Votre Formation Sur Mesure

MA PREMIERE APPLICATION WEB EN PHP

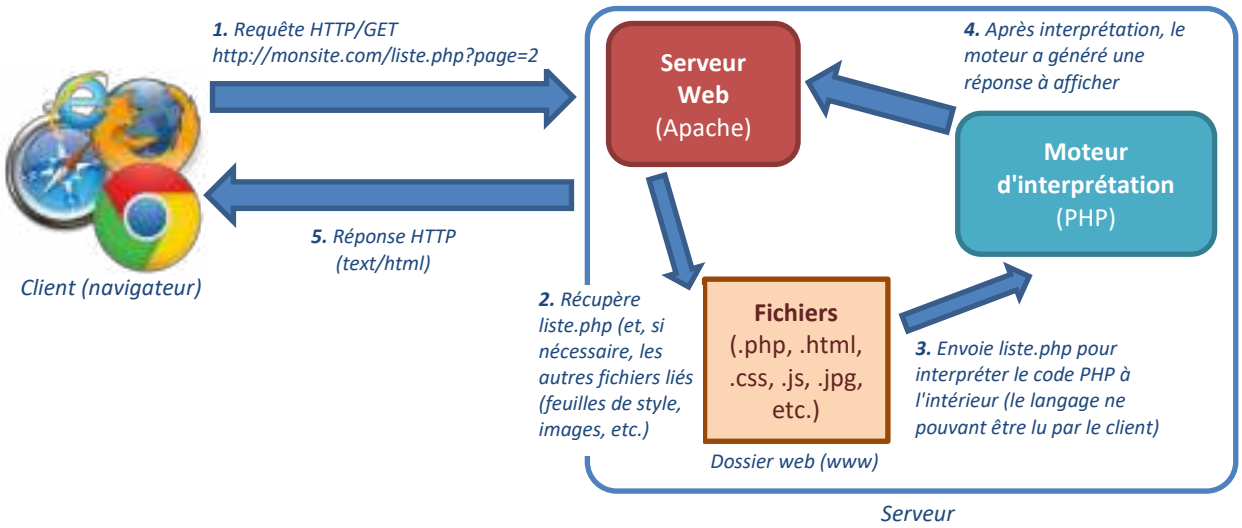
ELAN
14 rue du Rhône - 67100 STRASBOURG
☎ 03 88 30 78 30 elan@elan-formation.fr
www.elan-formation.fr

SAS ELAN au capital de 37 000 € -
RCS Strasbourg B 390758241 – SIRET 39075824100041 – Code APE : 8559A
N° déclaration DRTEFP 42670182967 - Cet enregistrement ne vaut pas agrément de l'Etat

SOMMAIRE

I. Introduction

En tant que langage de script coté serveur, PHP implémente nativement diverses fonctionnalités permettant d'assurer une bonne communication des données dans un contexte client-serveur. Cette communication passe naturellement dans ce contexte par le protocole HTTP, comme ce schéma l'illustre :



Ainsi, le moyen de communication entre le client et le serveur (le protocole HTTP) représente en quelque sorte l'intermédiaire, l'interprète des deux parties lors de leurs échanges respectifs.

Dans l'exemple ci-dessus, le client soumet une requête au serveur qui se compose de plusieurs éléments présents dans l'URL :

http://monsite.com/liste.php?page=2

Le protocole employé pour la communication	Le nom de domaine du serveur sur le réseau Internet	La ressource (fichier) demandée	Un paramètre de requête "page" avec pour valeur = 2

Le dernier élément nous intéresse : le paramètre "page" et sa valeur associée apportent au serveur une information supplémentaire, nécessitée par le script du fichier liste.php. Nous pouvons imaginer ici que ce paramètre oblige la "liste", découpée en plusieurs pages à la manière d'une pagination, à n'afficher que la deuxième page des éléments de cette liste.

Ce paramètre (appelé techniquement Query String Parameter) sera référencé dans une variable spécifique, fournie par le langage PHP et spécialement conçue pour récupérer les paramètres de requête présents dans une URL : **$_GET !**

II. Les superglobales PHP

Afin d'accéder à toutes les informations pouvant être transmises par le client au serveur, PHP dispose de plusieurs variables dites "superglobales". Toutes les superglobales sont du type tableau, proposant ainsi une manière simple d'y regrouper plusieurs informations sous forme de paires "clé / valeur".

Le terme "superglobales" signifie que ces variables sont disponibles dans n'importe quel script PHP : autrement dit, **il est inutile de vérifier si elles existent** (avec la fonction *isset()*, par exemple), elles sont créées automatiquement par le serveur. Néanmoins, elles peuvent être vides (et le seront si aucune donnée n'est transmise).

Ci-dessous la liste des superglobales et le type de données transmis par le client au serveur (en fonction de la méthode employée) :

- **$_GET**

Liée à la méthode HTTP GET, contient tous les paramètres ayant été transmis au serveur par l'intermédiaire de l'URL de la requête (Query String Parameters).

- **$_POST**

Liée à la méthode HTTP POST, contient toutes les données transmises au serveur par l'intermédiaire d'un formulaire (Form Data ou Request Body Parameters).

- **$_COOKIE**

Contient les données stockées dans les cookies du navigateur client.

- **$_REQUEST**

Regroupe les données transmises par les trois superglobales $_GET, $_POST et $_COOKIE.

- **$_SESSION**

Contient les données stockées dans la session utilisateur côté serveur (si cette session a été démarrée au préalable).

- **$_FILES**

Contient les informations associées à des fichiers uploadés par le client.

Cette variable est soumise à plusieurs conditions : le client a soumis un formulaire dans lequel un champ *<input type="file">* était présent (et rempli) **ET** que la balise <form> dudit formulaire comporte l'attribut *enctype="multipart/form-data".*

$_ENV et **$_SERVER**, elles, contiennent des informations relatives à l'environnement serveur (comme la version de l'OS, la version d'Apache, de PHP, le chemin du dossier web, etc.). Elles ne sont pas concernées par la transmission d'information du client vers le serveur, cependant.

Si nous reprenons l'URL de la page 3, voici ce que var_dump($_GET) (écrit dans le fichier liste.php) nous afficherait :

```
array (size=1)
  'page' => string '2' (length=1)
```

On constate ainsi que $_GET est un tableau (array), contenant une clé "page" associée à la valeur "2". La valeur est de type "string" même si c'est un nombre, puisque HTTP est un protocole de communication de texte (Hyper**Text** Transfer Protocol).

III. Cahier des charges

Dans la suite de ce cours, nous allons créer pas-à-pas une petite application web dont le cahier des charges est celui-ci :

L'application doit permettre à un utilisateur de renseigner différents produits par le biais d'un formulaire, produits qui seront consultables sur une page récapitulative. L'enregistrement en session de chaque produit est nécessaire. L'application ne nécessite pour l'instant aucun rendu visuel spécifique.

Trois pages sont nécessaires à cela :

1. **index.php**

Présentera un formulaire permettant de renseigner :

- Le nom du produit
- Son prix unitaire
- La quantité désirée

2. **traitement.php**

Traitera la requête provenant de la page index.php (après soumission du formulaire), ajoutera le produit avec son nom, son prix, sa quantité et le total calculé (prix × quantité) en session.

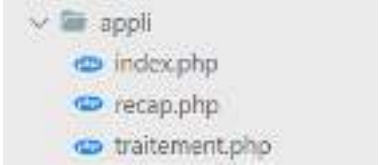
3. **recap.php**

Affichera tous les produits en session (et en détail) et présentera le total général de tous les produits ajoutés.

1. Création du dossier de l'application

En fonction de la solution de développement que vous utilisez (WAMP, MAMP, Laragon…), **créons un dossier "appli" dans notre serveur web**, au sein du dossier "www" ou "htdocs" selon cas.

Créons ensuite les trois fichiers, vides pour le moment, afin de baliser notre futur travail.



IV. Page index.php

Voici le code de la page *index.php*, contenant le formulaire qui nous permettra de renseigner les produits à l'application :

```
1  <!DOCTYPE html>
2  <html lang="en">
3      <head>
4          <meta charset="UTF-8">
5          <meta name="viewport" content="width=device-width, initial-scale=1.0">
6
7          <title>Ajout produit</title>
8      </head>
9      <body>
10
11         <h1>Ajouter un produit</h1>
12         <form action="traitement.php" method="post">
13             <p>
14                 <label>
15                     Nom du produit :
16                     <input type="text" name="name">
17                 </label>
18             </p>
19             <p>
20                 <label>
21                     Prix du produit :
22                     <input type="number" step="any" name="price">
23                 </label>
24             </p>
25             <p>
26                 <label>
27                     Quantité désirée :
28                     <input type="number" name="qtt" value="1">
29                 </label>
30             </p>
31             <p>
32                 <input type="submit" name="submit" value="Ajouter le produit">
33             </p>
34         </form>
35
36     </body>
37 </html>
```

Quelques précisions concernant certaines lignes de ce code :

↳ Ligne 12 : la balise *<form>* comporte deux attributs :

- ***action*** (qui indique la cible du formulaire, le fichier à atteindre lorsque l'utilisateur soumettra le formulaire)
- **method** (qui précise par quelle méthode HTTP les données du formulaire seront transmises au serveur)

La méthode employée ici est POST, pour ne pas "polluer" l'URL avec les données du formulaire. Il est néanmoins possible de soumettre un formulaire avec la méthode GET (par défaut si aucune méthode n'est précisée), les données renseignées dans les champs

du formulaire seraient par conséquent inscrites dans l'URL et limitées en nombre de caractères (selon le navigateur ou le serveur).

Ci-dessous, un exemple d'URL obtenue en validant ce formulaire avec l'attribut *method="get"* :

http://localhost/appli/traitement.php**?name=Pomme&price=2.5&qtt=2**

↳ Lignes 16, 22 et 28 :

Chaque input dispose d'un attribut "name", ce qui va permettre à la requête de classer le contenu de la saisie dans des clés portant le nom choisi. Ainsi, *var_dump($_POST)* donnera ceci après saisie et soumission du formulaire :

```
array (size=3)
  'name' => string 'Pomme' (length=5)
  'price' => string '2.5' (length=3)
  'qtt' => string '10' (length=2)
```

Les données sont structurées dans le tableau $_POST de la même manière que $_GET, sans être visibles dans l'URL.

↳ Ligne 32 :

Le champ <input type="submit">, représentant le bouton de soumission du formulaire, contient lui aussi un attribut "name". Nous verrons plus bas que ce choix permettra de vérifier côté serveur que le formulaire a bien été validé par l'utilisateur.

V. Mise en place de la page traitement.php

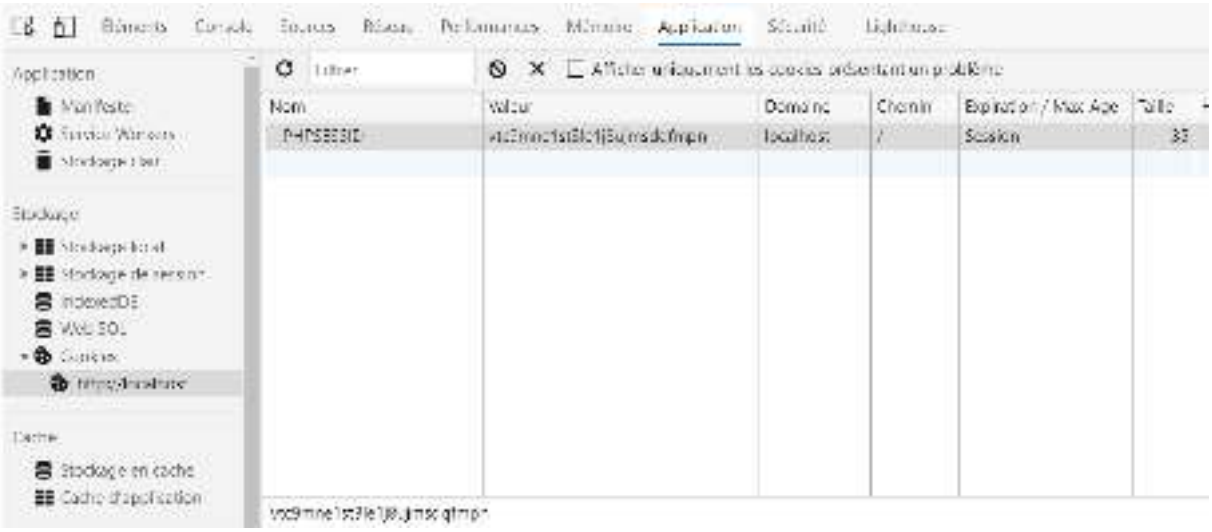
1. Démarrer une session

Nous souhaitons enregistrer les produits en session sur le serveur, pour cela PHP nous suggère, dans sa documentation officielle (source : https://www.php.net/manual/fr/intro.session.php), d'effectuer l'appel d'une fonction bien précise pour disposer d'une session : ***session_start()***.

Cette fonction a deux utilités : démarrer une session sur le serveur pour l'utilisateur courant, ou récupérer la session de ce même utilisateur s'il en avait déjà une. Cette deuxième fonctionnalité est rendue possible puisqu'au démarrage d'une session, **le serveur enregistrera un cookie PHPSESSID dans le navigateur client**, contenant l'identifiant de la session appartenant à celui-ci. Les cookies sont transmis au serveur avec chaque requête HTTP effectuée par le client.

La durée de vie du cookie dépend de la configuration serveur à ce sujet. Par défaut, le cookie expirera à la fermeture du navigateur (Expiration/Max-Age = Session).

La capture suivante présente le cookie en question (onglet *Application* du DevTools de Google Chrome) :



Note : si le cookie venait à être supprimé ou modifié, alors le serveur considérera la prochaine requête effectuée par le client comme devant démarrer une nouvelle session. Les données de la session précédente, tant que l'identifiant précédent ne sera pas retransmis dans une éventuelle prochaine requête, resteront inaccessibles. Le serveur, lui, conservera la session selon le paramètre de configuration session.cache_limiter, défini à 180 minutes par défaut.

Commençons alors notre code en appelant cette fonction :

```php
<?php
    session_start();
```

2. Vérifier l'existence d'une requête POST

Un utilisateur mal intentionné (ou trop curieux) pourrait atteindre le fichier *traitement.php* en saisissant directement l'URL de celui-ci dans la barre d'adresse, et ainsi **provoquer des erreurs sur la page qui lui présenterait des informations que nous ne souhaitons pas dévoiler**. Il faut donc limiter l'accès à *traitement.php* par les seules requêtes HTTP provenant de la soumission de notre formulaire.

Pour cela, une condition simple doit être effectuée :

```
1  <?php
2      session_start();
3
4      if(isset($_POST['submit'])){
5
6      }
7
8      header("Location:index.php");
9
```

Nous vérifions alors l'existence de la clé "submit" dans le tableau $_POST, celle clé correspondant à l'attribut "name" du bouton *<input type="submit" name="submit">* du formulaire. La condition sera alors vraie seulement si la requête POST transmet bien une clé "submit" au serveur.

Dans l'autre cas, la ligne 8 effectue une redirection grâce à la fonction *header()*. Il n'y a pas de "else" à la condition puisque dans tous les cas (formulaire soumis ou non), nous souhaitons revenir au formulaire après traitement.

Un mot sur la fonction *header("Location:…")*

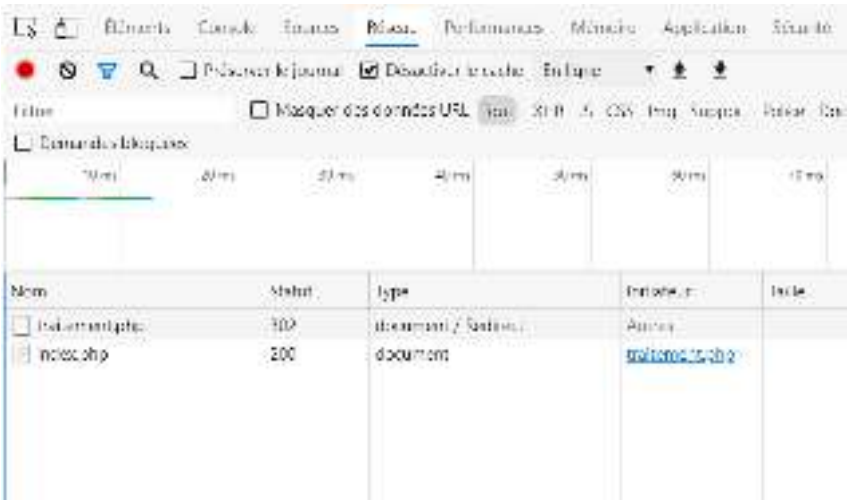
Cette fonction envoie un nouvel entête HTTP (les entêtes d'une réponse) au client. Avec le type d'appel "Location:", cette réponse est envoyée au client avec le status code 302, qui indique une redirection. Le client recevra alors la ressource précisée dans cette fonction.

Attention : l'utilisation de la fonction header() nécessite deux précautions :

- La page qui l'emploie **ne doit pas avoir émis un début de réponse avant *header()*** (afficher du HTML, appeler les fonctions *echo()* ou *print()* ou un autre *header()*…) sous peine de perturber la réponse à émettre au client (mauvaises entêtes HTTP…).
- **L'appel de la fonction *header()* n'arrête pas l'exécution du script courant**. Si le fichier effectue à la suite de la fonction d'autres traitements, ils seront exécutés. Il faut alors veiller à ce que *header()* soit la dernière instruction du fichier ou appeler la fonction *exit()* (ou *die()*) tout de suite après. De même, une fonction *header()* appelée successivement à une autre écrasera les entêtes de la première…

Essayons d'accéder à *traitement.php* sans passer par la validation du formulaire (en saisissant directement son URL dans la barre d'adresse du navigateur)… Nous nous

retrouvons sur *index.php* quoi qu'il arrive. Constatons alors, via l'outil DevTools du navigateur (onglet *Network*), ce qu'il s'est passé :



Le status code HTTP de la requête a bien répondu 302 (redirection) et, dessous, *index.php* est effectivement la page affichée (code 200) à l'initiative de *traitement.php*. Parfait !

3. Filtrer les valeurs transmises

Le point d'achoppement de tout développeur web est de permettre à un utilisateur d'introduire de mauvaises données sur un serveur. **Les formulaires sont à ce sujet le coupable idéal** !

Imaginons que l'utilisateur modifie dans le DevTools de son navigateur les attributs du formulaire, les types attendus par les champs ou même la ressource à atteindre : sans une vérification minutieuse des données transmises au serveur par le formulaire, **il existe un risque de provoquer des erreurs, voire pire, de pirater le serveur en injectant du code.** On appelle cela une faille par injection de code, comme XSS (pour Cross-Site Scripting) ou SQL Injection (en écrivant du SQL dans un champ afin de faire exécuter cela par la base de données).

Nous devons alors vérifier l'intégrité des valeurs transmises dans le tableau $_POST en fonction de celles que nous attendons réellement :

```php
<?php
    session_start();

    if(isset($_POST['submit'])){

        $name = filter_input(INPUT_POST, "name", FILTER_SANITIZE_STRING);
        $price = filter_input(INPUT_POST, "price", FILTER_VALIDATE_FLOAT, FILTER_FLAG_ALLOW_FRACTION);
        $qtt = filter_input(INPUT_POST, "qtt", FILTER_VALIDATE_INT);


    }

    header("Location:index.php");
```

La fonction PHP *filter_input()* permet, depuis la version 5.2.0, d'effectuer une validation ou un nettoyage de chaque donnée transmise par le formulaire en employant divers filtres[1]. Détaillons ceux que nous utilisons ici :

- **FILTER_SANITIZE_STRING** (champ "name") : ce filtre supprime une chaîne de caractères de toute présence de caractères spéciaux et de toute balise HTML potentielle ou les encode. Pas d'injection de code HTML possible !
- **FILTER_VALIDATE_FLOAT** (champ "price") : validera le prix que s'il est un nombre à virgule (pas de texte ou autre…), le drapeau **FILTER_FLAG_ALLOW_FRACTION** est ajouté pour permettre l'utilisation du caractère "," ou "." pour la décimale.
- **FILTER_VALIDATE_INT** (champ "qtt") : ne validera la quantité que si celle-ci est un nombre entier, au moins égal à 1.

filter_input() renvoie en cas de succès la valeur assainie correspondant au champ traité, *false* si le filtre échoue ou *null* si le champ sollicité par le nettoyage n'existait pas dans la requête POST. Ainsi, **pas de risque que l'utilisateur transmette des champs supplémentaires** !

À la suite de cela, nous disposons de trois variables *$name*, *$price* et *$qtt* censées contenir respectivement les valeurs nettoyées et/ou validées du formulaire. Il nous faut vérifier si les filtres ont tous fonctionné grâce à une nouvelle condition :

```php
<?php
    session_start();

    if(isset($_POST['submit'])){

        $name = filter_input(INPUT_POST, "name", FILTER_SANITIZE_STRING);
        $price = filter_input(INPUT_POST, "price", FILTER_VALIDATE_FLOAT, FILTER_FLAG_ALLOW_FRACTION);
        $qtt = filter_input(INPUT_POST, "qtt", FILTER_VALIDATE_INT);

        if($name && $price && $qtt){



        }
    }

    header("Location:index.php");
```

Sachant qu'un filtre renverra *false* ou *null* s'il échoue, et que nous ne pouvons pas anticiper la saisie de l'utilisateur à ce niveau, il suffit de vérifier implicitement si chaque variable contient une valeur jugée positive par PHP (du texte, des nombres, etc., autrement dit tout sauf *false* ou *null* ou *0)*. Voilà pourquoi la condition ne compare les variables à rien de précis.

[1] Il existe de nombreux autres filtres pour de nombreux autres cas de validation de données, n'hésitez pas à consulter la documentation officielle pour en apprendre plus : https://www.php.net/manual/fr/filter.filters.php

4. Enregistrement du produit en session

Il nous faut désormais stocker nos données en session, en ajoutant celles-ci au tableau *$_SESSION* que PHP nous fournit. Comme il est demandé de conserver chaque produit renseigné, nous devons au préalable décider de l'organisation de ces données au sein de la session.

Pour cela, nous allons construire pour chaque produit un tableau associatif *$product* :

```php
<?php
    session_start();

    if(isset($_POST['submit'])){

        $name = filter_input(INPUT_POST, "name", FILTER_SANITIZE_STRING);
        $price = filter_input(INPUT_POST, "price", FILTER_VALIDATE_FLOAT, FILTER_FLAG_ALLOW_FRACTION);
        $qtt = filter_input(INPUT_POST, "qtt", FILTER_VALIDATE_INT);

        if($name && $price && $qtt){

            $product = [
                "name"  => $name,
                "price" => $price,
                "qtt"   => $qtt,
                "total" => $price*$qtt
            ];


        }
    }

    header("Location:index.php");

```

$product comporte, comme stipulé dans le cahier des charges, une quatrième valeur "total", résultat de la multiplication du prix par la quantité du produit. Cette organisation des données nous permettra d'afficher nos produits plus tard de manière plus efficace et de rendre notre code plus explicite.

Il ne nous reste plus qu'à enregistrer ce produit nouvellement créé en session :

```php
<?php
    session_start();

    if(isset($_POST['submit'])){

        $name = filter_input(INPUT_POST, "name", FILTER_SANITIZE_STRING);
        $price = filter_input(INPUT_POST, "price", FILTER_VALIDATE_FLOAT, FILTER_FLAG_ALLOW_FRACTION);
        $qtt = filter_input(INPUT_POST, "qtt", FILTER_VALIDATE_INT);

        if($name && $price && $qtt){

            $product = [
                "name"  => $name,
                "price" => $price,
                "qtt"   => $qtt,
                "total" => $price*$qtt
            ];

            $_SESSION["products"][] = $product;
        }
    }

    header("Location:index.php");

```

Cette ligne 19 est particulièrement efficace car :

- On sollicite le tableau de session *$_SESSION* fourni par PHP.
- On indique la clé "products" de ce tableau. Si cette clé n'existait pas auparavant (ex: l'utilisateur ajoute son tout premier produit), PHP la créera au sein de $_SESSION.
- Les deux crochets "[]"[2] sont un raccourci pour indiquer à cet emplacement que nous ajoutons une nouvelle entrée au futur tableau "products" associé à cette clé. $_SESSION["products"] doit être lui aussi un tableau afin d'y stocker de nouveaux produits par la suite.

Notre page "traitement.php" est enfin terminée.

[2] Une autre syntaxe pour cette ligne aurait été : *array_push($_SESSION['products'], $product);* mais PHP nous conseille, dans sa documentation, d'utiliser les "[]" pour éviter le passage d'une fonction (plus lourd en termes de performances).

VI. Création de la page *recap.php*

recap.php devra nous permettre d'afficher de manière organisée et exhaustive la liste des produits présents en session. Elle doit également présenter le total de l'ensemble de ceux-ci.

1. Base de la page

Commençons alors la page comme ceci :

```php
<?php
    session_start();
?>
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>Récapitulatif des produits</title>
</head>
<body>

</body>
</html>
```

A la différence d'*index.php*, nous aurons besoin ici de parcourir le tableau de session, il est donc nécessaire d'appeler la fonction *session_start()* en début de fichier afin de récupérer, comme dit plus haut, la session correspondante à l'utilisateur.

2. Tests unitaires

Ecrivons ceci à la ligne 13 :

```
12  <body>
13      <?php var_dump($_SESSION); ?>
14  </body>
```

Nous allons dans un premier temps nous assurer que le tableau de session contienne des informations à afficher. Pour cela, voici trois cas de tests que nous allons appliquer successivement :

1. Accéder à *recap.php* sans avoir ajouté de produit précédemment
2. Accéder à *recap.php* après avoir ajouté deux produits
3. Accéder à *recap.php* après avoir ajouté ce produit[3] dans le formulaire :
   - *"name" => "Cliquez ici",*
   - *"price" => "A définir",*
   - *"qtt" => "0"*

Et voici ce que *recap.php* nous affiche dans chacun des cas de tests effectués :

Cas n°1 – aucun produit ajouté

Cas n°2 – 2 produits ajoutés

Cas n°3 – produit "bizarre" ajouté

Ces tests nous permettent de nous assurer d'une part du bon fonctionnement de *traitement.php*, et d'autre part de disposer désormais d'un visuel de la structure de notre tableau de session, ce qui nous aidera à le parcourir correctement.

Remarquons que l'ajout du produit "pirate" lors du cas n°3 n'a pas affecté le tableau de session. Pour cause : **les filtres ont bien fait leur boulot !**

[3] Pour ce faire, il a été nécessaire de modifier, via le DevTools du navigateur, quelques attributs du formulaire…

Modifions le code afin de prendre en compte tous ces cas de figure :

```php
<?php
    session_start();
?>
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>Récapitulatif des produits</title>
</head>
<body>
    <?php
        if(!isset($_SESSION['products']) || empty($_SESSION['products'])){
            echo "<p>Aucun produit en session...</p>";
        }
        else{

        }
    ?>
</body>
</html>
```

Nous rajoutons une condition qui vérifie :

- Soit la clé "products" du tableau de session *$_SESSION* n'existe pas : ***!isset()***
- Soit cette clé existe mais ne contient aucune donnée : ***empty()***

Dans ces deux cas, nous afficherons à l'utilisateur un message le prévenant qu'aucun produit n'existe en session. Il ne nous reste plus qu'à afficher le contenu de $_SESSION['products'] dans la partie *else* de notre condition.

3. Affichage dynamique des produits

A ce niveau, nous déciderons d'afficher nos produits dans un tableau HTML *<table>*, élément tout à fait légitime à présenter une liste de données ordonnée et bien décomposée.

```
12  <body>
13      <?php
14          if(!isset($_SESSION['products']) || empty($_SESSION['products'])){
15              echo "<p>Aucun produit en session...</p>";
16          }
17          else{
18              echo "<table>",
19                      "<thead>",
20                          "<tr>",
21                              "<th>#</th>",
22                              "<th>Nom</th>",
23                              "<th>Prix</th>",
24                              "<th>Quantité</th>",
25                              "<th>Total</th>",
26                          "</tr>",
27                      "</thead>",
28                      "<tbody>";
29
30              foreach($_SESSION['products'] as $index => $product){
31
32              }
33              echo "</tbody>",
34                  "</table>";
35          }
36      ?>
37  </body>
```

De la ligne 17 à la ligne 28, nous trouvons les balises HTML initialisant correctement un tableau HTML avec une ligne d'en-têtes *<thead>,* afin de bien décomposer les données de chaque produit.

Ligne 30, nous observons la boucle itérative *foreach()*[4] de PHP, particulièrement efficace pour exécuter, produit par produit, les mêmes instructions qui vont permettre l'affichage uniforme de chacun d'entre eux. Pour chaque donnée dans *$_SESSION['products']*, nous disposerons au sein de la boucle de deux variables :

- **$index** : aura pour valeur l'index du tableau *$_SESSION['products']* parcouru. Nous pourrons numéroter ainsi chaque produit avec ce numéro dans le tableau HTML (en première colonne).
- **$product** : cette variable contiendra le produit, sous forme de tableau, tel que l'a créé et stocké en session le fichier *traitement.php*.

[4] D'autres boucles pourraient être employées ici, mais foreach() permet de parcourir $_SESSION['products'] de A à Z et, pour ce faire, aucune autre boucle ne bénéficie d'une syntaxe aussi simple d'utilisation.

Utilisons ces variables à l'intérieur de la boucle *foreach()* et affichons chaque produit comme ceci :

```
30          foreach($_SESSION['products'] as $index => $product){
31              echo "<tr>",
32                      "<td>".$index."</td>",
33                      "<td>".$product['name']."</td>",
34                      "<td>".$product['price']."</td>",
35                      "<td>".$product['qtt']."</td>",
36                      "<td>".$product['total']."</td>",
37                  "</tr>";
38          }
```

La boucle créera alors une ligne *<tr>* et toutes les cellules *<td>* nécessaires à chaque partie du produit à afficher, et ce pour chaque produit présent en session.

Visuellement, notre navigateur web nous affichera ce résultat :

#	Nom	Prix	Quantité	Total
0	Banane	1.25	3	3.75
1	Pomme	2.5	10	25

Pour que les prix s'affichent sous un format monétaire plus lisible, modifions notre code en ce sens :

```
30          foreach($_SESSION['products'] as $index => $product){
31              echo "<tr>",
32                      "<td>".$index."</td>",
33                      "<td>".$product['name']."</td>",
34                      "<td>".number_format($product['price'], 2, ",", " ")." €</td>",
35                      "<td>".$product['qtt']."</td>",
36                      "<td>".number_format($product['total'], 2, ",", " ")." €</td>",
37                  "</tr>";
38          }
```

La fonction PHP number_format() permet de modifier l'affichage d'une valeur numérique en précisant plusieurs paramètres :

```
number_format(
   variable à modifier,
   nombre de décimales souhaité,
   caractère séparateur décimal,
   caractère séparateur de milliers[5]
);
```

En ajoutant avant la fermeture de la balise <td> un symbole €, nos montants s'affichent comme ceci :

#	Nom	Prix	Quantité	Total
0	Banane	1.25 €	3	3.75 €
1	Pomme	2.50 €	10	25.00 €

4. Calcul et affichage du total général

Il ne manque plus qu'une dernière étape pour couvrir le cahier des charges au complet : afficher le total général.

Nous aurons besoin de modifier notre code ainsi :

↳ Ligne 29 :

Dans un premier temps, avant la boucle, on initialise une nouvelle variable *$totalGeneral* à zéro.

↳ Ligne 38 :

À l'intérieur de la boucle, grâce à **l'opérateur combiné +=**, on ajoute le total du produit parcouru à la valeur de *$totalGeneral*, qui augmente d'autant pour chaque produit[6].

[5] Le caractère HTML * * est un espace insécable.
[6] Une autre syntaxe est, là encore, possible : $totalGeneral = $totalGeneral + $produit["total"];

↳ Lignes 40 à 43 :

Une fois la boucle terminée, nous affichons une dernière ligne avant de refermer notre tableau. Cette ligne contient deux cellules : une cellule fusionnée de 4 cellules (*colspan=4*) pour l'intitulé, et une cellule affichant le contenu formaté de *$totalGeneral* avec *number_format().*

#	Nom	Prix	Quantité	Total
0	Banane	1,25 €	3	3,75 €
1	Pomme	2,50 €	10	25,00 €
Total général :				**28,75 €**

Mission accomplie ! Notre cahier des charges est couvert à 100% et tout fonctionne correctement.

Cependant…