $files['upload']['name'][$key];
     if (move_uploaded_file($tmp_name, $upload_dir . $filename) === false)
     {
          $error++ ;
     }
     else
     {
          // appel avec $this de la méthode au sein d'une même classe
          $this->createThumbnail ($filename);
     }
}
```

Fig. 17

Testons l'application. Après avoir intégrer ce fonctionnement dans le nouveau script de l'*upload*, nous allons préparer l'intervention sur le nom des fichiers images.

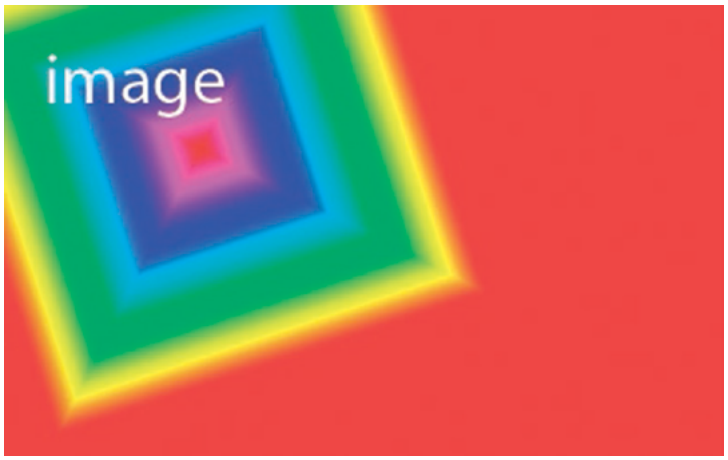


Fig. 18

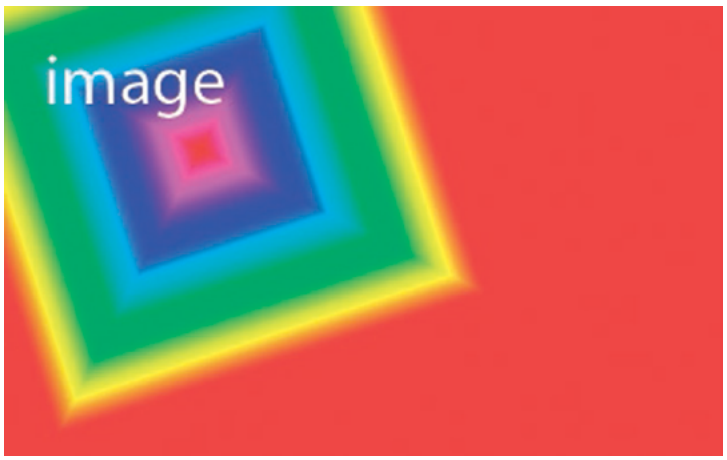


Fig. 19

Nous obtenons suite à l'*upload* le fichier original d'une dimension de 400 x 250 ainsi que, dans le répertoire *thumbnails*, la vignette correspondante de dimension 200 x 125. La largeur maximale est bien de 200 pixels et la hauteur est proportionnelle à celle du fichier source.

Il ne reste plus qu'à afficher dans le *template* des pages *Admin* et *Front* les vignettes au lieu des grandes images initiales en utilisant le fichier config.php auquel nous ajoutons sous forme de constantes les chemins des vignettes.

Fichier **config.php** :

```
<?php
// fichier de configuration
define ('WEB_TITLE', 'Projet Images');
define ('WEB_DIR_NAME', 'projet_image');
define ('WEB_DIR_URL', 'http://' . $_SERVER['HTTP_HOST'] . '/'. WEB_
DIR_NAME .'/');
define ('IMAGE_DIR_NAME', 'images');
define ('IMAGE_THUMB_NAME', 'thumbnails');
define ('IMAGE_DIR_PATH', $_SERVER['DOCUMENT_ROOT'] . WEB_DIR_NAME
.'/'. IMAGE_DIR_NAME .'/');
define ('IMAGE_DIR_URL', 'http://' . $_SERVER['HTTP_HOST'] . '/'.
WEB_DIR_NAME .'/'. IMAGE_DIR_NAME .'/');
define ('THUMB_DIR_PATH', $_SERVER['DOCUMENT_ROOT'] . WEB_DIR_NAME
.'/'. IMAGE_DIR_NAME .'/'.IMAGE_THUMB_NAME .'/');
define ('THUMB_DIR_URL', 'http://' . $_SERVER['HTTP_HOST'] . '/'.
WEB_DIR_NAME .'/'. IMAGE_DIR_NAME .'/'.IMAGE_THUMB_NAME .'/');
```

Fig.20

III. Renommage des images

Les noms des fichiers images peuvent être très variés et nous souhaitons les homogénéiser pour l'ensemble des fichiers que nous gérons avec notre application. Et nous souhaitons également ajouter le nom du site « *projet_image* » à toutes les images que nous « *uploadons* ».

Nous donnerons le même nom de fichier aux images et aux vignettes, sachant que nous les distinguerons par leur chemin (localisation du répertoire + nom de fichier).

Les noms de fichiers images devront donc tous être du type : **projet_image_nom_fichier.jpg**, le tout en bas de casse (lettre minuscule).

Nous allons donc créer une fonction qui va nettoyer les noms de fichiers et les redéfinir selon nos souhaits qui peuvent être traduits par le besoin de ne contenir que des caractères alphanumériques en bas de casse ainsi que des *underscores* pour remplacer les éventuels espaces. Ainsi le nom de fichier : « **Arc en Ciel.jpg** » devra être transformé en « **projet_image_arc_en_ciel.jpg** ». Cela signifie que nous devons :
- **remplacer** les majuscules par des minuscules ;
- **remplacer** les espaces par des *underscores* ;
- **ajouter** le suffixe *projet_image*.

Un autre nom de fichier peut être « **Image d'été.jpg** ». Dans ce cas nous devons en plus :
- **remplacer** le signe typographique « apostrophe » par rien ;
- **remplacer** la lettre « é » par la lettre « e », sans accent.

Puisque nous n'allons pas attendre que différents cas de nommage se présentent pour adapter notre code, nous devons prendre la décision de gérer l'ensemble des cas possibles.

Nous décidons que **toutes les lettres accentuées** ainsi que **tous les signes typographiques** seront remplacés.

Dans les fonctions sur les chaînes de caractères que nous avons déjà abordées, il existe la fonction *str_replace* qui remplace justement toutes les occurrences dans une chaîne. La fonction *str_replace* utilise 3 arguments minimum *$search* , *$replace* , *$subject*.

Page des fonctions sur les chaînes de caractères (rappel) :

http://fr2.php.net/manual/fr/ref.strings.php

Page dédiée à *str_replace* : http://fr2.php.net/manual/fr/function.str-replace.php

Le fonctionnement de *str_replace* est simple : les occurrences de *search* dans *subject* sont remplacées par replace. Effectuons quelques tests.

Exemple théorique, nous pouvons remplacer les lettres « a » de bateau par « A » :

```
$result = str_replace("a", "A", "Bateau");
echo $result;
```

Fig.21

Ce code affiche :

```
BAteAu
```

Fig.22

Le premier argument permet de **capturer** la lettre « a » ce que nous recherchons dans la chaîne de caractères.

Autre exemple, remplaçons les espaces vides par un *underscore* :

```
$result = str_replace($tab, "_", "il fait beau");
```

Fig.23

Remarquons que le premier argument est vide, ce qui va permettre de capturer les espaces vides de la chaîne de caractères.

Ce code affiche :

```
il_fait_beau
```

Fig.24

Le grand intérêt de la fonction *str_replace* est qu'elle permet d'utiliser aussi bien des chaînes de caractères que des tableaux de données comme arguments. C'est la raison pour laquelle le manuel PHP indique *mixed* dans sa description.

```
$tab = array('a','e','i','o','u', 'y');
$result = str_replace($tab, "_", "il fait beau");
```

Fig. 25

Le premier argument est donc un tableau contenant les voyelles. Ici nous demandons donc à ce que toutes les voyelles soient remplacées par un *underscore*, ce qui affiche :

```
_l f__t b___
```

Fig. 26

Enfin, nous pouvons utiliser deux tableaux de données pour remplacer les valeurs du premier tableau trouvées dans la chaîne de caractères par les valeurs du second tableau.

Pour effectuer le remplacement, les tableaux doivent avoir le même nombre de valeurs, et il faut bien faire correspondre les valeurs des deux tableaux.

Exemple
Remplaçons chaque voyelle en lettre minuscule par la même voyelle en lettre majuscules :

```
$tab = array('a','e','i','o','u', 'y');
$tab2 = array('A','E','I','O','U', 'Y');
$result = str_replace($tab, $tab2, "il fait beau");
```

Fig. 27

Résultat

```
Il fAIt bEAU
```

Fig. 28

Nous confirmons ainsi que la fonction *str_replace* semble bien pouvoir répondre à notre besoin de renommage des noms de fichiers.

Nous créons deux tableaux de données. Le premier tableau nommé *$special* contient tous les caractères accentués, ainsi que l'apostrophe et l'espace vide. Le second *$normal* faisant correspondre tous les caractères non accentués ainsi que *l'underscore* pour l'espace vide et « rien » pour l'apostrophe.

Et nous testons avec une chaîne de caractères contenant l'ensemble des occurrences que nous voulons remplacer.

```
$special = array(' ', '\'', 'á','à','ã','ä','â','å','ç','é','è','ë'
,'ê','í','ì','ï','î','ñ','ó','ò','õ','ö','ô','ú','ù','ü','û','ý','ÿ'
,'Á','À','Ã','Ä','Â','Å','Ç','É','È','Ë','Ê','Í','Ì','Ï','Î','Ñ','
Ó','Ò','Õ','Ö','Ô','Ú','Ù','Ü','Û','Ý');

$normal = array('_', '', 'a','a','a','a','a','a','c','e','e','e','e
','i','i','i','i','n','o','o','o','o','o','u','u','u','u','y','y','
A','A','A','A','A','A','C','E','E','E','E','E','I','I','I','I','N',
'O','O','O','O','O','U','U','U','U','Y');

$result = str_replace($special, $normal, "c'áàãäâåçéèëêíìïîñóòõö
ôúùüûýÁÀÃÄÂÅÇÉÈËÊÍÌÏÎÑÓÒÕÖÔÚÙÜÛÝ");
```

Fig. 29

– Nous ne gérons pas dans notre exemple l'intégralité des signes typographiques.

– Il serait aisé d'ajouter les remplacements voulus avec les expressions régulières que nous aborderons dans le prochain livret.

– Nous gardons les équivalences en majuscule, ce qui nous obligera dans notre exemple à effectuer un *strtolower*.

Finalisons maintenant, la méthode dédiée au nettoyage et au **renommage** dont nous avons besoin en l'intégrant dans une méthode de la classe *Image* que nous pouvons nommer *cleanText*.

Nous ajoutons donc dans la méthode le *strtolower* puis nous appelons cette méthode *cleanText* depuis la méthode *upload* avec la pseudo-variable *$this*.

```
$filename = $files['upload']['name'][$key];
$filename = $this->cleantText($filename);
```

Fig. 30

Effectuons un test avec un fichier dont le nom initial est **C'est l'été.jpg** et constatons que la vignette ainsi que la grande image sont bien renommées en **cest_lete.jpg**. Ce résultat semble convenir à nos attentes. Si le besoin devait évoluer, il faudrait simplement affiner le script de la méthode *cleanText*.

IV. Suppression des images

Dans le cadre d'une gestion de fichiers, nous pouvons naturellement être amenés à supprimer des fichiers. Pour ce faire, nous avons besoin de deux types d'éléments :
– les fichiers : images et vignettes ;
– les informations de la base.

La méthode *deleteImage* que nous allons créer s'occupera donc de gérer ces deux actions.

Nous réutilisons les constantes *IMAGE_DIR_PATH* et *THUMB_DIR_PATH* pour gérer les chemins d'accès aux fichiers qui possèdent les mêmes noms de fichiers.

La fonction **unlink** est utilisée pour supprimer les fichiers.
Nous utilisons également la fonction PHP **file_exists** pour définir un message d'erreur car elle permet de vérifier si un fichier ou un répertoire existe bien.

Les fonctions *unlink* et *files_exists* sont des fonctions de base à connaître dans le cadre d'une gestion de fichiers avec PHP.

Fonction *unlink* : http://fr2.php.net/manual/fr/function.unlink.php

Fonction *files_exists* : http://php.net/manual/fr/function.file-exists.php

Puisque nous devons effectuer trois suppressions, nous utiliserons par commodité un tableau de données pour gérer les messages d'erreur, ce qui nous permettra d'afficher l'ensemble des erreurs pouvant se produire.

Code de la méthode *deleteImage* :

```
public function deleteImage($filename)
{
// suppression des fichiers : images et vignette
$path_images = IMAGE_DIR_PATH . $filename ;
$path_thumbs = THUMB_DIR_PATH . $filename ;

if (file_exists($path_images))
{
      if(!unlink($path_images))
      {
            $msg_error[] =  'Une erreur est survenue lors de la supression du
fichier image';
      }
}
else
{
      $msg_error[] =  'Le fichier image n\'existe pas';
}

if (file_exists($path_thumbs))
{
      if(!unlink($path_thumbs))
      {
            $msg_error[] =  'Une erreur est survenue lors de la supression du
fichier vignette';
      }
}
else
{
      $msg_error[] =  'Le fichier vignette n\'existe pas';
}

// suppression des données de la base
$mysqli = new mysqli('localhost', 'root', '', 'projet_image');
if ($mysqli->connect_errno) {
      echo 'Echec de la connexion ' . $mysqli->connect_error ;
      exit();
}

if (!$mysqli->query('DELETE FROM image WHERE filename = "'. $filename .'"'))
{
      $msg_error[] =  'Une erreur est survenue lors de la supression des
données dans la base. <br /> Le message d\'erreur de MySQL est : ' .
$mysqli->error;
}

if(isset($msg_error))
{
      $msg_error = implode (' ', $msg_error);
      return $msg_error;
}
else
{
return true;
}
}
```

Fig.31

Notez l'utilisation de la fonction **implode** qui rassemble les éléments d'un tableau en une chaîne et permet donc par la suite d'afficher le contenu du tableau avec un simple *echo*.

Fonction *implode* : http://fr2.php.net/manual/fr/function.implode.php

Nous créons un fichier **process_delete.php** dont le rôle sera d'appeler la méthode et d'en récupérer le résultat.

Code du fichier **process_delete.php** :

```
<?php
if(isset($_GET['delete']))
{
      $filename = $_GET['delete'];
      $image = new Image();
      $deleteImage = $image->deleteImage($filename);
      if(true === $deleteImage)
      {
      $msg_success = 'Le fichier a bien été supprimé.';
      }
      else
      {
      $msg_error = $deleteImage ;
      }
}
```

Fig. 32