

Compte Rendu E-Commerce

Parisot Clément

Table des matières

Chapitre 11 : Ajouter une catégorie et détail d'un produit	2
Détail d'un Produit.....	2
Ajouter une Catégorie.....	3
Chapitre 12 : Update un produit.....	5
Chapitre 13 : Supprimer un produit.....	8
Chapitre 14 : Envoyer un fichier.....	9
Chapitre 16 : Pages d'administration.....	12

Chapitre 11 : Ajouter une catégorie et détail d'un produit

Détail d'un Produit

Pour afficher de détail d'un produit, on commence par créer un nouveau Controller ainsi s'une page Twig (sans oublier les routes) :

```
function showProductController($twig, $db)
{
    include_once '../src/model/ProductModel.php';

    $product = null;

    if (isset($_GET['product'])) {
        $product = getOneProduct($db, $_GET['product']);
    }

    echo $twig->render('showProduct.html.twig', [
        'product' => $product
    ]);
}

<div class="container">
    {% if product %}
    <div class="col-12">
        <h1 class="mb-5">{{ product.label }}</h1>
    </div>
    <div class="row">
        <div class="col-md-4">
            <div class="card p-2">
                
            </div>
        </div>
        <div class="col-md-5 text-secondary">
            <p>{{ product.description }}</p>
        </div>
        <div class="col-md-3 bg-light text-center p-5">
            <h5 class="h1 text-primary">{{ product.price }} €</h5>
        </div>
    </div>
    {% else %}
    <p>Aucun produit n'est défini !</p>
    {% endif %}
</div>
```

On ajoute le bouton de sélection du produit :

Carte Graphique

ASRock Radeon RX 66!

C'est vraiment une super cart

Voir le produit

Et on a ce résultat-là :

ASRock Radeon RX 6650 XT Challenger D 8GB OC



C'est vraiment une super carte graphique de oufff

600.5 €

Ajouter une Catégorie

Pour ajouter la possibilité d'ajouter une catégorie, il a suffi de copier tout ce qui a été fait pour le produit et le remplacer tous les « product » par « category ».

```
function getOneCategory($db, $idCategory)
{
    $query = $db->prepare("SELECT id, label FROM categories WHERE id = ?");
    $query->execute([
        'id' => $idCategory
    ]);

    $category = $query->fetch();

    return $category;
}

function getAllCategories($db)
{
    $query = $db->prepare("SELECT id, label FROM categories");
    $query->execute([]);

    $categories = $query->fetchAll();

    return $categories;
}

function saveCategory($db, $label) {
    $query = $db -> prepare("INSERT INTO categories (label) VALUES (?)");
    return $query -> execute([
        'label' => $label
    ]);
}

function getOneProduct($db, $idProduct)
{
    $query = $db->prepare("SELECT id, label FROM products WHERE id = ?");
    $query->execute([
        'id' => $idProduct
    ]);

    $product = $query->fetch();

    return $product;
}

function getAllProducts($db)
{
    $query = $db->prepare("SELECT id, label FROM products");
    $query->execute([]);

    $products = $query->fetchAll();

    return $products;
}

function saveProduct($db, $label, $description, $price, $category) {
    $query = $db -> prepare("INSERT INTO products (label, description, price, category) VALUES (?, ?, ?, ?)");
    return $query -> execute([
        'label' => $label,
        'descr' => $description,
        'price' => $price,
        'idCategory' => $category
    ]);
}
```

On a donc ce résultat où on peut ajouter les catégories :

Add Category

Label

Submit

Pour ensuite avoir dans la page « addProduct » toutes les catégories ajoutées automatiquement, il suffit d'ajouter dans le Contrôleur la commande créée pour récupérer toutes les catégories :

```
echo $twig->render('addProduct.html.twig', [
    'form' => $form,
    'categories' => getAllCategories($db)
]);
```

Puis, dans le Twig, ajouter une boucle For dans le select :

```
<select name="productCategory" id="productCategory" class="form-select">
    {% for category in categories %}
        <option value="{{category.id}}">{{category.label}}</option>
    {% endfor %}
</select>
```

Ceci récupère toutes les catégories et, en passant par toutes les cases du tableau, affiche et propose les catégories tout en n'envoyant que l'ID de la catégorie, comme ce qui avait été fait auparavant.

De même dans le HomeController pour avoir la catégorie qui s'affiche au-dessus du produit :

```
function homeController($twig, $db) {  
  
    include_once("../src/model/ProductModel.php");  
    include_once("../src/model/CategoryModel.php");  
  
    echo $twig -> render('home.html.twig', [  
        'products' => getAllProducts($db),  
        'categories' => getAllCategories($db)  
    ]);  
}
```

Puis, dans le Twig on rajoute une boucle pour détecter la catégorie du produit :

```
<div class="col-md-6 d-flex align-items-center">  
    <div class="card-body">  
        {% for category in categories %}  
            {% if category.id == product.idCategory %}  
                <p class="card-text">{{category.label}} </p>  
            {% endif %}  
        {% endfor %}  
        <h5 class="card-title">{{product.label}}</h5>  
        <p class="card-text">{{product.description}}</p>  
  
        <a href="?page=showProduct&product={{ product.id }}" class="btn btn-primary">  
            Voir le produit  
        </a>  
    </div>  
</div>
```

En passant par toutes les catégories de « categories », si les ID concordent, afficher le nom de la catégorie au-dessus du nom du produit.

Ce qui donne ce résultat :

Carte Graphique

ASRock Radeon RX 6650 XT

C'est vraiment une super carte graphi

Voir le produit

Carte Graphique

deuxième carte graph fjezj

C'est vraiment une super carte graphi

Voir le produit

Disque Dur

Disque dur 1To SSD de la mc

Ca va super vite brrrr

Voir le produit

Afin d'avoir les erreurs qui s'affichent dans le Twig, il faut ajouter des conditions dans celui-ci :

```
<div class="card px-3 bg-light" style="width: 25rem;">

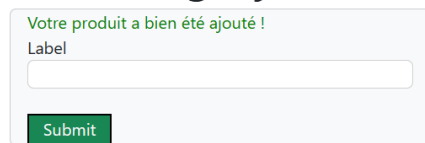
    {% if form.state == "success" %}
    <div style="color: green;">{{form.message}} </div>
    {% endif %}
    {% if form.state == "danger" %}
    <div style="color: red;">{{form.message}} </div>
    {% endif %}

    <label for="productLabel">Label</label>
    <input type="text" class="card" name="productLabel" id="pr
```

Si « form » signale un succès, alors afficher le message de succès, et si « form » signale un danger, afficher le message de danger.

Ce qui donne le résultat suivant dans le cas d'un succès :

Add Category



Chapitre 12 : Update un produit

Afin de modifier un produit, il nous faut une nouvelle page, en commençant par le Twig :

```
<div class="mb-3">
    <label for="productDescription" class="form-label">Description</label>
    <textarea class="form-control" id="productDescription" name="productDescription"
    rows="3">{{ product.description }}</textarea>
</div>

<div class="mb-3">
    <label for="productPrice" class="form-label">Prix</label>
    <input type="number" class="form-control" id="productPrice" name="productPrice" step="0.01"
    value="{{ product.price }}">
</div>

<div class="mb-3">
    <select class="form-select" name="productCategory">
        {% for category in categories %}
        <option value="{{category.id}}">{{category.label}}</option>
        {% endfor %}
    </select>
</div>

<button type="submit" class="btn btn-primary" name="btnPostProduct">Update</button>
</div>
```

Le code permettant d'afficher toutes les catégories a également été ajouté, le tableau « categories » ayant été transmis par le Controller :

```
echo $twig->render('updateProduct.html.twig', [
    'product' => $product,
    'page' => '?page=updateProduct&product=' . $product->id,
    'categories' => getAllCategories($db)
]);
```

On ajoute ensuite la fonction qui permet de modifier le produit dans la base de données :

```
function updateOneProduct($db, $id, $label, $description, $price, $category)
{
    $query = $db->prepare("UPDATE product SET label=:label,
`description`=:descr, price=:price, idCategory=:category WHERE id=:id");
    return $query->execute([
        'id' => $id,
        'label' => $label,
        'descr' => $description,
        'price' => $price,
        'category' => $category,
    ]);
}
```

Avec comme modification le bon nom de la table « product ».

Pour accéder à la page, on ajoute la route :

```
'addProduct' => 'addProductController',  
'updateProduct' => 'updateProductController',  
'addCategory' => 'addCategoryController'
```

Ainsi que le Controller en ajoutant « categories » afin de récupérer les catégories dans le Twig :

```
        $price = $_POST['productPrice'];  
        if (empty($price)) {  
            $price = 0.00;  
        }  
        $category = $_POST['productCategory'];  
        if (!empty($label) && !empty($description) && !empty($category)) {  
            updateOneProduct($db, $product['id'], $label, $description, $price, $category);  
        }  
        echo $twig->render('updateProduct.html.twig', [  
            'product' => $product,  
            'page' => '?page=updateProduct&product=' . $product['id'],  
            'categories' => getAllCategories($db)]  
        ]);  
    } else {  
        echo $twig->render('home.html.twig', []);  
    }  
}
```

Grâce à cela, on pourra avoir toutes les catégories dans la barre de sélection de la page de modification.

On se retrouve donc avec ce résultat :

Modifier un produit

Libellé

ASRock Radeon RX 6650 XT Challenger D 8GB OC

Description

C'est vraiment une super carte graphique de oufff

Prix

600,5

Carte Graphique

Update

Il ne manque que les affichages d'erreurs

Pour afficher les erreurs sur la page de modification de produit, il faut d'abord savoir quelles erreurs sont possibles d'afficher :

- Succès de la modification (pas vraiment une erreur mais rentre dans la même catégorie)
- Erreur lors de la modification
- Le produit n'existe pas (l'ID ne concorde avec rien qui est dans la base de données)

```

$form = [];

if (isset($_GET['product'])) {

    $product = getOneProduct($db, $_GET['product']);

    if (isset($_POST['btnPostProduct'])) {
        $label = $_POST['productLabel'];
        $description = $_POST['productDescription'];
        $price = $_POST['productPrice'];
        if (empty($price)) {
            $price = 0.00;
        }
        $category = $_POST['productCategory'];
        if (!empty($label) && !empty($description) && !empty($category)) {
            updateOneProduct($db, $product['id'], $label, $description, $price, $category);
            $form['state'] = 'success';
            $form['message'] = 'Votre produit a bien été modifié !';
        } else {
            $form['state'] = 'danger';
            $form['message'] = 'Veuillez remplir tous les champs !';
        }
    }
}

```

On commence par ajouter le tableau « form » et les assignations en fonction de la réussite ou de l'échec.

La première association étant pour le succès, la seconde est pour vérifier que tous les champs sont remplis.

```

if ($product == null) {
    $form['state'] = 'non-existent';
    $form['message'] = 'Le produit n\'existe pas !';

    echo $twig->render('updateProduct.html.twig', [
        'form' => $form
    ]);
    return 0;
}

echo $twig->render('updateProduct.html.twig', [
    'product' => $product,
    'page' => '?page=updateProduct&product=' . $product['id'],
    'categories' => getAllCategories($db),
    'form' => $form
]);
} else {
    $form['state'] = 'non-existent';
    $form['message'] = 'L\'identifiant de produit est erroné !';

    echo $twig->render('updateProduct.html.twig', [
        'form' => $form
    ]);
}
}

```

Puis on vérifie si le produit existe dans la base de données, s'il n'existe pas alors on renvoie la page d'erreur disant que le produit n'existe pas. Puis, pour pas que la fonction continue et affiche deux fois la page, on met un return 0.

Si le produit existe, le code continue et affiche la page normale avec le produit.

Si la variable « product » dans l'url n'existe pas, il renvoie vers une autre page d'erreur.

Les pages d'erreur « produit n'existe pas » et « url invalide » proposent un bouton renvoyant vers la page d'accueil. Ce qui nous donne ces résultats :

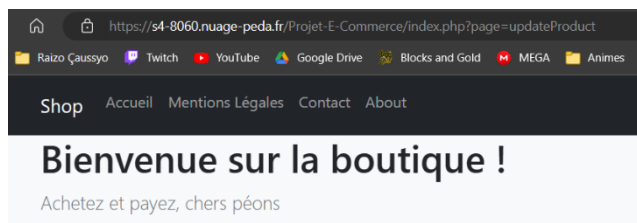
Modifier un produit

Votre produit a bien été modifié !

Libellé

ASRock Radeon RX 6650 XT Challenger D 8GB OC

Description



Modifier un produit

L'identifiant de produit est erroné !

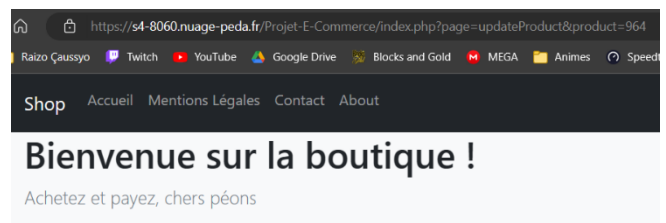
[Retourner à la page d'accueil](#)

Modifier un produit

Veuillez remplir tous les champs !

Libellé

Description



Modifier un produit

Le produit n'existe pas !

[Retourner à la page d'accueil](#)

Chapitre 13 : Supprimer un produit

Afin de supprimer un produit, on doit d'abord créer le Controller et la requête SQL :

```
function deleteOneProduct($db, $id)
{
    $query = $db->prepare("DELETE FROM product WHERE id=:id");
    $query->execute([
        'id' => $id
    ]);
    if ($query->rowCount() > 0) {
        $result = true;
    } else {
        $result = false;
    }
    return $result;
}
```

```
function deleteProductController($twig, $db)
{
    include_once '../src/model/ProductModel.php';

    $product = deleteOneProduct($db, $_GET['product']);

    if ($product) {
        $form = [
            'state' => 'success',
            'message' => 'L\'enregistrement ' . $_GET['product'] .
        ];
    } else {
        $form = [
            'state' => 'danger',
            'message' => 'L\'enregistrement ' . $_GET['product'] .
        ];
    }

    echo $twig->render('message.html.twig', [
        'form' => $form
    ]);
}
```

En cliquant sur le bouton de suppression d'un produit, on est renvoyé vers une page nous disant si le produit a bien été supprimé :

L'enregistrement 10 a bien été supprimé !

Chapitre 14 : Envoyer un fichier

Nous allons donc pouvoir envoyer des images, mais cela est source de faille de sécurité si non-traité.

On commence par créer l'input dans le Twig :

```
<div class="mb-3">
  <label for="productImage" class="form-label">Image</label>
  <input type="file" class="form-control" id="productImage" name="productImage">
</div>
```

Puis on ajoute de traitement de l'image et du titre dans le Controller :

```
if (!isset($_FILES["productImage"])) {  
    if (!empty($_FILES["productImage"]['name'])) {  
        $file_upload = explode(".", $_FILES["productImage"]['name']);  
        // Vérification de l'extension  
        if (in_array($file_upload[count($file_upload) - 1], $uploads['extensions'])) {  
            // Nettoyage des accents  
            $file_name = strtr(  
                $file_upload[0],  
                'AAAAAAÇÊËÉÎĨİŦŌŎŐŪŬŮŸYáàâãäåæçèéêëîïĩïöôõöüûÿý',  
                'AAAAAAAAEEEEIIIIIIOOOOUUUUYaaaaaaaceeeeeeiiiiiioooooouuuuyy'  
            );  
            // Nettoyage des caractères spéciaux  
            $file_name = preg_replace('/([^\.\a-z0-9]+)/i', '_', $file_name);  
            $file_name = $file_name . '.' . $file_upload[count($file_upload) - 1];  
            $file_path = $uploads['path'] . $file_name;  
            if (!file_exists($file_path)) {  
                // Déplacement du fichier  
                move_uploaded_file($_FILES['productImage']['tmp_name'], $file_path);  
                $uploads['state'] = true;  
            } else {  
                $msg = "Une image avec ce nom existe déjà !";  
            }  
        } else {  
            $msg = "L'extension du fichier n'est pas acceptée !";  
        }  
    } else {  
        $msg = "Veuillez choisir un fichier !";  
    }  
}  
  
if (!empty($label) && !empty($description) && !empty($category)) {  
    if ($uploads['state']) {  
        $form = [  
            'state' => 'success',  
            'message' => 'Votre produit a bien été ajouté !'  
        ];  
    } else {  
        $form = [  
            'state' => 'danger',  
            'message' => $msg  
        ];  
    }  
}
```

On crée un fichier dans `public/` qui servira à stocker les uploads, et on donne accès à ce dossier à tout le monde :

```
144  chmod -R 777 uploads/
145  ls -la
```

Il faudra plus tard passer les permissions de ce dossier en 755.

Il faut pouvoir stocker le chemin de l'image dans la base de données donc on ajoute une colonne « image » à la table des produits stockant une chaîne de caractères :

Nom	Type	Taille/Valeurs*	Valeur par défaut	Interclassement	Attributs	Null	Index
image	VARCHAR	255	Aucun(e)			<input checked="" type="checkbox"/>	---

Tout fonctionne lorsque l'on ajoute un nouveau produit avec une image :

On voit que les caractères ont également été retirés.

multicarte

Image

noire

Carte Graphique

deuxième carte g

C'est vraiment une sup

Voir le produit

M

Image

Disque Dur

Disque dur 1To S5

Ca va super vite brrrr

Voir le produit

M

Image

Disque Dur

Rgb

C'est du rgb

Voir le produit

M

Nous pouvons faire la même modification dans le Twig du showProduct :

```
</div>
<div class="row">
  <div class="col-md-4">
    <div class="card p-2">
      <div class="col-md-4">
        {% if product.image %}
        
        {% else %}
        
        {% endif %}
      </div>
    </div>
  </div>
  <div class="col-md-5 text-secondary">
    <p>{{ product.description }}</p>
  </div>
</div>
```

Afin de faciliter la réutilisation du code, on crée un dossier « service » qui sert à y poser toutes les fonctions qui peuvent être réutilisées dans plusieurs endroits, et un fichier « Upload » avec dedans comme fonction :

```
<?php
function VerifyImageName($image, $uploads)
{
    $file_name = null;

    $file_upload = explode(".", $image);
    // Vérification de l'extension
    if (in_array($file_upload[count($file_upload) - 1], $uploads['extensions'])) {
        // Nettoyage des accents
        $file_name = strtr(
            $file_upload[0],
            'ÀÁÂÃÄÅÇÈÉÊËÌÍÎÏÐÒÓÔÕÖÙÚÛÜÝàáâãäåçèéêëìíîïðóôõöùúûüýÿ',
            'AAAAAACCEEEIIIIIOOOOUUUUYaaaaaceeeiioooooouuuuyy'
        );
        // Nettoyage des caractères spéciaux
        $file_name = preg_replace('/([^\.\a-z0-9]+)/i', '_', $file_name);
        $file_name = $file_name . '.' . $file_upload[count($file_upload) - 1];
        $file_path = $uploads['path'] . $file_name;
        if (!file_exists($file_path)) {
            // Déplacement du fichier
            move_uploaded_file($image['tmp_name'], $file_path);
            $uploads['state'] = true;
        } else {
            $msg['state'] = False;
            $msg['msg'] = "Une image avec ce nom existe déjà !";
            return $msg;
        }
    } else {
        $msg['state'] = False;
        $msg['msg'] = "L'extension du fichier n'est pas acceptée !";
        return $msg;
    }

    $res = [
        "state" => True,
        "file" => $file_name
    ];
    return $res;
}
```

Cette fonction sert à vérifier le nom d'une image afin et de renvoyer l'état de l'upload de l'image, et de l'upload complètement dans le server. Elle renvoie également en cas de réussite le chemin absolu de l'image afin de pouvoir conserver ce chemin dans la base de données.

```
if (isset($_FILES["productImage"])) {
    if (!empty($_FILES["productImage"]['name'])) {
        $uploads = [
            'extensions' => ['png', 'jpg'],
            'path' => 'uploads/',
            'state' => false
        ];
        // Tu dois mettre la fonction demandée du coup mdr
        $verify = VerifyImageName($_FILES["productImage"], $uploads);
        $file_name = '';
        $msg = '';
        if ($verify['state']) {
            $file_name = $verify['file'];
        } else $msg = $verify['msg'];

        $uploads['state'] = $verify['state'];
    }
}
```

Mise en pratique, cette fonction permet de drastiquement réduire la taille du code et de faciliter la compréhension de ce qu'il se passe.

Chapitre 16 : Pages d'administration

Pour avoir des pages d'administrations, il faut commencer par créer ces pages en commençant par les routes :

```
'adminProducts' => 'adminProductsController',  
'adminProductCategories' => 'adminProductCategoriesController',
```

Puis on fait les Controller :

```
function adminProductCategoriesController($twig, $db)  
{  
    include_once '../src/model/ProductModel.php';  
  
    var_dump($_POST);  
  
    echo $twig->render('adminProductCategories.html.twig', []);  
}
```

Pour l'instant ils ne renvoient que la page Twig, sans faire de traitement de quoi que ce soit.

Et enfin les Twig avec simplement un titre pour débiter :

```
-E-Commerce > template > adminProductCategories.html.twig  
{% extends "layout.html.twig" %}  
  
{% block title %}Gestion des catégories de produit{% endblock %}  
  
{% block content %}  
<div class="container">  
|   <h1>Gestion des catégories de produit</h1>  
</div>  
{% endblock %}
```

Ensuite il faut récupérer les informations des produits et des catégories dans le Twig dans un tableau :

```
function adminProductsController($twig, $db)  
{  
    include_once '../src/model/ProductModel.php';  
    include_once '../src/model/CategoryModel.php';  
  
    var_dump($_POST);  
  
    echo $twig->render('adminProducts.html.twig', [  
        'products' => getAllProducts($db),  
        'categories' => getAllCategories($db)  
    ]);  
}
```

On récupère les informations des produits et des catégories afin d'avoir le nom des catégories liés aux catégories de produit.

On crée ensuite un tableau qui affiche tous les produits ligne par ligne, avec une condition qui récupère le libellé de la catégorie du produit :

```
<table class="table table-bordered table-striped">
  <thead>
    <tr>
      <th scope="col">ID</th>
      <th scope="col">Libellé</th>
      <th scope="col">Prix</th>
      <th scope="col">Catégorie</th>
      <th scope="col">Actions</th>
    </tr>
  </thead>
  <tbody>
    {% for product in products %}

    <tr>
      <th scope="row">{{product.id}}</th>
      <td>{{product.label}}</td>
      <td>{{product.price}}</td>

      {% for category in categories %}
      {% if product.idCategory == category.id %}
      <td>{{category.label}}</td>
      {% endif %}
      {% endfor %}

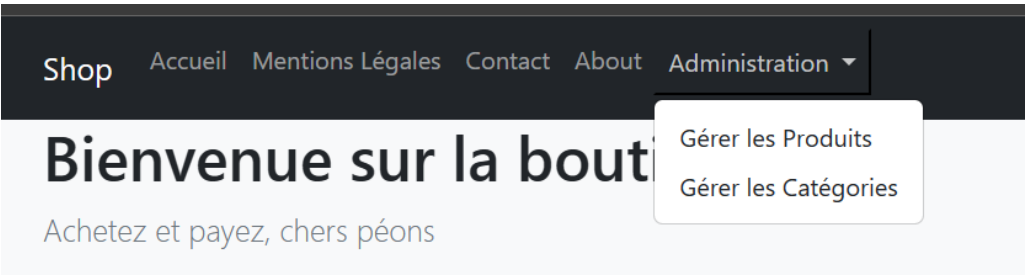
      <td><a href="?page=showProduct&product={{ product.id }}" class="btn btn-primary">Voir le produit</a>
      <a href="?page=updateProduct&product={{ product.id }}" class="btn btn-primary">Modifier le produit</a>
      <a href="?page=deleteProduct&product={{ product.id }}" class="btn btn-primary">Supprimer le produit</a>
      </td>
    </tr>
  </tbody>
</table>
```

On se retrouve avec ce résultat :

Gestion des produits

ID	Libellé	Prix	Catégorie	Actions
3	Disque dur 1To SSD de la mort qui tue	100.5	Disque Dur	<div>Voir le produit</div> <div>Modifier le produit</div> <div>Supprimer le produit</div>
14	d	51	Carte Graphique	<div>Voir le produit</div> <div>Modifier le produit</div> <div>Supprimer le produit</div>
15	d	5	Carte Graphique	<div>Voir le produit</div> <div>Modifier le produit</div> <div>Supprimer le produit</div>
16	d	5	Carte Graphique	<div>Voir le produit</div> <div>Modifier le produit</div> <div>Supprimer le produit</div>

En plus d'un dropdown menu pour accéder aux pages d'administration :



D'après vous, quelles sont les problématiques que vous pouvez observer à ce niveau du projet ?

On peut accéder aux modifications, ajout et suppression de produits et catégories à partir de n'importe quel appareil, tant qu'il est connecté à Internet et qu'il a l'URL d'hébergement du site.