

Projet HDMeet



Zakaria BELKACEMI

Formation Développeur Web

02/02/2024

Projet HDMeet (React, Node)



[COMMENCER](#)

[ADMINISTRATION](#)

Sommaire :

1- Compétences couvertes.....	3
2- Résumé du projet.....	3
3- Maquettage de l'application.....	4
4- Cahier des charges.....	4
5-Spécifications technique.....	5
5.1 Schéma de la base de données.....	7
5.2 Les clés des tables.....	9
5.3 Architecture MVC.....	10
6- Réalisation du projet PHP/MVC.....	11
6.1 Développement d'un CRUD complexe.....	15
6.2 Utilisation du programme.....	21
7- Réalisation du projet Symfony.....	24
7.1 Développement d'un CRUD complexe.....	30
7.2 Utilisation du programme.....	35
8- Hébergement du site.....	38
9- Sécurisation des données.....	39
10- Visuel de l'interface sur écrans.....	40
11- Les Tests.....	43
12- La veille technique.....	44
13- Recherche sur des sites anglophone.....	45
14- Bilan.....	46

1. Liste des compétences couvertes par le projet

Le SDBM est un projet back-end qui consiste la gestion de vente de bières. Ce projet permet de mettre en évidence plusieurs compétences techniques dans le titre professionnel développeur web - web mobile. Les compétences couvertes sont :

- Maquetter une application
- Réaliser une interface utilisateur web statique et adaptable
- Développer une interface utilisateur web dynamique
- Créer une base de données
- Développer les composants d'accès aux données
- Développer la partie back-end d'une application web ou web mobile

2. Résumé de Projet

Le projet consiste en la création d'un site web dynamiques qui va s'exécuter côté serveur pour une cave à bière. Ce site vise à offrir aux amateurs de bière une plateforme interactive pour découvrir, acheter et partager leur passion pour cette boisson.

Catalogue de Bières : Le site propose une liste exhaustive de bières avec des informations détaillées sur chaque produit, y compris le nom, le pays, la couleur et le continent.

Gestion de la Base de Données : Nous avons utilisé XAMPP pour gérer notre base de données et MySQL pour les créer, stockant des informations sur les bières, les utilisateurs, les commandes.

CRUD Fonctionnalités : Les administrateurs du site peuvent ajouter de nouvelles bières, mettre à jour les détails des produits existants, supprimer des éléments du catalogue.

Avantages :

Le site offre une expérience conviviale et informative aux amateurs de bière, en mettant en avant une grande variété de produits.

Les fonctionnalités CRUD facilitent la gestion du catalogue et des utilisateurs.

Les fonctionnalités ont été ajoutés pour protéger la base de données contre des failles XSS et de l'injection SQL.

Ce projet a été réalisé en 2 versions techniques différentes, d'une part en PHP, PDO et MVC et d'autre part avec le framework Symfony, pour chacune des version 1 site a été mis en ligne.

En PHP : <https://zakaribel.com/MBB>

En Symfony : https://zakaribel.com/MBB_S/public

La rédaction de ce dossier détaille en majorité la version PHP, PDO et MVC et va mettre en exergue les différences de développement avec Symfony, les possibilités complémentaires liées au Framework Symfony ainsi que les différentes méthodologies.

Avantages :

Le site offre une expérience conviviale et informative aux amateurs de bière, en mettant en avant une grande variété de produits.

Les fonctionnalités CRUD facilitent la gestion du catalogue et des utilisateurs.

(*En Symfony*)

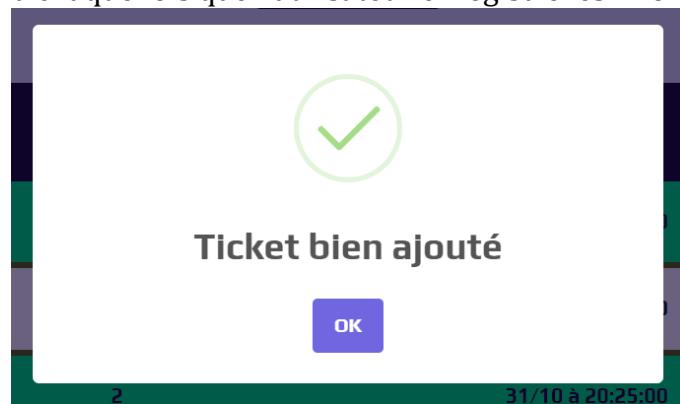
Un menu fixe en haut de page ainsi qu'une liste déroulante permet d'accéder à chaque table et affiche de liens textuels.



Chacune des pages dispose d'un titre, d'un tableau avec les actions réalisable sur la page (Afficher/Ajouter/ Modifier/Supprimer)

A screenshot of a page titled "Articles". At the top right is a "NOUVEL ARTICLE" button. Below it is a small illustration of a smiling yellow beer mug with foam. The main area shows a table with columns: "Code Article", "Nom De l'Article", "Prix d'Achat", "Volume", "Titrage", "Couleur", "Type", "Marque", and "Actions". One row is visible, showing "1", "das Echte Märzen", "2.14€", "33ml", "5.7%", "Blonde", "Bière de Saison", "das Echte Märzen", and two icons for search and edit. Above the table are page navigation controls: "1", "2", and "»".

Un message affichera à chaque fois que l'utilisateur enregistre les information.



3. MAQUETTAGE DE L'APPLICATION

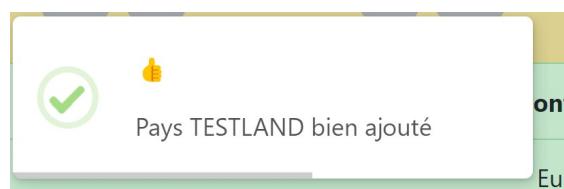
Un menu fixe en haut de page permet d'accéder à chaque table de la base de données.



Chacune des pages dispose d'un titre, d'un tableau avec les actions réalisable sur la page (Ajouter/ Modifier/Supprimer)

Code	Nom	Continent	Actions
1	ALLEMAGNE	Europe	

Un message affichera à chaque opération que l'utilisateur clique sur le bouton Valider.



4. CAHIER DES CHARGES

Les dirigeants de la Société de Distribution des Bières du Monde souhaitent développer un site web dédié à sa cave à bière pour promouvoir ses produits, faciliter les ventes en ligne et renforcer l'engagement de sa communauté de clients. Le site doit être convivial, informatif et sécurisé.

Ils veulent que l'on puisse classer les bières suivants différents critères :

Elles peuvent avoir une couleur (Blanche, Blonde, Brune...)

Elles peuvent avoir un type (Trappiste, Ale, Stout.....)

Elles sont rattachées à une marque.

Cette marque appartient alors à un fabricant.

Cette marque est originaire d'un pays.

Le site web doit atteindre les objectifs suivants :

Permettre aux administrateurs du site de gérer facilement le catalogue de bières en utilisant des fonctionnalités CRUD.

Le site devra inclure les fonctionnalités suivantes :

Développement des pages web dynamiques d'un site web en PHP selon architecture MVC (Model-View-Controller).

Développement des CRUD à partir d'une base de données en utilisant les PDO (PHP Data Objects).

La base de données utilisé sera le SDBM_V2.

Le site doit être adaptatif, c'est-à-dire qu'il doit être compatible avec les appareils mobiles et les ordinateurs de bureau.

Il doit être sécurisé, avec des mécanismes de protection contre les vulnérabilités telles que les injections SQL et le XSS.

Les données sensibles des utilisateurs doivent être stockées de manière sécurisée.

5. SPECIFICATIONS TECHNIQUES

Choix des technologies utilisées :

PARTIE FRONT-END

HTML :

Afin de créer et de représenter le contenu d'une page web et sa structure.

CSS :

Permet de créer des pages web à l'apparence soignée.

JAVASCRIPT :

Permet de créer du contenu mis à jour de façon dynamique.

JQUERY :

C'est une bibliothèque JS qui permet de créer des effets dynamiques sur la page web.

BOOTSTRAP :

C'est un framework CSS pour rendre mon site responsive design

FONTAWESOME :

J'ai utilisé cette librairie pour insérer des icônes pour illustrer rapidement mes pages.

SWEETALERT :

Permet de créer toutes sortes de messages d'alerte qui peuvent être personnalisés pour correspondre à l'apparence de notre propre site Web

PARTIE BACK-END

PHP (Hypertext PreProcessor)

J'ai utilisé ce langage de script pour créer des pages qui vont être générées dynamiquement. Nous allons pouvoir afficher des contenus différents sur une même page en fonction de certaines variables, le fait que l'utilisateur soit connu et connecté ou pas. Il va s'exécuter côté serveur.

La méthode **MVC (Modèle-Vue-Contrôleur)** en PHP est un motif de conception visant à faire le lien entre l'interface utilisateur et les modèles de données sous-jacents. L'objectif principal de l'architecture MVC est de séparer la logique de présentation, la logique d'application et la gestion des données dans des composants distincts, ce qui rend l'application plus organisée, évolutive et maintenable.

Le motif se compose de trois modules :

Modèle (Model) :

Un modèle représentant la structure logique sous-jacente des données dans une application logicielle, ainsi que la classe supérieure qui y est associée. Ce modèle d'objet ne contient aucune information sur l'interface utilisateur.

Vue (View) :

La vue, autrement dit un ensemble de classes représentant les éléments de l'interface utilisateur (tous ceux que l'utilisateur voit à l'écran et avec lesquels il peut interagir : boutons, boîtes de dialogue, etc.)

Contrôleur (Controller) :

Un contrôleur représentant les classes qui se connectent au modèle et à la vue, et servant à la communication entre les classes dans le modèle et la vue.

En résumé le modèle MVC en PHP permet d'organiser le code de manière modulaire et maintenable.

La classe PDO en PHP est un outil essentiel pour communiquer avec des bases de données de manière plus simple et sécurisée. Elle offre une interface unique pour interagir avec différentes bases de données, comme MySQL, PostgreSQL, etc.

PDO facilite également la protection contre les attaques de sécurité, en évitant les vulnérabilités d'injection SQL.

En outre, elle améliore la maintenance et l'évolutivité du code, car elle permet de changer de base de données plus facilement.

Elle offre également des fonctionnalités avancées comme la gestion des transactions et la réutilisation de requêtes préparées pour des performances optimales.

MYSQL (Structured Query Language)

Le système de gestion de base de données MySQL est un logiciel permettant d'introduire des données, de mettre à jour, de supprimer et d'accéder aux données stockées dans une base - CRUD (Create, Read, Update, Delete). Il utilise le langage SQL pour la manipulation des données des bases de données. Dans mon code, j'écrirai différentes requêtes SQL qui vont me permettre de créer notre base de données et d'enregistrer les données dedans.

MARIADB

C'est un système de gestion de base de données relationnelle open source qui est une alternative compatible avec MYSQL. Il est donc possible de migrer les bases de données MySQL vers MariaDB sans trop de difficultés.

COMPOSER

C'est un logiciel gestionnaire de dépendances libre écrit en PHP. Il permet à ses utilisateurs de déclarer et d'installer les bibliothèques dont le projet principal a besoin.

PUTTY

C'est un émulateur de terminal permettant la connexion à une machine distante par protocole ssh.

SYMFONY

C'est un Framework PHP reconnu dans le monde utilisé par les développeurs pour créer des sites ou des applications Web complexes. C'est un ensemble de composants PHP ainsi qu'un Framework MVC libre écrit en PHP. Il fournit des fonctionnalités modulables et adaptables qui permettent de faciliter et d'accélérer de développement d'un site web. Il intègre des mesures de sécurité préventives pour lutter contre les failles et attaques XSS, CSRF et injection SQL.

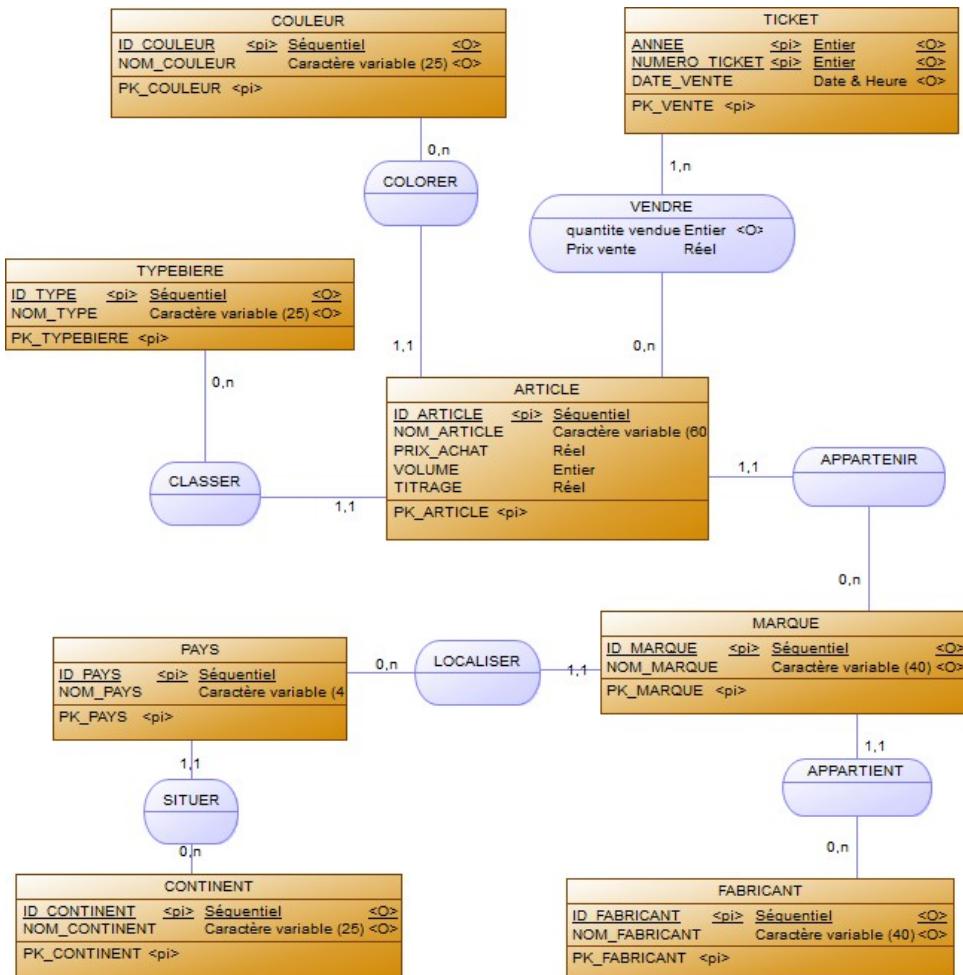
Symfony fournit aussi une interface en ligne de commande pour améliorer la productivité en créant un code de base modifiable à volonté.

5.1 La base de données et son Schéma

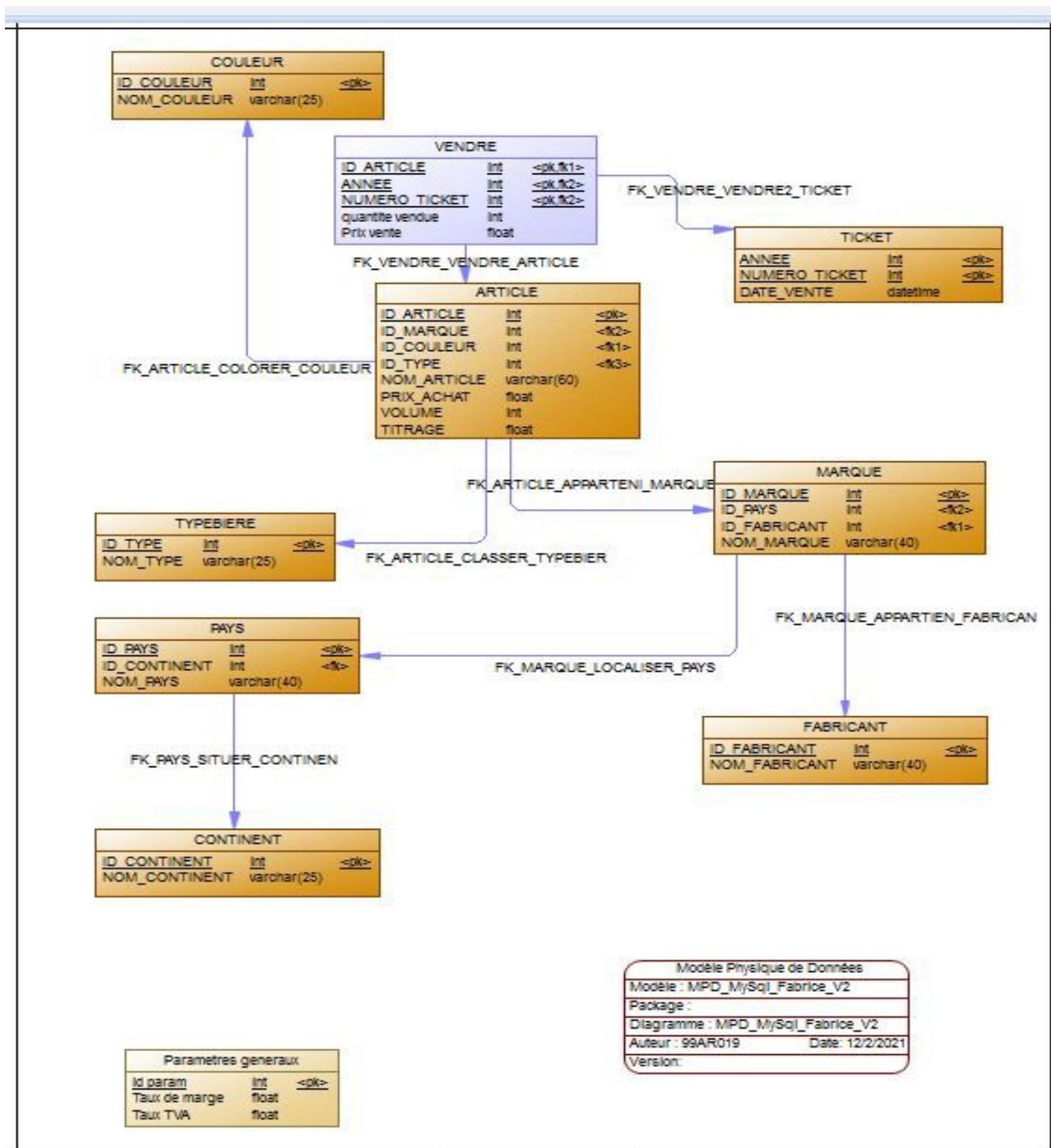
Qu'est-ce qu'une base de données ?

On les appelle "BDD" par commodité : les bases de données ont pour but de stocker, organiser et analyser les données.

Le schéma ci-dessous est le M.C.D (Modèle Conceptuel des données) issu de l'analyse du projet avec la méthode « MERISE » qui est l'acronyme pour « Méthode d'étude et de réalisation informatique pour les systèmes d'entreprise ».



Fais suite au MCD, ci-dessous est le M.P.D (Modèle Physique des Données). Il permet de construire la structure finale de la base de données. Voici le MPD de base de données sdbm_V2 qu'on va utiliser pour ce projet pour comprendre les relations entre les tables de la base de données.



5.2 Les clés des tables

Chaque table est constituée d'une clé primaire ou « pk » (primary key). Par exemple, la clé primaire ou « pk » de la table 'PAYS' est 'ID_PAYS'. La clé primaire ou « pk » de la table 'ARTICLE' est 'ID_ARTICLE'. Et ainsi de suite pour chacune des tables.

Certaines tables ne possèdent que des clé primaire « pk » comme les tables « CONTINENT », « FABRICANT », « COULEUR » et « TYPEBIERE ».

D'autres tables comme « ARTICLE », « MARQUE » possèdent une clé primaire « pk » et plusieurs clé étrangères « fk » (foreign key). Pour la table « PAYS » il a une clé primaire « pk » est une clé étrangère « fk » c'est « ID_CONTINENT ».

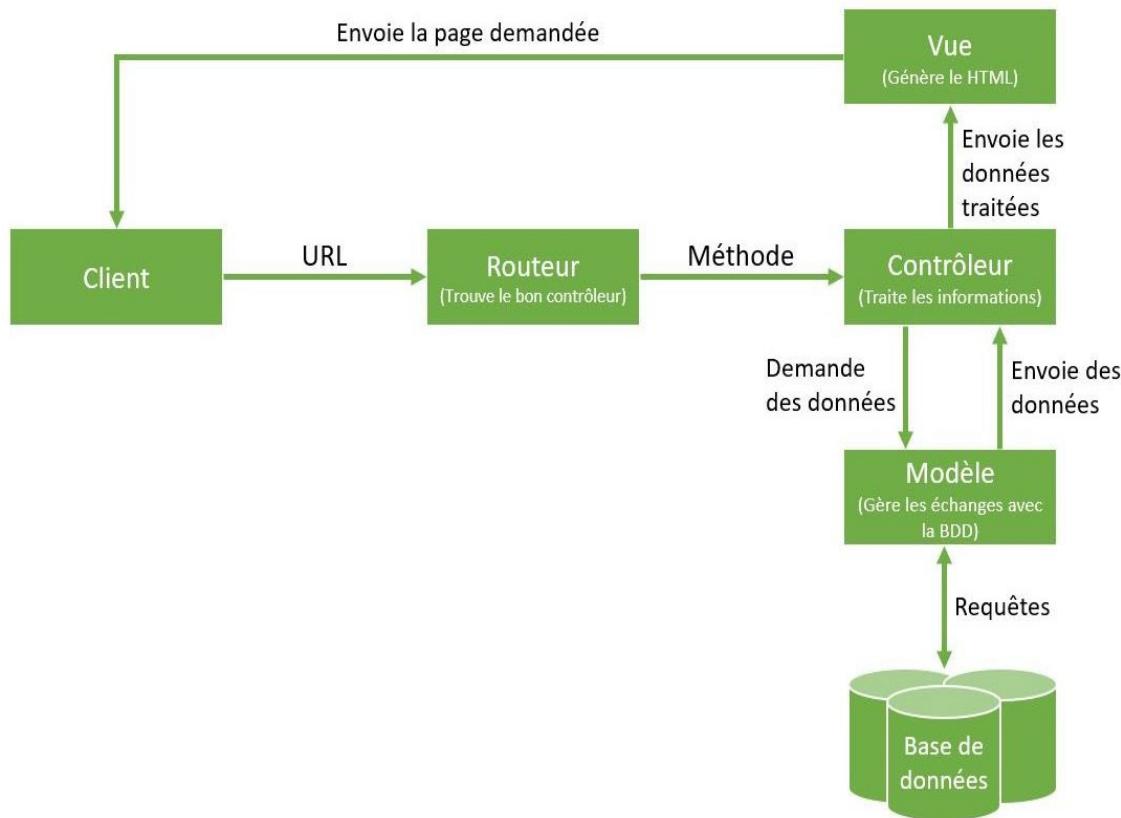
Une clé primaire permet d'assurer l'unicité de chaque ligne de la table. La clé primaire est souvent de type numérique et nommée id ou id_nomTable et sera auto-incrémentée.

Les clés primaires jouent un rôle vital pour maintenir l'ordre et l'unicité des données.

Dans une base de données, nous utilisons des clés étrangères pour créer des liens entre les données dans différentes tables. Ces liens nous permettent de récupérer des informations de manière cohérente et structurée. Les clés étrangères sont le ciment qui unit les tables et permet aux base de données de refléter les relations du monde réel.

5.3 SCHEMA ARCHITECTURE MVC (Model-View-Controller)

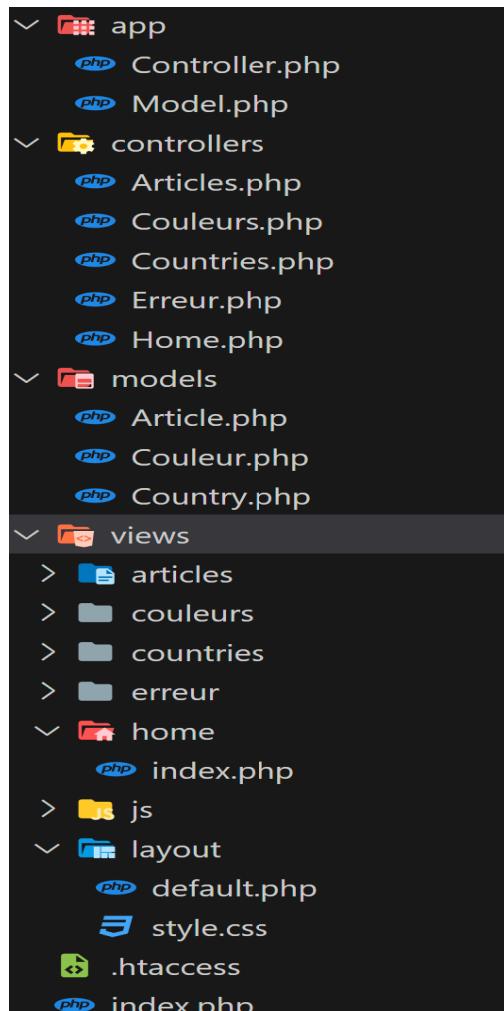
Pour ce projet, j'ai suivi l'architecture MVC. MVC est un principe de mise en place des projets applicatifs. MVC aide le développeur de bien structurer les différents éléments de son projet qui ont des fonctionnalités différentes.



Dans la structure MVC, un seul et unique fichier est le point d'entrée de l'application, quelle que soit la page affichée. Il est systématiquement appelé, et envoie la demande au bon contrôleur.

En résumé lorsqu'un client entre une url, il est dirigé vers le routeur qui décide quel est le contrôleur et la méthode appropriés, ce contrôleur communiquera avec le modèle correspondant et lui demandera d'effectuer une action sur la base de données, le modèle effectuera l'action et enverra les informations au contrôleur. Le contrôleur traitera les informations reçues du modèle et une fois qu'elles auront été traitées, il les enverra à la vue, qui élaborera la page web à envoyer au client.

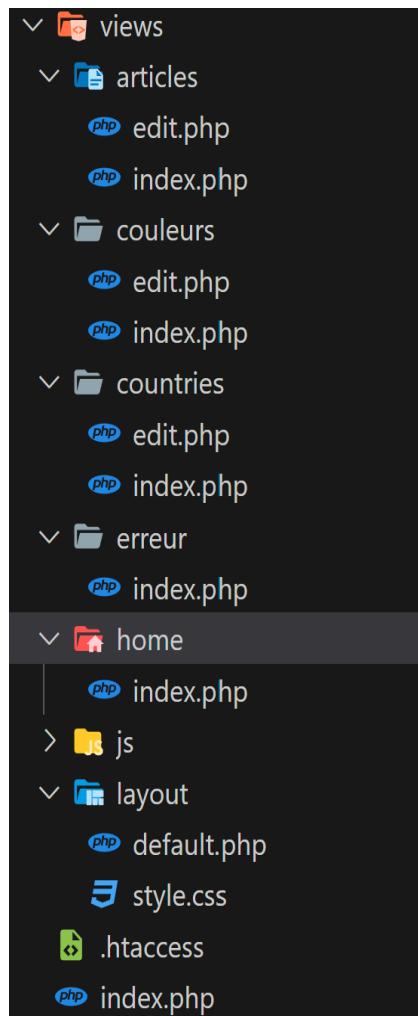
6. REALISATION DU PROJET SDBM_V2



Le dossier « **app** » contient deux fichiers qui s'appellent Controller.php et Model.php. Chaque une des deux fichiers définit une class « Controller » et « Model » respectivement.

Le dossier « **controllers** » contient des fichiers qui définissent les classes qui héritent la class « Controller ». Il y a autant des classes filles (de la class Controller) que les tables dans la base de données, plus deux, « Home », pour afficher la page index quand il n'y a pas de paramètres dans l'url et « Erreur », pour afficher diriger l'utilisateur vers cette page en cas d'erreur. Les controllers gèrent la responsabilité de communiquer avec autres éléments.

Par convention, les controllers sont écrites avec une majuscule au début et la lettre « s » à la fin, sauf « Erreur » et « Home ».



Le dossier « views » contient les dossiers pour rendre visible les différentes pages à l'utilisateur. Le fichier « default.php » dans « layout » gère la visibilité générale de toutes les pages. Les différents fichiers views affiche les données envoyés par le controller après avoir les traités.

Les fichiers « .htaccess » et « index.php » ensemble travaillent comme le « Routeur » qui est en charge pour trouver l'emplacement auquel il doit accéder, ça veut dire trouver le bon controller et envoie les commandes à ce controller. Ils aident à réécrire l'url et récupérer les paramètres.

Le « .htaccess » est un fichier apache qui est utilisé pour la **réécriture de l'url** pour ce projet.

```
htaccess
1 RewriteEngine On
2 RewriteRule ^([a-zA-Z0-9\-\_\\/]*$) index.php?p=$1 [L]
```

Avec « RewriteEngine On » on va préciser à Apache qu'on doit allumer le moteur de réécriture de url.

Ensuite on va créer une règle de réécriture avec « RewriteRule^()\$ », entre les parenthèses on doit ajouter « [] » et tous les caractères qu'on peut avoir:

A-Z : majuscules de la A à la Z

0-9 : chiffres 0 à 9

\-_\\/ : - (tiret), _ (underscore), et / (slash), échappées avec/ (antislash).

Le résultat final : [a-zA-Z0-9\-_\\/]

On ajoute * pour indiquer qu'on va répéter cette structure autant fois que nécessaire.

Dans le « index.php » on récupère les paramètres de l'url avec la méthode GET et instancie le controller correspondant. S'il existe plusieurs parties de la paramètres séparées par des « / », ça pointe vers les methods de la class de cette controller. On vérifie si le controller ou la method existe, sinon on instancie la class « Erreur ». Dans le cas, où il n'y a pas de paramètres on instancie la class « Home ».

Aussi, on a défini quelques constantes qui vont nous servir pendant le projet,dont « ROOT » qui contient le chemin vers la racine publique du projet, est utilisé toute de suite dans le fichier index.php.

```

index.php > ...
1  <?php
2  | function getUrlWithoutFilename() {
3  |   $url = $_SERVER["SCRIPT_NAME"];
4  |   $zones = explode("/", $url);
5  |   $resultat = "";
6  |   for ($i=1; $i < count($zones)-1 ; $i++) {
7  |     $resultat .= "/". $zones[$i];
8  |   }
9  |   return $resultat;
10 |
11 }
12 // On génère une constante contenant le chemin vers la racine publique du
13 define('ROOT', str_replace('index.php', '', $_SERVER['SCRIPT_FILENAME']));
14 // Mettre '' si le site est à la racine du serveur
15 define('PATH', getUrlWithoutFilename());
16
17 // On appelle le modèle et le contrôleur principaux
18 require_once(ROOT.'app/Model.php');
19 require_once(ROOT.'app/Controller.php');
20

```

```

try {
$params = explode('/', @$_GET['p']);

if($params[0] != ""){
    // On sauvegarde le 1er paramètre dans $controller en mettant
    $controller = ucfirst($params[0]);

    // On sauvegarde le 2ème paramètre dans $action si il existe,
    $action = isset($params[1]) ? $params[1] : 'index';

    // On appelle le contrôleur
    @require_once(ROOT.'controllers/'.$controller.'.php');

    $controller = new $controller();

    if(method_exists($controller, $action)){
        unset($params[0]);
        unset($params[1]);
        call_user_func_array([$controller,$action], $params);
    }
} else{
    require_once('controllers/Home.php');
    $home = new Home();
    $home->index();
}
} catch (Exception $e) {

    require_once(ROOT . 'controllers/Erreur.php');

    $controller = new Erreur($e);

    $controller->index();
}

```

6.1 DEVELOPPEMENT D'UN CRUD COMPLEXE

La création du CRUD complexe de la table « pays » de la base de données « sdbm_v2 ».

1. AFFICHAGE

Dans le menu de navigation, on suit ce lien :

```
<header>
  <div class="d-flex justify-content-center">
    <a href="#">  

      <button id="btnHome" class="btn">Accueil</button>  

    </a>

    <a class="" href="#">  

      <button id="btnArticles" class="btn">Articles</button>  

    </a>

    <a class="" href="#">  

      <button id="btnCountries" class="btn">Pays</button>  

    </a>

    <a class="" href="#">  

      <button id="btnCouleurs" class="btn">Couleurs</button>  

    </a>
  </div>
</header>
```

‘PATH’ est le chemin d'accès depuis le serveur, jusqu'au dossier racine du projet

```
index.php > ...
1  <?php
2  function getUrlWithoutFilename() {
3    $url = $_SERVER["SCRIPT_NAME"];
4    $zones = explode("/", $url);
5    $resultat = "";
6    for ($i=1; $i < count($zones)-1 ; $i++) {
7      $resultat .= "/". $zones[$i];
8    }
9    return $resultat;
10 }
11
12 // On génère une constante contenant le chemin vers la racine publique du
13 define('ROOT', str_replace('index.php','',$_SERVER['SCRIPT_FILENAME']));
14 // Mettre '' si le site est à la racine du serveur
15 define('PATH', getUrlWithoutFilename());
16
17 // On appelle le modèle et le contrôleur principaux
18 require_once(ROOT.'app/Model.php');
19 require_once(ROOT.'app/Controller.php');
```

Ceci permet d'éviter les erreurs en cas de renommage ou déplacement de dossiers pour le projet.

La class « Countries » étend une classe parente « Controller ». Elle à une propriété protégée de type « Country ». Ce code déclare une méthode « index() », une action qui sera exécutée lorsque la page d'index de la classe « Countries » est appelée. Le méthode ne retourne rien('void'). Et puis on appelle la méthode « loadModel() » pour charger le modèle « Country ». « allCountries() » récupérer la liste de tous les pays tandis que « allContinents() » récupère la liste de tous les continents. La méthode « render() » pour afficher une vue appelée « index » et lui transmet les données stockées dans les variables « \$allCountries,\$btnId et \$allContinents.

```

public function index($page = null): void
{
    if (isset($page)) {
        $this->loadModel("Country");
        $rows_per_page = $this->Country->getRowsPerPage();
        $this->Pagination($page, $rows_per_page);
    }
    $this->loadModel("Country");
    $allCountries = $this->Country->allCountries();
    $nbrCountries = $allCountries[0]['nbr_countries'];
    $allContinents = $this->Country->allContinents();
    $rows_per_page = $this->Country->getRowsPerPage();
    $pages = ceil($nbrCountries / $rows_per_page);
    $btnId = "btnCountries";
    $this->render('index', compact(
        'allCountries',
        'btnId',
        'allContinents',
        'page',
        'pages'
    ));
}

```

La méthode ci-dessous récupère toutes les informations sur les pays à partir de la base de données. Elle construit une requête SQL pour sélectionner des informations sur les pays en joignant la table « country » avec la table »continent » en utilisant l'ID du continent comme clé étrangère. Les 2 méthodes sont utilisées pour obtenir des données sur les pays et les continents à partir de la base de données. Les résultats de la requête sont renvoyés sous forme de tableau associatif en utilisant « fetchAll().

```

public function allCountries()
{
    $sql = "SELECT country.*, NOM_CONTINENT
    FROM " . $this->table . " country
    INNER JOIN continent NOM_CONTINENT ON country.ID_CONTINENT = NOM_CONTINENT.ID_CONTINENT
    ORDER BY country.ID_PAYS ASC
    LIMIT 20";
    $query = $this->_connexion->prepare($sql);
    $query->execute();
    return $query->fetchAll();
}

public function allContinents()
{
    $sql = "SELECT * FROM continent";
    $query = $this->_connexion->prepare($sql);
    $query->execute();
    return $query->fetchAll();
}

```

La fonction « render » permet de renvoyer sur la page index du dossier views.
 Le rendu de la page index est ceci :

The screenshot shows a web application interface. At the top, there is a navigation bar with four items: Accueil, Articles, Pays, and Couleurs. Below the navigation bar, the title "Liste des pays" is centered. Underneath the title is a blue button labeled "Ajouter". Below the button is a pagination control with arrows and the text "Page 1 sur 4". The main content is a table with the following data:

Code	Nom	Continent	Actions
1	ALLEMAGNE	Europe	
2	BELGIQUE	Europe	
3	ÉTATS-UNIS	Amérique	
4	PHILIPPINES	Europe	
5	BRÉSIL	Amérique	
6	ARGENTINE	Amérique	
7	DANEMARK	Europe	

2. AJOUT

Ce code traite la création d'un nouveau pays à partir d'une soumission de formulaire. Elle vérifie d'abord les données soumises, puis insère le nouveau pays dans la base de données à l'aide du modèle « Country ». Après l'insertion réussie, elle redirige l'utilisateur vers une autre page avec un message de succès, sinon elle le redirige vers une autre page en cas de données non valides.

```
public function newCountry(): void
{
    $newCountry = array();
    if (!empty($_POST['country'])) {
        $this->loadModel("Country");
        $newCountry['nom'] = htmlentities($_POST['country']);
        $newCountry['id'] = htmlentities($_POST['continent']);
        $this->Country->insert($newCountry);
        $this->redirectWithMessage("Pays " . $_POST['country'] . " bien ajouté", "success", '👍', true, 'countries');
    } else {
        header("Location: " . PATH . "/countries");
    }
}
```

On a ajouté les fonctions htmlentities dans le but de prémunir de failles XSS (Cross-site Scripting) en désactivant les balises HTML (&<et> étant, notamment, réécrits en & ;< ;/>).

La table pays possède un identifiant, un nom et 1 clé étrangère pour le continent. Il est important de charger le combobox permettant de choisir parmi le continent.

Liste des pays

Ajouter

Entrez un nouveau pays :

Europe ▾

- Europe
- Amérique
- Océanie
- Asie
- Afrique

Valider

« « < Page 1 sur 4 > » »

Nom	Continent

Sur cette page d'ajout, il suffit de remplir le formulaire et de le soumettre. La méthode action du formulaire fonctionne sur le même système que l'affichage de cette page.

3. La Modification

La modification fonctionne sur le même principe que l'ajout, une méthode updateCountry sur « controllers » charge un formulaire de modification et une méthode updateCountry sur « models » lance la requête SQL “updateCountry”.

Ce code traite la mise à jour d'un pays existant à partir d'un formulaire soumis. Elle vérifie d'abord les données soumises, puis remplit un tableau avec ces données. Ensuite, elle charge le modèle **Country** pour effectuer la mise à jour du pays en utilisant les données fournies. Après la mise à jour réussie, elle redirige l'utilisateur vers une autre page avec un message informatif.

```
public function updateCountry(): void
{
    $updatedCountry = array();
    if (!empty($_POST['updatedCountry'])) {
        $updatedCountry['idContinent'] = $_POST['updatedContinent'];
        $updatedCountry['updatedCountry'] = $_POST['updatedCountry'];
        $updatedCountry['idCountry'] = $_POST['id'];

        $this->loadModel("Country");
        $this->Country->update($updatedCountry);
        $this->redirectWithMessage('Pays bien modifié', 'info', '&#129299;', true, 'countries');
    } else {
        header("Location: " . PATH . "/countries");
    }
}
```

Cette méthode « updateCountry » permet de mettre à jour les informations d'un pays dans la base de données en utilisant les données fournies dans le tableau « \$updatedCountry ». Elle construit une requête SQL d'Update, prépare cette requête, puis l'exécute avec les valeurs appropriées.

```
////////////////////////////////////////////////////////////////
public function updateCountry(array $updatedCountry)
{
    $sql = "UPDATE " . $this->table . " set NOM_PAYS=? , ID_CONTINENT=? WHERE ID_PAYS=?";
    $query = $this->connexion->prepare($sql);
    $query->execute([
        $updatedCountry['updatedCountry'],
        intval($updatedCountry['idContinent']),
        intval($updatedCountry['idCountry'])
    ]);
}
```

Le formulaire doit être remplir comme ceci :

The screenshot shows a web page with a light beige background. At the top, there is a navigation bar with four buttons: 'Accueil' (Home), 'Articles', 'Pays' (highlighted in grey), and 'Couleurs'. Below the navigation bar, the main content area has a heading 'Saisissez vos modifications' (Enter your modifications). There is a text input field labeled 'Entrez un nouveau pays :'. To the right of the input field is a dropdown menu with the following options: Europe, Amérique, Océanie, Asie, and Afrique. The 'Europe' option is currently selected. Next to the dropdown is a 'Valider' (Validate) button.

4. LA SUPPRESSION

Cette méthode permet de gérer la suppression d'un pays existant en fonction de l'identifiant fourni en paramètre. Elle charge le modèle « Country », utilise la méthode « delete() » pour effectuer la suppression dans la base de données, puis redirige l'utilisateur vers une autre page avec un message pour indiquer la suppression réussie.

```
public function deleteCountry(int $id): void
{
    $this->loadModel("Country");
    $this->Country->delete(htmlentities($id));
    $this->redirectWithMessage(
        'Pays numéro ' . $id . ' bien supprimé',
        'warning',
        'Aurevoir petit pays... &#128577;',
        true,
        'countries');
}
```

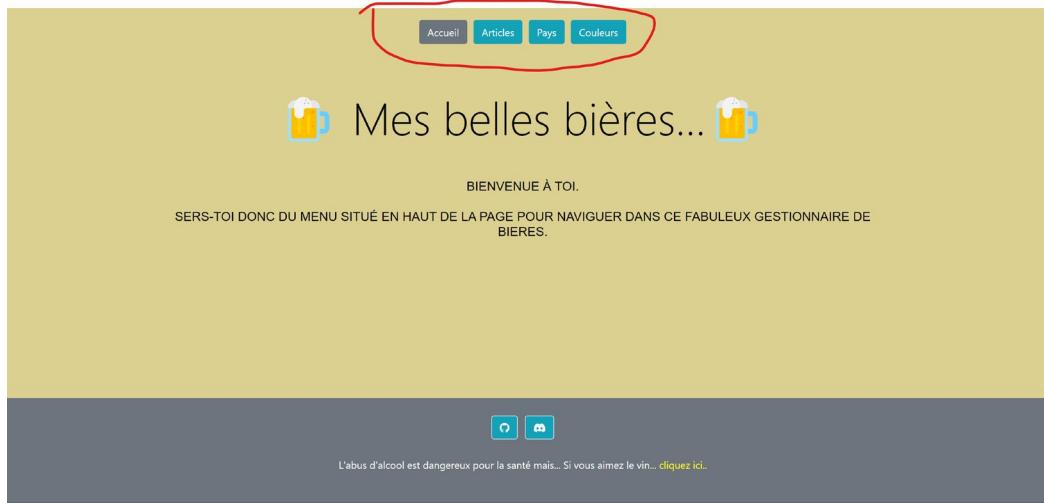
The screenshot shows a table of countries with columns for Nom, Continent, and Actions. A confirmation dialog box is overlaid on the table, asking "Etes vous sur de vouloir supprimer ce pays ?" with OK and Annuler buttons. The table data is as follows:

Nom	Continent	Actions
SUÈDE	Europe	
LAOS	Asie	
TCHÉQUE, RÉPUBLIQUE	Europe	
PORUGAL	Europe	
AUTRICHE	Europe	
PHILIPPINES	Asie	
TESTLAND	Europe	
TESTLAND	Europe	

At the bottom, there are navigation icons: <<, <, >, >>, and a page indicator "Page 4 sur 4".

7. UTILISATION DE PROGRAMME

Image ci - dessus : Page d'accueil avec les boutons de navigation.



Une clique sur un bouton s'affiche une liste correspondante aux tables avec la possibilité de l'afficher, ajouter, modifier et supprimer.

Code	Nom Article	Prix	Volume	Titrage	Marque	Type	Couleur	Actions
1	das Echte Märzen	2,14 €	33	5,7	das Echte Märzen	Blonde	Bière de Saison	
2	das Helle	2,17 €	33	5	das Helle			
3	das Schwarze	1,9 €	33	4,9	das Schwarze	Brune		
4	Dinkel Acker Märzen	2,59 €	33	5,6	Dinkel Accker	Blonde	Bière de Saison	
5	Dinkel Acker Privat	2,61 €	33	5,1	Dinkel Accker	Blonde		
6	Franziskaner Hefe-Weissbier Dunkel	2,32 €	33	5	Franziskaner	Brune		
7	Franziskaner Hefe-Weissbier Hell	1,71 €	33	5	Franziskaner	Blanche		

FONCTIONNEMENT « AJOUTER »

La clique sur le bouton «Ajouter» fera apparaître une formulaire d'ajout.

The screenshot shows a web application interface titled "Liste des couleurs". At the top, there are navigation links: Accueil, Articles, Pays, and Couleurs. Below the title is a blue "Ajouter" button. A search bar contains the placeholder "Entrez une nouvelle couleur:" followed by a "Valider" button. The main area displays a table with four rows of color data:

Code	Nom	Actions
1	Blonde	
2	Brune	
3	Blanche	
4	Ambrée	

Un message de confirmation s'affiche chaque fois que l'utilisateur valide l'ajout d'un nouveau type de bière.

The screenshot shows the same "Liste des couleurs" page after adding a new color. A modal dialog box is centered on the screen, displaying a green checkmark icon, a small orange flame icon, and the text "Couleur Noire bien ajoutée". The main table now includes a fifth row for "Noire". Each row has an "Actions" column with edit and delete icons.

FONCTIONNEMENT « MODIFIER »

Il faut cliquer sur le bouton pour modifier : un autre page avec un formulaire pour modifier une type de bière affiche, comme ci-dessous :

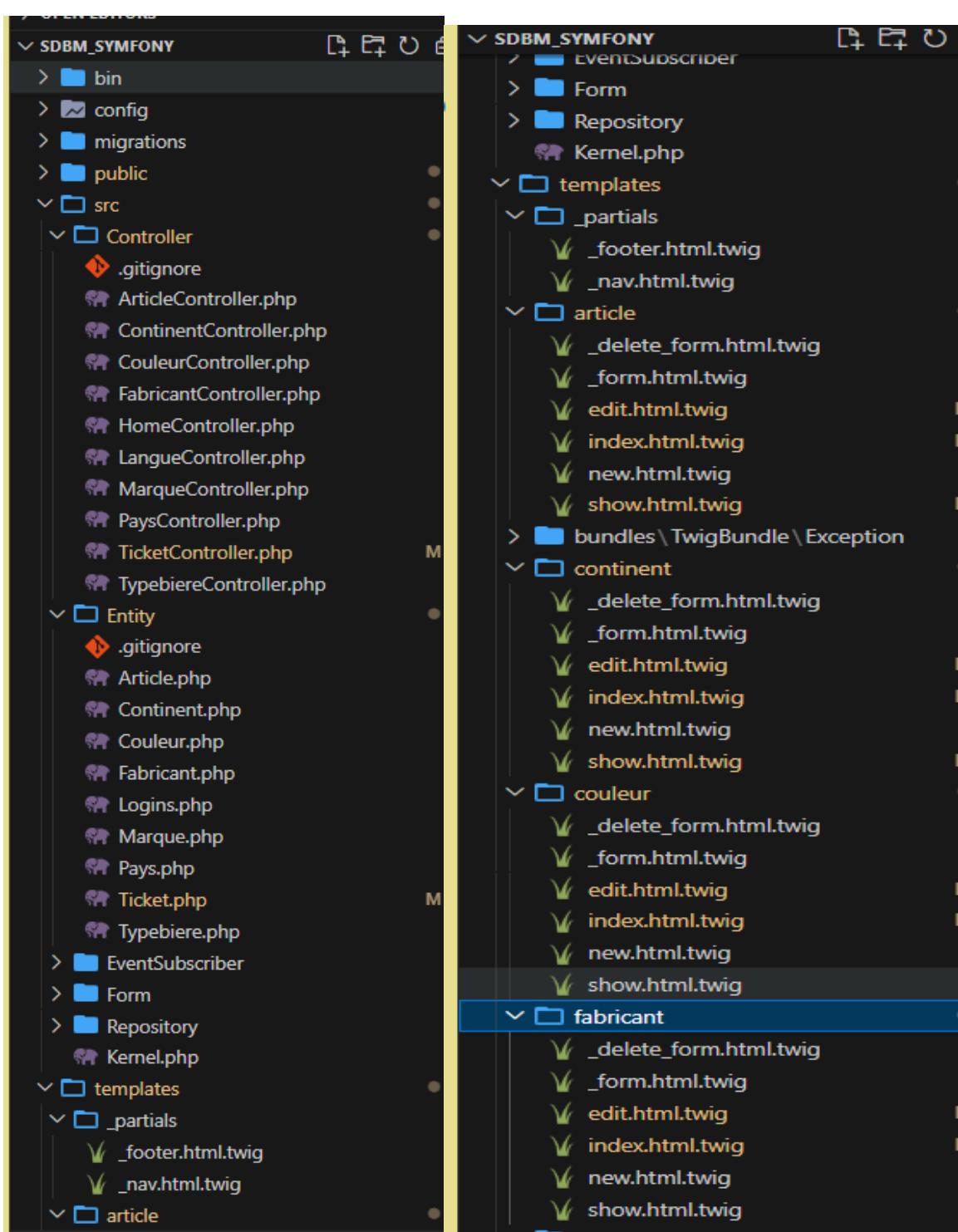
The screenshot shows the "Liste des couleurs" page again. The "Actions" column for the last row ("Noire") has two icons: a blue edit icon and a red delete icon. A red circle highlights the edit icon for the fourth row ("Ambrée").

**FONCTIONNEMENT « SUPPRIMER »**

Pour le cas de suppression , la clique sur le bouton affiche une alerte comme l'image ci-dessous demandant la confirmation de suppression. Si l'utilisateur clique sur « OK », la ligne va être supprimée.



7. REALISATION DU PROJET SDBM_V2 (SYMFONY)



Pour réaliser ce projet, nous devons installer la dernière version de Symfony 6.3 et créer un nouveau projet.

Création d'un projet avec Symfony CLI avec la commande :

```
symfony new SDBM_SYMFONY --version="6.3.*" --webapp
```

Serveur de développement :

symfony server : start -d

→ Ecoute sur <http://localhost>: 8000

LES ENTITES

On doit générer les entités à partir des table d'une BDD avec la commande :

symfony console doctrine : mapping : import "App\Entity" annotation --path=src/Entity

On doit générer aussi les getter/setter sur les entity avec la commande:

composer require doctrine/annotations

Pour les CRUD, On génère automatiquement sur une Entite avec la commande :

php bin/console make :crud

Important ! Vidage du cache pour une bonne prise en compte de nouvelles routes des CRUD avec la commande :

symfony console cache : clear

La partie serveur est coupée en trois morceaux :

Le **modèle** qui implémente des services (ex. gestion de la base de données) ;

Le **contrôleur** qui prend les décisions pour générer les pages demandées (la partie « algorithmique ») ;

La **vue**(templates) qui organise la présentation (affichage).

MVC en pratique

Contrôleurs en PHP (orienté objets) :

-le kernel Symfony fait le chef d'orchestre (*front controller*)

-selon l'url reçue, invoque un contrôleur(*controller*)pour générer la page demandée
-gestion des url par annotations

- Un contrôleur pour chaque « section » du site (article, continent, couleur, fabricant, pays , home, langue, marque, ticket et type de bière)

- un contrôleur (classe PHP) : plusieurs actions, une action pour chaque page à générer

-Une action d'un contrôleur (fonction) : implémente la logique de génération d'une page
- récupération des données / de l'input, traitement et envoi aux vues.

Vues en Twig :

Une vue par « type de page»

- langage de templates simple et puissant
- décrit la présentation de la page en fonction de plusieurs paramètres passés par le contrôleur
- séparation logique de « calcul » de la page/affichage mais très flexible: langage riche, possibilité d'inclure de la logique dans la présentation (p.ex. embedded controllers)
- héritage

Modèle :

Un ensemble de services (p.ex. gestion de la base de données, des formulaires, de la sécurité(authentification), des messages et mails, etc.)

- la plupart des services courants fournis par Symfony

- service de gestion de la bd(mysql) : Doctrine
 - correspondance: données dans la bd↔objets PHP
- possibilité de créer ses propres services (accessibles par tous les contrôleurs)

ORGANISATION DES FICHIERS

Deux répertoires pour développer : src (code PHP) et app (le reste : configuration, templates, etc.)

- src/AppBundle/Controller : les contrôleurs
- src/AppBundle/Entity : les classes pour la base de données
- app/Ressources /views : les vues(templates)
- app/config : les fichiers de configuration

Autres répertoires :

public : contient les fichiers publiquement accessibles (images, CSS, etc .)

bin : contient les exécutables(notamment console)

var : cache, logs, etc

tests : pour les tests automatiques

vendor : les modules additionnels

translations : messages.en.xlf

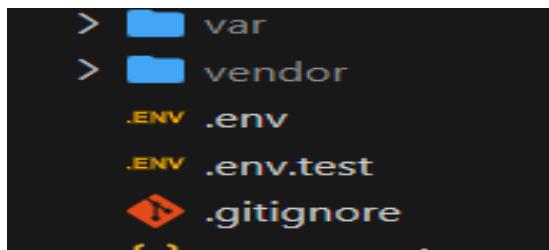
Base.html.twig

Le template de base, avec les blocs à remplir :

```
templates > ✓ base.html.twig
  1  !DOCTYPE html
  2  <html>| The title filter converts a string to title case.
  3  |   <head>| {{ 'foo bar'|title }}| 
  4  |   <title>
  5  |       (% block title %)Accueil
  6  |       (% endblock %)- SDBM</title>
  7  |
  8  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  9  <link rel="icon" href="data:image/svg+xml,<svg xmlns='http://www.w3.org/2000/svg' viewBox='0 0 128 128'><text y='221.2em' font-size='2296%22'>○</text></svg>">
10 <link rel="stylesheet" href="{{ asset('css/style.css') }}>
11 <link rel="preconnect" href="https://fonts.googleapis.com">
12 <link rel="preconnect" href="https://fonts.gstatic.com" crossorigin>
13 <link href="https://fonts.googleapis.com/css2?family=LibreBaskerville:wght@700&display=swap" rel="stylesheet">
14 <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/6.4.2/css/all.min.css" integrity="sha512-z3glpd7yknf1YoNbCzqRKc4qyor8gaKU1qm+CSxbuBusANIQpRohGReCFKxLhei659CQXFebbkuLg0DA==" crossorigin="anonymous" referrerpolicy="no-referrer" />
15 <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.2/dist/css/bootstrap.min.css" rel="stylesheet" integrity="sha384-T3c6CoI6uLrA9TneNEoa7RxnatzjcDSCmG1MxSR1GAsXEV/DwykczMPK8M2HN" crossorigin="anonymous">
16
17      (% block stylesheets %){% endblock %}
18
19      (% block javascripts %){% endblock %}
20
21  </head>
22  <body class="d-flex flex-column min-vh-100">
23      (% include "_partials/_nav.html.twig" %)
24  <div class="flex-grow-1">
25      (% for label, messages in app.flashes %)
26          (% for message in messages %)
27              <div class="alert alert-{{ label }} alert-dismissible">
28                  <button type="button" class="btn-close" data-bs-dismiss="alert"></button>
29                  <strong>{{ label|trans }}</strong> {{ message|trans }}
30              </div>
31          (% endfor %)
32      (% endfor %)
33
34      (% block body %){% endblock %}
35
36  </div>
37  (% block footer %)
38      (% include "_partials/_footer.html.twig" %)
39      (% endblock %)
```

Template appelé avec l'objet articles en argument :

```
templates > article > index.html.twig
1   {% extends 'base.html.twig' %}
2
3   {% block title %}SDBM Articles
4   {% endblock %}
5
6   {% block body %}
7   <div class = bg>
8     <h1>% trans % Liste des Articles%</h1>    <br>
9     <a class="btn btn-outline-success" role="button" href="{{ path('app_article_new') }}">% trans % Ajouter un nouvel Article(% endtrans %)</a>
10    <div class="table-wrapper">
11      <table class="fl-table">
12        <thead>
13          <tr>
14            <th>% trans % ID(% endtrans %)</th>
15            <th>% trans % Article(% endtrans %)</th>
16            <th>% trans % Prix d'Achat(% endtrans %)</th>
17            <th>% trans % Volume % endtrans %</th>
18            <th>% trans % Titrage(% endtrans %)</th>
19            <th>% trans % Couleur(% endtrans %)</th>
20            <th>% trans % Marque % endtrans %</th>
21            <th>% trans % Type % endtrans %</th>
22            <th>% trans % Actions(% endtrans %)</th>
23        </tr>
24      </thead>
25      <tbody>
26        {% for article in articles %}
27          <tr>
28            <td>{{ article.idArticle }}</td>
29            <td>{{ article.nomArticle }}</td>
30            <td>{{ article.prixAchat }}</td>
31            <td>{{ article.volume }}</td>
32            <td>{{ article.titrage }}</td>
33            <td>{{ article.getNomCouleur }}</td>
34            <td>{{ article.getNomMarque }}</td>
35            <td>{{ article.getNomType }}</td>
36            <td>
37              <a href="{{ path('app_article_show', {'idArticle': article.idArticle}) }}><i class="fa-solid fa-magnifying-glass fa-fade fa-lg" style="color: #370028;"></i></a>
38              <a href="{{ path('app_article_edit', {'idArticle': article.idArticle}) }}><i class="fa-solid fa-pen-to-square fa-lg" style="color: #9e0e40;"></i></a>
39            </td>
40          </tr>
41        {% endfor %}
42      </tbody>
43    </table>
44  </div>
```



Dans le fichier **.env**, il faut relier le projet à la BDD en mettant les infos correctement afin de se connecter à la BDD MySQL de XAMPP.

```
| DATABASE_URL="mysql://root@127.0.0.1:3306/sdbm_v2?serverVersion=8.0.32&charset=utf8mb4"
| APP_SECRET="3f2d95a2a0c343a29a760a10a7a2a0a0"
| APP_NAME="SDBM"
```

Routage et code simples (Home Controller) :

```
Controller > elephant HomeController.php > ...
<?php

namespace App\Controller;

use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;
use Symfony\Component\HttpFoundation\Response;
use Symfony\Component\Routing\Annotation\Route;

class HomeController extends AbstractController
{
    #[Route('/', name: 'app_home')]
    public function index(): Response
    {
        return $this->render('home/index.html.twig', [
            'controller_name' => 'HomeController',
        ]);
    }
}
```

Le contrôleur est dans le namespace 'App\Controller', ce qui signifie que la classe 'HomeController' est définie dans ce namespace.

- `use Symfony\Bundle\FrameworkBundle\Controller\AbstractController` : Ce contrôleur hérite de la classe AbstractController fournie par Symfony. Cette classe abstraite offre des fonctionnalités de base pour la création de contrôleurs.
- `use Symfony\Component\HttpFoundation\Response` : Cette classe est utilisée pour représenter une réponse HTTP qui sera renvoyée au client.
- `use Symfony\Component\Routing\Annotation\Route` : Cette classe est utilisée pour définir des annotations de routage qui permettent de spécifier comment les URL doivent être gérées par ce contrôleur.

Définition de la classe : La classe HomeController étend la classe AbstractController, ce qui signifie qu'elle hérite des fonctionnalités de base nécessaires pour gérer les requêtes.

En résumé, ce code définit un contrôleur Symfony pour gérer l'URL de la page d'accueil de l'application. Lorsque l'URL '/' est demandée, la méthode index est exécutée, elle rend un modèle Twig et renvoie le contenu HTML correspondant au client. Le modèle Twig utilisé affiche le nom du contrôleur 'HomeController'.

Routage avec paramètre

```

> Controller > ArticleController.php > ArticleController > index
<?php

namespace App\Controller;

use App\Entity\Article;
use App\Form\ArticleType;
use Doctrine\ORM\EntityManagerInterface;
use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;
use Symfony\Component\HttpFoundation\Request;
use Symfony\Component\HttpFoundation\Response;
use Symfony\Component\Routing\Annotation\Route;
use Symfony\Contracts\Translation\TranslatorInterface;

#[Route('/article')]
class ArticleController extends AbstractController
{
    #[Route('/', name: 'app_article_index', methods: ['GET'])]
    public function index(EntityManagerInterface $entityManager): Response
    {
        $articles = $entityManager
            ->getRepository(Article::class)
            ->findBy([], ['idArticle' => 'DESC'], 25);

        return $this->render('article/index.html.twig', [
            'articles' => $articles,
        ]);
    }
}

```

Espace de noms (**namespace**) : Dans Symfony, les espaces de noms (namespace) sont utilisés pour organiser le code de manière logique. Le code est placé dans le **namespace App\Controller**, ce qui signifie que la classe ArticleController est située dans le répertoire correspondant à ce namespace.

Utilisation des classes :

- **use App\Entity\Article** : Cette ligne importe la classe Article, qui est une entité (généralement liée à une table de base de données) utilisée pour représenter des articles dans l'application.
- **use App\Form\ArticleType** : Cette ligne importe la classe ArticleType, qui est un formulaire utilisé pour créer ou éditer des articles.
- **use Doctrine\ORM\EntityManagerInterface** : Cette ligne importe l'interface EntityManagerInterface de Doctrine, qui est utilisée pour interagir avec la base de données et gérer les entités.
- **use Symfony\Contracts\Translation\TranslatorInterface** : Cette ligne importe l'interface TranslatorInterface, qui permet de gérer la traduction de chaînes de texte dans l'application.

D'autres classes comme AbstractController, Request, Response, et Route sont également importées pour être utilisées dans le code.

Méthode index (#[Route('/', name: 'app_article_index', methods: ['GET'])]) :

Cette méthode est annotée avec **#[Route('/', name: 'app_article_index', methods: ['GET'])]**. Cela signifie que cette méthode sera exécutée lorsque l'URL '/article/' est demandée en utilisant la méthode HTTP GET.

La méthode reçoit un paramètre \$entityManager de type EntityManagerInterface, qui sera automatiquement injecté par Symfony grâce à l'injection de dépendance. Cet objet est utilisé pour interagir avec la base de données.

À l'intérieur de la méthode, elle récupère les articles depuis la base de données en utilisant le entityManager. Elle les récupère en fonction de la classe Article, trie les articles par identifiant décroissant ('idArticle' DESC), et limite les résultats à 25 articles.

Ensuite, la méthode rend une vue Twig en utilisant le modèle 'article/index.html.twig' et passe les articles à afficher sous la clé 'articles'.

En résumé, ce code représente un contrôleur Symfony appelé ArticleController qui gère les fonctionnalités liées aux articles de l'application. La méthode index est responsable de récupérer les articles depuis la base de données et de les afficher en utilisant un modèle Twig. Cette méthode est accessible via l'URL '/article/', et les articles seront affichés lorsque l'utilisateur accédera à cette URL en utilisant la méthode GET.

7.1 DEVELOPPEMENT D'UN CRUD COMPLEXE DE TABLE ARTICLE (en SYMFONY) CREATE (AJOUTER)

```
#Route('/new', name: 'app_article_new', methods: ['GET', 'POST'])
public function new(Request $request, EntityManagerInterface $entityManager, TranslatorInterface $translator): Response
{
    $article = new Article();
    $form = $this->createForm(ArticleType::class, $article);
    $form->handleRequest($request);

    if ($form->isSubmitted() && $form->isValid()) {
        $entityManager->persist($article);
        $entityManager->flush();

        // Génération du message d'information
        $this->addFlash([
            'success',
            $translator->trans('L\'article a bien été ajouté !')
        ]);

        return $this->redirectToRoute('app_article_index', [], Response::HTTP_SEE_OTHER);
    }

    return $this->render('article/new.html.twig', [
        'article' => $article,
        'form' => $form,
    ]);
}
```

#Route('/new', name: 'app_article_new', methods: ['GET', 'POST']) : Cette annotation définit une route pour cette méthode. L'URL correspondante pour cette route est '/article/new'. Le nom de la route est 'app_article_new', ce qui signifie que vous pouvez vous y référer ailleurs dans le code. La méthode peut être invoquée à la fois avec les méthodes HTTP GET et POST.

Signature de la méthode :

public function new(Request \$request, EntityManagerInterface \$entityManager, TranslatorInterface \$translator): Response : Cette méthode s'attend à recevoir trois paramètres :

\$request : Il s'agit d'un objet Request qui contient les informations sur la requête HTTP entrante, y compris les données du formulaire si la méthode est appelée avec POST.

\$entityManager : C'est une instance de **EntityManagerInterface**, qui permet d'interagir avec la base de données pour gérer les entités.

\$translator : Il s'agit d'un objet **TranslatorInterface**, utilisé pour gérer la traduction de chaînes de texte dans l'application.

Création d'une nouvelle instance d'article :

\$article = new Article(); : Une nouvelle instance de la classe Article est créée. Cela prépare un objet pour stocker les informations d'un nouvel article.

Création d'un formulaire :

\$form = \$this->createForm(ArticleType::class, \$article); : Un formulaire est créé en utilisant la classe **ArticleType** (un formulaire défini ailleurs dans l'application) et l'objet \$article créé précédemment. Le formulaire est prêt à être utilisé pour la création d'un nouvel article.

Cette méthode gère la création d'un nouvel article. Elle crée un formulaire, traite les soumissions, effectue une validation, persiste l'article en base de données, ajoute un message de succès, puis redirige l'utilisateur vers la liste des articles. Si le formulaire n'est pas soumis ou s'il y a des erreurs de validation, il affiche à nouveau le formulaire pour permettre à l'utilisateur de corriger les erreurs.

READ(AFFICHER)

```
#[Route('/{idArticle}', name: 'app_article_show', methods: ['GET'])]
public function show(Article $article): Response
{
    return $this->render('article/show.html.twig', [
        'article' => $article,
    ]);
}
```

Annotation de routage :

#[Route('/{idArticle}', name: 'app_article_show', methods: ['GET'])] : Cette annotation définit une route pour cette méthode. L'URL correspondante pour cette route est '/article/{idArticle}', où

{idArticle} est une partie variable de l'URL. Le nom de la route est 'app_article_show', ce qui signifie que vous pouvez vous y référer ailleurs dans le code. La méthode ne répond qu'aux requêtes HTTP de type GET.

Rendu de la page d'affichage de l'article :

À l'intérieur de la méthode, elle renvoie une réponse HTTP en rendant un modèle Twig en utilisant `return $this->render('article/show.html.twig', ['article' => $article]);`.

Le modèle Twig 'article/show.html.twig' est utilisé pour générer la page d'affichage de l'article, et l'objet \$article est passé au modèle pour être affiché.

```

private $idType;

public function getIdArticle(): ?int
{
    return $this->idArticle;
}

public function getNomArticle(): ?string
{
    return $this->nomArticle;
}

public function setNomArticle(string $nomArticle): static
{
    $this->nomArticle = $nomArticle;
    return $this;
}
    
```

En résumé, ces méthodes sont utilisées pour accéder et modifier les attributs de l'entité Article, en particulier l'identifiant (idArticle) et le nom de l'article (nomArticle). Les méthodes "getter" permettent de récupérer les valeurs de ces attributs, tandis que la méthode "setter" permet de définir une nouvelle valeur pour l'attribut nomArticle. L'utilisation de méthodes "getter" et "setter" est courante dans les entités Doctrine pour assurer l'encapsulation des données et le respect des bonnes pratiques de programmation orientée objet.

UPDATE (MODIFICATION)

```

#[Route('/{idArticle}/edit', name: 'app_article_edit', methods: ['GET', 'POST'])]
public function edit(Request $request, Article $article, EntityManagerInterface $entityManager, TranslatorInterface $translator): Response
{
    $form = $this->createForm(ArticleType::class, $article);
    $form->handleRequest($request);

    if ($form->isSubmitted() && $form->isValid()) {
        $entityManager->flush();

        // Génération du message d'information
        $this->addFlash(
            'warning',
            $translator->trans('L\'article a bien été modifié !')
        );

        return $this->redirectToRoute('app_article_index', [], Response::HTTP_SEE_OTHER);
    }

    return $this->render('article/edit.html.twig', [
        'article' => $article,
        'form' => $form,
    ]);
}
    
```

Annotation de routage :

#[Route('/{idArticle}/edit', name: 'app_article_edit', methods: ['GET', 'POST'])] : Cette annotation définit une route pour cette méthode. L'URL correspondante pour cette route est '/article/{idArticle}/edit', où {idArticle} est un paramètre variable dans l'URL qui représente l'identifiant de l'article à éditer.

Signature de la méthode :

public function edit(Request \$request, Article \$article, EntityManagerInterface \$entityManager, TranslatorInterface \$translator): Response : Cette méthode s'attend à recevoir plusieurs paramètres :

\$request : C'est un objet Request qui contient des informations sur la requête HTTP entrante, notamment les données du formulaire si la méthode est appelée avec POST.

\$article : C'est un objet de type Article qui représente l'article à éditer. L'objet est automatiquement récupéré à partir de l'identifiant {idArticle} présent dans l'URL grâce à la capacité de paramétrage automatique de Symfony.

\$entityManager : C'est une instance de EntityManagerInterface, qui est utilisée pour interagir avec la base de données et mettre à jour l'entité Article.

\$translator : C'est un objet TranslatorInterface utilisé pour gérer la traduction de chaînes de texte dans l'application.

Création d'un formulaire de modification :

\$form = \$this->createForm(ArticleType::class, \$article); : Un formulaire est créé en utilisant la classe ArticleType, qui est probablement un formulaire Symfony défini ailleurs dans l'application. Le formulaire est prérempli avec les données de l'article spécifié par l'objet \$article.

Gestion de la soumission du formulaire :

- **\$form->handleRequest(\$request);** : Le formulaire traite la requête HTTP pour récupérer les données du formulaire. Cela permet de mettre à jour l'objet \$article avec les nouvelles données soumises.

Validation du formulaire :

if (\$form->isSubmitted() && \$form->isValid()) : Cette condition vérifie si le formulaire a été soumis et si les données soumises sont valides. Si c'est le cas, le code à l'intérieur du bloc sera exécuté.

Mise à jour en base de données :

- **\$entityManager->flush();** Les changements apportés à l'objet \$article sont sauvegardés dans la base de données.

Ajout d'un message "flash" de confirmation :

En résumé, cette méthode permet de mettre à jour un article existant. Elle crée un formulaire prérempli avec les données de l'article à éditer, traite la soumission du formulaire, effectue une validation, met à jour l'article en base de données, ajoute un message de confirmation, puis redirige l'utilisateur vers la liste des articles. Si le formulaire n'est pas soumis ou s'il y a des erreurs de validation, il affiche à nouveau le formulaire pour permettre à l'utilisateur de corriger les erreurs.

DELETE (SUPPRESSION)

```
#Route('/{idArticle}', name: 'app_article_delete', methods: ['POST'])
public function delete(Request $request, Article $article, EntityManagerInterface $entityManager, TranslatorInterface $translator): Response
{
    if ($this->isCsrfTokenValid('delete' . $article->getIdArticle(), $request->request->get('_token'))) {
        $entityManager->remove($article);
        $entityManager->flush();

        // Génération du message d'information
        $this->addFlash(
            'danger',
            $translator->trans('L\'article a bien été supprimé !')
        );
    }

    return $this->redirectToRoute('app_article_index', [], Response::HTTP_SEE_OTHER);
}
```

Annotation de routage:

#Route('/{idArticle}', name: 'app_article_delete', methods: ['POST']) : Cette annotation définit une route pour cette méthode. L'URL correspondante pour cette route est '/article/{idArticle}', où {idArticle} est un paramètre variable dans l'URL qui représente l'identifiant de l'article à supprimer.

Signature de la méthode :

public function delete(Request \$request, Article \$article, EntityManagerInterface \$entityManager, TranslatorInterface \$translator): Response : Cette méthode s'attend à recevoir plusieurs paramètres :

Vérification du jeton CSRF :

if (\$this->isCsrfTokenValid('delete' . \$article->getIdArticle(), \$request->request->get('_token'))) : Cette condition vérifie si le jeton CSRF (Cross-Site Request Forgery) est valide. Le jeton CSRF est une mesure de sécurité pour protéger contre les attaques de type CSRF. Il est généré lors de l'affichage du formulaire de suppression et doit correspondre au jeton envoyé avec la requête POST. Si le jeton est valide, le code à l'intérieur du bloc sera exécuté.

Suppression de l'article en base de données :

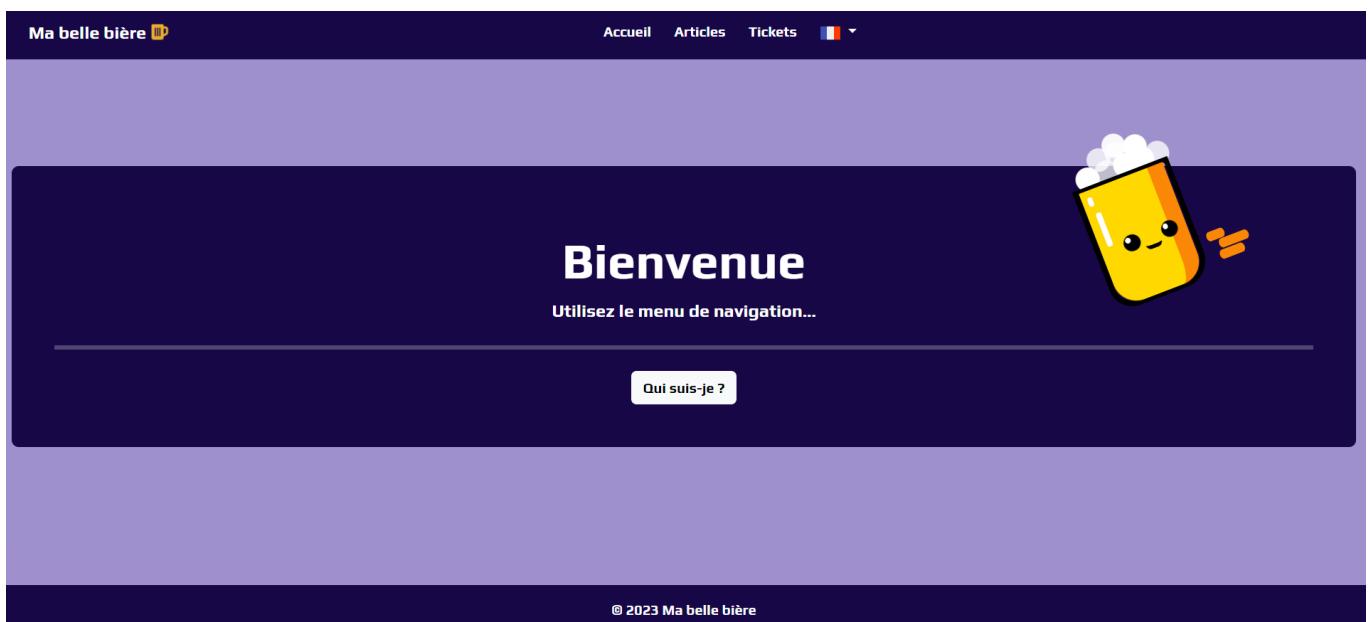
\$entityManager->remove(\$article); : L'objet \$article est marqué pour suppression.

Redirection vers la liste des articles :

return \$this->redirectToRoute('app_article_index', [], Response::HTTP_SEE_OTHER); : Une fois que l'article a été supprimé avec succès, l'utilisateur est redirigé vers la liste des articles (la page d'index des articles) en utilisant la route 'app_article_index'.

7.2 UTILISATION DE PROGRAMME SYMFONY 6.3

Page d'accueil avec les boutons de navigation



Une clique sur un bouton affiche le choix des langues.



FONCTIONNEMENT « AJOUTER » (CREATE)

Lorsque on clique sur ajouter un nouvel article, une autre fenêtre affichera pour demander plus de détails de l'article à ajouter.

A screenshot of a modal window titled 'Créer un nouvel article'. The window contains fields for entering article details: 'Nom de l'article', 'Prix d'achat', 'Volume', 'Titrage', 'Marque', 'Couleur', and 'Type'. Each field has a corresponding input box. A cartoon beer glass icon is positioned on the right side of the form. At the bottom is a blue 'Enregistrer' (Save) button.

Un message de confirmation s'affichera à chaque fois que l'utilisateur enregistre de l'ajout d'un nouvel article de bière



FONCTIONNEMENT « AFFICHER » (READ)

Il faut cliquer sur le bouton pour afficher : un autre page avec un formulaire pour afficher un article de bière, comme ci-dessous :

Code Article	Nom De l'Article	Prix d'Achat	Volume	Tirage	Couleur	Type	Marque	Actions
1	das Echte Märzen	2.14€	33ml	5.7%	Blonde	Bière de Saison	das Echte Märzen	
2	das Helle	2.17€	33ml	5%	non renseigné	non renseigné	das Helle	
3	das Schwarze	1.9€	33ml	4.9%	Brune	non renseigné	das Schwarze	

das Echte Märzen

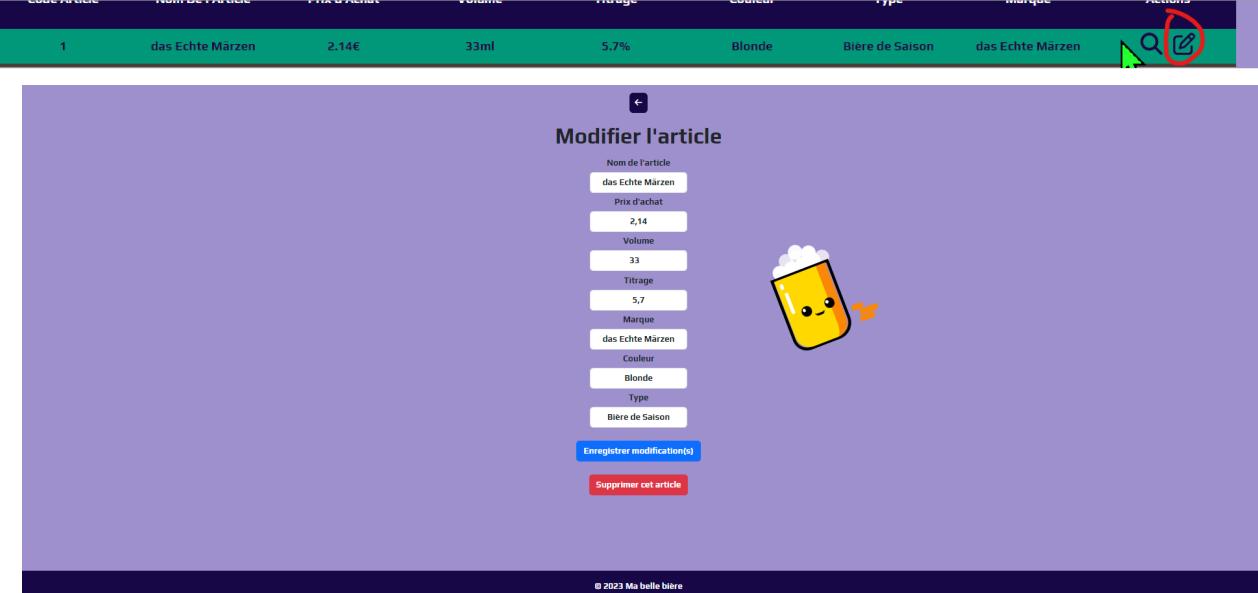
Code Article	1
Nom Article	das Echte Märzen
Prix Achat	2.14€
Volume	33ml
Tirage	5.7%
Couleur	Blonde
Marque	das Echte Märzen
Type	Bière de Saison

[Modifier cet article](#)

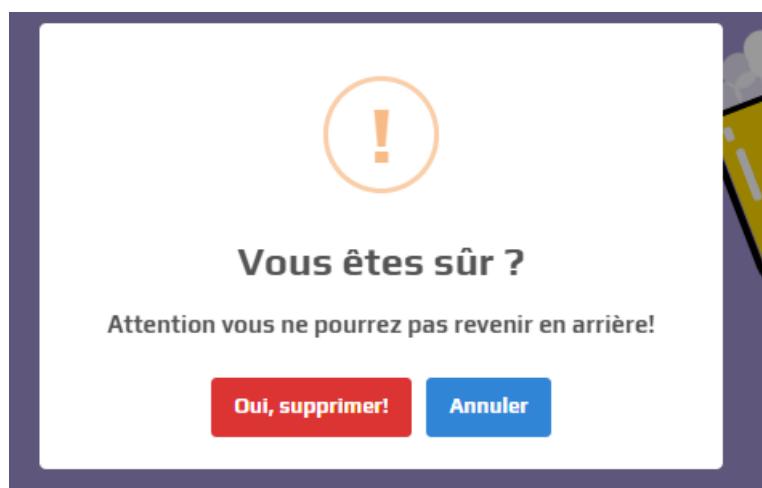
[Supprimer cet article](#)

FONCTIONNEMENT « MODIFIER »

Il faut cliquer sur le bouton  pour modifier : un autre page avec un formulaire pour modifier un article de bière, comme ci-dessous :

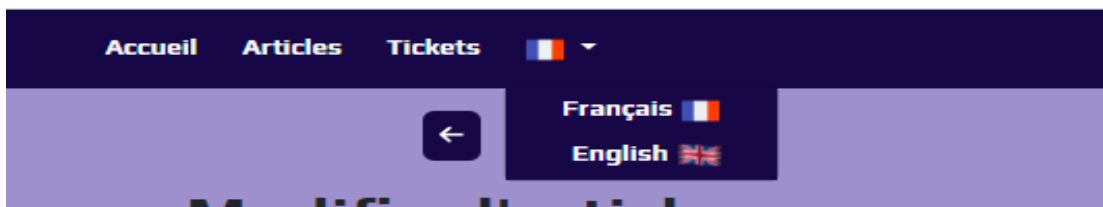
**FONCTIONNEMENT « SUPPRESSION »**

Pour le cas de la suppression d'un article, la clique sur le bouton  affiche une alerte comme l'image ci-dessous demandant la confirmation de suppression. Si l'utilisateur clique sur « OK », la ligne va être supprimée.



CHOIX DE LANGUES :

Un utilisateur pourrait choisir la langue qui lui convient, 2 choix ont été traduit : la langue anglais et la langue française. Il suffit de cliquer sur le menu déroulant (l'image ci-dessous) :



8. HEBERGEMENT DU SITE

Quand le projet est prêt pour héberger, « FileZilla » est utilisé pour transférer le répertoire qui contient tous les dossiers et les fichiers du projet vers l'hébergeur « Hostinger » dans un sous répertoire « MBB » à l'intérieur du répertoire « public_html » du domaine « zakaribel.com ».

Alors, l'adresse url de cette application est zakaribel.com/MBB Puis une base de données (BDD) MySQL qui s'appelle « u481835991_sdbm » avec le nom d'utilisateur « u481835991_root » et son propre mot de passe est créée chez « Hostinger».

Base de données MySQL	Utilisateur MySQL	Créé le	Site Web	
u481835991_sdbm 16 MB	u481835991_root	2023-10-05	zakaribel.com	Accédez à phpMyAdmin ⋮

Comme le nom de la BDD, le nom de l'utilisateur et le mot de passe sont différents chez l'hébergeur que chez le server local, il a fallu faire les changements des attributs de la class « Model » dans le fichier « Model.php ».

En server de XAMPP (localhost)

```
app > 🏷 Model.php > 📄 Model
 1  <?php
 2  abstract class Model{
 3      // Informations de La base de données
 4      private $host = "localhost";
 5      private $db_name = "sdbm_v2";
 6      private $username = "root";
 7      private $password = "";
```

Chez Hostinger

```
<?php
abstract class Model{
    private $host = "localhost";
    private $db_name = "u481835991_sdbm";
    private $username = "u481835991_root";
    private $password = "*****";
```

HEBERGEMENT DU SITE (SYMFONY)

Quand le projet est prêt pour héberger, « FileZilla » est utilisé pour transférer le répertoire qui contient tous les dossiers et les fichiers du projet vers l'hébergeur « Hostinger » dans un nouveau sous répertoire « **MBB_S** » à l'intérieur du répertoire « **public_html** » du domaine « **zakaribel.com** ».

On doit transférer tous les fichiers sur « FileZilla » sauf le **vendor**, le **var** et le **.env.local**. On utilise PuTTY comme terminal permettant la connexion à « Hostinger » par protocole ssh et on doit installer composer2 pour faciliter le transfert des fichiers. Les 2 commandes nécessaires sont : **composer2 install** (pour avoir un fichier vendor) et puis **composer2 require symfony/apache-pack** (pour avoir .htaccess)
On utilise la même base de données **sdbm_v2**.

L'adresse url de cette application:

https://zakaribel.com/MBB_S/public

9. LA SECURISATION DES DONNEES

Comment se protéger des failles XSS (Cross-site Scripting)?

L'encodage d'entités HTML est sûrement la mesure la plus essentielle car les scripts malveillants sont souvent injectés via des balises HTML. Il s'agit donc d'encoder la plupart des entités HTML pouvant constituer un risque afin qu'elles soient interprétées comme fiables.

Comment se protéger des injections SQL ?

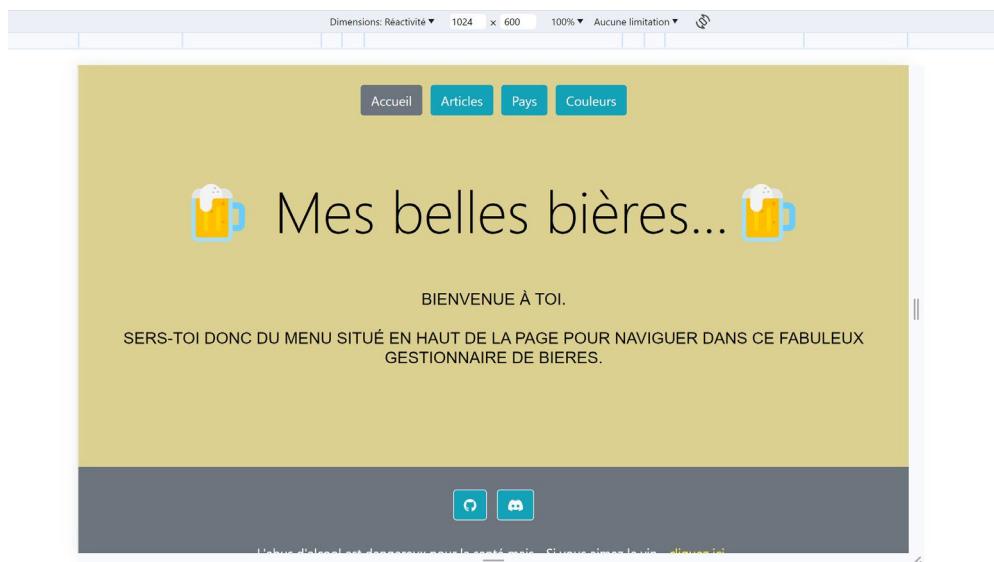
Pour de protéger des injections SQL, nous allons utiliser la fonction **prepare** de PDO.

Cette méthode consiste à séparer la requête les valeurs envoyées. Il est possible de ne pas nommer ses clés et d'utiliser des points d'interrogation « ? » à la place. Dans ce cas, le tableau de valeurs n'est pas indexé et l'ordre des données doit correspondre à l'ordre d'apparition des points d'interrogation.

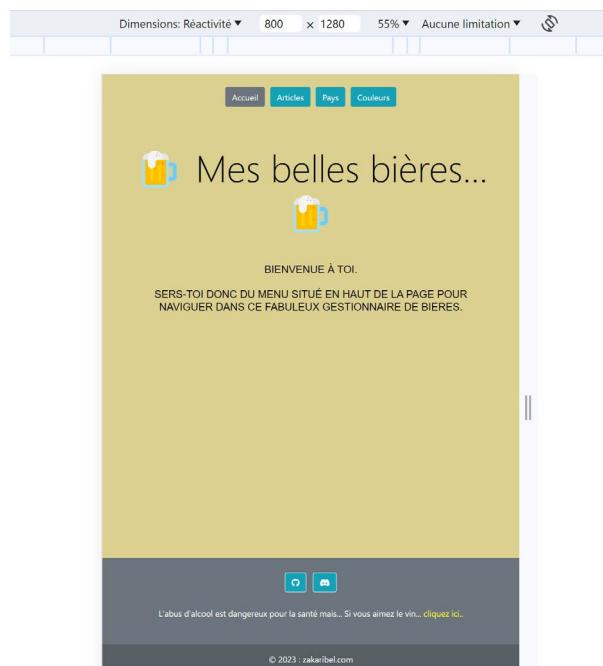
10. Visuel de l'interface sur

écrans Responsive Test

Ordinateur portable



Tablette

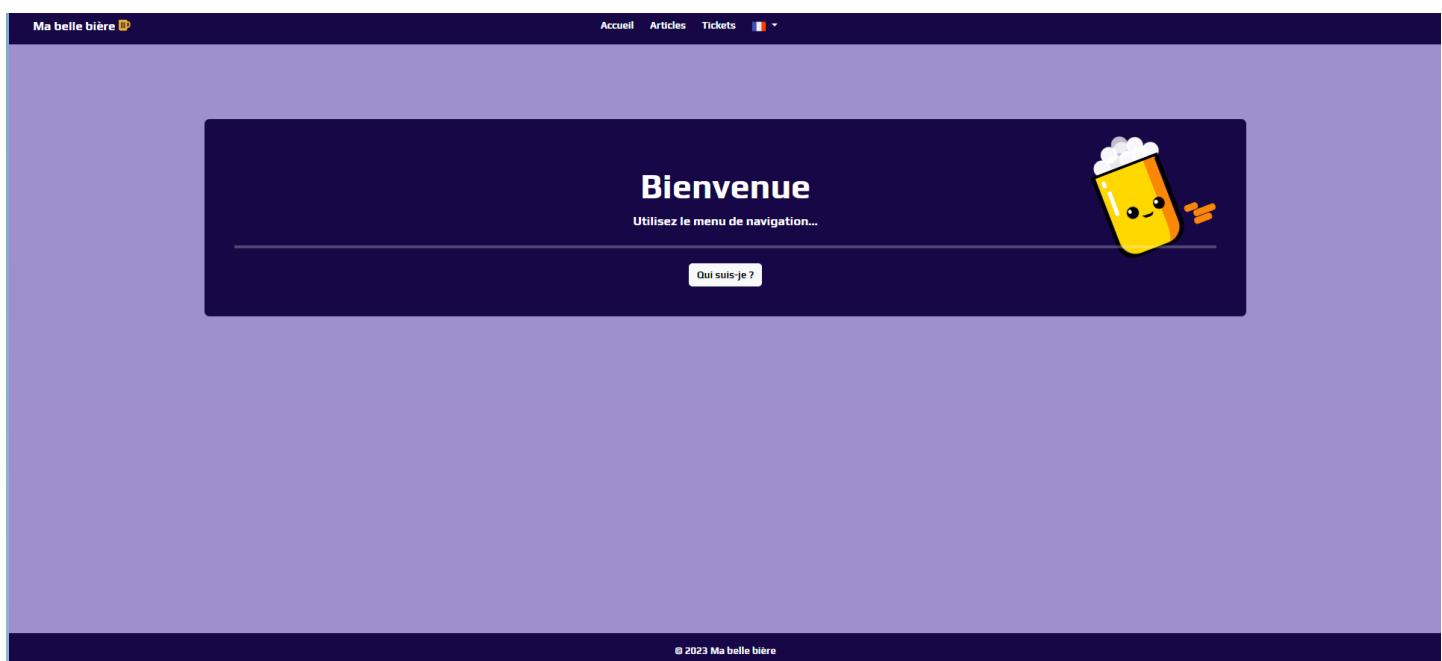


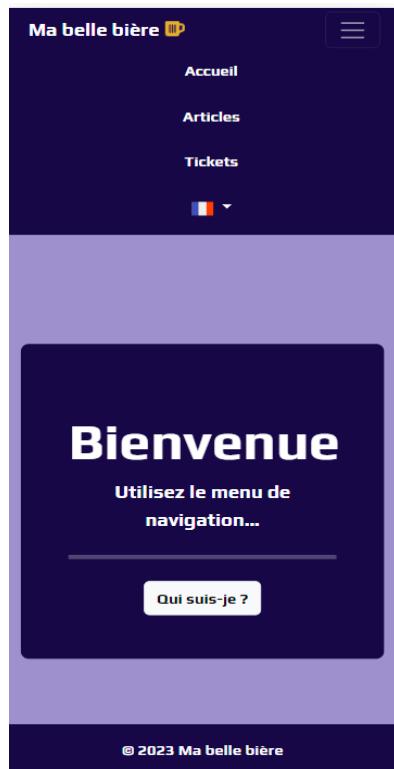
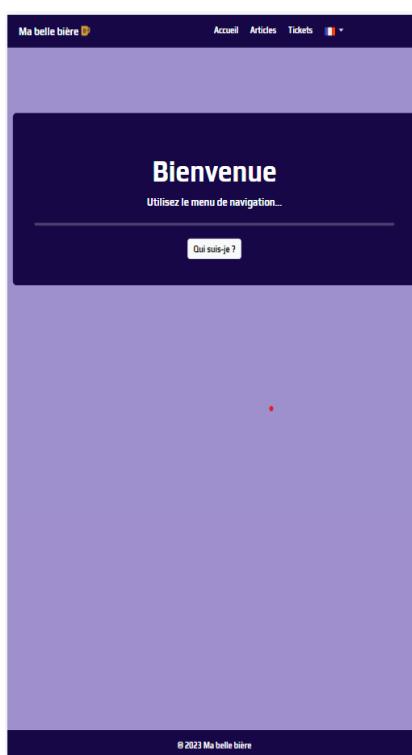
Mobile



RESPONSIVE TEST (Projet Symfony)

Ordinateur portable



Mobile**Tablette**

11. LES TESTS PHP MVC

Test	Affichage	Nouveau	Modification	Suppression
Accueil	Ok			
Gestion des Pays	OK	Ok	Ok	OK
Gestion des Articles	OK	OK	OK	OK
Gestion de Couleurs	OK	OK	OK	OK

SYMFONY

Test	Affichage	Nouveau	Modification	Suppression
Accueil	Ok			
Gestion des Articles	OK	Ok	Ok	OK
Gestion des Tickets	OK	OK	OK	OK
Type de vente	OK	OK	OK	OK

12 LA VEILLE TECHNIQUE

https://www.w3schools.com/php/php_mysql_intro.asp

The screenshot shows a web browser displaying the 'PHP MySQL Database' tutorial from w3schools.com. The page header includes the w3schools logo, navigation links for Tutorials, Exercises, Get Certified, Services, and a search bar. The main content area is titled 'PHP MySQL Database' and contains an introduction stating: 'With PHP, you can connect to and manipulate databases. MySQL is the most popular database system used with PHP.' Below this, a section titled 'What is MySQL?' lists several bullet points about the MySQL database system.

- MySQL is a database system used on the web
- MySQL is a database system that runs on a server
- MySQL is ideal for both small and large applications
- MySQL is very fast, reliable, and easy to use
- MySQL uses standard SQL
- MySQL compiles on a number of platforms
- MySQL is free to download and use
- MySQL is developed, distributed, and supported by Oracle Corporation

<https://www.php.net/manual/en/function.htmlentities.php>

Description

```
htmlentities(
    string $string,
    int $flags = ENT_QUOTES | ENT_SUBSTITUTE | ENT_HTML401,
    ?string $encoding = null,
    bool $double_encode = true
): string
```

This function is identical to [htmlspecialchars\(\)](#) in all ways, except with [htmlentities\(\)](#), all characters which have HTML character entity equivalents are translated into these entities. The [get_html_translation_table\(\)](#) function can be used to return the translation table used dependent upon the provided **flags** constants.

If you want to decode instead (the reverse) you can use [html_entity_decode\(\)](#).

Parameters

string
The input string.

Flags

Search bar: Search

- crr
- chunk_split
- convert_uudecode
- convert_uuencode
- count_chars
- crc32
- crypt
- echo
- explode
- fprintf
- get_html_translation_table
- hebrev
- hexbin
- html_entity_decode
- » htmlentities**
- htmlspecialchars_decode
- htmlspecialchars
- implode
- join
- lfirst
- levenshtein
- localeconv
- ltrim
- md5_file
- md5
- metaphone
- money_format
- nl_langinfo

<https://stackoverflow.com/questions/68019647/left-join-from-multiple-tables>

Left join FROM multiple tables

Asked 2 years, 3 months ago Modified 2 years, 3 months ago Viewed 372 times

1 Something like this:

```
SELECT *
FROM table1, table2
LEFT JOIN other_table
ON other_table.id = table2.id
AND other_table.otherId = table1.otherId
```

mysql sql database syntax left-join

Share Improve this question Follow asked Jun 17, 2021 at 12:44 helper12345 31 4

The Overflow Blog

- Open Discussion: What can be done to reduce infrastructure-as-code complexity?

Featured on Meta

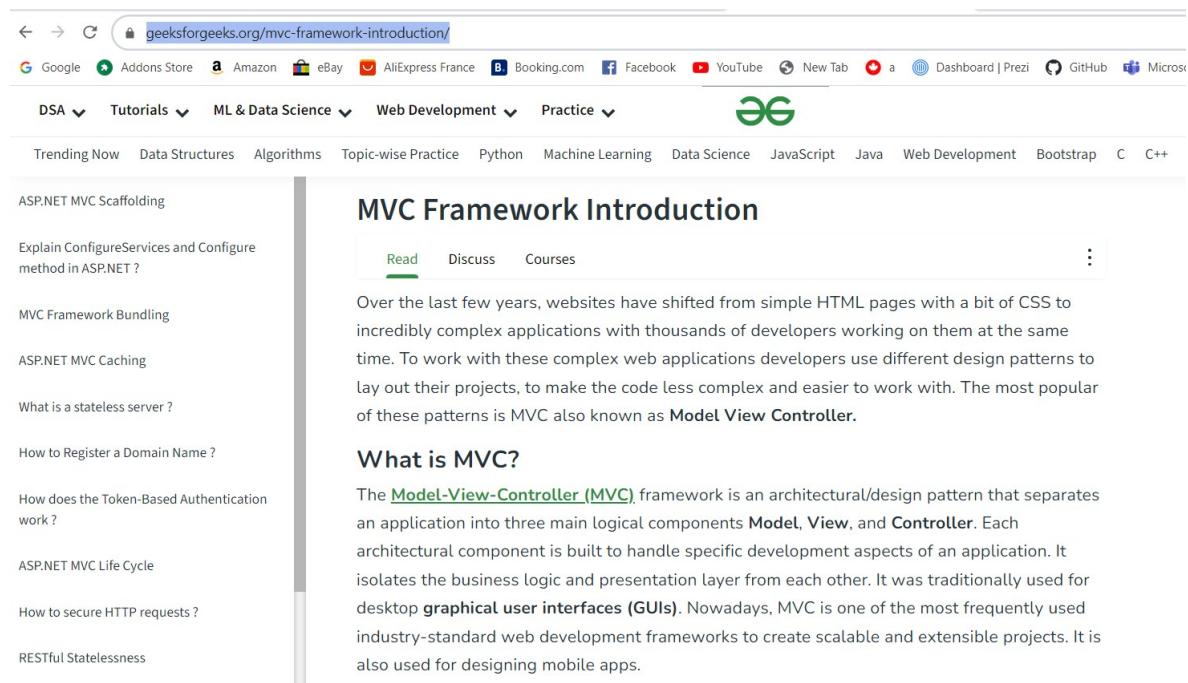
- Alpha test for short survey in banner ad slots starting on week of September...
- What should be next for community events?
- Temporary policy: Generative AI (e.g., ChatGPT) is banned
- Expanding Discussions: Let's talk about curation
- Update on Collectives and Discussions
- OverflowAI Search is now available for alpha testing (September 13, 2023)

Related

Join Stack Overflow to find the best answer to your technical question, help others answer theirs.

13. RECHERCHE SUR DES SITES ANGLOPHONE

<https://www.geeksforgeeks.org/mvc-framework-introduction/>



The screenshot shows a web browser displaying the URL [geeksforgeeks.org/mvc-framework-introduction/](https://www.geeksforgeeks.org/mvc-framework-introduction/). The page title is "MVC Framework Introduction". Below the title, there are three tabs: "Read" (which is selected), "Discuss", and "Courses". The main content area starts with a paragraph about how websites have shifted from simple HTML pages to complex applications. It then defines the Model-View-Controller (MVC) framework as an architectural pattern that separates an application into three components: Model, View, and Controller. The text explains that MVC isolates business logic and presentation layer, making it suitable for desktop GUIs and mobile apps.

ASP.NET MVC Scaffolding

Explain ConfigureServices and Configure method in ASP.NET ?

MVC Framework Bundling

ASP.NET MVC Caching

What is a stateless server ?

How to Register a Domain Name ?

How does the Token-Based Authentication work ?

ASP.NET MVC Life Cycle

How to secure HTTP requests ?

RESTful Statelessness

MVC Framework Introduction

Read Discuss Courses

Over the last few years, websites have shifted from simple HTML pages with a bit of CSS to incredibly complex applications with thousands of developers working on them at the same time. To work with these complex web applications developers use different design patterns to lay out their projects, to make the code less complex and easier to work with. The most popular of these patterns is MVC also known as **Model View Controller**.

What is MVC?

The **Model-View-Controller (MVC)** framework is an architectural/design pattern that separates an application into three main logical components **Model**, **View**, and **Controller**. Each architectural component is built to handle specific development aspects of an application. It isolates the business logic and presentation layer from each other. It was traditionally used for desktop **graphical user interfaces (GUIs)**. Nowadays, MVC is one of the most frequently used industry-standard web development frameworks to create scalable and extensible projects. It is also used for designing mobile apps.

14 BILAN

Les deux projets m'ont beaucoup intéressé, car la réalisation d'un site Internet en PHP MVC et SYMFONY m'ont permis de développer mes connaissances dans ce langage de programmation, ce qui me servira dans ma vie professionnelle.

Les deux sites réalisés répondent aux objectifs de ce projet, développement des pages dynamiques d'un site Web en PHP MVC et Symfony qui affichent les CRUD créés à partir des tables d'une base de données selon l'architecture MVC en utilisant les PDO.

Les leçons apprises sont :

- Utilisation des classes, l'héritage des classes, leurs attributs et leurs méthodes en PHP.
- Structurer un projet selon l'architecture MVC (Model View Controller).
- Préparer une base de données pour l'utiliser dans un projet.
- Accéder une base de données en utilisant le PDO (PHP Data Objects).
- Faire les requêtes SQL pour dialoguer avec la base de données.
- Import d'une base de données vers une base de données d'un autre serveur.

En résumé, ce projet m'a permis d'acquérir un ensemble de compétences techniques et pratiques essentielles dans le développement web et la gestion de bases de données, tout en offrant une expérience concrète de la création d'un site web interactif. Cela m'a encouragé à poursuivre d'autres projets et opportunités dans le domaine du développement web pour continuer à apprendre et créer du contenu interactif et convivial.