Qu’est ce que le CRUD et ses grandes étapes.

Qu’est ce que le CRUD ?

L'acronyme informatique anglais **CRUD** (pour *create*, *read*, *update*, *delete*) (parfois appelé SCRUD avec un "S" pour *search*) désigne les quatre opérations de base pour la persistance des données, en particulier le stockage d'informations en base de données.

Soit :

* **c**reate : créer
* **r**ead : lire
* **u**pdate : mettre à jour
* **d**elete : supprimer

Plus généralement, il désigne les opérations permettant la gestion d'une collection d'éléments.

Ce terme est aussi un jeu de mot en anglais sur l'adjectif **crude** (en français **brut** ou **rudimentaire**).

1. **C**reate : product

1.La première étape consiste à créer une table de base de données qui sera manipulée en utilisant l’accès CRUD. Pour cela, il faut soumettre la requête suivante dans votre base de données MySQL :  
  
<?php  
**public** **function** addProductBySupplier() {  
 //Soumission de la requête au serveur de la base de données  
 $results = $**this**->pdo->db->prepare('INSERT INTO `c19v12\_products` (`name`, `reference`, `price`, `picture`, `id\_c19v12\_categories`)  
 VALUES (:name, :reference, :price, :picture, :id\_c19v12\_categories)');  
 // Associe une valeur à chaque marqueur nominatif de la requête $results->bindValue(':name', $**this**->name, **PDO**::PARAM\_STR); $results->bindValue(':reference', $**this**->reference, **PDO**::PARAM\_STR); $results->bindValue(':price', $**this**->price, **PDO**::PARAM\_STR); $results->bindValue(':picture', $**this**->picture, **PDO**::PARAM\_STR); $results->bindValue(':id\_c19v12\_categories', $**this**->id\_c19v12\_categories, **PDO**::PARAM\_INT);  
// Exécution de la requête  
**try** {  
**return** $results->execute(); // Méthode execute() toujours avec prepare() sinon query()  
}   
**catch** (**Exception** $e) {  
**die**('échec de la connexion :' . $e->getMessage());  
}  
}  
?>  
  
  
La table est utilisée pour collecter des informations sur l’utilisateur : le nom, la référence, le prix, l’imagetable. Chaque entrée reçoit automatiquement une **clé primaire** (*AUTO\_INCREMENT PRIMARY KEY*), c’est-à-dire son propre ID unique.

2. Il est ensuite nécessaire de configurer la connexion et de supprimer la base de données. Il faut créer un fichier PHP sous le nom database.php, il est ensuite nécessaire d’insérer le script suivant avec la classe « Database » pour gérer les connexions de base de données :  
  
« Database » pour gérer les connexions de base de données :

<?php

**class** Product **extends** Database {

**public** $pdo;

//attributs

//(seront utilisés lorsque l'on récupèrera des données à partir du formulaire)

**public** $id;

**public** $name;

**public** $reference;

**public** $price;

**public** $picture;

**public** $id\_c19v12\_categories;

**public** $formErrors = **array**();

**public** **function** \_\_construct() {

$**this**->pdo = Database::getInstance(); // Création de l'instance de connexion

}

**public** **function** \_\_destruct() {

$**this**->pdo->db = NULL;

}

/\*

\* Méthode démarrant une transaction SQL

\*/

**public** **function** beginTransaction() {

**return** $**this**->pdo->db->beginTransaction();

}

/\* \* Méthode appliquant la transaction courante et rendant ses modifications permanentes \*/  
**public** **function** commit() {

**return** $**this**->pdo->db->commit();

}

/\*

\* Méthode faisant un rollback de la transaction courante et annulant ses modifications

\*/

**public** **function** rollBack() {

**return** $**this**->pdo->db->rollBack();

}

/\*

\* Method retournant l'id de la dernière rangée insérée en base de données

\*/

**public** **function** lastInsertId() {

**return** $**this**->pdo->db->lastInsertId();

}

?>

Afin de pouvoir utiliser ici la classe définie pour l’accès à la base de données avec PDO, il faut **spécifier exactement les valeurs pour les quatre entrées $database** (nom de la base de donnée utilisée), **$host** (nom de l’hôte sur lequel la base de données est en cours d’exécution), **$username** (nom d’utilisateur) et **$password** (mot de passe de l’utilisateur).

Dans ce script, la classe de base de données comporte trois fonctions : *\_\_construct()*, le **constructeur de classe** permet que la classe rappelle aux utilisateurs que l’initialisation (c’est-à-dire l’assignation d’une valeur initiale ou de départ) n’est pas autorisée. Ensuite *connect()* est la fonction principale de la classe qui régule la connexion enfin au contraire *disconnect()* est utilisée pour mettre fin à la connexion.

<?php  
// Création d'une instance de classe Product. Instance de classe = OBJET $product = **new** Product();  
// Récupération des données du formulaire  
// Affectation de chaque champ de formulaire à l'attribut auquel il est associé  
$product->name = **isset**($\_POST['name']) ? htmlspecialchars($\_POST['name']) : '';  
$product->price = **isset**($\_POST['price']) ? htmlspecialchars($\_POST['price']) : '';  
$product->reference = **isset**($\_POST['reference']) ? htmlspecialchars($\_POST['reference']) : ''; $product->id\_c19v12\_categories = **isset**($\_POST['category']) && $\_POST['category'] > 0 ? $\_POST['category'] : ''; // input name = category  
//Toutes regex pour le formulaire  
$regexNameProduct = '/^[a-zA-Z0-9À-ÿ\' -]+$/';  
$regexReference = '/^[A-Z]{1,5}[0-9]{1,5}$/';  
$regexPrice = '/^\d+(\d{3})\*(\.\d{1,2})?$/';  
…  
  
// Validation de picture  
**if** (**isset**($\_FILES['picture']) && $\_FILES['picture']['error'] == 0) { $filesInfo = basename($\_FILES['picture']['name']);  
$extension\_upload = $filesInfo['extension'];  
$extension\_allowed = **array**('jpg','png','bmp');  
$product->picture = $filesInfo;  
**if** (in\_array($extension\_upload, $extension\_allowed)) {  
**echo** 'Le fichier transmis est bien un fichier jpg';  
} **else** {  
**echo** '/!\ vous devez sélectionner un fichier JPG /!\ ';  
}  
}  
// Insertion du product dans la base de données  
**if** (**empty**($product->formErrors)) {  
$success = $product->addProductBySupplier();  
?>

La page *createUser.php* est désormais activée, permettant ainsi d’entrer les informations utilisateur. Le script garantit que toutes les **données entrées et les messages d’erreur** sont enregistrés et que les messages d’erreur correspondants apparaissent bien quand une **entrée est incorrecte**, enfin le script assure aussi la **transmission des données à la base de données.**

1. **R**ead : product

La procédure Read lit les enregistrements d’une table sur la base de clé primaire indiqué en paramètre d’entrée. Mes 2 fichiers permettant de lire dans mon CRUD User les enregistrements d’une table sont “listUser.php” et “profilUser.php”.  
  
Ma vue “listProduct.php” permet d’afficher la liste des produits. Mon code :  
  
<?php

session\_start();

**require\_once** '../init/functions.php';

**require\_once** '../init/credentials.php';

**require\_once** '../models/database.php';

**require\_once** '../models/product.php';

**require\_once** '../models/category.php';

**require\_once** '../controllers/listProductCtrl.php';

//require\_once '../controllers/listProductSearchCtrl.php';

**require\_once** 'header.php';

?>

<**div** id="detailsProduct" class="h1 text-center mt-4">LISTE DES PRODUITS

</**div**>

<**div** class="row d-flex justify-content-center mt-5">

<**input** type="hidden" name="id" value="<?= **isset**($\_GET['id']) ? $\_GET['id'] : '' ?>" />

<?php **foreach** ($productList **as** $product) { ?>

<**div** class="col-md-4 d-flex justify-content-center mt-3">

<**div** class="card text-black text-center bg-secondary mb-3" style="**max-width**: 20rem;">

<**div** class="card-header"><?= $product->id ?><?= $product->productName ?></**div**>

<**div** class="card-body">

<**div**><**img** src="../assets/img/<?= $product->picture ?>" class="picture" alt="vatminton" /></**div**>

<**h4** class="card-title">référence : <?= $product->reference ?></**h4**>

<**p** class="card">Catégorie de produit : <?= $product->categoryName ?></**p**>

<**p** class="card">Prix : <?= $product->price ?> €</**p**>

<**a** class="px-3" href="detailsProduct.php?id=<?= $product->id ?>">Détails</**a**>

<?php **if** (isUser() || isAdmin()) { ?>

<**a** class="px-3" href="addToCart.php?id=<?= $product->id ?>">Ajouter à mon panier</**a**>

<?php } ?>

</**div**>

</**div**>

</**div**>

<?php }

?>

</**div**>

Mon contrôleur “listProductCtrl.php” permet de faire toutes les vérifications nécessaires avant d’envoyer les données à la vue. Mon code :

<?php  
$product = **new** Product();  
$productList = $product->getProductList();  
?>

1. **U**pdate : product

La procédure Update utilise la commande sql UPDATE sur la table en se basant bien sûr la clé primaire spécifié dans la clause WHERE. Tout comme la procédure Create il possède un paramètre pour chaque colonne dans la table. La requête *“prepare” évite l’injection de code SQL.*Voici mon script permettant de modifier le profil de l’utilisateur et de le mettre à jour dans la base de données “product.php” :  
  
<?php

**public** **function** updateProfilProduct() {

//Soumission de la requête au serveur de la base de données

$results = $**this**->pdo->db->prepare('UPDATE `c19v12\_products` SET `name` = :name, `reference` = :reference, `price` = :price, `picture` = :picture

WHERE `id` = :id');

// :mail est un marqueur nominatif, pour éviter l'injection de code SQL

// Associe une valeur à chaque marqueur nominatif de la requête

$results->bindValue(':id', $**this**->id, **PDO**::PARAM\_INT);

$results->bindValue(':name', $**this**->name, **PDO**::PARAM\_STR); //dans On nomme ce qu'on veut dans bindvalue

$results->bindValue(':reference', $**this**->reference, **PDO**::PARAM\_STR);

$results->bindValue(':price', $**this**->price, **PDO**::PARAM\_STR);

$results->bindValue(':picture', $**this**->picture, **PDO**::PARAM\_STR);

$results->bindValue(':name', $**this**->name, **PDO**::PARAM\_STR);

// Exécution de la requête

**try** {

**return** $results->execute(); // Méthode execute() toujours avec prepare() sinon query()

// execute retourne true ou false

} **catch** (**Exception** $e) {

**die**('échec de la connexion :' . $e->getMessage());

}

}

?>

Ma vue “update.php” permet de modifier un produit. Mon code :

<?php

session\_start();

**require\_once** '../init/functions.php';

**require\_once** '../init/credentials.php';

**require\_once** '../models/database.php';

**require\_once** '../models/product.php';

**require\_once** '../models/category.php';

**require\_once** '../models/role.php';

**require\_once** '../controllers/updateProductCtrl.php';

**include** 'header.php';

?>

<**div** id="formulary" class="h1 text-center my-4">Espace fournisseur</**div**>

<**h2** class="text-center mb-4">Modifier un produit</**h2**>

<**div** class="container">

<**div** class="row">

<**div** id="mainForm" class="col-md-12">

<**div** class="row justify-content-center">

<**div** class="col-md-12">

<**div** class="formA p-4 mt-2">

<**form** method="POST" action="" enctype="multipart/form-data">

<!-- Nom -->

<**div** class="form-row">

<**div** class="form-group col-md-6 text-center">

<**label** for="name">Nom : </**label**>  
 <**input** type="text" class="form-control text-center text-info" name="name" placeholder="" id="name" value="<?= **isset**($detailsProduct->productName) ? $detailsProduct->productName : '' ?>" />

<**small** class="text-warning"><?= **isset**($profilProduct->formErrors['name']) ? $profilProduct->formErrors['name'] : '' ?></**small**>

</**div**>

<!-- Price -->

<**div** class="form-group col-md-6 text-center">

<**label** for="price">Prix : </**label**>

<**input** type="text" class="form-control text-center text-info" name="price" id="price" placeholder="" value="<?= **isset**($detailsProduct->price) ? $detailsProduct->price : '' ?>" />

<**small** class="text-warning"><?= **isset**($profilProduct->formErrors['price']) ? $profilProduct->formErrors['price'] : '' ?></**small**>

</**div**>

<!-- Reference -->

<**div** class="form-group col-md-8 text-center">

<**label** for="reference">Référence du produit : </**label**>

<**input** type="text" class="form-control text-center text-info" name="reference" id="reference" placeholder="" value="<?= **isset**($detailsProduct->reference) ? $detailsProduct->reference : '' ?>">

<**small** class="text-warning"><?= **isset**($profilProduct->formErrors['reference']) ? $profilProduct->formErrors['reference'] : '' ?></**small**>

</**div**>

<!-- Categories -->

<**div** class="form-group col-md-4 text-center">

<**label** for="categories">Catégorie : </**label**>

<**select** id="categories" name="category" class="form-control"> <!-- le name="category" est lié à $category dans le controller -->

<**option** value="<?= $detailsProduct->id\_c19v12\_categories ?>" selected><?= $detailsProduct->categoryName ?></**option**>

<?php **foreach** ($categoryList **as** $item) { ?>

<**option** class="text-white" value="<?= $item->id ?>"><?= ucfirst($item->name) ?></**option**> <!-- uppercase first letter -->

<?php } ?>

</**select**>

<**small** class="text-warning"><?= **isset**($profilProduct->formErrors['categories']) ? $profilProduct->formErrors['categories'] : '' ?></**small**>

</**div**>

<**div** class="row">

<**div** class="form-group col-md-12 text-center">

<**input** type="file" name="picture" />

</**div**>

<**input** type="submit" name="submit" value="Enregistrer" />

</**div**>

</**div**>

</**form**>

</**div**>

</**div**>

</**div**>

</**div**>

</**div**>

Mon contrôleur “updateProductCtrl.php” permet de faire toutes les vérifications nécessaires avant d’envoyer les données à la vue. Mon code :

<?php

$product = **new** Product();

$category = **new** Category();

$product->id = **isset**($\_GET['id']) ? $\_GET['id'] : '';

$detailsProduct = $product->detailsProduct();

$categoryList = $category->getCategories();

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

// Récupération des données du formulaire

// Affectation de chaque champ de formulaire à l'attribut auquel il est associé

$product->name = **isset**($\_POST['name']) ? htmlspecialchars($\_POST['name']) : '';

$product->price = **isset**($\_POST['price']) ? htmlspecialchars($\_POST['price']) : '';

$product->reference = **isset**($\_POST['reference']) ? htmlspecialchars($\_POST['reference']) : '';

$product->id\_c19v12\_categories = **isset**($\_POST['category']) && $\_POST['category'] > 0 ? $\_POST['category'] : ''; // input name = category

//Toutes regex pour le formulaire

$regexNameProduct = '/^[a-zA-Z0-9À-ÿ\' -]+$/';

$regexReference = '/^[A-Z]{1,5}[0-9]{1,5}$/';

$regexPrice = '/^\d+(\d{3})\*(\.\d{1,2})?$/';

// Validation du nom du produit

**if** (**empty**($product->name)) {

$product->formErrors['name'] = 'Ce champ est vide';

} **elseif** (!preg\_match($regexNameProduct, $product->name)) {

$product->formErrors['name'] = 'Merci de rentrer un nom de produit valide';

} **elseif** (strlen($product->name) < 1 || strlen($product->name) > 26) {

$product->formErrors['name'] = 'Le nom doit comporter entre 2 et 25 caractères';

}

// Validation des catégories (select)

**if** (**isset**($\_POST['categories'])) {

**if** (**empty**($product->categories)) {

$product->formErrors['categories'] = 'Ce champ est vide';

} **else** {

$product->formErrors['categories'] = 'champ est vide';

}

}

// Validation du prix

**if** (**empty**($product->price)) {

$product->formErrors['price'] = 'Ce champ est vide';

} **elseif** (!preg\_match($regexPrice, $product->price)) {

$product->formErrors['price'] = 'Merci de rentrer un prix valide';

}

// Validation de la référence

**if** (**empty**($product->reference)) {

$product->formErrors['reference'] = 'Ce champ est vide';

} **elseif** (!preg\_match($regexReference, $product->reference)) {

$product->formErrors['reference'] = 'Merci de rentrer un numéro de référence valide';

}

// Validation de picture

**if** (**isset**($\_FILES['picture']) && $\_FILES['picture']['error'] == 0) {

$filesInfo = basename($\_FILES['picture']['name']);

$extension\_upload = $filesInfo['extension'];

$extension\_allowed = **array**('jpg','png','bmp');

$product->picture = $filesInfo;

**if** (in\_array($extension\_upload, $extension\_allowed)) {

**echo** 'Le fichier transmis est bien un fichier jpg, png, bmp';

} **else** {

**echo** '/!\ vous devez sélectionner un fichier JPG, PNG, BMP /!\ ';

}

}

// Insertion du product dans la base de données

**if** (**empty**($profilProduct->formErrors)) {

$success = $product->updateProfilProduct();

header('Location: ../../views/listProduct.php');

//header('Location: ../../views/detailsProduct.php?id=' . $product->id);

**exit**();

}

}

?>

1. **D**elete : user

Cette procédure supprime une ligne spécifiée dans la clause WHERE.

Mon code dans le modèle product.php :

<?php

**public** **function** deleteProduct() {

//Définition de la requête SQL

$results = $**this**->pdo->db->prepare('DELETE FROM `c19v12\_products` WHERE `id` = :id');

// :mail est un marqueur nominatif, pour éviter l'injection de code SQL

// Associe une valeur à chaque marqueur nominatif de la requête

$results->bindValue(':id', $**this**->id, **PDO**::PARAM\_INT);

//Soumission de la requête au serveur de la base de données

**try** {

**return** $results->execute();

} **catch** (**Exception** $e) {

**die**('échec de la connexion :' . $e->getMessage());

}

}

?>

Ma vue “delete.php” permet de modifier un produit. Mon code :

<?php

session\_start();

**require\_once** '../init/functions.php';

**require\_once** '../init/credentials.php';

**require\_once** '../models/database.php';

**require\_once** '../models/product.php';

**require\_once** '../controllers/deleteProductCtrl.php';

**require\_once** '../views/header.php';

?>

<**div** class="row">

<**div** class="col-md-12 text-center my-5">Êtes-vous sûr de vouloir supprimer le produit ?</**div**>

</**div**>

<**div** class="row">

<**div** class="col-md-6 text-center">

<**a** class="btn bg-success mx-5" href="../views/detailsProduct.php">Annuler</**a**>

</**div**>

<**div** class="col-md-6 text-center">

<**a** href="deleteProduct.php?id=<?= $product->id ?>&confirm=true" class="btn bg-danger mx-5">Confirmer</**a**>

</**div**>

</**div**>

Mon contrôleur “deleteProductCtrl.php” permet de faire toutes les vérifications nécessaires avant d’envoyer les données à la vue. Mon code :

<?php

**if** (**isset**($\_GET['id']) && $\_GET['id'] > 0 && $\_GET['confirm'] == true) {

$product = **new** Product();

$product->id = **isset**($\_GET['id']) ? htmlspecialchars($\_GET['id']) : '';

$success = $product->deleteProduct();

**if** ($success) {

$\_SESSION['message'] = 'Le produit a bien été supprimé';

} **else** {

$\_SESSION['message'] = 'Le produit n\'a pas été supprimé';

}

header('Location: ../views/listProduct.php');

**exit**();

} **else** {

$product = **new** Product();

$product->id = **isset**($\_GET['id']) ? htmlspecialchars($\_GET['id']) : '';

}

?>

Sécurité de mon site:

Pour la sécurité de mon site, j’ai utilisé le htmlspecialchars car cela m’a permis l’injection de code HTML et JS. J’ai également créé des REGEX (expression régulière) et un fichier php que j’ai nommé “functions”. Voici quelques lignes de mes REGEX :

<?php

$regexName = '/^[a-zA-ZÀ-ÿ\' -]+$/';

$regexAddress = '/[0-9]{1,3}(**?:**(**?:**[,. ]){1,2}[-a-zA-ZÀ-ÿ]+)+$/';

$regexBirth = '/^(19|20)[0-9]{2}-[0-9]{2}-[0-9]{2}$/';

$regexPhone = '/^[0-9]{2}(-|\ ?)[0-9]{2}(-|\ ?)[0-9]{2}(-|\ ?)[0-9]{2}(-|\ ?)[0-9]{2}$/';

$regexMail = '/^[^\W][a-zA-Z0-9]+(.[a-zA-Z0-9]+)@[a-zA-Z0-9]+(.[a-zA-Z0-9]+).[a-zA-Z]{2,4}$/';

$regexPassword = '/^(**?=**.\*[a-z])(**?=**.\*[A-Z])(**?=**.\*[0-9])(**?=**.\*\W).{3,}$/';

?>

Voici les fonctions que j’ai utilisé :

1. “function debug($data) et debugPrint($data)”
2. Ma fonction debug($data) m’a permis de faire un var\_dump pour identifier une erreur dans mon code et ainsi trouver une solution. Les codes de retour pour ces exceptions ont des noms au format SQL\_ERROR\_XXX dont voici le script :

<?php

**public** **function** \_\_construct() {

**try** {

// -- IMPORTANT --

// Les tables doivent être du type innoDB pour prendre en charge les transactions.

// Les tables de type MyISAM ne gèrent pas les transactions.

$**this**->db = **new** **PDO**('mysql:host=' . $**this**->host .';dbname=' . $**this**->database . ';charset=utf8',

$**this**->username,

$**this**->password);

// En mode ERRMODE\_EXCEPTION, si un échec survient, le script est

// interrompu, la connexion fermée, et mysql effectue un roll back

// sur la transaction

$**this**->db->setAttribute( **PDO**::ATTR\_ERRMODE, **PDO**::ERRMODE\_EXCEPTION );

} **catch** (**Exception** $e) {

**die**('Error : ' . $e->getMessage());

}

}

?>

La balise <pre> représente du texte préformaté, généralement écrit avec une police à chasse fixe. Le texte est affiché tel quel, les espaces utilisés dans le document HTML seront retranscrits (mets à la ligne).

1. Ma fonction debugPrint(), à la différence de la fonction debug() elle permet d’avoir des informations sommaires.

2) ”function debugSession()”

Cette fonction permet d’afficher l’id du *“role”.*

3) les 3 fonctions : isAdmin(), isUser() et isSupplier()

Ses 3 fonctions me permettent d’afficher certains éléments sur une page pour un *“role”* et non pour les autres. Elles permettent également de limiter les données à chaque *“role”.*

*a)isAdmin()*

*<?php*

*// Méthode certifiant que l'utilisateur est un administrateur*

***function*** *isAdmin() {*

***if*** *(****isset****($\_SESSION['user\_role']) && $\_SESSION['user\_role'] == 1) {*

***return*** *TRUE;*

*}* ***else*** *{*

***return*** *FALSE;*

*}*

*?>*

b)isUser()

<?php

// Méthode certifiant que l'utilisateur est un client enregistré

**function** isUser() {

**if** (**isset**($\_SESSION['user\_role']) && $\_SESSION['user\_role'] == 2) {

**return** TRUE;

} **else** {

**return** FALSE;

}

}

?>

c)isSupplier()

<?php

// Méthode certifiant que l'utilisateur est un fournisseur

**function** isSupplier() {

**if** (**isset**($\_SESSION['user\_role']) && $\_SESSION['user\_role'] == 3) {

**return** TRUE;

} **else** {

**return** FALSE;

}

}

?>

WEB MOBILE (responsive)

Le responsive Web design est une approche de conception Web qui vise à l’élaboration de sites offrant une expérience de lecture et de navigation optimales pour l’utilisateur quelle que soit sa gamme d’appareil (téléphones mobiles, tablettes, liseuses, moniteurs d’ordinateur de bureau).

Une expérience utilisateur “Responsive” réussie implique un minimum de redimensionnement (zoom), de recadrage, et de défilements multidirectionnels de pages.

Voici un échantillon de mon script :

<?php

<div **class**="row d-flex justify-content-center my-4">

<div **class**="col-lg-12">

<div **class**="news text-center my-3 p-0">Actualité n°1</div>

</div>

</div>

<!-- 1er article -->

<div **class**="row lee">

<div **class**="col-md-6 d-flex justify-content-center py-3">

<img src="../assets/img/leeChongWei.jpg" **class**="wei" alt="" /></div>

<div **class**="col-md-5">

<h1 **class**="display text-center">Atteint d’un cancer, Lee Chong Wei ne participera pas aux Mondiaux</h1>

<p **class**="lead text-center">Atteint d’un cancer depuis l’été dernier, la star du badminton Lee Chong Wei ne participera pas aux Mondiaux en Suisse au mois

…

<p **class**="lead mt-2">Tombé à la 113e place mondiale, Lee Chong Wei a repris l’entraînement au mois de janvier, mais a repoussé plusieurs fois depuis son retour à la compétition.</p>

</div>

</div>

?>

Quelques explications :

on retrouvera le plus fréquemment des règles pour :

* agrandir la taille du texte
* agrandir la taille des contrôles et zones cliquables (pour une utilisation au doigt)
* faire passer le contenu sur une seule colonne
* masquer ou afficher des éléments spécifiques
* ajuster les dimensions et marges

Voici un échantillon de mon css :

<?php

@media only screen and (max-width: 600px) {

img {

width:100%;

display: block;

float: none;

margin-left: auto;

margin-right: auto;

}

body {

width:100%;

background-color: #FFFFFF;

display: block;

float: none;

margin-left: auto;

margin-right: auto;

}

#carousel{

width: 85%;

display: block;

float: none;

margin-left: auto;

margin-right: auto;

}

}

?>

DESCRIPTION D’UNE SITUATION DE TRAVAIL A PARTIR DE SITE ANGLOPHONE