

Projet Professionnel

Fatima EL MOUHINE



FORMATION DÉVELOPPEUR WEB ET WEB MOBILE

Sommaire

I.	Compétences du référentiel couvertes par le projet	4
II.	Résumé	5
III.	Cahier des charges	6
IV.	Spécifications techniques	9
V.	Réalisations	11
VI.	Jeu d'essai	19
VII.	Veille sur les vulnérabilités de sécurité	27
VIII.	Recherche anglophone	30
	ANNEXES	31

I. Compétences du référentiel couvertes par le projet

Le projet couvre les compétences énoncées ci-dessous.

Pour l'activité 1, “Développer la partie front-end d'une application web et web mobile en intégrant les recommandations de sécurité”:

- ❖ Maquetter une application
- ❖ Réaliser une interface utilisateur web ou mobile statique et adaptable
- ❖ Développer une interface utilisateur web dynamique

Pour l'activité 2, “Développer la partie back-end d'une application web ou web

mobile en intégrant les recommandations de sécurité.”:

- ❖ Créer une base de données
- ❖ Développer les composants d'accès aux données
- ❖ Développer la partie back-end d'une application web ou web mobile

II. Résumé

Mille Et Une Deco (1001 DECO) est une entreprise, basée dans le 3ème arrondissement de Marseille depuis 2012, qui vend des objets du quotidien (vaisselles, appareils électroménagers , produits d'entretiens, d'hygiène, etc) pour des professionnels et des particuliers.

Cependant, dû à l'accroissement de leur stock et leur variété de produits, faire l'inventaire est devenu un vrai casse-tête. Entre les stocks qui sortent du dépôt pour le magasin et ceux vendus au grossiste, une erreur peut vite arriver.

Pour optimiser son organisation et son temps de comptage d'inventaire, la société m'a sollicité pour créer une interface administrateur où elle pourra gérer ses stock de produits, les modifier et les supprimer. Mon client a voulu aussi voir ses fournisseurs et les associer au produit.

J'ai également eu, dans un second temps, pour mission de créer un catalogue présentable à ses clients professionnels, avec une solution de création de commande et de génération de facture.

La société m'a également demandé de développer cette solution sous format tablette pour qu'elle puisse l'utiliser sur ses appareils.

III. Cahier des charges

Périmètre du projet

Le site sera réalisé en français et ce dernier devra être accessible sur tablette et ordinateur en priorité.

Arborescence du site

L'arborescence du site se décline comme suit :

- ❖ Page d'accueil
- ❖ Page connexion (accessible que pour l'administrateur)
- ❖ Page inscription (accessible que pour l'administrateur)
- ❖ Page catalogue
- ❖ Page produit
- ❖ Page paramètre (qu'accessible pour l'admin , compte profil pour modifier ses accès)

Une partie back-office est également prévue afin de permettre la gestion du site.

Description des fonctionnalités

1. Catalogue produit et filtre

Une page doit permettre d'afficher l'ensemble des produits disponibles. Cet

affichage devra comprendre la photo du produit, le nom du produit ainsi que son prix.

Un filtre doit être implémenté afin de trier les articles en fonction de leur catégorie mais également en fonction du prix.

2. Fiche produit

Les clients devront être en mesure d'accéder à une fiche produit. Cette dernière devra comprendre:

- ❖ Le nom du produit
- ❖ Le prix
- ❖ La description du produit
- ❖ un bouton d'ajout au panier par unité de produit ou par carton

3. Panier

Les utilisateurs devront être en mesure de sélectionner un article et de le mettre dans son panier dans le but final que l'administrateur passe commande depuis son espace.

Ce panier devra afficher les éléments suivants:

- ❖ Une photo de l'article
- ❖ Le prix de l'article
- ❖ La quantité demandée par le client
- ❖ Le total de la commande
- ❖ Un bouton de validation du panier débouchant sur le processus de commande.

Les clients devront être en mesure d'augmenter ou diminuer la quantité demandée. Il aura aussi la possibilité de supprimer le un article du panier, voir l'intégralité du panier.

4. Fonctionnalité de recherche produit

Les clients devront être en mesure d'effectuer une recherche de produit ou de catégorie dans un champ prévu à cet effet. Afin de faciliter la recherche, un système de recherche avec autocompletion doit être mis en place.

5. Back office

Le gérant du site devra avoir accès à un espace sécurisé lui permettant d'administrer le site.

Pouvoir ajouter/modifier/supprimer (CRUD) :

A. Des catégories et sous-catégories du produit

B. Des produits

- ❖ Sur son inventaire de produit, une barre de recherche sera disponible pour gagner du temps
- ❖ Lors de la création du produit, ce dernier sera en mesure de:
 - ◆ Définir une référence et un titre au produit
 - ◆ Définir le prix de revient du produit à l'unité et à la vente
 - ◆ Définir le nombre de produit par carton ainsi que le nombre de carton
 - ◆ Définir la quantité total de son stock
 - ◆ Uploader une image du produit

C. Des fournisseurs

L'administrateur pourra également lier ses produits à des fournisseurs au préalablement ajouté

D. Des clients

Une fois la commande du client ajoutée au panier, l'administrateur devra se connecter pour alors lui créer une fiche contenant ses informations tel que son adresse et son contact pour faciliter la facturation et la livraison. Et s'il existe déjà en base de données, il pourra le sélectionner.

E. des commandes

- ❖ L'administrateur pourra voir les commandes passées et les traiter.
- ❖ Depuis ce même espace, il pourra également télécharger la facture du client qui sera automatiquement générer.

6. Notification du client

Un mail devra être envoyé au client afin de lui confirmer que sa commande a bien été passée. Ce mail devra contenir un récapitulatif de la commande.

IV. Spécifications techniques

Choix techniques et environnement de travail

Pour mener à bien ce projet, plusieurs outils ont été utilisés tels que :

- ❖ **Visual Studio Code** comme éditeur de code
- ❖ **Github** pour le versionning
- ❖ **Figma** pour le maquettage
- ❖ **Draw.io** pour la conception de la base de données (MCD¹, MLD²)

Les technologies utilisées sur la partie front-end sont:

- ❖ **HTML** et **CSS**.
- ❖ **Bootstrap** afin de faciliter le responsive
- ❖ **JavaScript** afin de dynamiser le site et d'améliorer l'expérience utilisateur.

Les technologies utilisées sur la partie back-end sont:

- ❖ **PHP**
- ❖ Base de données **SQL**
- ❖ Gestionnaire de dépendance: **Composer**
 - ◆ **Altorouter**
 - ◆ **Whoops** pour améliorer l'affichage des erreurs et débuguer plus facilement
 - ◆ **Dotenv** pour mettre en place des variables globales d'environnement
 - ◆ **Var_dumper**
 - ◆ **Dompdf** pour la création de pdf

Le projet a été développé en **Programmation orientée objet (POO)**, cela permet :

- ❖ que les programmes sont plus faciles à tester et à maintenir
- ❖ un développement plus rapide du code (développé en classes parallèles plutôt que séquentiellement).

Du point de vue de l'organisation, j'ai utilisé **Trello** afin de découper le projet en une multitude de tâches à réaliser et de définir leur ordre de priorité.

¹ MCD : Modèle Conceptuel de Données

² MLD : Modèle Logique de Données

Architecture du projet

J'ai utilisé un design pattern de type **MVC**³ pour segmenter mes fichiers.

Modèle : noyau de l'application qui gère les données, permet de récupérer les informations dans la base de données, de les organiser pour qu'elles puissent ensuite être traitées par le contrôleur.

Vue : composant graphique de l'interface qui permet de présenter les données du modèle à l'utilisateur.

Contrôleur : composant responsable des prises de décisions, gère la logique du code, il est l'intermédiaire entre le modèle et la vue.

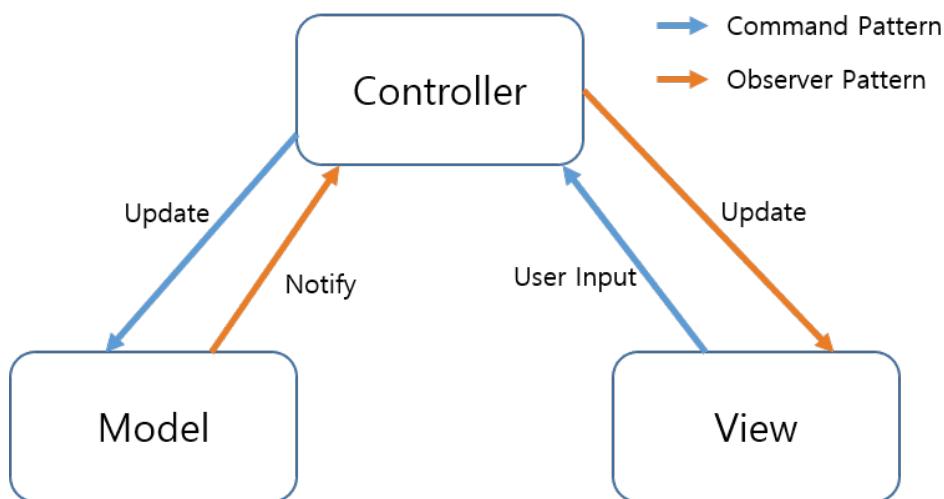


Schéma illustrant le design pattern MVC

J'ai aussi mis en place en place un routeur à l'aide du package **altorouter**, qui va rediriger toutes les requêtes vers un fichier index.php qui servira de carrefour, et qui inclura les bons fichiers en fonction de l'URL.

La création d'un fichier **.htaccess**⁴ a été nécessaire pour faire de la réécriture d'URL et rediriger toutes les requêtes vers le fichier index.php cité précédemment.

³ Model View Controller

⁴ fichier de configuration de server qui est propre à **apache**

V. Réalisations

A. Charte graphique

Le client n'avait pas vraiment d'idée de visuel, il m'a seulement indiqué qu'il voulait un site plutôt sobre

J'ai utilisé les polices :

- ❖ Roboto pour le texte
- ❖ Allerta Stencil pour les logos



B. Maquette

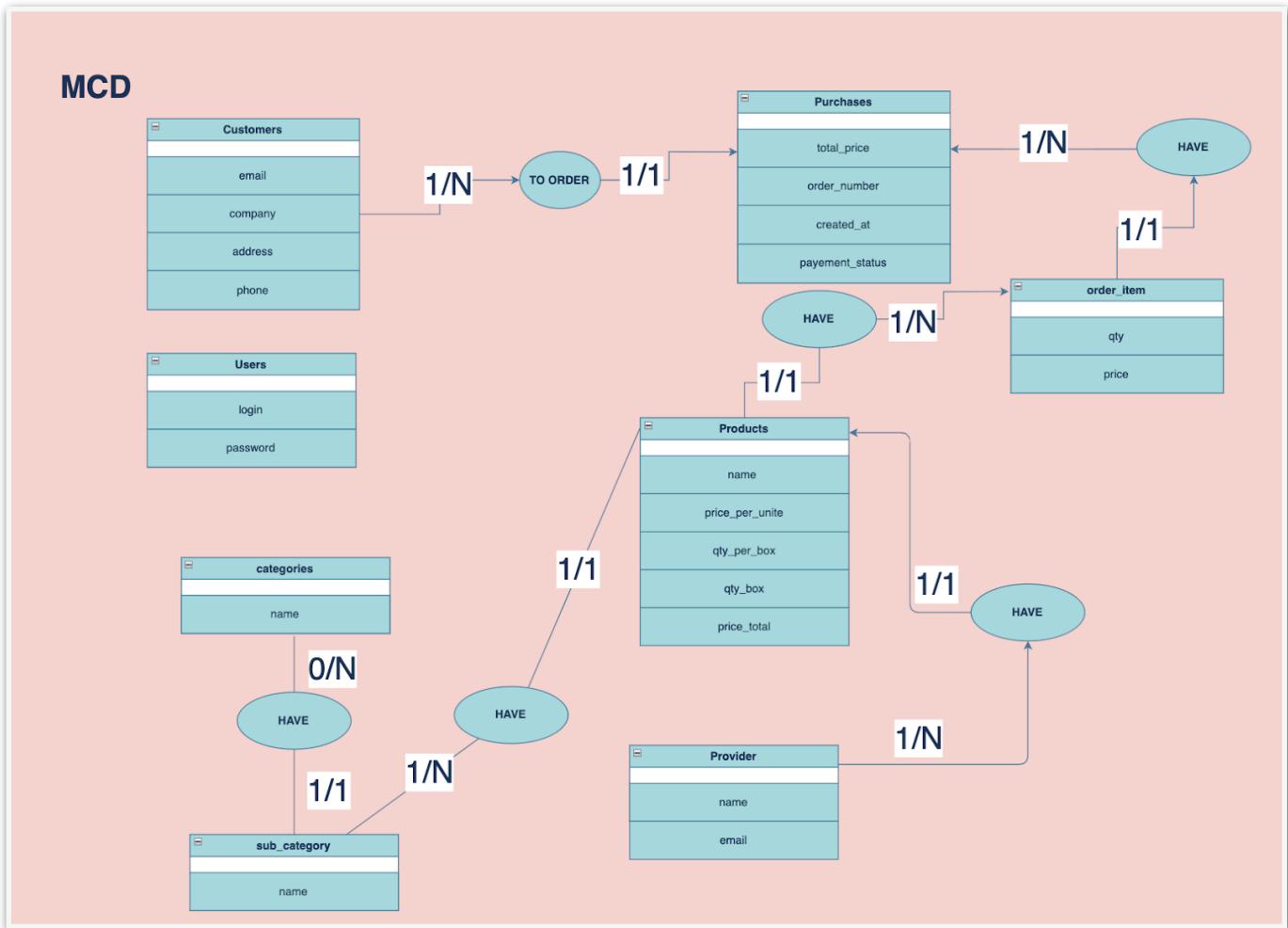
J'ai réalisé les maquettes avec **Figma**. Pour réaliser le design de la futur application, je me suis inspirée de sites de vente de grossiste qui plaisaient au client.

Les maquettes sont disponibles en annexe du dossier (p.32 et p.34)

C. Conception de la base de données

Durant ce projet, j'ai appliqué la méthode d'analyse, de conception et de gestion de projet informatique, **Merise**.

Au regard des fonctionnalités demandées par l'entreprise, j'ai développé la base de données suivante



Modèle conceptuel des données

Vous trouverez également en annexe le modèle logique de données. (p.31)

La base de données tourne majoritairement autour de la table **products**. Grâce au MCD, j'ai pu établir les différentes relations aux autres tables plus facilement.

Par exemple, la table **products** est relié à table **sub_category** par un verbe pour matérialiser leur relation comme on peut le voir illustré au dessus :

- ❖ **1 produit** a au **minimum 1 sous-catégorie** et au **maximum 1 sous-catégorie**.
- ❖ **1 sous-catégorie** doit contenir au **minimum 1 produit** et au **maximum N produits**.

J'ai créé également une table **users** pour pouvoir connecter l'administrateur.

La table **customers** stockera les informations des clients pour la facturation.

La table **purchases** et **order_items** contiendront les informations liée à la commande passée comme les produits sélectionnés, leurs quantités, la date, etc.

D. Extraits du code significatifs

Le dashboard admin était la première fonctionnalité que la société m'a demandé et elle m'a semblé la plus appropriée à aborder dans cet extrait.

Dans un souci d'optimisation de mon code, j'ai créé un fonction `run()` que j'ai utilisé tout le long de mon projet.

```
public function run($sql, $args = []){

    if (!$args)
    {
        $stmt = $this->db->query($sql);
    }else{
        $stmt = $this->db->prepare($sql);
        $stmt->execute($args);

    }
}
```

Cette fonction prend en paramètre la requête SQL à exécuter, si on ne lui passe pas d'autre argument alors elle utilisera la fonction `query()` pour l'exécuter. Sinon elle passera la requête dans la fonction `prepare()` et l'argument de type tableau dans la fonction `execute()`.

Les fonctions `query`, `prepare` et `execute` sont des fonctions de la classe **PDO**.

J'ai créé une classe contrôleur `AdminController` et une classe `Admin` pour le modèle qui contient le CRUD que l'administrateur va pouvoir effectuer sur son interface.

1. Création des produits

Pour la création du produit, j'ai d'abord créer une route en GET sur laquelle mon client pourra se rendre.

```
$router->map('GET', '/admin/addItems', function(){ AdminController::showAddItems();});  
$router->map('POST', '/admin/addProduct', function(){ AdminController::addProduct();});
```

Ajouter un produit

Référence

Désignation

Catégorie

Fournisseurs

Quantité par carton

Quantité de carton

Prix unitaire

Quantité total

Prix de vente

Photo du produit

Ajouter

```

$product = new Admin();

$reqCheck = $product->checkProduct($name, $reference);

if (!$reqCheck) {
    $product->reset();

    if(isset($_FILES) AND $_FILES['img']['size'] != 0 ){
        $img = $_FILES['img'];
        $img = Render::checkImage($img, 'product');
        $req = $product->insertProduct($reference, $name, $subCategoryId, $providerId, $qtyPerBox, $qtyOfBox, $priceByUnite,
$qtyTotal, $salePrice, $img);
    }else{
        $req = $product->insertProduct($reference, $name, $subCategoryId, $providerId, $qtyPerBox, $qtyOfBox, $priceByUnite,
$qtyTotal, $salePrice, $img);
    };
    $msg = "success";
}else{
    $msg = 'Cette référence avec ce nom de produit existe déjà';
}

echo json_encode($msg);

```

Une fois rendu sur cette page qui se présente comme ci dessus et le formulaire rempli et soumis, je vérifie si le produit avec ce nom et cette référence n'existe pas déjà en base de données. Si la vérification renvoie **true**, j'envoie un message d'erreur en JSON que je récupère dans une promesse en **Javascript** et que j'affiche dans une popup sans recharger la page.

Si la vérification renvoie **false**, ce produit n'existe pas, et après avoir vérifié que l'image était conforme, je peux l'ajouter en base de données. Pour cela dans mon modèle je crée une fonction **insertProduct()**.

```

$sql = "INSERT INTO $this->table (`reference`, `name_product`, `qty_per_box`, `qty_of_box`, `price_by_unite`, `qty_total`, `sale_price`, `img`, `id_sub_category`, `id_provider`)
VALUES (:reference, :name, :qtyPerBox, :qtyOfBox, :priceByUnite, :qtyTotal, :salePrice, :img, :subCategoryId, :providerId)";

$args = [
    ':reference' => $reference,
    ':name' => $name,
    ':qtyPerBox' => $qtyPerBox,
    ':qtyOfBox' => $qtyOfBox,
    ':priceByUnite' => $priceByUnite,
    ':qtyTotal' => $qtyTotal,
    ':salePrice' => $salePrice,
    ':img' => $img,
    ':subCategoryId' => $subCategoryId,
    ':providerId' => $providerId
],
    $req = $this->db->run($sql, $args);
]

```

J'utilise pour cela la requête **SQL INSERT**, et je créer un tableau contenant les valeurs à faire passer dans l'**execute()**.
Un message est alors envoyé à l'utilisateur pour le prévenir que son produit a bien été créé .

2. Modification des catégories

Pour la modification, j'effectue des actions assez similaires à l'ajout de produit, je vérifie dans la base de données si le nom de la catégorie est déjà pris en envoyant une requête en **POST** côté **JavaScript**. Si ce n'est pas le cas, je modifie la catégorie en base de donnée par le champ envoyé.



```

function fetchEdit(url, formdata){
    fetch(url, {
        method: 'POST',
        body: formdata
    })
    .then(res => {
        return res.json();
    })
    .then(data => {

```

```

buttonEditGroup.forEach(element => {
    element.addEventListener('click', ()=>{
        var id_category = element.getAttribute('data-id'),
            input = document.querySelector('input[data-id="'+ id_category +'"]'),
            formdata = new FormData();
        formdata.append("newCategorieName", input.value);
        var url = '/admin/categories/edit/' + id_category;
        fetchEdit(url, formdata)
    })
});

```

#	Nom	Action
1	Céramique	
3	Cuisson	
2	Thermos	
10	Verrerie	

#	Catégorie	Nom	Action
1	Cuisson	Céramique	
3	Céramique	Mug	
10	Verrerie	Mug	
12	Thermos	Thé	

Bravo

Votre catégorie a bien été modifiée

3. Suppression des fournisseurs

Pour la suppression, j'envoie dans **fetch()** une requête en **DELETE**, l'url correspond à une route que j'ai créée en amont.

```

$router->map("DELETE", '/admin/providers/delete/[i:id]', function($id){ AdminController::deleteProv($id);});

```

Avant d'envoyer la requête, je demande la **confirmation** de l'administrateur par mesure de sécurité et qu'il soit sûr de supprimer l'élément.

Une fois confirmé, je récupère **l'id** de l'élément que j'avais stocké au préalable dans l'attribut **data-id**.

```
var msgConfirm = "Êtes-vous sûr de vouloir supprimer cet élément ? ";
if (confirm(msgConfirm) = true) {
    var id_provider = element.getAttribute('data-id');
    var url = '/admin/providers/delete/'+id_provider;
    fetchDelete(url)
}
```

```
function fetchDelete(url){

    fetch(url , {
        method: 'DELETE'
    })
    .then(res =>{
        return res.json();
    })
    .then((res) =>{
```

```
static function deleteProv($id){
    $provider= new Admin();

    $provider->deleteProvider($id);

    echo json_encode("Votre fournisseur a bien été supprimé");

}
```

```
public function deleteProvider($id) {  
    $req = $this->db = DB::delete($this->table4, $id)  
}
```

*\$this->table4 correspond à la table **Providers**

IV. Jeu d'essai

1. Panier

J'ai choisi de stocker le panier de l'utilisateur en session. Pour ce faire, j'ai tout d'abord créé une classe '**Cart**' que j'ai appelée dans une classe enfant '**CartController**' , dans laquelle j'ai implémenté différentes méthodes pour ajouter un article, le supprimer, modifier les quantités.



```
//CART  
  
$router->map('GET', '/cart', function(){ CartController::cartView();});  
$router->map('POST', '/product/[i:id]/addToCart', function($id){ CartController::addToCart($id);});  
$router->map('GET', '/cart/delete/[i:id]', function($id){ CartController::deleteFromCart($id);});  
$router->map('POST', '/cart/update/[i:id]', function($id){ CartController::updateQty($id);});
```

J'ai préparé différentes routes en fonction des actions que l'utilisateur va effectuer sur le site.



```
static function addToCart($id){  
    if(isset($_POST['qtyByUnity']) && isset($_POST['qtyByBox']) && ($_POST['qtyByUnity'] >= 1 ||  
$_POST['qtyByBox'] >= 1 )){  
        if (!isset($_SESSION['allInfo'][$id])) {  
            $_SESSION['allInfo'][$id] = [];  
        }  
        if(isset($_SESSION['allInfo'][$id]) && in_array($_POST['id'], $_SESSION['allInfo'][$id])) {  
            $_SESSION['allInfo'][$id][1]+=$_POST['qtyByUnity'];  
            $_SESSION['allInfo'][$id][2]+=$_POST['qtyByBox'];  
        }else{  
            foreach( $_POST as $post){  
                array_push($_SESSION['allInfo'][$id], $post);  
            }  
        }  
    }  
}
```

fonction d'ajout au panier

J'ajoute le produit sélectionné dans mon tableau de session, après avoir vérifié s'il n'était pas déjà dedans. S'il est déjà présent, uniquement la quantité se mettra à jour.

```
qtyByUnityGroup.forEach(element =>{  
    element.addEventListener('click', ()=>{  
        var field = element.getAttribute('name')  
        console.log(field)  
        formdata.append(field, element.value)  
        var id_product = element.getAttribute('id')  
        fetch('/cart/update/'+id_product, {  
            method: 'POST',  
            body: formdata  
        })  
        .then((res)=>{  
            return res.json()  
        })  
    })  
}
```

Une modification plus précise se fera depuis la page panier grâce à un **fetch** en JavaScript. Une fois la requête envoyée, je fais la modification des quantités en session. Puis je renvoie un tableau d'informations avec les nouvelles valeurs pour les modifier dynamiquement.



```
static function updateQty($id){
    if(isset($_POST['qtyByUnity'])){
        $newQty = htmlentities($_POST['qtyByUnity']);
        $_SESSION['allInfo'][$id][1] = $newQty;
        $_SESSION['allInfo'][$id]['total'] = $newQty * $_SESSION['allInfo'][$id][4];
    }elseif(isset($_POST['qtyOfBox'])){
        $newQty = htmlentities($_POST['qtyOfBox']);
        $_SESSION['allInfo'][$id][2] = $newQty;
        $_SESSION['allInfo'][$id]['total'] = $newQty * $_SESSION['allInfo'][$id][6] * $_SESSION['allInfo'][$id][4];
    }
    $sum = self::sumTotal();
    $params = ['sum' => $sum, 'order_sum' => $_SESSION['allInfo'][$id]['total']];
    echo json_encode($params);
}
```

fonction de modification des quantités des éléments du panier

2. Crédation de la commande

Dans un second temps, une fois le panier de l'utilisateur confirmé, l'administrateur est redirigé vers un espace de connexion s'il n'est pas connecté.

Une fois connecté, il va créer la commande de son client. Pour cela, il devra choisir son client parmi une liste récupérée en base de données. Si le client n'est pas inscrit, il peut l'ajouter via un formulaire mis à disposition. S'il essaye de l'ajouter mais qu'il existe déjà une vérification en base de données est effectué et un message est renvoyé en JSON et est affiché grâce au JavaScript.

Panier

Mes achats

2 items

Shirt
test produit1

À l'unité

2

Au carton

3

€ 76,00

Shirt
test produit2

À l'unité

2

Au carton

4

€ 48,00

[← Back to shop](#)

Récapitulatif

Test produit1 € 76,00

Test produit2 € 48,00

PRIX TOTAL € 124,00

Nouvelles Commandes

Déjà client ? Nouveau client ?

Nom de la société

TEST

Email

test@client.fr

Adresse

11 rue d'hozier

Téléphone

0909090909

 Une erreur

X

Cette email existe déjà

```

static function chooseClient()
{
    if (!isset($_SESSION['client'])) {
        $_SESSION['client'] = [];
    }

    if (isset($_POST['clientEmail'])) {
        $clientEmail = htmlentities($_POST['clientEmail']);
        $clientAddress = htmlentities($_POST['clientAddress']);
        $clientCompany = htmlentities($_POST['clientCompany']);
        $clientPhone = htmlentities($_POST['clientPhone']);
        $client = new Admin();

        $reqCheck = $client->checkClient($clientEmail);

        if (!$reqCheck) {
            $client->reset();
            $req = $client->insertClient($clientEmail, $clientAddress, $clientCompany,
$clientPhone);
            $_SESSION['client'] = $req;

            $msg = 'success';
        } else {
            $msg = 'Cette email existe déjà';
            // header('Location: /admin/cart/newPurchase');
        }
        echo json_encode($msg);
    } elseif (isset($_POST['listClient'])) {
        $_SESSION['client'] = htmlentities($_POST['listClient']);
        header('Location: /admin/cart/recap');
    }
}

```

Ensuite, après la sélection ou la création de son client, il est redirigé vers une page de récapitulatif.

Je stocke en session l'id du client pour pouvoir afficher sur ce récapitulatif ses informations de facturation.

Récapitulatif

Informations facture:

Amala
Amala@hotmail.fr
8 rue d'hozier 13014
0991881199

Total à payer
€124,00

Informations produit:

	Shirt	U: 2	Ctn: 3
test produit1			
€ 76,00			
Quantité total d'unité 38			
	Shirt	U: 2	Ctn: 4
test produit2			
€ 48,00			
Quantité total d'unité 26			

Valider

Une fois cette étape validée, une commande est créée et le stock en base de données est mis à jour et le panier et les variables de session liées sont vidés.

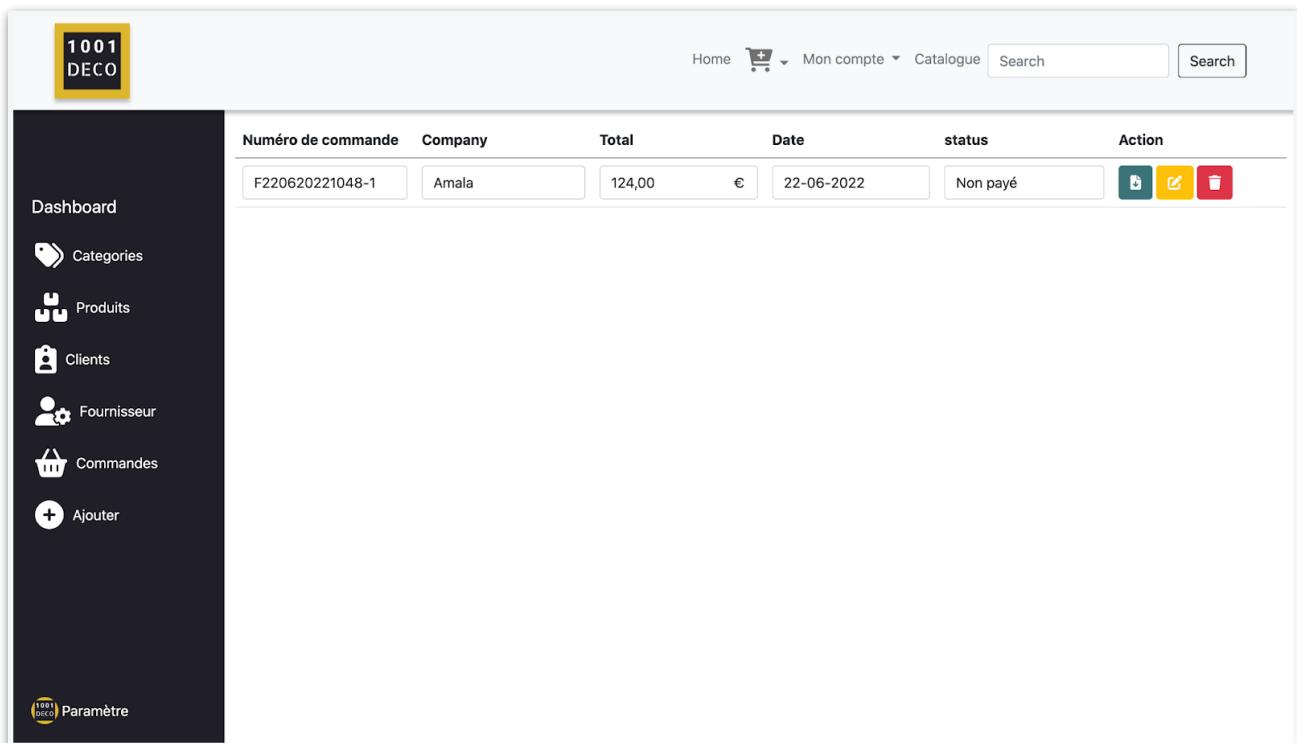
```
$idPurchase = $purchases->insertPurchase($idClient, $total, $orderRef);

//insert commande order_items
foreach ($_SESSION['allInfo'] as $info) {
    $id_product = $info[0];
    $refProduct = $info[7];
    $qtyTotal = ($info[2] * $info[6]) + $info[1];
    $infoProduct = "reference:$info[7]|price_sale:$info[1]|name:$info[5]|qty_per_box:$info[6]|total:$qtyTotal";
    $qty = "unity:" . htmlentities($info[1]) . "|box:" . htmlentities($info[2]);
    $price = htmlentities($info['total']);
    $orders = new Cart();
    $order = $orders->insertOrderItem($qty, $price, $idPurchase, $infoProduct);
    $product = new Cart;
    $req = $product->updateStock($id_product, $qtyTotal);
}
```

J'ai choisi de stocker en dur certaines informations liées à la commande pour pouvoir générer par la suite un pdf avec des informations immuables.

3. Génération de PDF

Une fois la commande enregistrée, depuis la section commandes dans le dashboard administrateur, on peut voir les commandes passées. J'ai mis à disposition un bouton pour pouvoir télécharger la facture en pdf. Pour cela j'ai utilisé la **librairie Dompdf**.



The screenshot shows a Laravel application's admin dashboard. At the top left is a yellow header bar with the text "1001 DECO". To its right are links for "Home", "Mon compte", "Catalogue", and a search bar. On the far right of the header is another search bar with the word "Search". The main content area has a dark sidebar on the left containing icons and labels for "Dashboard", "Categories", "Produits", "Clients", "Fournisseur", "Commandes", "Ajouter", and "Paramètre". The main area displays a table with columns: "Numéro de commande", "Company", "Total", "Date", "status", and "Action". A single row is shown with values: F220620221048-1, Amala, 124,00 €, 22-06-2022, Non payé, and three action buttons (edit, download, delete).

J'ai d'abord créé une route en **GET** que j'ai reliée à une fonction static **generatePdf()** que j'ai conçue dans dans le controller PurchaseController.



```
$router->map('GET', '/admin/purchase/{i:id}', function($id){ PurchaseController::generatePdf($id);});
```

Dans cette fonction, j'instancie un nouvel objet de la **classe Dompdf**. Je récupère également les informations de la commande sélectionnée par l'id passé en paramètre de la route.



```
static function generatePdf($id)
{
    $dompdf = new Dompdf();
    ob_start();
    $purchases = new Admin;
    $purchase = $purchases->getOnePurchase($id);

    $orders = $purchases->getOrdersByIdPurchase($id);
    include "invoice/invoice.php";

    $html = ob_get_clean();
    $dompdf->loadHTML($html);
    $dompdf->setPaper('A4', 'portrait');

    $dompdf->render();
    header("Content-type: application/pdf");
    header("Content-Disposition: inline; filename=Facture.pdf");
    $dompdf->stream("Facture.pdf", array("Attachment" => true));
}
```

Je crée également un template html qui contiendra ces informations et grâce aux fonctions de php **ob_start()** et **ob_get_clean()**, cela m'a permis de mettre en mémoire tampon, de lire le contenu courant du tampon de sortie puis de l'effacer.

La fonction **loadHTML()** de Dompdf m'a permis de préciser le contenu que l'on veut convertir.

J'ai pu aussi définir le format / l'orientation de rendu du pdf et le nom qui va être généré.

Vous pouvez voir le résultat du pdf en annexe (p.34)

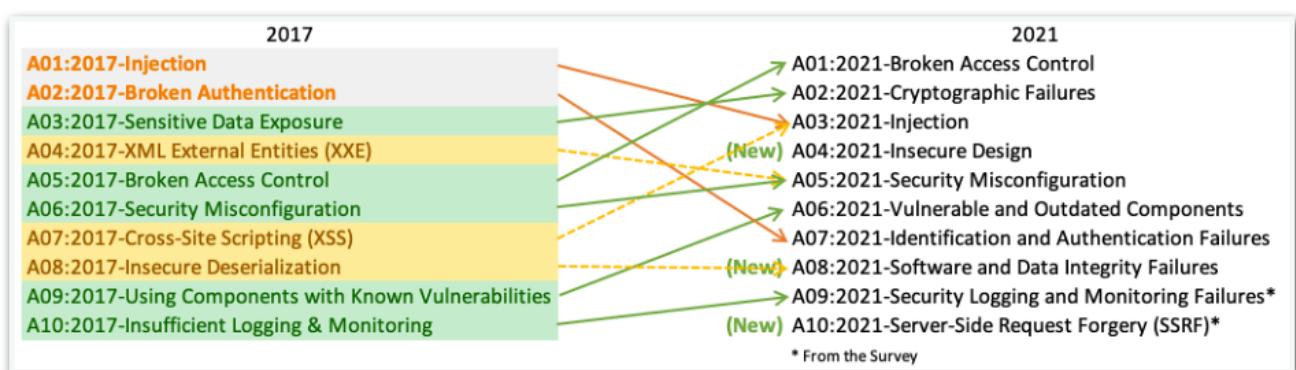
VII. Veille sur les vulnérabilités de sécurité

Les sites web sont des outils de plus en plus complexes qui permettent à des millions de personnes de traiter/échanger/stocker/consulter des données.

Ces systèmes ne sont malheureusement pas infaillibles et une personne malveillante peut vite prendre l'avantage.

C'est pour lutter contre ces failles, sensibiliser et nous former que des structures comme OWASP existent.

OWASP (Open Web Application Security Project) est une communauté en ligne partageant des outils, documents et méthodes libres pour la sécurisation des applications web. Elle publie régulièrement des rapports permettant d'évaluer les plus fréquentes vulnérabilités rencontrées dans le web, propose des solutions pour garantir la sécurité des applications et données et assure son intégrité face à l'évolution des menaces.



Top 10 des vulnérabilités web publié par OWASP en 2021

Je vais maintenant aborder quelque risque que j'ai pu contrer dans ce projet.

1. Injection SQL

Les attaques par injection de commandes SQL exploitent les failles de sécurité d'une application qui interagit avec des bases de données. L'attaque SQL consiste à modifier une requête SQL en cours par l'injection d'un morceau de requête non prévu, souvent par le biais d'un formulaire. Le hacker peut ainsi accéder à la base de données, mais aussi modifier le contenu et donc compromettre la sécurité du système.

Pour contrer cette faille, j'utilise des requêtes SQL paramétrées : des requêtes à trous sont envoyées au serveur SQL avec la fonction **prepare()** de la classe **PDO** qui les complétera à l'aide des paramètres qui lui sont envoyés avec la fonction **execute()**.

2. Faille XSS (Cross-site scripting)

C'est une attaque dans laquelle un hacker va injecter dans un site web de bonne foi un script malveillant en profitant d'une faille dans les contrôles de validité du site. Lors d'une visite de la page piégée ou lors du clic sur un lien piégé le script sera chargé à l'insu de l'utilisateur en même temps que les données, et il s'exécutera sur son ordinateur.

Ce script pourra faire diverses choses, comme afficher un popup qui abusera l'utilisateur sur son origine, détourner celui-ci vers un site de phishing, effectuer une usurpation d'identité qui permettra au pirate, par exemple, de consulter le compte bancaire de la victime, ou de faire une commande sur internet en se faisant passer pour lui.

Pour contrer cette faille, j'ai traité les informations récupérées des formulaires avec la fonction **htmlentities()** de php qui permet de convertir tous les caractères éligibles en **entités HTML**. Donc les balises seront converties.

Par exemple :

Ce que la personne a injecté dans le formulaire :

```
&lt;img src="./chat.jpg" title="Une image" />&lt;script&gt;alert('hackndo');&lt;/script&gt;&lt;p class="" /&gt;
```

```
<script>alert('hackndo');</script><p class="" />
```

Ce que la base de données reçoit :

3. Faille Upload

La faille upload est une faille permettant d'uploader des fichiers avec une extension non autorisée, cette faille est due à la mauvaise configuration du script d'upload ou à l'absence complète de sécurité. Celle-ci est généralement présente dans les scripts d'upload d'images. C'est une des failles les plus dangereuses.

Pour contrer cette faille, j'ai tout d'abord commencé par mettre un attribut accept qui permet de préciser quelle extension l'input accepte. Mais ce n'est pas suffisant puisque l'utilisateur peut le désactiver via l'inspecteur. J'ai donc créé une fonction qui vérifie **l'extension, le poids et le type MIME** du fichier.

Les fichiers uploadés n'ont pas à être de grande résolution, j'ai donc limité la taille des fichiers acceptés à 10Mo.

```

static function checkImage($image, $where){
    $image = $_FILES['img'];
    $maxSize = 10485760;
    $name = uniqid();
    $validExtension = array('jpg','jpeg','png','gif');
    if($_FILES['img']['size'] <= $maxSize){
        $extentionUpload = strtolower(substr(strrchr($_FILES['img']['name'], '.'), 1));
        if(in_array($extentionUpload,$validExtension)){
            if($where == 'product'){
                $path = 'images/uploads/products/'.$name.".".$extentionUpload;
            }else{
                $path = 'images/uploads/users/'.$name.".".$extentionUpload;
            }
            $moveAvatar = move_uploaded_file($_FILES['img']['tmp_name'], $path);
            if($moveAvatar){
                $name = "$name.$extentionUpload";
                return $name;
            }else{
                echo 'Une erreur est survenue lors de l\'importation de votre image';
            }
        }else{
            echo 'Votre image doit etre au format jpg, jpeg, png ou gif ! ';
        }
    }else{

```

4. Cryptage des données

Protéger les accès de mon client fut une priorité également , j'ai donc pour éviter de stocker en clair ses mot de passe, je les ai crypté grâce à la fonction **password_hash()** de php qui crée une clé de hachage pour le mot de passe et qui utilise l'algorithme de cryptage sécurisé **Bcrypt**.

J'ai également mis en place une expression régulième (**regex**) en place, lors de la création de son compte administrateur pour créer des mot de passe assez solide composé au minimum de 12 caractères dont 3 chiffres, 3 majuscules, 3 minuscules et 3 caractères spéciaux.

VII. Recherche anglophone

Lors de l'utilisation de la librairie **Dompdf**, j'ai rencontré le problème de faire afficher mon logo en svg sur la facture pdf.

Après avoir lu la documentation officielle sur le **github** de la dépendance, je n'ai pas trouvé d'élément de réponse.

J'ai donc mené plusieurs recherches en anglais pour trouver une solution.

Grâce à **stackoverflow**, le forum des développeurs et passionnés, quelqu'un avait rencontré le même problème et est arrivé à concevoir sa propre solution qui a fonctionné.

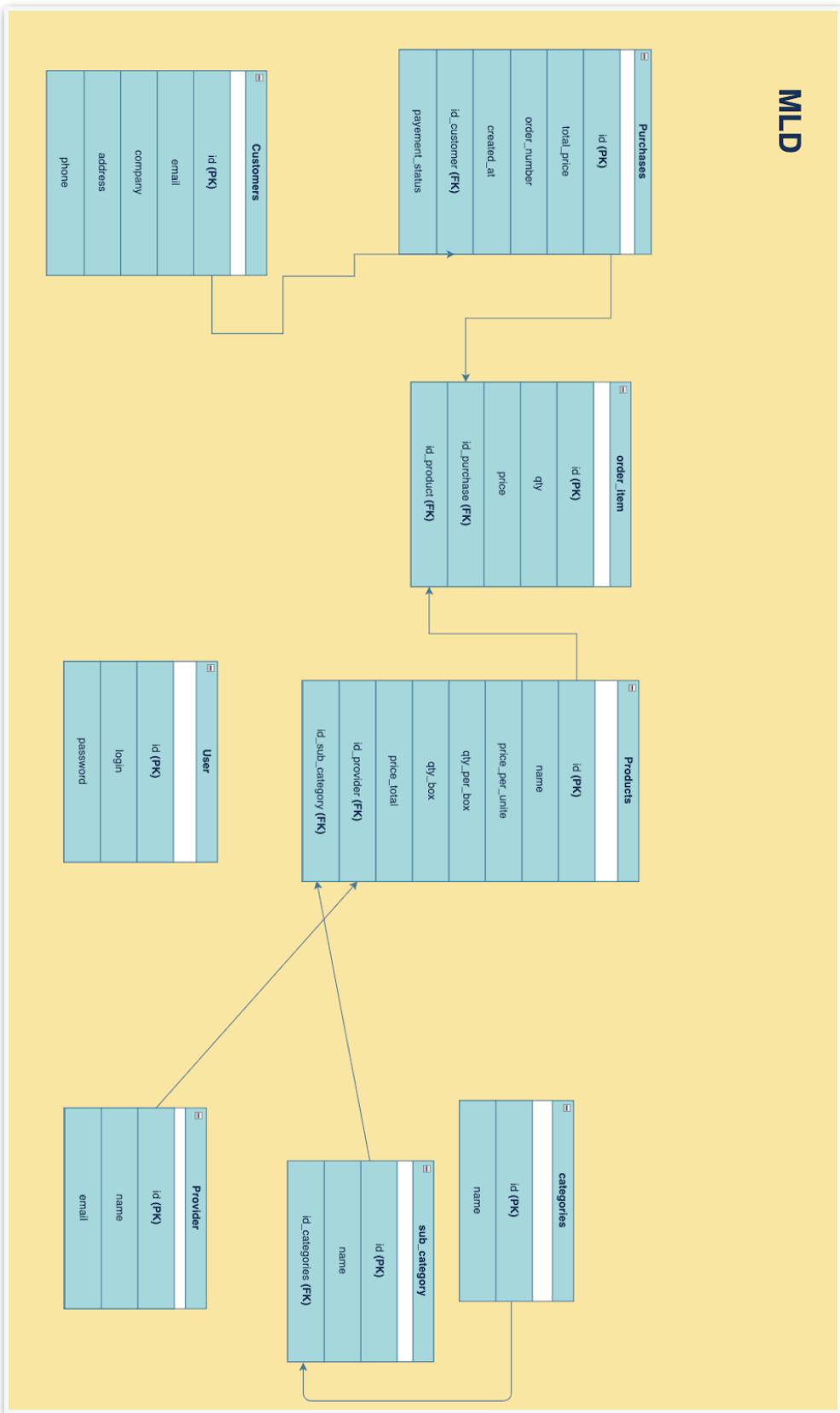
The screenshot shows a Stack Overflow post. The sidebar on the left includes links for Home, PUBLIC, Questions (which is selected), Tags, Users, Companies, COLLECTIVES, Explore Collectives, TEAMS, and Stack Overflow for. The main content area shows a question with 18 answers. One answer has been expanded, showing the following text and code:

I searched FOREVER until I finally came up with my own solution. I thought I should share in case it could be useful to anyone else. If you are using **PHP** the best solution to inline SVG images (for me) is to output it using **BASE64_ENCODE()** on the actual svg, then wrapping it in an image using the src as follows:

```
$svg = '<svg xmlns="http://www.w3.org/2000/svg" xmlns:xlink="http://www.w3.org/1999/xlink">
  <path class="circle-bg"
    stroke-width="1"
    fill="none"
    stroke="#ddd"
    d="M18 2.0845
      a 15.9155 15.9155 0 0 1 0 31.831
      a 15.9155 15.9155 0 0 1 0 -31.831"
  />
</svg>';
$html = '';
```

ANNEXES

Modèle logique de données



Maquette format tablette

Team project / 1001 DECO ▾

home

1001 DECO

Catégories

Assiettes Ustensiles Marmites

Assiettes Ustensiles Marmites

Nos tendances

Couvert Moule Verre à thé

Couvert Moule Verre à thé

products

1001 DECO

Produits

Catégories

- Thermos
- Verrerie
- Céramique
- Cuisson
- Ustensile à gâteau
- Alimentaires
- Couvert
- Cosmétique

Thé Café Eau

Mug Saladier Plat à four

Assiette Service orangeade

Inscription

Identifiant

Mot de passe

Confirmez

Connexion

Maquette format Mobile

The image displays a mobile wireframe for a 'Mille & Une Déco' application, organized into six screens:

- login**: A screen for entering credentials. It features a logo for 'MILLE & UNE DECO', fields for 'Identifiant' and 'Mot de passe', and a 'Connexion' button. A small illustration of a person sitting at a desk is visible.
- home**: The main dashboard screen. It includes a search bar, four main navigation buttons ('produit', 'catégorie', 'inventaire', 'fournisseur'), and a central figure of a person interacting with a screen.
- inventory**: A screen listing inventory items. Each item is represented by a thumbnail image, a checkbox, and a detailed description table. The descriptions include:
 - Assiette en porcelaine 18" - Prix unité : 1€ - Stock : 2000 pièces (50 carton) - Voir plus de détails
 - Assiette en porcelaine 18" - Prix unité : 1€ - Stock : 2000 pièces (50 carton) - Voir plus de détails
 - Assiette en porcelaine 18" - Prix unité : 1€ - Stock : 2000 pièces (50 carton) - Voir plus de détails
 - Assiette en porcelaine 18" - Prix unité : 1€ - Stock : 2000 pièces (50 carton) - Voir plus de détails
 - Assiette en porcelaine 18" - Prix unité : 1€ - Stock : 2000 pièces (50 carton) - Voir plus de détails
 - Assiette en porcelaine 18" - Prix unité : 1€ - Stock : 2000 pièces (50 carton) - Voir plus de détails
- product**: A detailed view of a product. It shows a large image of a decorative plate, a table with product details, and icons for trash and edit.
- new**: A screen for adding new products. It includes fields for 'Nom du produit', 'Qté par carton', 'Qté de carton', 'Fournisseur', and a 'Ajouter' button. A small illustration of a person holding a clipboard is present.
- success**: A confirmation screen stating 'Votre produit a bien été ajouté. Vous pouvez dès à présent le voir dans votre inventaire.' It features a 'Inventaire' button and a small illustration of two people with balloons.

Résultat du PDF



Facture

Adresse de facturation

Amala
Amala@hotmail.fr
8 rue d'hozier 13014
0991881199

1001 DECO
240 BD National
13003, Marseille
France
0491000008
contact@1001deco.fr

Facture N° F220620221048-1
Commandé le 22-06-2022
Status du paiement :
non payé

Référence	Désignation	Prix Unitaire TTC	Quantité Unité	Quantité Carton	Quantité Total	Total
ROPDDDF	test produit1	2.00 €	2	3	38 pcs	76.00 €
RO330013	test produit2	2.00 €	2	4	26 pcs	48.00 €
				Total € (TTC)	124.00€	

1001 DECO, Vente en gros et détail
240 Bd National 13003 , Marseille , France