



Webdesigner

Une formation de Bruxelles Formation CEPEGRA

Introduction à PHP

Nicolas Bauwens
n.bauwens@bruxellesformation.be

Contenu

Objectifs	3
Langage client vs langage serveur	4
Quelles sont les différences ?	4
Comment ça marche ?	4
Qu'est-ce que ça permet de faire ?	5
Introduction à PHP	6
Outils nécessaires	6
Rendre une page web dynamique	6
Placer les pages PHP dans votre serveur local	6
La balise PHP	7
Les éléments du langage	8
Cas concrets	9
Inclusion de pages	9
Transmettre des données entre les pages	11
Gérer l'envoi d'un formulaire	14
Envoyer un e-mail	17
Introduction aux bases de données	20
MySQL	20
phpMyAdmin	20

Objectifs

Le but de ce module n'est pas de vous faire devenir des *webdeveloper* mais bien de vous faire comprendre les possibilités que le développement dit *back-end* offrent et ainsi comprendre le métier de vos futurs collègues.

Pour joindre l'utile à l'utile, nous verrons des cas concrets qui vont servir d'une grande aide pour comprendre comment WordPress fonctionne, cas que vous pourrez réutiliser dans le cadre de petits projets.

Au terme de ce module, vous serez capable de :

- Comprendre la différence entre langage client et langage serveur
- Créer des « templates »
- Récupérer des données depuis un URL
- Récupérer des données depuis un formulaire
- Envoyer un mail
- Comprendre le rôle de MySQL

Langage client vs langage serveur

Quelles sont les différences ?

Langage client (front-end)

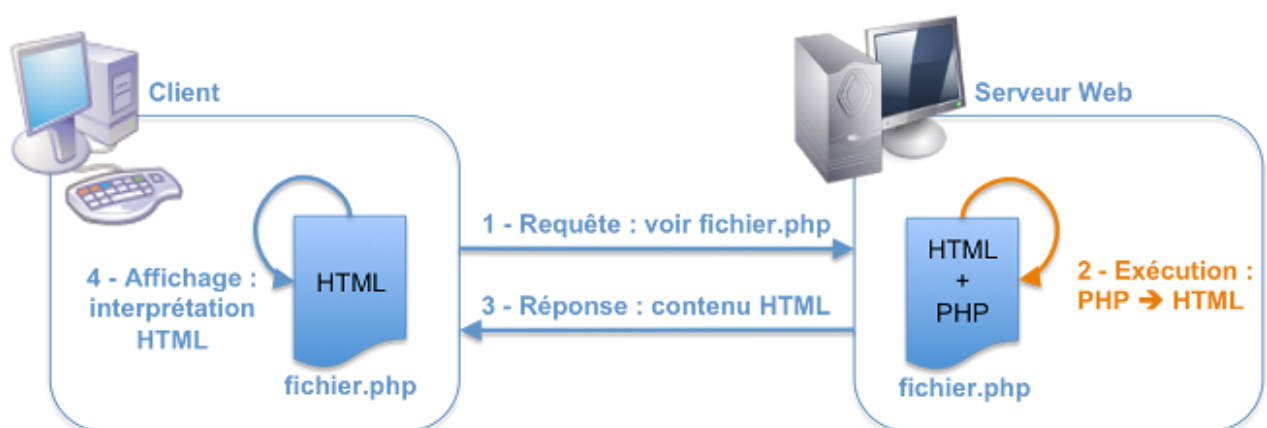
- JavaScript
- Interprété par le navigateur, sur la machine du visiteur
- Modifie l'HTML existant renvoyé par le serveur

Langage serveur (back-end)

- ASP.NET, Java, PHP, ASP, Coldfusion, Python, Ruby, Node.js, ...
- Exécuté/interprété par un serveur web
- Génère de l'HTML
- Indispensable pour :
 - Sauver des informations dans une base de données (si ça ne vous dit rien, voyez ça comme un gros fichier Excel ☺) ou sur le serveur
 - Envoyer des e-mails
 - Gérer la sécurité (login, autorisations,...)
 - Générer du contenu (HTML ou visuel) dynamiquement, c-à-d en fonction de certaines conditions ou critères

Comment ça marche ?

1. La machine client fait une requête vers une page PHP (ou dans une autre technologie *back-end*)
2. Le serveur exécute le code PHP, qui se trouve généralement « inclus » dans une page HTML standard. Le résultat, c'est du code HTML pur et dur.
3. Ce contenu HTML est renvoyé à la machine client
4. Le navigateur interprète la page HTML et l'affiche



Qu'est-ce que ça permet de faire ?

L'avantage principal d'un langage côté serveur c'est qu'il permet de créer des pages web *dynamiques*.

Une seule et même page PHP peut afficher du contenu différent, provenant d'une base de données, en fonction de paramètres qui lui sont envoyés dans l'URL: un ID de produit, un numéro de page, une direction de tri ou des filtres, un terme de recherche... Il suffit de se rendre sur une page produit d'Amazon, sur un profil Facebook, sur un article du Soir pour s'en rendre compte.

C'est aussi un moyen puissant et rapide de générer de longues listes, par exemple une liste d'article sur Amazon ou d'articles sur WordPress : en « bouclant » sur chaque donnée provenant de la base de données et en les formatant avec un bout de code HTML ; il suffit donc de préciser 1 seule fois quel sera le code HTML à répéter.

Mais PHP et les autres langages serveur peuvent faire bien d'autres choses :

- lire des données envoyées par un formulaire
- lire et sauver des informations et des fichiers sur le disque dur du serveur ou une base de données
- envoyer des e-mails
- permettre à des utilisateurs de s'enregistrer et de se connecter au site
- de gérer les droits et la sécurité d'un site internet (+ page 404, 403, etc...)
- exposer des données du site ou accéder à des données exposées par d'autres sites (via [webservices](#))
- générer du contenu HTML différent en fonction du device ([RESS](#))
- prendre plusieurs langues en compte dans le cas de sites multilingues
- etc...

Les CMS comme WordPress ou autre dépendent donc très fort de langages serveur comme PHP.

Introduction à PHP

Outils nécessaires

Pour « faire tourner » du PHP sur votre ordinateur, rien de plus simple, il suffit de le « transformer » en serveur web !

C'est très simple et possible en installant :

- **Apache** : c'est ce qu'on appelle un serveur web. Il s'agit du plus important de tous les programmes, car c'est lui qui est chargé de délivrer les pages web aux visiteurs. Cependant, Apache ne gère que les sites web statiques (il ne peut traiter que des pages HTML). Il faut donc le compléter avec d'autres programmes.
- **PHP** : c'est un plug-in pour Apache qui le rend capable de traiter des pages web dynamiques en PHP. En clair, en combinant Apache et PHP, votre ordinateur sera capable de lire des pages web en PHP.

Et si vous voulez stocker vos données dans une base de données :

- **MySQL** : un logiciel de gestion de bases de données qui fonctionne à merveille avec PHP. Il permet d'enregistrer des données de manière organisée sur le serveur.

Il existe des produits clé en main qui installent tout ça pour vous, comme [WAMP Server](#) ou [EasyPHP](#) ; voici comment [Installer WAMP Server](#)

Il est impossible de connaître toutes les fonctionnalités de PHP, la documentation vous sera donc toute aussi importante, vous pourrez trouver tout ce qu'il vous faut sur <http://www.php.net/manual/fr/index.php>

Rendre une page web dynamique

Placer les pages PHP dans votre serveur local

Une fois tous les logiciels installés, vous pouvez commencer à écrire votre première page PHP.

Wamp Server a créé un répertoire sur votre disque dur, `c:\wamp`. Ce répertoire contient un sous-répertoire `www`. C'est dans ce répertoire `c:\wamp\www` que vous allez placer vos pages PHP.

Les pages placées dans ce répertoire seront accessibles dans votre navigateur via l'adresse **`http://localhost/`**, c'est l'adresse de votre serveur web local.

En gros, si vous placez une page « `contact.php` » dans le répertoire `c:\wamp\www`, vous pourrez y accéder en tapant l'adresse `http://localhost/contact.php` dans votre navigateur.

Vous pouvez créer des sous-dossiers au sein du dossier `www`. La logique suivra celle des URLs à plusieurs niveaux.

La balise PHP

Comment rendre une page HTML statique en une page PHP ? Il suffit simplement de changer l'extension .html en .php ! Mais ça ne fera rien de plus... Si vous voulez inclure du code PHP dans la page, il vous faut inclure ce code entre les **balises PHP**. C'est en voyant cette balise que le serveur saura qu'il doit exécuter le code compris dedans.

```
<?php    <div class="firstName">
          <?php /* Le code PHP se met ici */ ?>
        </div>
```

Il faut aussi que la page soit prise en compte par Apache, le serveur web, il faut donc la placer dans le répertoire approprié (avec Wamp, en général c:\php\www).

- Pour faire écrire du texte à PHP, il faut utiliser l'instruction *echo*.

Le code PHP :

```
<?php    <div class="firstName">
          <?php echo 'Nicolas'; ?>
        </div>
```

Génère l'HTML suivant :

```
</>    <div class="firstName">
        Nicolas
    </div>
```

- ❖ Comme pour JavaScript, il est possible d'afficher la valeur d'une variable

```
<?php    <div class="firstName">
          <?php $firstName = 'Nicolas';
              echo $firstName;
          ?>
        </div>
```



Contrairement à JavaScript, *var* n'est pas utilisé en PHP - pour utiliser ou déclarer une variable, il faut la faire précéder du symbole \$.

- ❖ Il est également possible de générer de l'HTML, comme ceci

```
<?php    <div class="firstName">
          <?php echo '<span>Nicolas</span>'; ?>
        </div>
```

- ❖ Enfin, il est possible de générer de l'HTML qui contiendrait la valeur d'une (ou plusieurs) variables :

- Soit en *concaténant* (en « collant » les bouts de texte ensemble) l'HTML et les variables

```
<?php <div class="firstName">
    <?php $firstName = 'Nicolas';
        echo '<span>'. $firstName .'
```



Contrairement à JavaScript, le symbole `.` est utilisé pour la *concaténation*, au lieu du `+`

- Soit, plus simple, en incluant les variables dans le texte HTML **mais vous devez utiliser impérativement les guillemets !**

```
<?php <div class="firstName">
    <?php $firstName = 'Nicolas';
        echo "<span>$firstName</span>"; ?>
</div>
```

Les éléments du langage

PHP est un langage de programmation, au même titre que JavaScript. Et il permet de faire les mêmes choses :

- Déclarer et assigner des variables, de type
 - Numérique
 - Texte
 - Booléen
 - Array
 - Objet
- D'utiliser des opérateurs de comparaison (plus grand, plus petit,...) et logique (et, ou, contraire)
- Définir des conditions de type if-else/etc...
- Définir des boucles while/for
- Déclarer et appeler des fonctions

Ces concepts ayant été vu dans le cours JavaScript, ils ne seront pas abordés ici. Par contre, bien que JavaScript et PHP se ressemblent très forts au niveau de la syntaxe, il existe certaines différences dans la syntaxe et l'implémentation de certaines fonctionnalités. Nous en verrons lors des différents exercices.

Cas concrets

Inclusion de pages

Si votre site contient sur toutes les pages :

- la même navigation ou header/footer
- les mêmes CSS
- les mêmes JavaScripts
- etc...

Ça devient vite barbant de copier-coller ces blocs HTML et de devoir passer dans toutes les pages pour le modifier.

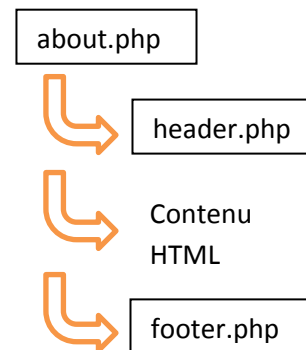
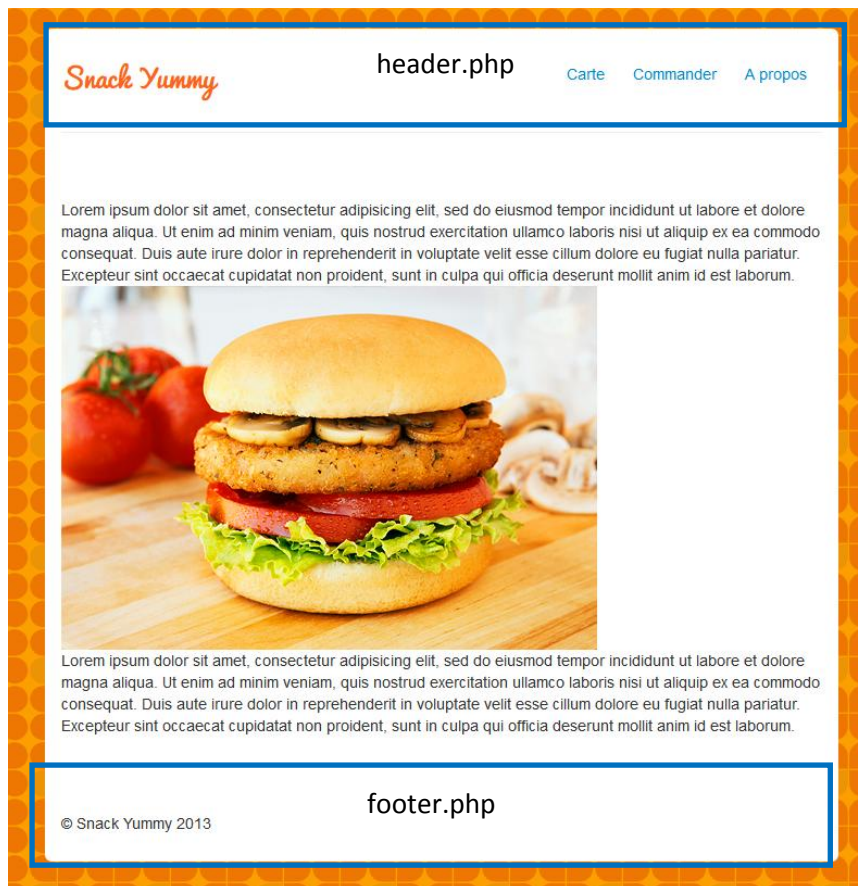
Une possibilité intéressante de PHP, c'est de générer des pages HTML à partir de plusieurs « blocs », ou sous-pages.

Imaginons une page « about.php », elle peut inclure des portions d'autres pages pour, au final, générer le code HTML de la page complète.

Elle pourrait faire appel à :

- header.php qui comprend le code HTML du header
- footer.php qui comprend le code HTML du footer (et pourquoi pas les balises scripts)

Au milieu de ces 2 inclusions, elle comprendrait le contenu HTML propre à elle.




Dans la pratique, il vous suffit d'utiliser l'instruction *include* à l'endroit où vous voulez que la portion de code HTML s'affiche

<code><?php</code>	<code><?php include("header.php"); ?></code>
-----------------------	--

Cette instruction dit : « **Insère ici le contenu de la page `header.php`** ».

Le chemin entre guillemets est soumis aux mêmes principes qu'en HTML, si voulez mettre vos portions de page dans un sous-dossier nommé *includes* il vous faudra écrire :

<code><?php</code>	<code><?php include("includes/header.php"); ?></code>
-----------------------	---

	<p>A noter qu'une page dont l'extension est <code>.php</code> peut très bien ne contenir aucune balise PHP. Dans ce cas, cela redevient une page HTML classique qui n'est pas modifiée avant l'envoi.</p>
---	---

Transmettre des données entre les pages

Envoyer des paramètres dans l'URL

Vous avez déjà certainement rencontré des adresses de ce genre :

<https://www.google.be/search?q=licorne&source=lnms&tbm=isch>
http://www.cssdesignawards.com/search.php?search_term=V%26M

Toutes les données qui suivent le ? sont les données qui sont envoyées à la page.

Il peut n'y en avoir qu'une, comme il peut y en avoir plusieurs, il n'y a pas de limites. Dans ce cas, chaque donnée est séparée par un &

`http://www.mysite.com/results.php?lang=fr&page=2`

La partie avant le = symbolise le **nom** de la donnée, et ce qui suit le =, la **valeur** de la donnée. Dans cet exemple, 2 paramètres sont envoyés à la page results.php :

- *lang*, qui représente la langue choisie par l'utilisateur – ici « fr » pour Français
- *page*, qui représente le numéro de page à montrer dans le cas d'une liste d'éléments affichés – ici 2

En fonction des paramètres et de leur valeur, la page affichera des informations différentes. C'est donc bien pratique pour mettre en place un **comportement générique** et faire prendre en charge chaque cas particulier par PHP.

Récupérer les paramètres dans l'URL

Maintenant qu'on sait comment envoyer des données, reste à savoir comment la page cible peut les récupérer.

Cela est possible via un *array* un peu spécial qui s'appelle `$_GET`. Chaque « case » de cet *array* contient un paramètre qui a été envoyé dans l'URL. On peut accéder à la valeur du paramètre via son nom, comme ceci

<code><?php</code>	<code><?php echo \$_GET['lang']; ?></code>
-----------------------	--

Voice un exemple avec un switch, pour afficher un message différent en fonction de la langue passée en paramètre à la page

<code><?php</code>	<pre>switch(\$_GET['lang']){ case 'fr': echo 'Bonjour, monde !'; break;</pre>
-----------------------	---

	<pre> case 'nl': echo 'Dag, wereld !'; break; default: echo 'Hello, world !'; break; } </pre>
--	---

« Transformer » les données

Dans le cas du paramètre « page », on récupèrera via `$_GET` le **texte « 2 »**, ce qui peut ne pas correspondre avec le type attendu ; ici on s’attend plutôt à ce que « page » soit plutôt le **chiffre 2** – ce qui nous permettrait de calculer quelle est la page précédente (« page – 1) et suivante (« page » + 1).

C’est possible de « transformer » une donnée d’un type vers une autre, ici, de texte à numérique. Cela peut s’écrire de cette manière :

<code><?php</code>	<code><?php \$_GET['page'] = (int)\$_GET['page']; ?></code>
-----------------------	---

On a précisé via ce **(int)** (*int* est le raccourci pour *integer* en anglais, ou “nombre entier” en français) qu’on veut « transformer » le paramètre « page » de texte (son type original) vers un nombre entier.

Si « page » venait à contenir des caractères autres que des nombres (par exemple « page » vaudrait « blalblalba »), le résultat de la transformation donnera tout simplement 0.

Validation des données

Etant donné que les données se trouvent dans l’URL, n’importe qui peut les changer... ou les enlever ! Ce qui comporte plusieurs risques :

- Votre page risque de donner un résultat inattendu voire carrément une grosse erreur
- Des petits filous pourraient essayer de hacker votre site en *injectant* dans l’URL des choses pas très catholiques comme du JavaScript et d’autres joyeusetés.

Il convient donc entre autres de vérifier si :

- Les données « requises » pour que la page puisse fonctionner sont présentes, via la fonction **isset()**
- Les données ont le bon format, en appliquant des [transformations de types](#)
- Les données ne contiennent pas de balises `<script>` en utilisant la méthode **htmlspecialchars()**

isset()

La fonction `isset()` renverra *true* si le paramètre est bien dans l'URL et *false* sinon. A utiliser dans un `if`, de cette manière :

<code><?php</code>	<code><?php if(isset(\$_GET['lang'])){ //code à executer } ?></code>
-----------------------	--

htmlspecialchars()

`htmlspecialchars()` va transformer les chevrons des balises HTML `<>` en `<` et `>` respectivement. Cela provoquera l'affichage de la balise plutôt que son exécution. Par exemple, faire ceci

<code><?php</code>	<code><?php echo htmlspecialchars("<script>alert('coucou')</script>"); ?></code>
-----------------------	--

Encoder le texte "`<script>alert('coucou')</script>`" comme ceci, une fois l'HTML généré :

`<script>alert('coucou')</script>`

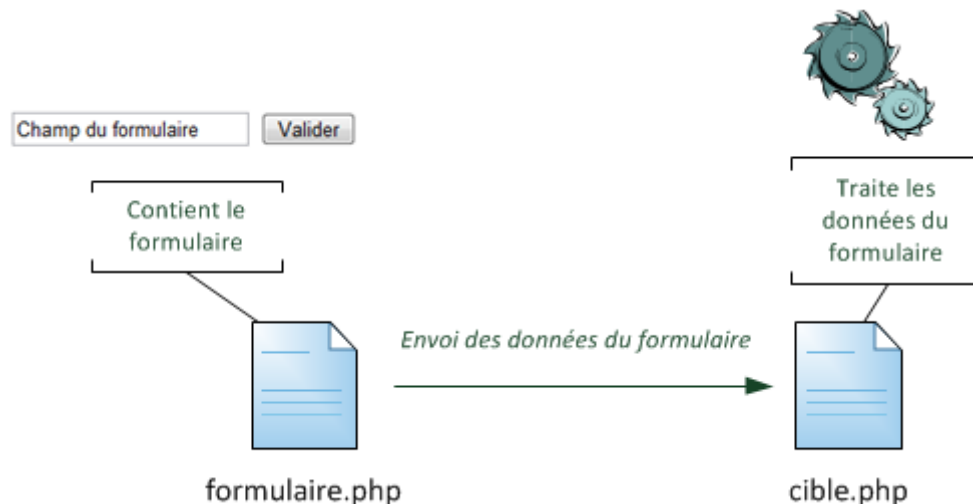
Ce qui aura pour but d'afficher le texte "`<script>alert('coucou')</script>`" dans la page et non pas d'exécuter le script malicieux !

Gérer l'envoi d'un formulaire

Vous avez déjà vu comment créer un formulaire en HTML, comment le valider en JavaScript, mais nous nous sommes arrêtés là.

Comment « envoyer » les données du formulaire pour pouvoir les traiter ?

Il faut dire à notre formulaire qu'il poste les données vers une page PHP, c'est cette page PHP qui va récupérer les données et ensuite envoyer un mail, les sauvegarder en base de données, ou autre.



Cela se fait via 2 attributs de la balise `<form>` : **action** et **method**

```
</> <form method="post" action="cible.php">

    <p>
        On insèrera ici les éléments de notre formulaire.
    </p>

</form>
```

- **action** va préciser à quelle page il faut envoyer les données, il faut donc insérer l'URL d'une page qui existe sur votre site, rien de plus.
- **method** va préciser comment envoyer les données. Cet attribut peut avoir comme valeur :
 - **get** les données du formulaire seront envoyées à la page cible via l'URL, comme vu dans le chapitre [Envoyer des données dans l'URL](#). Cette méthode est rarement utilisée.
 - **post** les données du formulaire seront envoyées vers la page cible mais ne seront pas visibles dans l'URL, c'est cette méthode qui est le plus souvent utilisée. **Attention**, ça n'est pas parce que les données ne sont pas visibles dans la barre d'adresse qu'elles ne sont pas modifiables par des petits filous ! 😊

Valider les données du formulaire

Même si vous avez mis en place un système de validation *côté client* en JavaScript, rien ne vous permet de savoir, une fois que vous souhaitez traiter les données en PHP, qu'elles sont valides.

En effet, le visiteur pourrait avoir désactivé le JavaScript, ou une personne mal intentionnée aurait pu modifier les données envoyées, même si la méthode *post* a été utilisée – c'est possible via des outils comme Fiddler.

Comme dans le cas de [transmission de données par l'URL](#), il faudra impérativement vérifier la validité des données envoyées par le formulaire.

En résumé, nous pouvons dire :

Validation client (JavaScript) -> expérience utilisateur et premier filtrage

Validation serveur (PHP) -> pour s'assurer de l'intégrité des données

Récupérer les données d'un formulaire

La démarche est similaire à la récupération de données dans l'URL. Si *method* est mis à **get** dans votre formulaire, vous pouvez de nouveau utiliser l'array **\$_GET**.

Si maintenant vous utilisez plutôt *post*, ça sera possible via l'array **\$_POST**. Il s'utilise de la même manière que **\$_GET**.

Par exemple, si vous avez dans votre formulaire comme ceci :

```
</> <form action="pacman.php" method="get">
      <label>Nombre de fantômes</label>
      <input type="text" name="ghosts">
      <button type="submit">GO PACMAN !</button>
    </form>
```

Dans la page `pacman.php`, nous pourrions récupérer le contenu du champ `input` via **\$_GET['ghosts']** – le nom à préciser dans l'array correspond à la valeur du champ *name*.

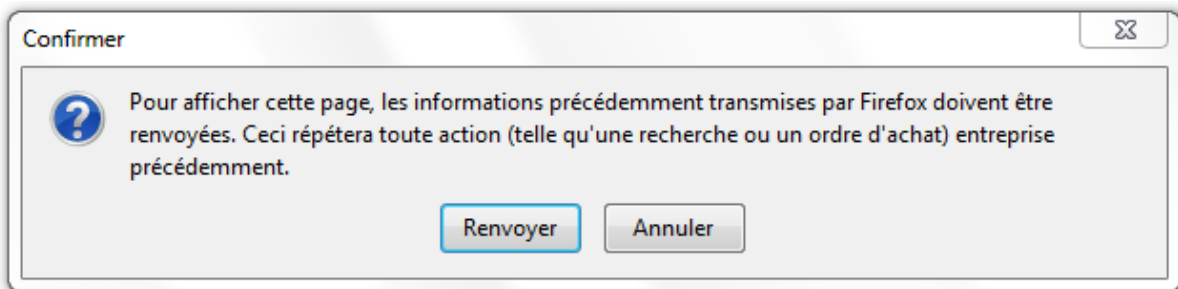
Si vous utilisez, *post*, comme ceci :

```
</> <form action="pacman.php" method="post">
      <label>Nombre de fantômes</label>
      <input type="text" name="ghosts">
      <button type="submit">GO PACMAN !</button>
    </form>
```

Dans la page pacman.php, nous pourrions récupérer le contenu du champ input via `$_POST['ghosts']` – le nom à préciser dans l'*array* `$_POST` correspond à la valeur du champ *name*.

Rediriger vers une autre page pour éviter le double envoi

Vous avez déjà certainement faire face à ce scénario : vous avez envoyé des données à un site web via un formulaire, et puis pour une raison ou une autre, vous rafraichissez la page et voyez ce message d'alerte



Pas très rassurant n'est-ce pas ? On se demande ce qu'il va se passer ! La commande sera envoyée 2 fois ? Est-ce que j'aurai acheté 4 billets de concerts au lieu de 2 ? Gloups...

Tout dépendra de comment le développeur aura conçu son site web, mais ce genre de comportement n'est pas impossible.

Pour éviter que vos utilisateurs soient confrontés à ce genre de problème, il faut mettre en place une redirection lorsque l'action effectuée par votre envoi de formulaire s'est bien passée et est terminée.

Ce type de fonctionnement est ce qu'on appelle le [Post-Redirect-Get](#), le principe est le suivant :

1. L'utilisateur clique sur le bouton « Envoyer » de votre formulaire
2. Une requête « POST » est envoyée vers le serveur, et la page PHP correspondante va traiter l'envoi de formulaire (envoi de mail, insertion en base de données, etc...)
3. La page PHP a fait le boulot demandé, elle redirige l'utilisateur vers une autre page, une simple page de confirmation

Si l'utilisateur rafraichit la page de confirmation, comme elle n'est plus liée au formulaire, la page ne fera que se rafraichir.

Comment l'implémenter facilement dans votre code ? Via du JavaScript ! ☺ Voici un exemple :

```
<?php
<?php
    //code d'envoi du formulaire, la variable showConfirmation me permet
    de savoir si l'envoi est OK
    if($showConfirmation == false){
?>
        //code HTML du formulaire
    <?php
    } else {
?>
        <script type='text/javascript'>
            document.location.href = "confirmation.html";
        </script>
    <?php
    }
?>
```

Envoyer un e-mail

Une des fonctionnalités intéressante des langages serveur comme PHP, c'est d'envoyer des e-mails.

Pour envoyer un mail, il faut fournir plusieurs informations à PHP :

- Un e-mail expéditeur (*from*)
- Un ou plusieurs e-mails destinataires (*to*)
- Un sujet (*subject*)
- Un message (*body*), en texte ou en HTML
- Des attachements éventuels
- L'adresse d'un serveur SMTP

Un serveur SMTP, c'est un serveur capable d'envoyer des e-mails. Via votre hébergeur, il est normalement possible d'utiliser le serveur SMTP de l'hébergeur – il faudra peut-être indiquer des *credentials* (login et mot de passe pour que le serveur SMTP accepte votre demande d'envoi).

Au niveau technique, nous allons utiliser le plug-in **PHPMailer**

[Tutoriel PHPMailer](#)

[Page GitHub](#)

Installer PHPMailer

Le .zip de PHPMailer contient à la fois le code du plug-in, mais aussi de la documentation et plein d'exemples. Par soucis de facilité, gardez tout le contenu du zip et extrayez-le dans un sous-dossier « phpmailer » dans votre site web.

Une fois que c'est fait, vous devez « inclure » le plug-in dans votre code PHP. Pour ce faire, mettez cette ligne dans votre code, **avant** le code qui sera utilisé pour envoyer le mail.

```
<?php require("phpmailer/class.phpmailer.php"); ?>
```

Configurer l'envoi de mail

Nous sommes prêts à configurer l'envoi d'e-mail, c'est-à-dire fournir toutes les informations nécessaires pour que le mail parte. Le code ci-dessous va envoyer un mail depuis « info@monsite.com » à « pacman@hotmail.com » avec comme sujet « Confirmation de votre commande ». Le corps du message (*body*) sera en HTML et non en texte pur.

Vous pouvez bien sûr utiliser des variables en lieu et place de texte entre guillemets.

Le serveur SMTP utilisé est celui de Bruxelles Formation (« smtp.brufor.be »), attention, ça ne marchera que sur le réseau de Bruxelles Formation, à vous de trouver l'adresse du serveur SMTP de votre hébergeur ☺

```
<?php
<?php

$mail = new PHPMailer();
$mail->IsSMTP();
$mail->SMTPDebug = 0;
$mail->Host = "smtp.brufor.be";
$mail->Port = 25;
$mail->SetFrom("info@monsite.com");
$mail->AddAddress("pacman@hotmail.com");
$mail->IsHTML(true);
$mail->Subject = "Confirmation de votre commande";
$mail->Body = "<h1>Merci</h1><p>Et bonne journée</p>";

?>
```

Envoyer le mail

Maintenant que le mail est configuré, il suffit de l'envoyer, grâce à la méthode `Send()`.

Evidemment, il se peut que l'envoi de mail ne se passe pas comme prévu, par exemple si le serveur SMTP ne répond pas.

C'est possible de le savoir, et de tester ce cas de figure, en effet la méthode `Send()` renverra *true* si l'envoi a pu avoir lieu, et *false* sinon. L'erreur exacte, si ça vous intéresse, est disponible via la propriété `ErrorInfo`.

Dans le cas où une erreur survient, il est préférable d'en informer le visiteur (par exemple avec un message « Nous n'avons pas pu vous envoyer de mail »). Tout comme un message de confirmation est également le bienvenu.

<?php	<pre data-bbox="320 183 1420 674"><?php if(!\$mail->Send()) { echo "Erreur lors de l'envoi du mail: " . \$mail->ErrorInfo; } else { echo "Votre e-mail est bien parti !"; } ?></pre>
-------	--

Introduction aux bases de données

MySQL

MySQL est ce qu'on appelle un SGBD, ou *système de gestion de bases de données*, mais on préfère le terme plus court « base de données ».

Une base de données permet de stocker dans un seul endroit des informations et les liens entre ces informations. Il existe des concurrents à MySQL comme Oracle ou SQL Server ou même Access.

Un CMS comme WordPress dépend fortement d'une base de données pour pouvoir stocker toutes les pages, images et autres fichiers encodés dans WordPress par le webmaster. C'est un élément incontournable d'un site dynamique.

La lecture et l'écriture dans la base de données (on peut la voir comme un gros fichier) se fait via PHP.

phpMyAdmin

phpMyAdmin est un outil, une interface, installé lorsque vous installez WAMP Server, qui vous permet d'accéder à MySQL afin d'aller :

- Créer la structure de la base de données MySQL
- De visualiser les données
- D'ajouter des données
- De modifier/effacer les données
- Etc...

Si vous utiliser un CMS comme WordPress, vous allez rarement utiliser phpMyAdmin, car WordPress créer la base de données tout seul et vous n'avez guère d'intérêt à aller plus loin. Mais cet outil se révèle très utile lorsque vous faites un site qui n'utilise pas de CMS mais qui nécessiterait une base de données.

Pour accéder à phpMyAdmin, cliquez sur l'icône de Wamp Server dans la barre des tâches et cliquez sur « phpMyAdmin ».

Pour se connecter, le nom d'utilisateur de l'administrateur (vous) est *root*, et par défaut, il n'y a aucun mot de passe.

Si ce n'est pas très grave pour du développement, il faudra impérativement [changer ce mot de passe](#) dès l'instant où vous mettrez votre site en ligne !