



DOCUMENTATION TECHNIQUE

Stage 2021

Lycée Hôtelier Jean Monnet
12 Rue Louis Armstrong, 87000 Limoges

Table des matières

1. Description du projet	3
2. Connexion à la base de données	4
3. Structure de nos tables	5
4. Utilisation d'une architecture MVC	6
5. Modèles de connexion	7
5.1. Procédure de connexion	7
5.2. Système de hachage pour le mot de passe	7
6. Utilisation du front-end framework : Bootstrap	8
7. Les scripts Javascript	9
7.1. AutoComplete : client	9
7.2. AutoComplete : composition des productions	10
8. Outil de génération de PDF	11
9. Génération de statistiques	12



DOCUMENTATION TECHNIQUE

1. Description du projet

Réalisation d'une application de gestion de la boutique des ventes à emporter.

La boutique des ventes à emporter permet la vente de productions alimentaires réalisées par un groupe en formation (du lycée, de l'UFA ou du GRETA).

La liste des produits peut être connue par avance ou se complète en direct.

Certains ateliers travaillent pour une distribution en boutique (charcuterie, boucherie, pâtisserie, boulangerie).

Les cuisines distribuent généralement en restaurant mais peuvent avoir des restes à fournir à la boutique.

Création d'un catalogue des productions afin de permettre un suivi des productions et des ventes associées. Chaque production appartient à une famille de produits et ce classement est en lien avec la tarification (exemple : boulangerie, charcuterie cuite...). Ces familles de produits peuvent être « alimentées » par les différents ateliers.

A chaque produit (production) est affecté une dénomination, un atelier de provenance, une famille de produits et donc une catégorie, professeur et 1 classe, un prix de vente normal, un prix forfaitaire, une température de conservation, une date de fabrication, une date de péremption, une quantité.

Le catalogue doit permettre : de rentrer une nouvelle production, de la modifier, de la supprimer, de pouvoir visualiser et imprimer les ventes par productions, d'imprimer ce catalogue.

Les clients doivent se caractériser par leur nom, prénom, numéro de téléphone et/ou mail.

Liste de clients, le logiciel doit prévoir une liste des clients avec le nom, l'adresse (uniquement ville), le mail ou téléphone, la possibilité de rajouter, modifier ou supprimer un client, la possibilité de consulter l'historique des factures réglées par le client.

Ils existent différents moyens de règlements : espèces, chèques, carte bleue, transfert atelier, transfert self, destruction (la destruction est actuellement un moyen de valorisation des produits arrivés à terme de DLC et détruits), le prix doit être un prix forfaitaire et non de vente client.

Lors de l'enregistrement des ventes et la facture, chaque client se voit attribuer une facture où sont indiqués : le nom, l'adresse et le numéro de téléphone et de fax de l'établissement, le nom du client, la date de la facturation finale, le numéro de facture, la désignation des produits achetés, la portion, la date limite de consommation, la quantité, le prix unitaire du produit, le sous total de chaque produit, le total de la facture, le mode de règlement.

Cette facture doit pouvoir restée ouverte sur plusieurs jours : pour un client qui prend un produit le mardi, en réserve pour le jeudi et passe finalement chercher ses produits le vendredi et règle sa facture à ce moment là.



2. Connexion à la base de données

Nous avons fait le choix d'utiliser la technologie PDO afin de relier la base de données avec notre script. Car si nous devons un jour utiliser un autre système de base de données, le changement sera beaucoup plus simple que si nous avions tout développé sous MySQLi auquel cas nous devrions réécrire le code dans son ensemble.

De plus, PDO est orienté objet, il supporte les requêtes préparées qui servent à se prémunir des injections SQL.

Utilisation dans notre projet :

```
function connexionPDO() {  
    $login = "root";  
    $mdp = "root";  
    $bd = "gestion_boutique";  
    $serveur = « localhost »;  
  
    try {  
        $conn = new PDO("mysql:host=$serveur;dbname=$bd", $login, $mdp,  
array(PDO::MYSQL_ATTR_INIT_COMMAND => 'SET NAMES \'UTF8\''));  
        $conn->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);  
        return $conn;  
    } catch (PDOException $e) {  
        print "Erreur de connexion PDO ";  
        die();  
    }  
}
```

Nous utiliserons cette fonction de connexion dans l'ensemble de nos requêtes vers la base de données.

Premièrement en incluant la page liée à cette fonction :

```
include_once(« pdo.php »);
```

Puis dans chacune de nos fonctions incluant la requête vers la base de données, nous stockons le résultat de notre fonction connexionPDO(), dans une variable que nous utiliserons.

```
$conn = connexionPDO();
```

3. Structure de nos tables

Afin de gérer nos données nous allons utiliser phpMyAdmin. Il s'agit d'une application Web de gestion pour les systèmes de gestion de base de données MySQL et MariaDB, réalisée principalement en PHP et distribuée sous licence GNU GP.

Notre base de données n'inclût aucune clé étrangère car elles auraient présentées trop d'inconvénients.

gestion_boutique production id : int(11) # id_classe : int(11) # id_produit : int(11) # id_atelier : int(11) # id_prof : int(11) # temperature : int(11) # date_fabrication : varchar(255) # date_peremption : varchar(255) # quantite : float # conditionnement : int(11) # etat_congelation : tinyint(1) # date_destruction : varchar(255) # etat_affichage : tinyint(1) # etat_transfert : int(11) # date_transfert : date	gestion_boutique mode_reglement id : int(11) # nom : varchar(255) gestion_boutique facture id : int(11) # id_moyen_reglement : int(11) # id_client : int(11) # date : date # date_reglement : date # total : float # etat_annulation : int(11) gestion_boutique classe id : int(11) # nom : varchar(255)	gestion_boutique role id : int(11) # role : varchar(250) gestion_boutique client id : int(11) # nom : varchar(255) # prenom : varchar(255) # ville : varchar(255) # telephone : varchar(255) # mail : varchar(255) # id_categorie : int(11) gestion_boutique vente id_facture : int(11) id_production : int(11) # quantite : int(11) # reservation : int(10) # prix_unitaire : float # prix_total : float
gestion_boutique atelier id : int(11) # nom : varchar(255) gestion_boutique categorie_client id : int(11) # nom : varchar(255) gestion_boutique type_produit id : int(11) # nom : varchar(255) # prix : float # id_famille : int(11) # id_unite : int(11)	gestion_boutique produit id : int(11) # id_famille : int(11) # id_unite : int(11) # denomination : varchar(255) # prix : float # etat_affichage : tinyint(1) gestion_boutique utilisateur id : int(11) # nom : varchar(255) # mot_de_passe : varchar(255) # role : int(11)	gestion_boutique total_atelier id : int(11) # nom : varchar(255) # id_facture : int(11) # id_production : int(11) # prix_total : float gestion_boutique professeur id : int(11) # nom : varchar(255) gestion_boutique famille_produit id : int(11) # nom : varchar(255) gestion_boutique unite id : int(11) # nom : varchar(255)

Chacune de nos tables a une clé primaire, avec comme spécificité l'utilisation **AUTO_INCREMENT**.

La commande **AUTO_INCREMENT** est utilisée dans le langage SQL afin de spécifier qu'une colonne numérique avec une clé primaire (PRIMARY KEY) sera incrémentée automatiquement à chaque ajout d'enregistrement dans celle-ci.

Comme indiqué précédemment aucune clé étrangère n'est renseignée or, il existe tout de même des liens avec les tables (exemple : dans la table facture, id est en lien avec id_facture de la table vente).

Chaque champs de chaque table est intitulé de manière explicite de façon à ce que l'explication exhaustive de ces champs en devient inutile.

4. Utilisation d'une architecture MVC

Nous avons utilisé l'architecture MVC pour notre projet car il s'agit d'un des plus célèbres design patterns, ce qui signifie **Modèle - Vue - Contrôleur**. C'est celui que nous allons découvrir maintenant.

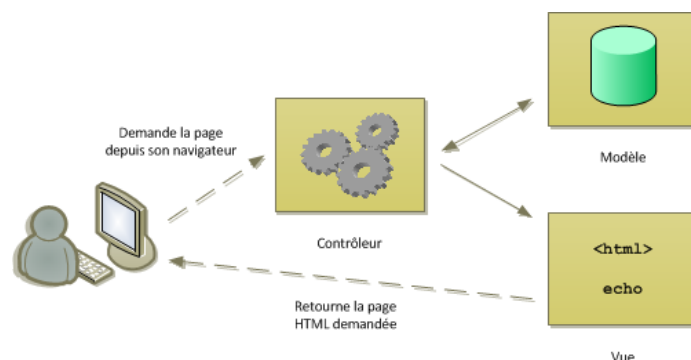
Le pattern MVC permet de bien organiser son code source. Il va vous aider à savoir quels fichiers créer, mais surtout à définir leur rôle. Le but de MVC est justement de séparer la logique du code en trois parties que l'on retrouve dans des fichiers distincts.

- **Modèle** : cette partie gère les données de votre site. Son rôle est d'aller récupérer les informations « brutes » dans la base de données, de les organiser et de les assembler pour qu'elles puissent ensuite être traitées par le contrôleur. On y trouve donc entre autre les requêtes SQL.
- **Vue** : cette partie se concentre sur l'affichage. Elle ne fait presque aucun calcul et se contente de récupérer des variables pour savoir ce qu'elle doit afficher. On y trouve essentiellement du code HTML mais aussi quelques boucles et conditions PHP très simples, pour afficher par exemple une liste de messages.
- **Contrôleur** : cette partie gère la logique du code qui prend des décisions. C'est en quelque sorte l'intermédiaire entre le modèle et la vue : le contrôleur va demander au modèle les données, les analyser, prendre des décisions et renvoyer le texte à afficher à la vue. Le contrôleur contient exclusivement du PHP. C'est notamment lui qui détermine si le visiteur a le droit de voir la page ou non (gestion des droits d'accès).

La figure suivante schématise le rôle de chacun de ces éléments.



Concrètement, le visiteur demandera la page au contrôleur et c'est la vue qui lui sera retournée, comme schématisée sur la figure suivante. Bien entendu, tout cela est transparent pour lui, il ne voit pas tout ce qui se passe sur le serveur. Il s'agit du type d'architecture utilisé par un grand nombre de sites professionnels !



5. Modèles de connexion

5.1. Procédure de connexion

La fonction ci-dessous va permettre la procédure de connexion et vérifier si l'utilisateur peut se connecter avec son nom.prénom et son mot de passe.

Divers paramètres sont pris en compte :

- * @param nom \$nom : Nom de l'utilisateur (nom.prénom)
- * @param mot_de_passe \$mot_de_passe : Mot de passe de l'utilisateur
- * @return boolean \$connectionApprouvee : Si "true" l'utilisateur peut se connecter

```
function ConnectionUtilisateur($nom, $mot_de_passe){  
  
    $connectionApprouvee = false;  
    $conn = connexionPDO();  
    $chercherUser = $conn->prepare("SELECT * FROM utilisateur WHERE nom  
= ?");  
    $chercherUser->execute(array($nom));  
  
    if($chercherUser->rowCount() != 0){  
        $ligne = $chercherUser->fetch(PDO::FETCH_OBJ);  
        if(password_verify($mot_de_passe, $ligne->mot_de_passe)){  
            session_start();  
            $_SESSION['user'] = $ligne;  
            $connectionApprouvee = true;  
        }  
    }  
    return $connectionApprouvee;  
}
```

5.2. Système de hachage pour le mot de passe

```
boolean password_verify ( string $password , string $hash )
```

Vérifie que le hachage fourni correspond bien au mot de passe fourni.

La fonction `password_hash()` retourne l'algorithme, le "cost", et le salt comme parties du hachage retourné. Toutefois, toutes les informations nécessaires pour vérifier le hachage y sont incluses. Ceci permet à la fonction de vérifier le hachage sans avoir besoin d'un stockage séparé pour les informations concernant l'algorithme et le salt.

6. Utilisation du front-end framework : Bootstrap

Bootstrap est un framework développé par l'équipe du réseau social Twitter. Proposé en open source (sous licence MIT), ce framework utilisant les langages HTML, CSS et JavaScript fournit aux développeurs des outils pour créer un site facilement. Ce framework est pensé pour développer des sites avec un design responsive, qui s'adapte à tous les types d'écrans, et en priorité pour les smartphones. Il fournit des outils avec des styles déjà en place pour des typographies, des boutons, des interfaces de navigation et bien d'autres encore. On appelle ce type de framework un "Front-End Framework".

Nous avons fait le choix de Bootstrap car il nous permet, par exemple, de rendre facilement un site responsive design (adapté à tous les écrans : ordinateur, mobile, tablette). L'installation sur le projet est viable grâce à des liens entre Bootstrap et notre application.

```
<!-- CSS only -->
<link href="https://cdn.jsdelivr.net/npm/bootstrap@5.1.0/dist/css/
bootstrap.min.css" rel="stylesheet" integrity="sha384-
KyZXEAg3QhQLMpG8r+8fhAXLRk2vvoC2f3B09zVXn8CA5QIVfZ0J3BCsw2P0p/We"
crossorigin="anonymous">
<!-- JavaScript Bundle with Popper -->
<script src="https://cdn.jsdelivr.net/npm/bootstrap@5.1.0/dist/js/
bootstrap.bundle.min.js" integrity="sha384-U1DAWAznBHeqEILVSCgzq+c9gqGAJn5c/
t99JyeKa9xxaYpSvHU5awsuZVVFIhvj" crossorigin="anonymous"></script>
```

Nous utilisons de nombreux outils graphiques développés par Bootstrap, nous allons en décrire un exemple ci dessous;

*Les messages d'alerte

Success! This alert box indicates a successful or positive action.

Info! This alert box indicates a neutral informative change or action.

Warning! This alert box indicates a warning that might need attention.

Danger! This alert box indicates a dangerous or potentially negative action.

```
<div class="alert alert-success">
  <strong>Success!</strong> Indicates a successful or positive action.
</div>

<div class="alert alert-info">
  <strong>Info!</strong> Indicates a neutral informative change or action.
</div>

<div class="alert alert-warning">
  <strong>Warning!</strong> Indicates a warning that might need attention.
</div>

<div class="alert alert-danger">
  <strong>Danger!</strong> Indicates a dangerous or potentially negative action.
</div>
```


7. Les scripts Javascript

7.1. AutoComplete : client

L'autocomplete, aussi appelé autocomplétion, est une fonctionnalité répandue en recherche d'information qui consiste à proposer à l'utilisateur des options de saisie au fur et à mesure de sa frappe.

Nous avons eu le choix d'utiliser l'autocomplétion développé en JavaScript afin de lister et rechercher les clients afin de faire une vente.

Pour utiliser ceci nous devons transmettre un tableau PHP à notre script Javascript. Pour ce faire, il faut déjà récupérer à l'aide d'une requête l'ensemble des clients. Que nous stockons ensuite dans une variable :

```
$lesResultatsClients = AffichageClients();
```

Nous créons ensuite un tableau \$clients, que nous allons transmettre à notre script javascript. Or, il est nécessaire de stocker les nom et prénom en les concaténant ensemble.

```
$clients = array();  
foreach($lesResultatsClients as $leResultatClient){  
    array_push($clients, $leResultatClient->nom . " " . $leResultatClient->prenom);  
}
```

Pour transmettre ce tableau php au script javascript, nous utilisons cette ligne :

```
<script> var clients = <?= json_encode($clients); ?>; </script>
```

La fonction de saisie autocompletion prend deux arguments, nous exécutons la fonction lorsque quelqu'un écrit dans le champ de texte. Nous fermons ensuite toutes les listes déjà ouvertes de valeurs auto-complétées. Puis nous créons un élément DIV qui contiendra les éléments (valeurs). On ajoute l'élément DIV en tant qu'enfant de l'autocompletion. Nous faisons ensuite une boucle qui parcourt tous les éléments du tableau, on vérifie si l'élément commence par les mêmes lettres que la valeur du champ de texte :

```
if (arr[i].substr(0, val.length).toUpperCase() ==val.toUpperCase())
```

Pour chaque élément trouvé nous créons un div, et nous mettons les lettres correspondantes en gras.

Pour finir nous exécutons une fonction lorsque quelqu'un clique sur la valeur de l'élément (élément DIV).

7.2. AutoComplete : composition des productions

```
<input name='tags' class='form-group' placeholder='Composition' value='' of  
Objects)></input>
```

Petit sac de transport papier

Grand sac de transport papier

Charcuterie à cuire à base de porc

Charcuterie cuite : pâté, terrine et galantine de porc

Charcuterie cuite : pâté, terrine et galantine de volaille et lapin

Charcuterie cuite : terrine de poisson et/ou fruits de mer

Charcuterie cuite : terrine festive

Charcuterie cuite : produits tripiers

Charcuterie cuite : saucisserie

Charcuterie cuite : saucisserie sèche

a

Composition

Charcuterie cuite : terrine festive x Entrées froides : poisson fumé x

Composition

Composition

patte x Composition

La liste de suggestions est un tableau de chaînes ou d'objets de la liste blanche qui a été définie dans les paramètres Object lors de la création de l'instance, et peut être défini directement sur l'instance : `tagifyInstance.whitelist = ["tag1", "tag2", ...]`.

La liste déroulante des suggestions sera ajoutée à l'<body>élément du document et sera rendue par défaut dans une position en dessous de l'élément. L'utilisation des flèches haut/bas du clavier met en surbrillance une option de la liste et appuyez sur la touche Entrée pour la sélectionner.

Il est possible d'ajouter un `maxItems` qui limite le nombre d'éléments.

```
var input = document.querySelector('input'),  
    tagify = new Tagify(input, {  
      whitelist: ['aaa', 'aaab', 'aaabb', 'aaabc', 'aaabd', 'aaabe', 'aaac', 'aaacc'],  
      dropdown: {  
        classname: "color-blue",  
        enabled: 0, // afficher la liste déroulante immédiatement  
        maxItems: 5,  
        position: "text", // la liste déroulante près du texte tapé  
        closeOnSelect: false, // laisse la liste déroulante ouverte après avoir sélectionné une suggestion  
        highlightFirst: true  
      }  
    });
```

Ceci rendra donc :

```
<div class="tagify__dropdown tagify__dropdown--text" style="left:993.5px; top:106.375px;  
width:616px;">  
  <div class="tagify__dropdown_wrapper">  
    <div class="tagify__dropdown_item tagify__dropdown_item--active" value="aaab">aaab</div>  
    <div class="tagify__dropdown_item" value="aaabb">aaabb</div>  
    <div class="tagify__dropdown_item" value="aaabc">aaabc</div>  
    <div class="tagify__dropdown_item" value="aaabd">aaabd</div>  
    <div class="tagify__dropdown_item" value="aaabe">aaabe</div>  
  </div>  
</div>
```

8. Outil de génération de PDF

Afin de répondre à ce besoin nous avons eu besoin de dompdf qui est un convertisseur de HTML vers PDF écrit en PHP, qui essaie de respecter au mieux les styles CSS tout en assurant des fonctionnalités utiles pour l'impression (saut de page, numéro de page, couleurs CMJN, format de page, résolution des images, etc). Le principe de cette bibliothèque est de télécharger et lire les feuilles de style externes, les balises, et les attributs des éléments individuels HTML pour ensuite les insérer dans le document PDF résultant. Il supporte également la plupart des attributs de présentation de HTML 4 pour rester compatible avec les documents HTML sans style CSS.

Pour son installation :

Nous avons créé un spacename nommé Dompdf, puis nous installons et utilisons la classe. Nous pouvons y configurer le format, l'orientation du papier.

```
use Dompdf\Dompdf;  
$dompdf = new Dompdf();  
$dompdf->loadHtml('la page html');  
$dompdf->setPaper('A4', 'landscape');
```

Puis nous rendons le format HTML en PDF, et nous générons avec comme sortie le navigateur.

```
$dompdf->render();  
$dompdf->stream();
```

Nous utilisons DOMPDF afin de générer les factures des clients :

1 / 1 | - 100% + | [] ↻



Boutique des ventes
12 rue Louis Armstrong
87065 LIMOGES
Tél : 05 55 35 06 73
Mail : boutique.jean.monnet@gmail.com
N°SIRET : 19 870 999 0000 16

ROUSSELOT Sandra
HHH

Facture N° 432
Date 2021-06-21



A ÉTÉ RÉSERVÉE

Désignation	Portions	Conso. avant	Quantité	P. Unité	Total
test1	1	25/06/2021	0	27€	27€
test2	1	27/06/2021	0	11€	11€

Régler par Carte Bancaire

Total
38€

1 / 1 | - 100% + | [] ↻



Boutique des ventes
12 rue Louis Armstrong
87065 LIMOGES
Tél : 05 55 35 06 73
Mail : boutique.jean.monnet@gmail.com
N°SIRET : 19 870 999 0000 16

RODRIGUES anthony
drytf



Facture N° 437
Date 2021-08-31

Désignation	Portions	Conso. avant	Quantité	P. Unité	Total
Chou	1	01/09/2021	1	11€	11€

Régler par Carte Bancaire

Total
11€

1 / 1 | - 100% + | [] ↻



Boutique des ventes
12 rue Louis Armstrong
87065 LIMOGES
Tél : 05 55 35 06 73
Mail : boutique.jean.monnet@gmail.com
N°SIRET : 19 870 999 0000 16

ROUSSELOT Sandra
HHH

Facture N° 435
Date 2021-06-24

ANNULÉE

Désignation	Portions	Conso. avant	Quantité	P. Unité	Total
alouette	3	24/09/2021	0	7.2€	7.2€

Veuillez régler votre facture dans les meilleurs délais

Total
0€

9. Génération de statistiques

Pour la génération de statistiques nous avons opté pour Chart.js qui est une bibliothèque JavaScript open source gratuite pour la visualisation des données, qui prend en charge 8 types de graphiques: barre, ligne, zone, tarte, bulle, radar, polaire et diffusion.

```
var cte = document.getElementById('stat').getContext('2d');
var stat = new Chart(cte, {
  type: 'bar',
  data: {
    labels: <?= $nom ?>,
    datasets: [{
      label: 'Montant en euros ',
      data: <?= $result ?>,
      backgroundColor: [
        'rgba(255, 0, 188, 0.3)',...
      ],
      borderColor: [
        'rgba(255, 0, 188, 1)',...
      ],
      borderWidth: 1
    }]
  },
  options: {
    plugins: {
      legend: {
        display: false,
      }
    }
  }
});
```

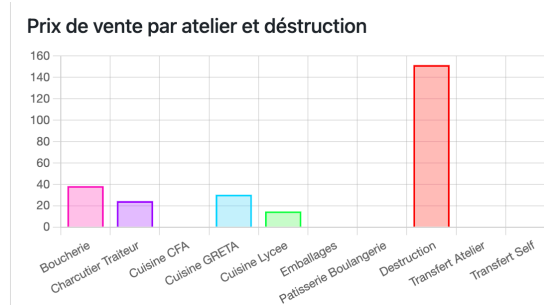
Nous utilisons deux variables (\$nom, \$result) sous forme de tableau afin de les utiliser pour notre graphique.

Le calcul de ces variables a été fait en amont puis encodé en JSON permettant une utilisation par notre script JavaScript.

```
$calculAtelier = $calculAtelier + (($RecuperationConditionnementResultatetat_transfert->conditionnement -
$CalculNbVentesResultatetat_transfert->quantite_vendue) * $RecuperationPrixUnitaireResultatetat_transfert->prix);
}

$jsonprix[] = $calculAtelier;

$result = json_encode($jsonprix);
```



La documentation technique sera régulièrement complétée après les diverses mises à jour de notre application.