

# PHP-MySQL\*

## Partie 2

### R3.01 : Développement web - TP2

## 1 Écoutez la requête de vos utilisateurs grâce aux URL

Grâce aux URL et à PHP, nous allons rendre le formulaire de contact (du site de recettes) dynamique en retournant une page de prise en compte de la demande à chaque personne qui soumettra le formulaire.



URL signifie Uniform Resource Locator. C'est en fait une adresse sur le Web.

Toutes les adresses que vous voyez en haut de votre navigateur, comme : `//www.google.com` sont des URL.

Lorsque vous faites une recherche sur Google pour trouver par exemple le site de Wikipedia en tapant le mot "Wikipedia", la barre d'adresse contient une URL un peu longue qui ressemble à ceci :

`https://www.google.com/search?q=wikipedia&rlz=1C1LZQR_frFR1070FR1070&oq=wikipedia...`

Les informations après le point d'interrogation ? sont en réalité des données que l'on fait transiter d'une page à une autre.

### 1.1 Envoyez des paramètres dans l'URL

#### 1.1.1 Formez une URL pour envoyer des paramètres

Imaginons que votre site s'appelle `www.monsite.com` et qu'il possède une page PHP de contact `contact.php`.

Pour accéder à la page de contact, vous devez aller à l'URL suivante :

`http://www.monsite.com/contact.php`

Nous allons envoyer des informations à la page `contact.php`.

Ces informations vont figurer à la fin de l'URL, comme ceci :

`http://www.monsite.com/contact.php?nom=Dupont&prenom=Jean`

---

\*<https://openclassrooms.com/fr/courses/918836-concevez-votre-site-web-avec-php-et-mysql>



Ce que vous voyez après le point d'interrogation, ce sont des **paramètres** que l'on envoie à la page PHP.

Cette dernière récupère ces informations dans des variables dites « superglobale ».

Le point d'interrogation sépare le nom de la page PHP des paramètres.

Les paramètres s'enchaînent selon la forme **nom=valeur** et sont séparés les uns des autres par le symbole **&**.

On peut théoriquement écrire autant de paramètres que l'on veut.

Il suffit de les séparer par des **&**. On peut donc voir une URL de la forme :

```
page.php?param1=valeur1&param2=valeur2&param3=valeur3&param4=valeur4...
```

La seule limite est la longueur de l'URL.

En général, il n'est pas conseillé de dépasser les 256 caractères, mais les navigateurs arrivent parfois à gérer des URL plus longues.

### 1.1.2 Créez un lien avec des paramètres

Nous pouvons donc maintenant créer des liens en HTML, qui transmettent des paramètres d'une page vers une autre.

Imaginons que vous ayez deux fichiers sur votre site : **index.php** (l'accueil) et **bonjour.php**.

Nous voulons faire un lien de **index.php** à **bonjour.php** pour transmettre des informations dans l'URL :

Pour cela, créez un fichier **index.php** et insérez-y par exemple le code suivant :

```
<a href="bonjour.php?nom=Dupont&prenom=Jean">Dis-moi bonjour !</a>
```



Comme vous le voyez, le **&** dans le code a été remplacé par **&amp;** dans le lien.

Ça n'a rien à voir avec PHP : simplement, en HTML, on demande à ce que les **&** soient écrits **&amp;** dans le code source.

Si vous ne le faites pas, le code ne passera pas la validation W3C.

Ce lien appelle la page **bonjour.php** et lui envoie deux paramètres : **nom** de valeur **Dupont** et **prenom** de valeur **Jean**.

### 1.1.3 Créez un formulaire avec la méthode HTTP GET

La deuxième solution pour faire passer des informations dans l'URL, c'est de proposer à l'utilisateur de soumettre un formulaire avec la méthode HTTP GET.

Nous pouvons faire cela avec une balise `form` qui a pour l'attribut `method` la valeur GET.

```
<form action="contact.php" method="GET">
  <!-- données à faire passer à l'aide d'inputs -->
  <input name="nom">
  <input name="prenom">
</form>
```

Les données soumises à l'aide du formulaire se retrouveront dans l'URL, comme pour un lien.

Nous allons voir maintenant comment faire dans la page `contact.php` ou dans `bonjour.php` pour récupérer ces informations.

## 1.2 Récupérez les paramètres en PHP

Nous savons maintenant comment faire pour envoyer des paramètres vers une autre page.

Intéressons-nous maintenant à la page qui réceptionne les paramètres.

Pour notre besoin, nous avons un formulaire de contact (`contact.php`) que nous allons soumettre à une autre page (`submit_contact.php`) qui affichera un message de bonne réception.

Voici le formulaire de contact `contact.php` de la séance précédente avec deux modifications :

- la balise `form` est complétée ;
- le nom de la balise `textarea` contenant le message s'appelle désormais `message`.

```
<form action="submit_contact.php" method="GET">
  <div>
    <label for="email">Email</label>
    <input type="email" name="email">
  </div>
  <div>
    <label for="message">Votre message</label>
    <textarea placeholder="Exprimez vous" name="message"></textarea>
  </div>
  <button type="submit">Envoyer</button>
</form>
```

Par exemple, si l'utilisateur complète le formulaire de contact avec pour e-mail « utilisateur@exemple.com » et comme message « Bonjour », alors le formulaire va être converti en lien vers :  
`submit_contact.php?email=utilisateur%40exemple.com&message=Bonjour`

Et ces informations pourront être récupérées par PHP dans le fichier `submit_contact.php`.



Lors de la soumission, une variable [superglobale](#) appelée `$_GET` va contenir les données envoyées.  
Il s'agit d'un tableau associatif dont les clés correspondent aux noms des paramètres envoyés dans l'URL : `$_GET['email']` a pour valeur `'utilisateur@exemple.com'` et `$_GET['message']` a pour valeur `'Bonjour'`.

On peut donc récupérer ces informations, les traiter, les afficher, bref, faire ce que l'on veut avec.

### 1.2.1 Travail à faire



Pour le site de recettes, créez une copie du fichier `index.php` et renommez la `submit_contact.php`. Puis placez dans ce nouveau fichier le code suivant de manière à obtenir le résultat ci-après lorsqu'on remplit et valide le formulaire de la page `contact.php`.

```
<h1>Message bien reçu !</h1>

<div class="card">
  <div class="card-body">
    <h5 class="card-title">Rappel de vos informations</h5>
    <p class="card-text"><b>Email</b> : <?php echo $_GET['email']; ?></p>
    <p class="card-text"><b>Message</b> : <?php echo $_GET['message']; ?></p>
  </div>
</div>
```

Site de recettes



## Site de recettes

### Message bien reçu !

#### Rappel de vos informations

**Email** : alexandre.brabant@wanadoo.fr

**Message** : Bonjour



Créez une nouvelle version de votre projet avec l'enchaînement de commandes ci-après :

```
> git add --all  
> git commit -m "Récupérez les paramètres en PHP"  
> git push -u origin main
```

## 1.3 Ne faites jamais confiance aux données reçues !



| Il ne faut jamais faire confiance aux variables qui transitent de page en page, comme `$_GET`.

### 1.3.1 Tous les visiteurs peuvent trafiquer les URL

Si vous faites les tests des codes précédents, vous devriez tomber sur une URL de la forme :

`http://localhost/submit_contact.php?email=utilisateur%40exemple.com&message=Bonjour`

Essayez par exemple de modifier l'adresse pour :

`http://localhost/submit_contact.php?email=utilisateur%40exemple.com&message=Mouahah`

Vous devez obtenir :



| Cela montre qu'on ne peut pas avoir confiance dans les données qu'on reçoit.  
N'importe quel visiteur peut les changer.

### 1.3.2 Testez la présence d'un paramètre

Allons plus loin. Qu'est-ce qui empêche le visiteur de supprimer tous les paramètres de l'URL ?

Par exemple, il peut très bien tenter d'accéder à :

`http://localhost/submit_contact.php`


Faites le test ! Le navigateur va afficher quelque chose comme :

[Site de recettes](#) [Home](#) [Contact](#)

## Site de recettes

### Message bien reçu !


**Rappel de vos informations**  
**Email :**

 **Notice: Undefined index: email in C:\Users\alexa\UwAmp\www\php-mysql\submit\_contact.php on line 33**

**Call Stack**

#	Time	Memory	Function	Location
1	0.0011	360704	{main}()	...\submit_contact.php:0

**Message :**

 **Notice: Undefined index: message in C:\Users\alexa\UwAmp\www\php-mysql\submit\_contact.php on line 34**

**Call Stack**

#	Time	Memory	Function	Location
1	0.0011	360704	{main}()	...\submit_contact.php:0

© 2021 Copyright: [Alexandre Brabant](#)

Explication :

On a essayé d'afficher la valeur de `$_GET['email']` et celle de `$_GET['message']`.

Mais comme on vient de les supprimer de l'URL, ces variables n'ont pas été créées, et donc elles n'existent pas ! PHP nous avertit qu'on essaie d'utiliser des variables qui n'existent pas, d'où les « Undefined index ».

Pour résoudre ce problème, on peut faire appel à une fonction un peu spéciale : `isset()`.



| La fonction `isset()` teste si une variable existe.

Nous allons nous en servir pour afficher un message spécifique, dans le cas où le nom ou le prénom serait absent.

Le code ci-après teste si les variables `$_GET['email']` et `$_GET['message']` existent.

- Si elles existent, on affiche la confirmation de prise en compte du message.
- S'il nous manque une des variables (ou les deux), on affiche un message d'erreur : « Il faut un e-mail et un message pour soumettre le formulaire », et on arrête l'exécution de la page.

## Travail à faire



Appliquez ce code à votre script `submit_contact.php` pour bien gérer le cas où le visiteur aurait retiré les paramètres de l'URL.

Essayez ensuite d'accéder à la page `submit_contact.php` sans les paramètres :

Site de recettes



### Site de recettes

**Il faut un email et un message  
pour soumettre le formulaire.**



Créez une nouvelle version de votre projet avec l'enchaînement de commandes ci-après :

```
> git add --all
> git commit -m "Testez la présence d'un paramètre"
> git push -u origin main
```

### 1.3.3 Contrôlez la valeur des paramètres

Une fois les valeurs soumises et correctement récupérées, il faut vérifier qu'elles correspondent à ce qui est attendu !

Imaginez par exemple que l'utilisateur se soit trompé et que l'e-mail envoyé ne soit pas valide : vous ne pourrez donc pas le recontacter pour répondre à son besoin !

Il va donc falloir :

1. contrôler si l'e-mail passé est bien valide, à l'aide de la fonction [filter\\_var](#) ;
2. et vérifier que le message n'est pas vide, à l'aide de la fonction [empty](#).

Voici donc le code final sécurisé, qui prévoit tous les cas pour éviter d'être pris au dépourvu par un utilisateur mal intentionné :

```
<?php
if (
    (!isset($_GET['email']) || !filter_var($_GET['email'], FILTER_VALIDATE_EMAIL))
    || (!isset($_GET['message']) || empty($_GET['message'])))
)
{
    echo('Il faut un email et un message valides pour soumettre le formulaire.');
```



Nous n'avons pas détaillé le fonctionnement des fonctions `filter_var` et `empty`.

N'hésitez pas à consulter tous [les exemples](#) de la documentation PHP pour bien comprendre son fonctionnement.

## Travail à faire



Appliquez ce code à votre script `submit_contact.php` pour bien contrôler la valeur des paramètres.

Testez.



Créez une nouvelle version de votre projet avec l'enchaînement de commandes ci-après :

```
> git add --all  
> git commit -m "Contrôlez la valeur des paramètres"  
> git push -u origin main
```

## 2 Administrez des formulaires de façon sécurisée

Les formulaires constituent le principal moyen pour vos visiteurs d'entrer des informations sur votre site. Dans la section précédente, nous avons d'ailleurs abordé un premier formulaire de contact assez basique pour envoyer une URL avec des paramètres.

Dans cette section, vous allez apprendre à faire des formulaires plus sécurisés.

Cette section est particulièrement importante ; nous réutiliserons ce que nous avons appris ici dans toute la suite du cours. Soyez attentif !

### 2.1 Rappelez-vous : la base du formulaire



En HTML, pour insérer un formulaire, on se sert de la balise `<form>`.  
On l'utilise de la manière suivante :



```
<!-- index.php -->
<form method="post" action="submit_form.php">

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

</form>
```

Il y a deux attributs très importants à connaître pour la balise `<form>` :

- La méthode : `method`
- Et la cible : `action`

### 2.1.1 Privilégiez la méthode `post` pour envoyer les données du formulaire



Il existe deux méthodes pour envoyer les données du formulaire :

- `get` : les données transiteront par l'URL, comme on l'a vu précédemment. On pourra les récupérer grâce au tableau (array) `$_GET`.  
Cette méthode est assez peu utilisée car on ne peut pas envoyer beaucoup d'informations dans l'URL (il est préférable de ne pas dépasser 256 caractères).
- `post` : les données ne transiteront pas par l'URL, l'utilisateur ne les verra donc pas passer dans la barre d'adresse.  
Cette méthode permet d'envoyer autant de données que l'on veut, ce qui fait qu'on la privilégie le plus souvent.  
Néanmoins, les données ne sont pas plus sécurisées qu'avec la méthode `GET`, et il faudra toujours vérifier si tous les paramètres sont bien présents et valides, comme on l'a fait dans le chapitre précédent.  
On ne doit pas plus faire confiance aux formulaires qu'aux URL.

La bonne pratique consiste généralement à privilégier la méthode `post` pour les formulaires.

### Travail à faire



Modifiez vos scripts `contact.php` et `submit_contact.php` pour que la méthode `post` soit utilisée à la place de la méthode `get`.  
Testez.



Créez une nouvelle version de votre projet avec l'enchaînement de commandes ci-après :

```
> git add --all
> git commit -m "Privilégiez la méthode post"
> git push -u origin main
```

### 2.1.2 Choisissez la page appelée par le formulaire en définissant la cible



L'attribut `action` sert à définir la page appelée par le formulaire. C'est cette page qui recevra les données du formulaire, et qui sera chargée de les traiter.

Selon la méthode utilisée, ce ne sera pas la même variable spéciale qui aura accès aux données :

- Si la méthode est `GET` alors c'est la supervariable `$_GET` qui aura les données ;
- Si la méthode est `POST` (bonne pratique), alors c'est la supervariable `$_POST` qui recevra les données.

Retenez donc bien que vous travaillez normalement sur deux pages différentes :

- La page qui contient le formulaire (`contact.php` dans notre projet fil rouge) ;
- Et celle qui reçoit les données du formulaire pour les traiter (`submit_contact.php`).

### 2.1.3 Ajoutez un ou plusieurs champs `input` avec un attribut `name`



La variable `$_GET` (ou `$_POST`) sera complétée avec la valeur de l'attribut `name` en tant que clé, et aura pour valeur ce qui aura été soumis par l'utilisateur :

```
<input type="text" name="nom" value="Mateo21" />

<?php

// Après soumission du formulaire
echo $_GET['nom']; // "Mateo21"

// OU

echo $_POST['nom']; // "Mateo21"
```

Il y aura autant de clés dans le tableau `$_GET` que de champs avec un attribut `name` dans le formulaire soumis.

### 2.1.4 Faites attention avec les champs cachés



Un champ caché est un code dans votre formulaire qui n'apparaîtra pas aux yeux du visiteur, mais qui va quand même créer une variable avec une valeur. On peut s'en servir pour transmettre des informations fixes.

Par exemple, supposons que vous ayez besoin de retenir que le pseudo du visiteur est « Mateo21 ». Vous allez taper ce code :

```
<input type="hidden" name="pseudo" value="Mateo21" />
```

À l'écran, sur la page web on ne verra rien. Mais dans la page cible, une variable `$_POST['pseudo']` sera créée, et elle aura la valeur « Mateo21 ».

C'est apparemment inutile, mais vous verrez que vous en aurez parfois besoin.



On croit par erreur que, parce que ces champs sont cachés, le visiteur ne peut pas les voir. C'est faux ! En effet, n'importe quel visiteur peut afficher le code source de la page pour voir et modifier les champs cachés en lisant le code HTML.

## 2.2 Ne faites jamais confiance aux données reçues : la faille XSS

Les mises en garde que faites dans la section précédente ne concernent pas que les paramètres qui transitent par l'URL : tout cela vaut aussi pour les formulaires !

Vous avez vu comment on peut modifier l'URL, maintenant vous allez voir comment peut faire un visiteur pour modifier le formulaire de mon site et trafiquer les données.

### 2.2.1 La faille XSS : attention au code HTML que vous recevez !



La faille XSS (pour cross-site scripting) est une technique qui consiste à injecter du code HTML contenant du JavaScript dans vos pages, pour le faire exécuter à vos visiteurs.

Par exemple, reprenons la page de récapitulatif qui affiche l'e-mail et le message qu'on a soumis dans la section précédente :

```
<h5>Rappel de vos informations</h5>
<p><b>Email</b> : <?php echo $_POST['email']; ?></p>
<p><b>Message</b> : <?php echo $_POST['message']; ?></p>
```

Si le visiteur décide d'écrire du code HTML dans la zone de message, cela fonctionnera très bien !

Par exemple, imaginons qu'il écrive dans le champ « Votre message » le code :

```
<strong>Badaboum</strong>
```

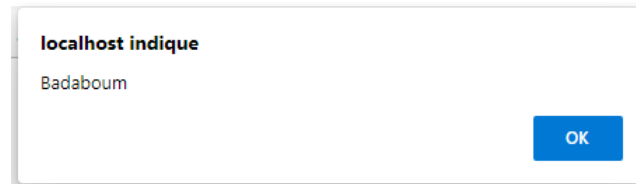
Le code source HTML qui sera généré par PHP sera le suivant :

```
<p><b>Message</b> : <strong>Badaboum</strong></p>
```

Il peut aussi ouvrir des balises de type `<script>` pour faire exécuter du code JavaScript au visiteur qui visualisera la page !

```
<p><b>Message</b> : <script>alert('Badaboum')</script></p>
```

Tous les visiteurs qui arriveront sur cette page verront la boîte de dialogue JavaScript suivante s'afficher :



La sécurité des applications web est un sujet très important à maîtriser lors de votre apprentissage.

Si vous souhaitez en savoir plus sur le sujet, consultez le site de l'Open Web Application Security Project, ou OWASP, c'est est une référence pour le développement d'applications web !

### 2.2.2 Sécurisez votre code en bloquant l'exécution de code JavaScript



Pour ignorer le code HTML, il suffit d'utiliser la fonction `htmlspecialchars`.

Elle va transformer les chevrons des balises HTML `<` et `>` en `&lt;` et `&gt;` respectivement.

On dit qu'on « échappe » le code HTML.

Cela provoquera l'affichage de la balise plutôt que son exécution.

```
<p><b>Message</b> : <?php echo htmlspecialchars($_POST['message']); ?></p>
```

Le code HTML qui en résultera sera propre et protégé, car les balises HTML insérées par le visiteur auront été échappées :

```
<p><b>Message</b> : &lt;strong>Badaboum&lt;/strong> ?></p>
```



Il faut penser à utiliser cette fonction sur tous les textes envoyés par l'utilisateur qui sont susceptibles d'être affichés sur une page web.

Bref, tout ce qui est affiché et qui vient, à la base, d'un visiteur, vous devez penser à le protéger avec `htmlspecialchars`.



Si vous préférez retirer les balises HTML que le visiteur a tenté d'envoyer plutôt que de les afficher, utilisez la fonction `strip_tags`.

## Travail à faire



Sécurisez votre code en appliquant une des deux méthodes `htmlspecialchars` ou `strip_tags` au choix afin de bloquer toute exécution de code HTML ou `avaScript`.

Testez.



Créez une nouvelle version de votre projet avec l'enchaînement de commandes ci-après :

```
> git add --all
> git commit -m "Sécurisez votre code"
> git push -u origin main
```

## 3 Activez le partage de fichiers

Imaginez qu'une personne essaie de vous contacter car elle est victime d'un bug d'utilisation sur votre site : elle souhaiterait peut-être vous partager une capture d'écran, ce qui faciliterait beaucoup votre travail de débogage.

Pour cela, vous allez devoir proposer la soumission de fichiers dans votre formulaire de contact !

Vous aviez découvert les superglobales `$_GET` et `$_POST` dans les sections précédentes, c'est maintenant au tour de la supervariable `$_FILES` de faire son entrée dans votre projet !

## 3.1 Donnez la possibilité d'envoyer des fichiers

Envoyer des fichiers grâce aux formulaires se passe en deux temps :

1. Le visiteur arrive sur votre formulaire et le remplit (en indiquant le fichier à envoyer). Une simple page HTML suffit pour créer le formulaire.
2. PHP réceptionne les données du formulaire et, s'il y a des fichiers dedans, il les « enregistre » dans un des dossiers du serveur.



Attention : l'envoi du fichier peut être un peu long si celui-ci est gros. Il faudra dire au visiteur de ne pas s'impatienter pendant l'envoi.

Vous allez devoir adapter votre formulaire de contact pour autoriser l'envoi et la soumission.

### 3.1.1 Paramétrez le formulaire d'envoi de fichier

#### Travail à faire



Dès l'instant où votre formulaire propose aux visiteurs d'envoyer un fichier, il faut ajouter l'attribut `enctype="multipart/form-data"` à la balise `<form>`.

```
<form action="submit_contact.php" method="POST" enctype="multipart/form-data">  
    <!-- champs de formulaire -->  
</form>
```

Grâce à `enctype`, le navigateur du visiteur sait qu'il s'apprête à envoyer des fichiers. Maintenant que c'est fait, nous pouvons ajouter à l'intérieur du formulaire une balise permettant d'envoyer un fichier.

C'est une balise très simple de type `<input type="file" />`.

Il faut donner un nom à ce champ de formulaire (grâce à l'attribut `name` ) pour que PHP puisse reconnaître le champ par la suite.

```
<form action="submit_contact.php" method="POST" enctype="multipart/form-data">
  <!-- Ajout des champs email et message -->
  [...]
  <!-- Ajout champ d'upload ! -->
  <div class="mb-3">
    <label for="screenshot" class="form-label">Votre capture d'écran</label>
    <input type="file" class="form-control" id="screenshot" name="screenshot" />
  </div>
  <!-- Fin ajout du champ -->
  <button type="submit" class="btn btn-primary">Envoyer</button>
</form>
```

Testez :



Créez une nouvelle version de votre projet avec l'enchaînement de commandes ci-après :

```
> git add --all
> git commit -m "Paramétrez le formulaire d'envoi de fichier"
> git push -u origin main
```

### 3.1.2 Traitez l'envoi en PHP

Il faut que l'on ajoute du code dans la page `submit_contact.php` pour traiter l'envoi du fichier.



En fait, au moment où la page PHP s'exécute, le fichier a été envoyé sur le serveur mais il est stocké dans un **dossier temporaire**.

C'est à vous de décider si vous acceptez définitivement le fichier ou non.

Vous pouvez par exemple vérifier si le fichier a la bonne extension (si vous demandez une image et qu'on vous envoie un « .txt », vous devrez refuser le fichier).

Pour chaque fichier envoyé, une variable `$_FILES['nom_du_champ']` est créée.

Dans notre cas, la variable s'appellera `$_FILES['screenshot']`.

Cette variable est un tableau (associatif) qui contient plusieurs informations sur le fichier dont les suivantes qui nous seront utiles :

Clé	Valeur
<code>name</code>	nom du fichier
<code>type</code>	type du fichier (image/gif par exemple)
<code>size</code>	taille en octets (PHP limite à 8 Mo)
<code>tmp_name</code>	emplacement temporaire sur le serveur
<code>error</code>	vaut 0 s'il n'y a pas eu d'erreur dans l'envoi

Nous allons faire les vérifications suivantes pour décider si l'on accepte le fichier ou non :

1. Vérifier tout d'abord si le visiteur a bien envoyé un fichier, en testant la variable `$_FILES['screenshot']` avec `isset()` et s'il n'y a pas eu d'erreur d'envoi, grâce à `$_FILES['screenshot']['error']`.
2. Vérifier si la taille du fichier ne dépasse pas 1 Mo par exemple (environ 1 000 000 d'octets), grâce à `$_FILES['screenshot']['size']`.
3. Vérifier si l'extension du fichier est autorisée (il faut interdire à tout prix que les gens puissent envoyer des fichiers PHP, sinon ils pourraient exécuter des scripts sur votre serveur).  
Dans notre cas, nous autoriserons seulement les images (fichiers .png, .jpg, .jpeg et .gif).  
Nous analyserons pour cela la variable `$_FILES['screenshot']['name']`.

Vous allez donc faire une série de tests dans votre page `submit_contact.php`.

## Travail à faire



1. Commencez par vérifier qu'un fichier a été envoyé :

Pour cela, on va tester si la variable `$_FILES['screenshot']` existe avec `isset()`.  
On vérifie dans le même temps s'il n'y a pas d'erreur d'envoi.



```
<?php
// Testons si le fichier a bien été envoyé et s'il n'y a pas d'erreur
if (isset($_FILES['screenshot']) && $_FILES['screenshot']['error'] == 0)
{

}
?>
```

2. On veut interdire que le fichier dépasse 1 Mo, soit environ 1 000 000 d'octets.  
On doit donc tester `$_FILES['screenshot']['size']` :

```
<?php
// Testons si le fichier a bien été envoyé et s'il n'y a pas d'erreur
if (isset($_FILES['screenshot']) && $_FILES['screenshot']['error'] == 0)
{
    // Testons si le fichier n'est pas trop gros
    if ($_FILES['screenshot']['size'] <= 1000000)
    {

    }

}
?>
```

3. Vérifiez l'extension du fichier :

On peut récupérer l'extension du fichier dans une variable grâce à ce code :

```
<?php
$fileInfo = pathinfo($_FILES['screenshot']['name']);
$extension = $fileInfo['extension'];
?>
```

La fonction `pathinfo` renvoie un tableau (array) contenant entre autres l'extension du fichier dans `$fileInfo['extension']`.

On stocke ça dans une variable `$extension`.

Une fois l'extension récupérée, on peut la comparer à un tableau d'extensions autorisées, et vérifier si l'extension récupérée fait bien partie des extensions autorisées à l'aide de la fonction `in_array()`.

On obtient ce code :

```

<?php
// Testons si le fichier a bien été envoyé et s'il n'y a pas d'erreur
if (isset($_FILES['screenshot']) AND $_FILES['screenshot']['error'] == 0)
{
    // Testons si le fichier n'est pas trop gros
    if ($_FILES['screenshot']['size'] <= 1000000)
    {
        // Testons si l'extension est autorisée
        $fileInfo = pathinfo($_FILES['screenshot']['name']);
        $extension = $fileInfo['extension'];
        $allowedExtensions = ['jpg', 'jpeg', 'gif', 'png'];
        if (in_array($extension, $allowedExtensions))
        {
            }
        }
    }
}
?>

```

#### 4. Validez l'upload du fichier :

Si tout est bon, on accepte le fichier en appelant la fonction [move\\_uploaded\\_file](#).

Cette fonction prend deux paramètres :

- Le nom temporaire du fichier (on l'a avec `$_FILES['screenshot']['tmp_name']`).
- Le chemin qui est le nom sous lequel sera stocké le fichier de façon définitive.  
On peut utiliser le nom d'origine du fichier `$_FILES['screenshot']['name']` ou générer un nom arbitraire.

Placez donc le fichier dans un sous-dossier « Uploads » que vous aurez pris soin de créer au préalable dans le répertoire où se trouve le script « submit\_contact.php ». On gardera le même nom de fichier que celui d'origine.

Comme `$_FILES['screenshot']['name']` contient le chemin entier vers le fichier d'origine (C:\dossier\fichier.png, par exemple), il nous faudra extraire le nom du fichier. On peut utiliser pour cela la fonction `basename` qui renverra juste « fichier.png ».

```

<?php
// Testons si le fichier a bien été envoyé et s'il n'y a pas d'erreur
if (isset($_FILES['screenshot']) && $_FILES['screenshot']['error'] == 0)
{
    // Testons si le fichier n'est pas trop gros
    if ($_FILES['screenshot']['size'] <= 1000000)
    {
        // Testons si l'extension est autorisée
        $fileInfo = pathinfo($_FILES['screenshot']['name']);
        $extension = $fileInfo['extension'];
        $allowedExtensions = ['jpg', 'jpeg', 'gif', 'png'];
        if (in_array($extension, $allowedExtensions))
        {
            // On peut valider le fichier et le stocker définitivement
            move_uploaded_file($_FILES['screenshot']['tmp_name'],
                'uploads/' . basename($_FILES['screenshot']['name']));
            echo "L'envoi a bien été effectué !";
        }
    }
}
?>

```



Créez une nouvelle version de votre projet avec l'enchaînement de commandes ci-après :

```

> git add --all
> git commit -m "Traitez l'envoi en PHP"
> git push -u origin main

```

### 3.1.3 Pour ceux qui pédalent vite



Si le nom du fichier à télécharger contient des espaces ou des accents, ça posera un problème une fois envoyé sur le Web.

Ou si quelqu'un envoie un fichier qui a le même nom que celui d'une autre personne, l'ancien sera écrasé !

Une solution peut consister à « choisir » vous-mêmes le nom du fichier stocké sur le serveur, plutôt que de nous servir du nom d'origine. Vous pouvez faire un compteur qui s'incrémente : 1.png, 2.png, 3.jpg, etc.



Créez une nouvelle version de votre projet avec l'enchaînement de commandes ci-après :

```
> git add --all  
> git commit -m "Allez plus loin dans le traitement de l'envoi d'un fichier"  
> git push -u origin main
```