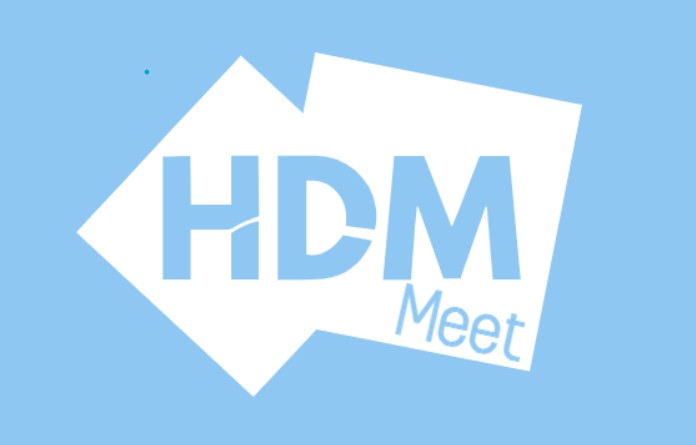
Projet HDMeet



**Zakaria BELKACEMI**Formation Développeur Web 02/02/2024

Projet HDMeet du 02/02/2024

**Projet HDMeet (React, Node)**

**Sommaire :**

|  |
| --- |
| 1- Compétences couvertes 3 |
| 2- Résume 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 |

# Liste des compétences couvertes par le projet

- 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

- Gérer l’authentification des utilisateurs et attribution de ROLES

- Etablir une connexion websocket entre les peers notamment via WEBRTC

# Résumé de Projet

L'objectif du projet est de développer une application de vidéoconférence en utilisant les technologies React pour le front-end et Node/Express pour le back-end, afin de remplacer l'utilisation de la version gratuite de Zoom dans l'entreprise actuelle. Cette transition est motivée par la limitation de 45 minutes imposée par Zoom sur les sessions gratuites.

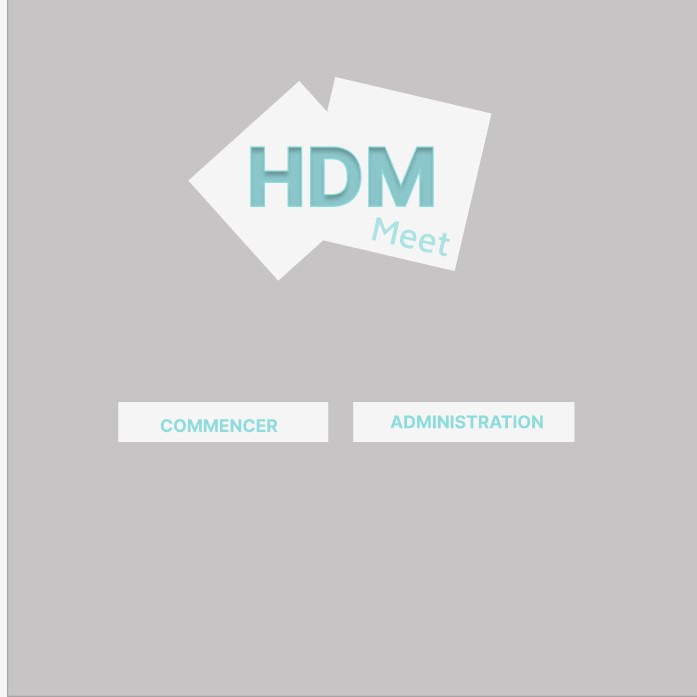
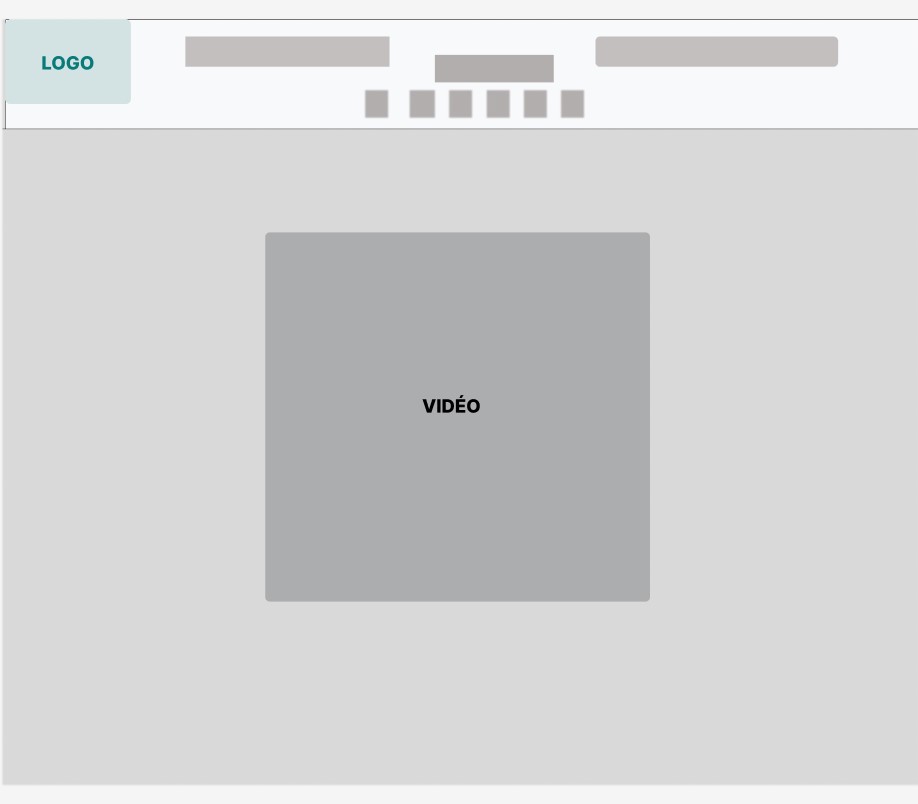
Le processus de développement a débuté par une analyse approfondie du cahier des charges, permettant de définir les besoins spécifiques de l'entreprise en matière de vidéoconférence. En se basant sur ces exigences, un Modèle Conceptuel de Données (MCD) a été élaboré pour comprendre la structure de la base de données MySQL qui sera utilisée.

Le front-end de l'application sera construit en utilisant React, assurant une interface utilisateur moderne et réactive. Du côté du back-end, Node.js avec Express sera employé pour créer un serveur robuste et efficace, permettant la gestion des sessions de vidéoconférence.

Le développement de cette application de vidéoconférence a suivi une approche Agile pour assurer une gestion flexible et itérative du projet. Une partie authentification a également été mise en place, au même titre qu’un panel Admin afin de gérer les droits des utilisateurs.

Les technologies Socket.IO et WebRTC ont été cruciales, permettant l'établissement de connexions en temps réel entre les pairs. Socket.IO a facilité la communication bidirectionnelle en temps réel, tandis que WebRTC a été essentiel pour la transmission de flux audio et vidéo, garantissant une expérience de vidéoconférence fluide et efficace. Cette combinaison de technologies modernes et bien établies aboutit à une application performante et adaptée aux besoins spécifiques de communication de l'entreprise.

# MAQUETTAGE DE L’APPLICATION

# CAHIER DES CHARGES

HDM souhaite se détacher de l’application Zoom pour ses vidéo conférence à cause de la contrainte suivante : les utilisateurs sont expulsés de la conférence au bout de 45 minutes. Cela est dû au fait qu’ils utilisent la version gratuite de Zoom. C’est dans ce contexte que HDM m’a demandé la création d’une application de vidéo conférence. Ainsi, les utilisateurs ne seraient alors plus expulsés durant les conférences.

Les utilisateurs doivent pouvoir s’identifier à l’aide de leurs adresse email. Une adresse email est lié à un rôle, un rôle est lié à un email.

A partir du moment où une adresse email est présente dans la base de données, cette adresse email a accès à l’application. Seuls les emails avec le rôle ADMIN ont la possibilité d’ajouter un nouvel accès ou de changer un rôle.

L’application doit être facile d’utilisation et le plus proche possible de l’application Zoom en termes de fonctionnalités (chat, partage d’écran, mute, et camera on/off)

Le développement des pages web dynamiques en utilisant les technologies MYSQL pour la gestion de la base de données, Réact en front-end et Node.js / Express en back-end.

Développement des CRUD à partir d’une base de données.

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 les failles XSS.

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

# SPECIFICATIONS TECHNIQUES

**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.

**REACT** :

React est une bibliothèque Javascript open-source permettant la construction d’interfaces utilisateur interactives et réactives.

**SOCKET.IO :**

Socket.IO est une bibliothèque javascript qui permet la mise en œuvre de la communication en temps réel bidirectionnelle entre le serveur et le client, facilitant le développement d’application web interactives

**BOOTSTRAP :**

C’est un framework CSS pour rendre mon site responsive design.

**ANTD :**

Ant Design (antd) est une bibliothèque de conception de composants React qui offre un ensemble complet de composants d'interface utilisateur (UI) préconçus et stylisés. J’ai principalement utilisé cette bibliothèque pour mes alertes messages.

**MATERIAL-UI :**

Material-UI est une bibliothèque de composants React basée sur les principes du design Material, créée par Google. J’ai utilisé m-ui pour mes boutons et mes inputs.

**WebRTC:**

WebRTC (Web Real Time communication) est un ensemble de normes et de protocoles permettant la communication en temps réel directement entre navigateurs web (c’est pourquoi c’est utilisé seulement côté front) sans nécessiter de plugins ou d’installations tierces.

**MYSQL :**

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 de données. Il utilise le langage SQL

**EXPRESS :**

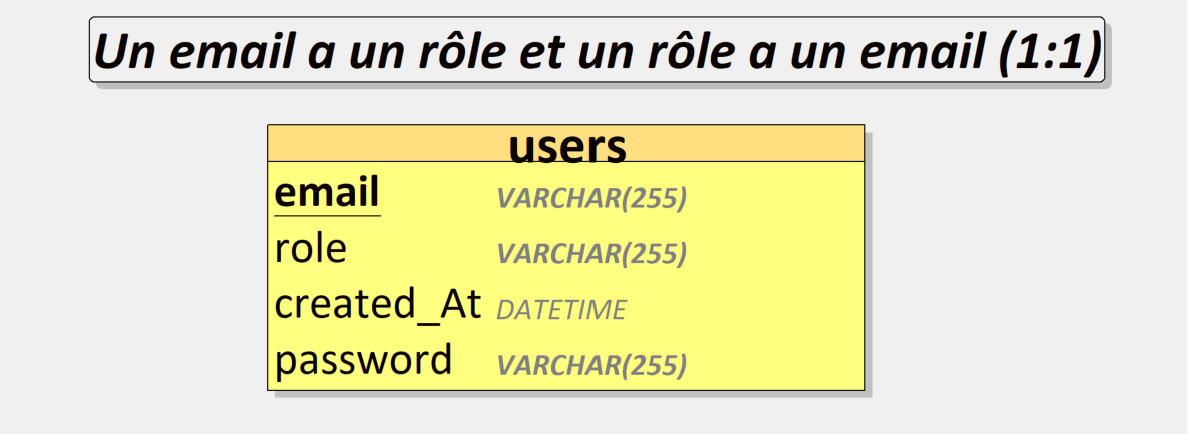
Express est un framework web pour Node.js, conçu pour simplifier le processus de création d'applications web et d'API.

**NodeJS**

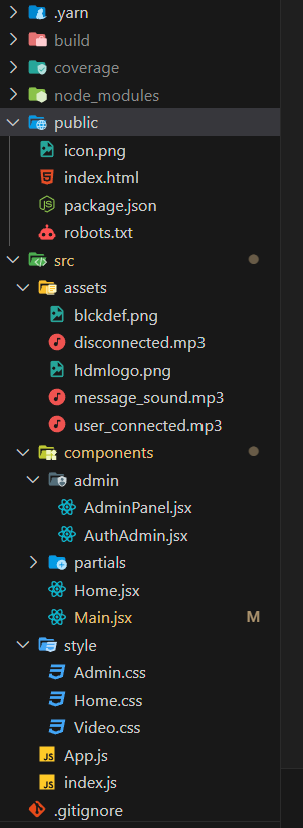
Node.js permet d’exécuter du code JavaScript côté serveur, alors que traditionnellement Javascript était principalement utilisé côté client dans les navigateurs web. Cela unifie le langage de programmation entre le côté client et le côté serveur.

# La base de données et son Schéma Qu’est-ce qu’une base de données ?

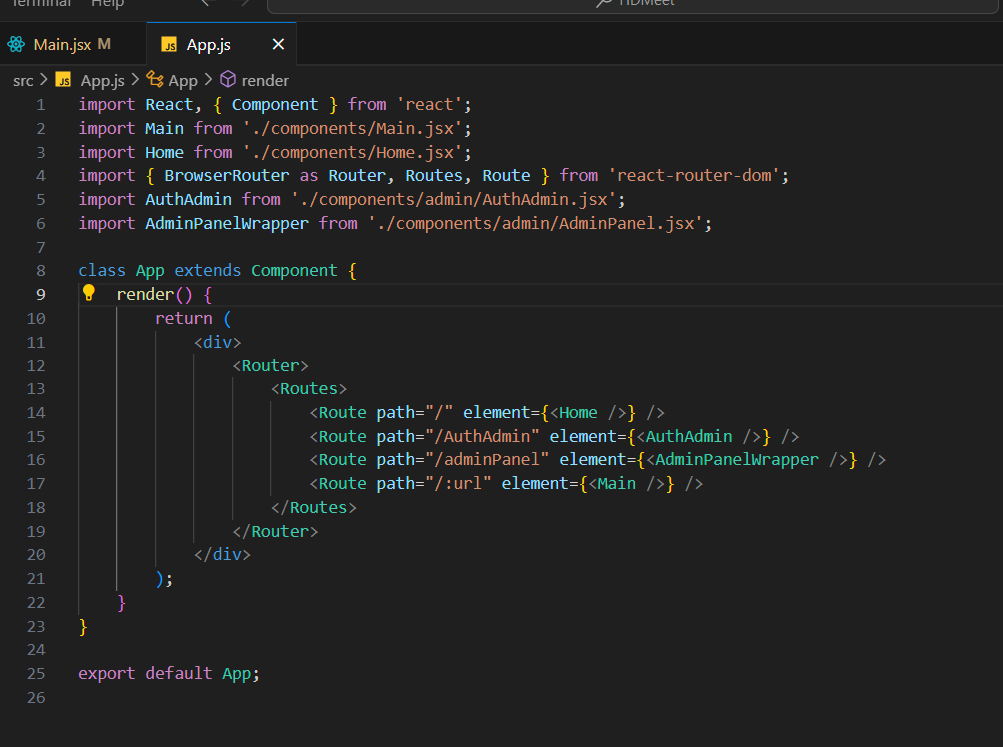
On les appelle “BDD” par commodité : les bases de données ont pour but de stocker, organiser et analyser les données.

Le schéma ci-dessous est le M.C.D (Modèle Conceptuel des données) 

# REALISATION DU PROJET HDM Meet

1. 

**App.js :**

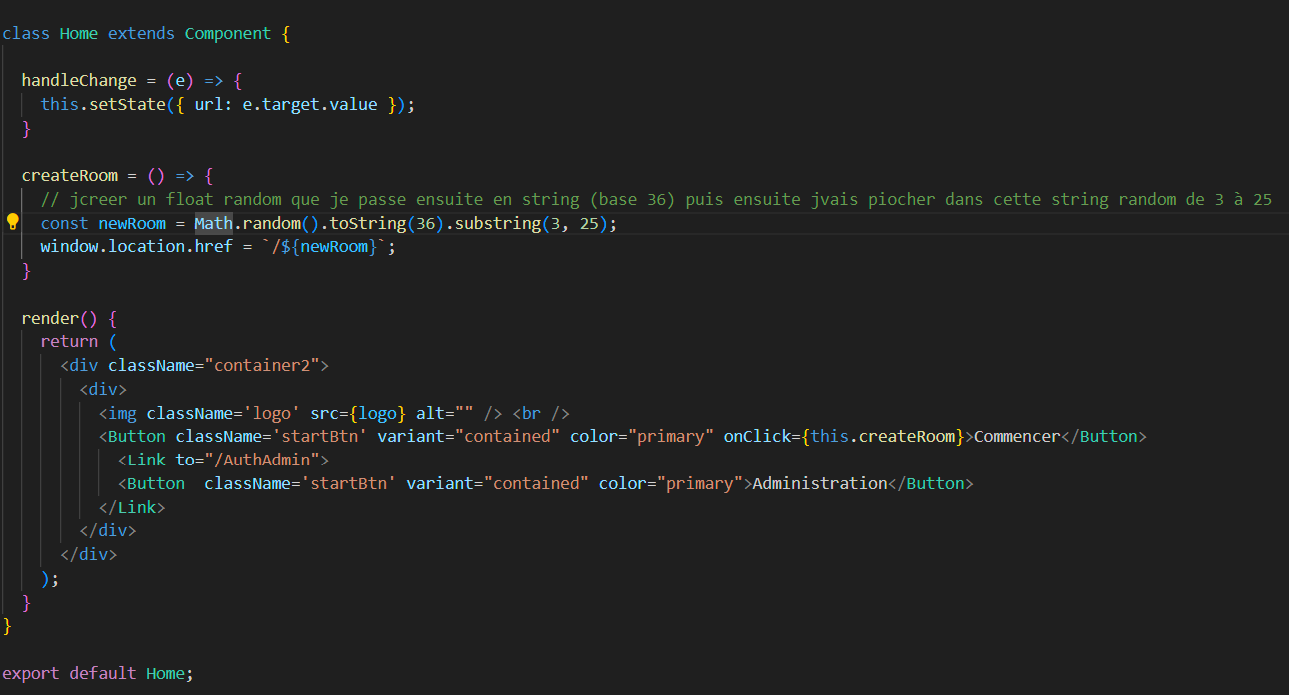


Le fichier **app.js** constitue le point d'entrée principal de notre application React. Il orchestre le routage et la navigation au sein de l'application en utilisant la bibliothèque **react-router-dom**.

* **Composants Importés :** J’importe les composants principaux, tels que **Main**, **Home**, **AuthAdmin**, et **AdminPanelWrapper**, qui représentent les différentes sections de notre application.
* **Configuration du Routage :** En utilisant **BrowserRouter**, **Routes**, et **Route** de **react-router-dom**, nous définissons les différentes routes de notre application. Chaque route est associée à un chemin spécifique et rend un composant particulier.
* **Routes Définies :**
  + La route "/" est associée au composant **Home**.
  + La route "/AuthAdmin" est associée au composant **AuthAdmin**.
  + La route "/adminPanel" est associée au composant **AdminPanelWrapper**.
  + La route "/:url" est dynamique, associant le chemin à un composant **Main**.

Ce fichier centralise la structure de navigation de notre application, facilitant la compréhension et la gestion des différentes sections. Il offre une vision claire de la logique de routage qui guide le rendu des composants en fonction des chemins d'URL spécifiés.

**Home.jsx :**



Le fichier **Home.jsx** définit le composant **Home**, qui représente la page d'accueil de notre application. Voici un aperçu des fonctionnalités clés de ce composant :

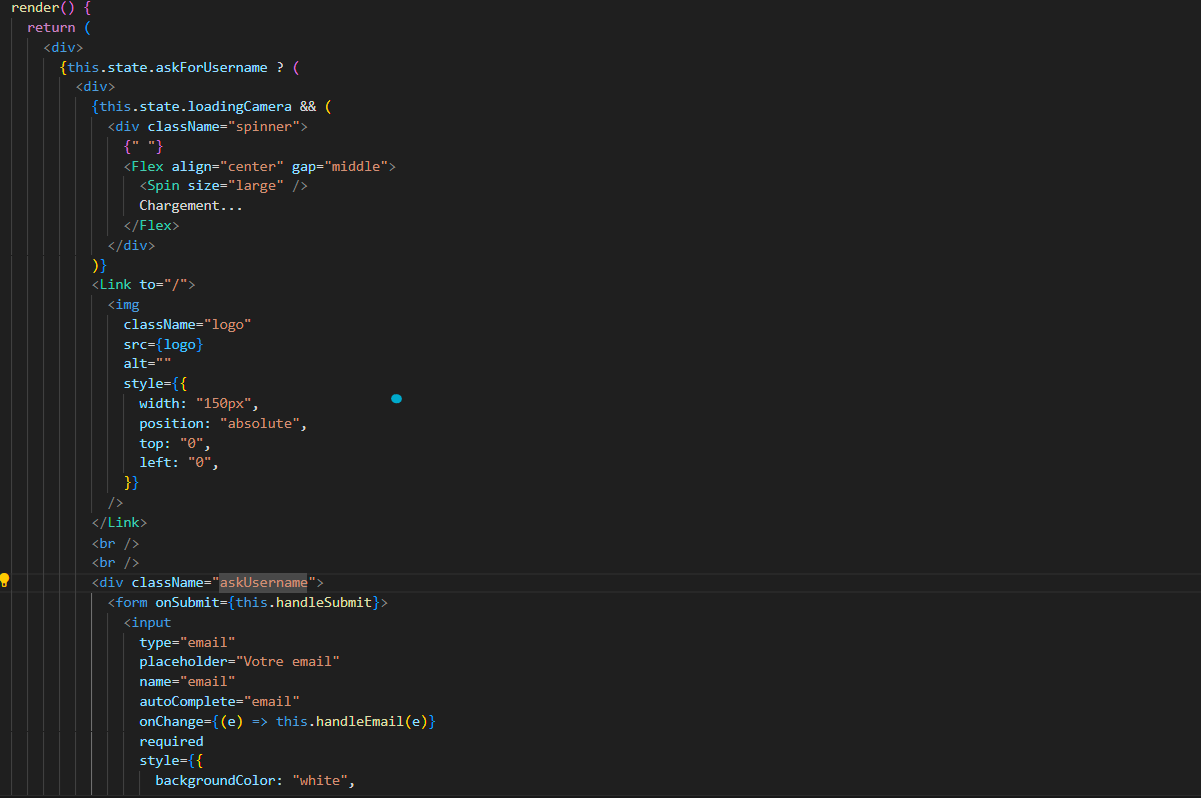
* **Imports :** Nous importons les dépendances nécessaires, y compris le composant **Button** de **@material-ui/core**, une image de logo, des styles CSS à partir de **Home.css**, et le composant **Link** de **react-router-dom**.
* **Méthode handleChange :** Cette méthode réagit aux changements dans le champ de texte, mais elle semble inutilisée dans le code actuel. Elle pourrait être utilisée pour mettre à jour un état en fonction des changements de saisie utilisateur.
* **Méthode createRoom :** Lorsque l'utilisateur clique sur le bouton "Commencer", cette méthode génère un identifiant de salle unique de manière aléatoire. Elle utilise un nombre aléatoire converti en une chaîne de caractères en base 36, puis extrait une sous-chaîne de cette valeur pour obtenir un identifiant unique. Ensuite, l'utilisateur est redirigé vers la nouvelle salle en utilisant **window.location.href**.
* **Rendu :** La méthode **render** génère l'interface utilisateur de la page d'accueil, comprenant le logo, un bouton "Commencer" qui déclenche la création d'une nouvelle salle, et un bouton "Administration" qui redirige vers la page d'authentification pour les administrateurs.

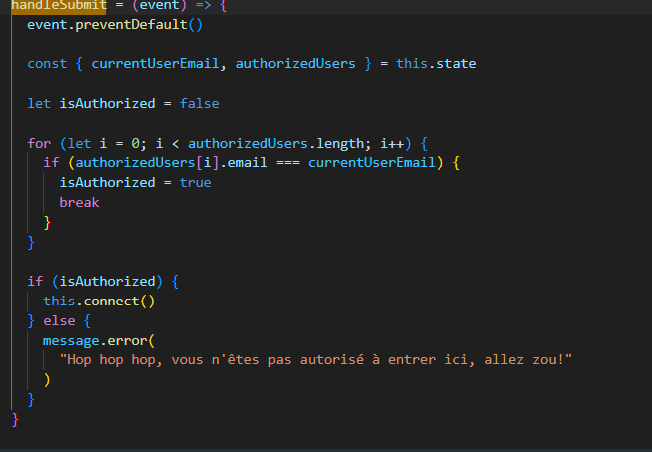
Ce composant constitue la première interaction de l'utilisateur avec l'application, offrant des fonctionnalités clés telles que la création de nouvelles salles et l'accès à l'administration. L'utilisation de boutons visuels améliore l'expérience utilisateur, et la navigation entre les pages est gérée de manière transparente grâce à **react-router-dom**.

**Main.jsx :**

Le composant **Main.jsx** gère la logique principale de notre application.

Depuis la page home, lorsque l’utilisateur a cliqué sur « commencer » il atterrit dans une partie d’authentification

. 



* **Condition askForUsername :** La section est conditionnée par **this.state.askForUsername**, qui contrôle si l'application doit demander à l'utilisateur de fournir son email et son nom d'utilisateur.
* **Formulaire de Connexion :** Un formulaire est rendu avec deux champs : un champ d'email et un champ de nom d'utilisateur. Ces champs sont stylisés et obligatoires (**required**). Les valeurs saisies sont gérées par les méthodes **handleEmail** et **handleUsername**.
* **Bouton "Se connecter" :** Un bouton "Se connecter" déclenche la méthode **handleSubmit** lorsqu'il est cliqué.
* **Vidéo en Arrière-plan :** Une vidéo en arrière-plan est rendue, affichant le flux vidéo de l'utilisateur. La vidéo est également cliquable pour zoomer, déclenchant la méthode **handleVideoClick** qui utilise l’API fullscreen des navigateurs.

**Vérification saisie :** La méthode **handleSubmit** est déclenchée lors de la soumission du formulaire où l'utilisateur entre son email et son nom d'utilisateur.

**event.preventDefault() :** Cette ligne empêche le comportement par défaut du formulaire, évitant ainsi le rechargement de la page lors de la soumission.

* **Extraction des Variables :** Les variables **currentUserEmail** et **authorizedUsers** sont extraites de l'état actuel (**this.state**).
* **Vérification de l'Autorisation :** La méthode itère à travers la liste des utilisateurs autorisés (**authorizedUsers**) pour vérifier si l'email actuel (**currentUserEmail**) est présent parmi les utilisateurs autorisés. Si une correspondance est trouvée, la variable **isAuthorized** est définie sur **true**.
* **Connexion Autorisée :** Si l'utilisateur est autorisé (si **isAuthorized** est **true**), la méthode **connect()** est appelée, impliquant probablement la transition vers la phase suivante de la vidéoconférence ou l'activation de fonctionnalités spécifiques.
* **Refus d'Accès :** Si l'utilisateur n'est pas autorisé, un message d'erreur est affiché à l'aide de la bibliothèque "antd" (**message.error**). Cela informe l'utilisateur qu'il n'a pas l'autorisation d'accéder à la vidéoconférence.

Cette méthode garantit que seuls les utilisateurs autorisés peuvent accéder à la vidéoconférence. La vérification se fait en comparant l'email actuel avec la liste des utilisateurs autorisés, renforçant ainsi la sécurité et la gestion des accès à l'application.

Lorsque l'utilisateur entre ses informations et clique sur "Se connecter", la logique de l'application passe à la deuxième partie où la vidéoconférence est activée et d'autres fonctionnalités sont disponibles. Cette approche offre une transition transparente entre l'inscription initiale et la participation active à la conférence.

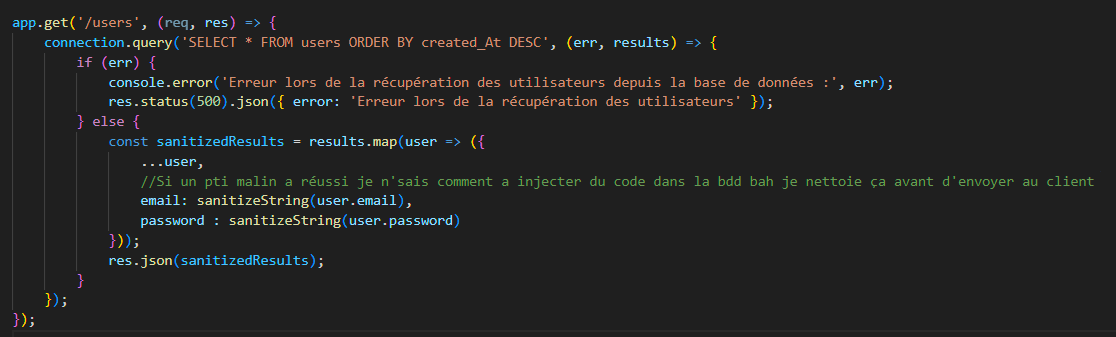
Déterminer si un utilisateur est autorisé ou non implique un call API afin d’alimenter une « state » qui sera ensuite traitée dans handleSubmit.



* Une requête HTTP GET est effectuée sur mon endpoint :"<http://localhost:4001/users>" pour obtenir la liste des utilisateurs autorisés.
* En cas de succès (**then**), la réponse est utilisée pour mettre à jour l'état **authorizedUsers** avec les données récupérées depuis l'API.
* En cas d'erreur (**catch**), un message d'erreur est affiché dans la console.

A noter que cet événement se produit dès l’initialisation du composant étant donné que mon call API est situé dans le constructeur du composant (j’aurai également pu mettre ce call dans un componentDidMount).

**Concernant la partie back-end de mon endpoint « users », il s’agit d’une requête SQL à ma base de données afin de récupérer tout de la table « users ».**



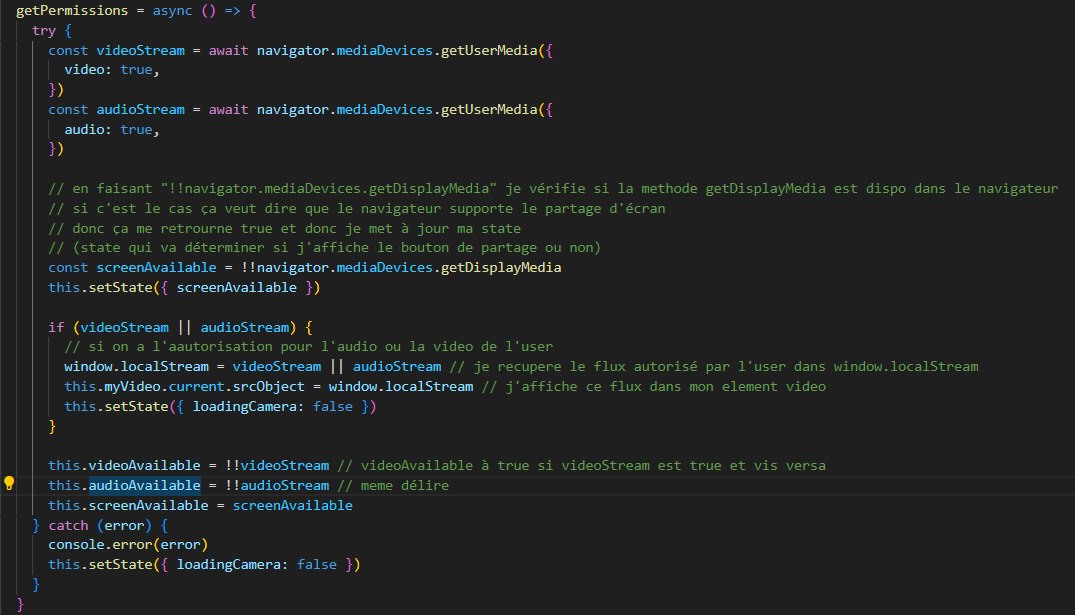
Cette route permet de récupérer la liste des utilisateurs depuis une base de données MySQL. Voici les points clés :

1. **Requête SQL :** La route effectue une requête SQL pour sélectionner tous les utilisateurs dans la table 'users', les triant par ordre décroissant de création.
2. **Gestion des Erreurs :** En cas d'erreur lors de l'exécution de la requête SQL, un message d'erreur est affiché dans la console, et une réponse JSON avec le statut 500 (Internal Server Error) est renvoyée au client.
3. **Sanitisation des Données :** Les résultats de la requête sont traités pour la sanitisation des données avant d'être renvoyés au client. La sanitisation implique le nettoyage des valeurs sensibles, telles que les emails et les mots de passe, pour prévenir les attaques d'injection de code malveillant.
4. **Réponse JSON :** En cas de succès, les résultats traités sont renvoyés au client sous forme de réponse JSON.

En résumé, cette route offre une manière sécurisée de récupérer la liste des utilisateurs, en veillant à nettoyer les données sensibles avant de les transmettre au client.

A présent, permettez moi de vous décrire les étapes pour établir une connexion entre pairs à l’aide de Socket.IO et WebRTC.

Dans un premier temps, lorsque l’utilisateur va créer sa « room » nous allons récupérer sa permission afin de pouvoir utiliser sa source audio/video. Pour cela , j’ai créée la methode getPermissions. Cette méthode est appelée dans le constructeur.



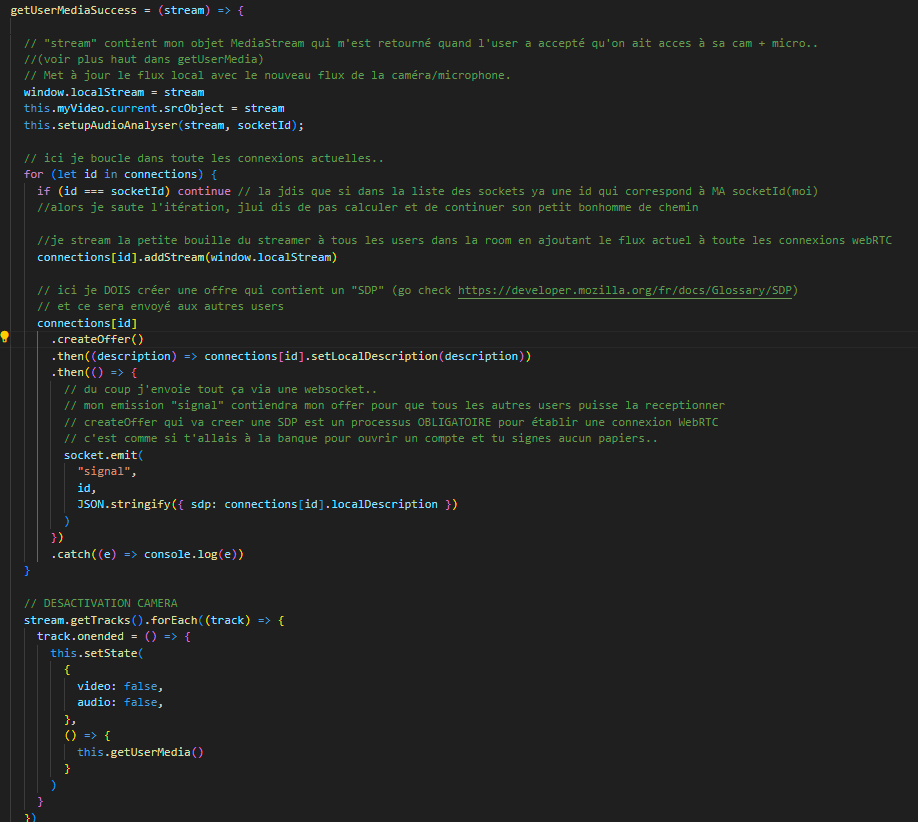
***getPermissions*** permet d'obtenir les autorisations de l'utilisateur pour accéder à sa caméra et à son microphone, et détermine également si le partage d'écran est disponible dans le navigateur.

**Autorisations de la Caméra et du Microphone :** La méthode utilise l'API **navigator.mediaDevices.getUserMedia** pour demander à l'utilisateur l'autorisation d'accéder à sa caméra (**video: true**) et à son microphone (**audio: true**).

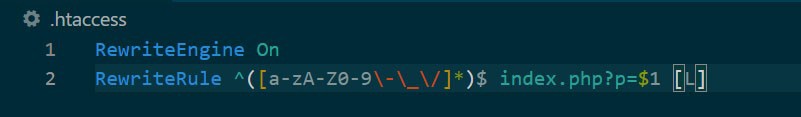
1. **Vérification de la Disponibilité du Partage d'Écran :** La méthode vérifie la disponibilité de la fonction **navigator.mediaDevices.getDisplayMedia** pour déterminer si le navigateur supporte le partage d'écran. La variable **screenAvailable** est mise à jour en conséquence.
2. **Mise à Jour de l'État :** Les résultats des autorisations pour la caméra et le microphone sont vérifiés. Si au moins l'un d'entre eux est autorisé, le flux autorisé est récupéré (**window.localStream**) et affiché dans un élément vidéo (**this.myVideo.current**). L'état de chargement de la caméra est également mis à jour.
3. **Mise à Jour de l'Indicateur de Disponibilité :** Les indicateurs **videoAvailable** et **audioAvailable** sont mis à jour en fonction de la disponibilité des flux vidéo et audio.
4. **Gestion des Erreurs :** En cas d'erreur lors de la demande d'autorisations, un message d'erreur est affiché dans la console, et l'état de chargement de la caméra est mis à jour pour refléter le problème.

En résumé, cette méthode gère la demande d'autorisations pour la caméra et le microphone de l'utilisateur, détermine la disponibilité du partage d'écran, et met à jour les états en conséquence. Elle assure une initialisation appropriée des flux vidéo et audio pour la vidéoconférence.

Une fois les permissions récupérées j’appelle getUserMediaSuccess.



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



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:

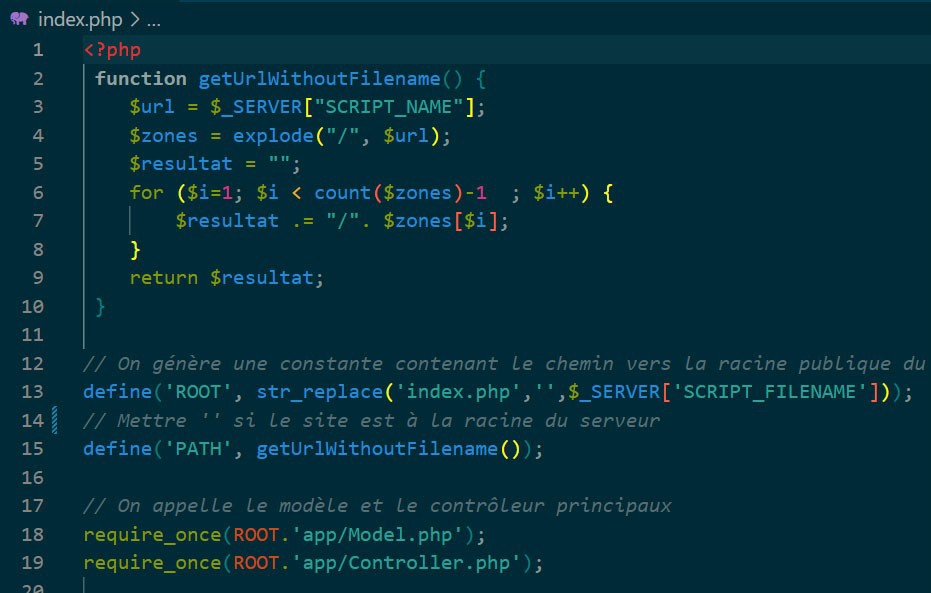
A-Z : majuscules de la A à la Z 0-9 : chiffres 0 à 9

\-\\_\/ : - (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 GETet 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.





# DEVELOPPEMENT D’UN CRUD COMPLEXE

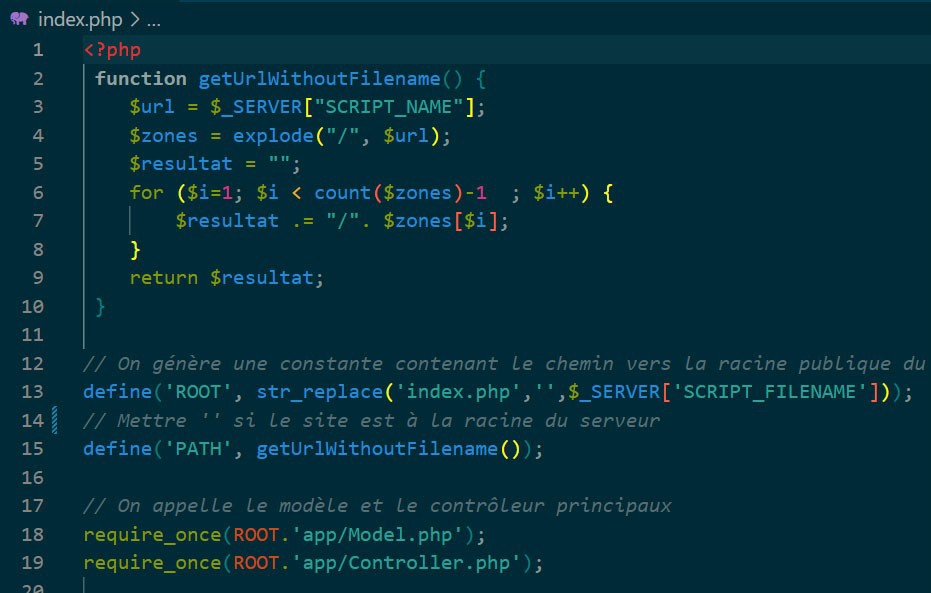
La création du CRUD complexe de la table « pays » de la base de données

« sdbm\_v2 ».

## AFFICHAGE

Dans le menu de navigation, on suit ce lien :

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



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 propriéte 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.

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 donées. Les résultats de la requête sont renvoyés sous forme de table associatif en utilisant « fetchAll().

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

# 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.



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

&amp ;&lt ;/&gt).

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.



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.

# La Modification

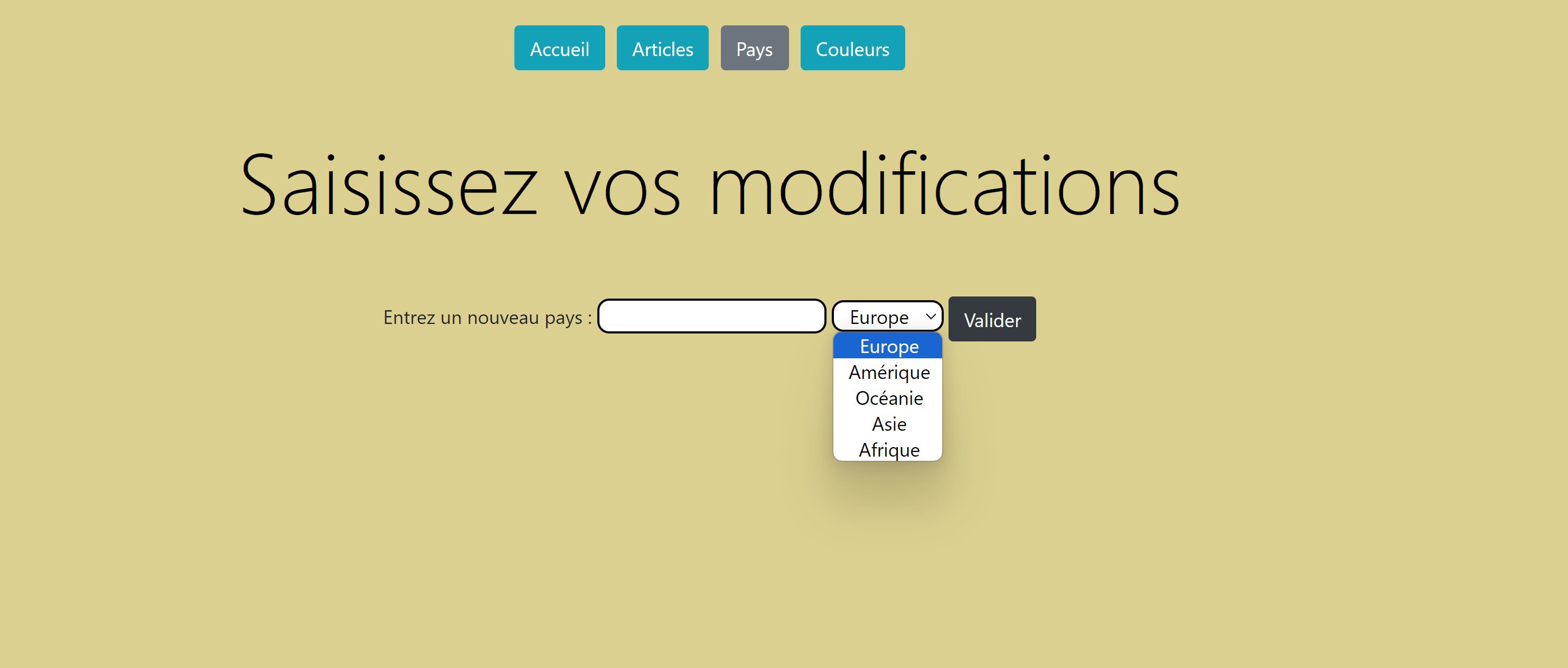
La modification fonctionne sur le même principe que l’ajout, une méthode updateCountry sur « controllers » charge un formulaire de modification et une méthode updateCountry sur « models » lance la requête SQL “updateCountry”.

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.



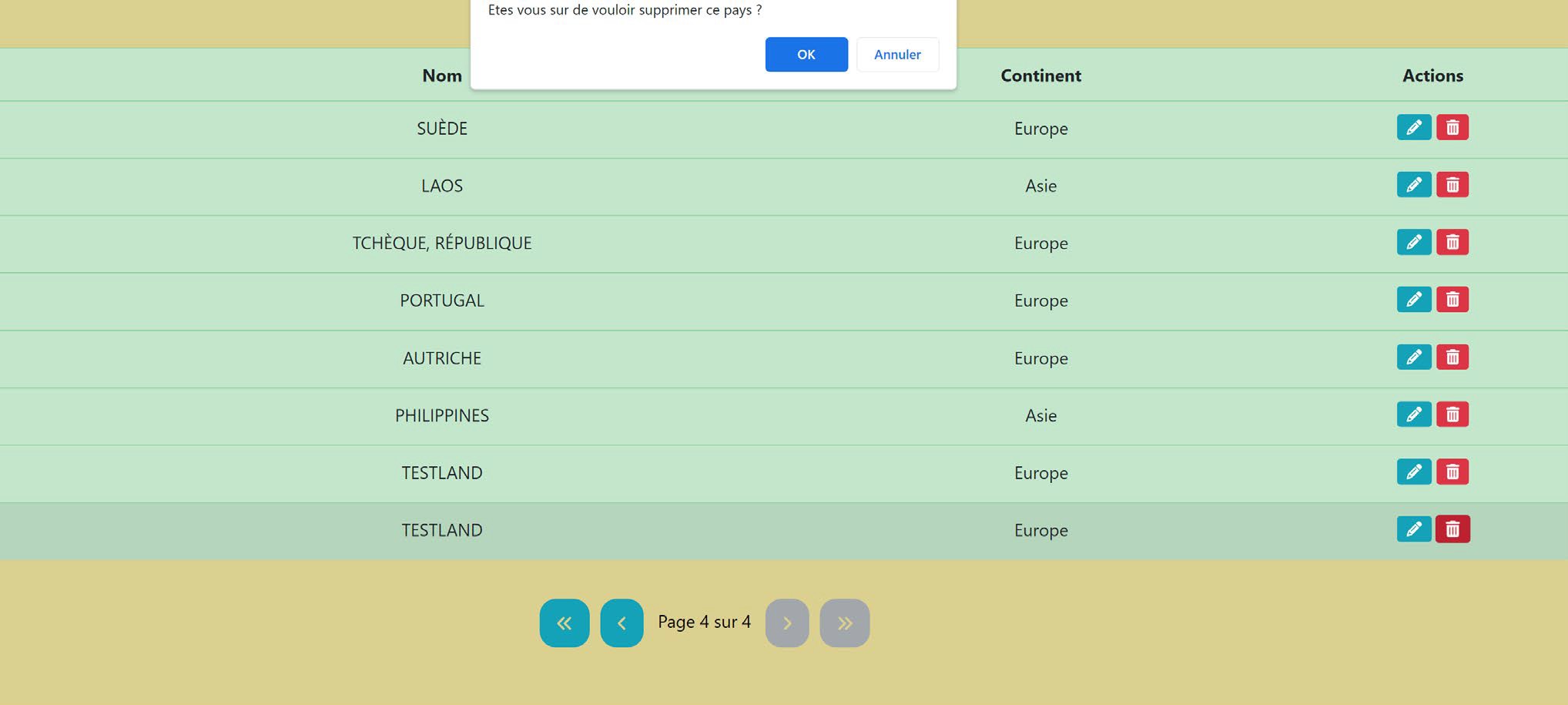
Cette méthode « updateCountry » permet de mettre à jour les informations d’un pays dans la base de données en utilisant les données fournies dans le tableau « $updatedCountry ». Elle construit une requête SQL d’Update, prépare cette requête, puis l’exécute avec les valeurs appropriées.

Le formulaire doit être remplir comme ceci :



# 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.



# 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.



## FONCTIONNEMENT « AJOUTER »

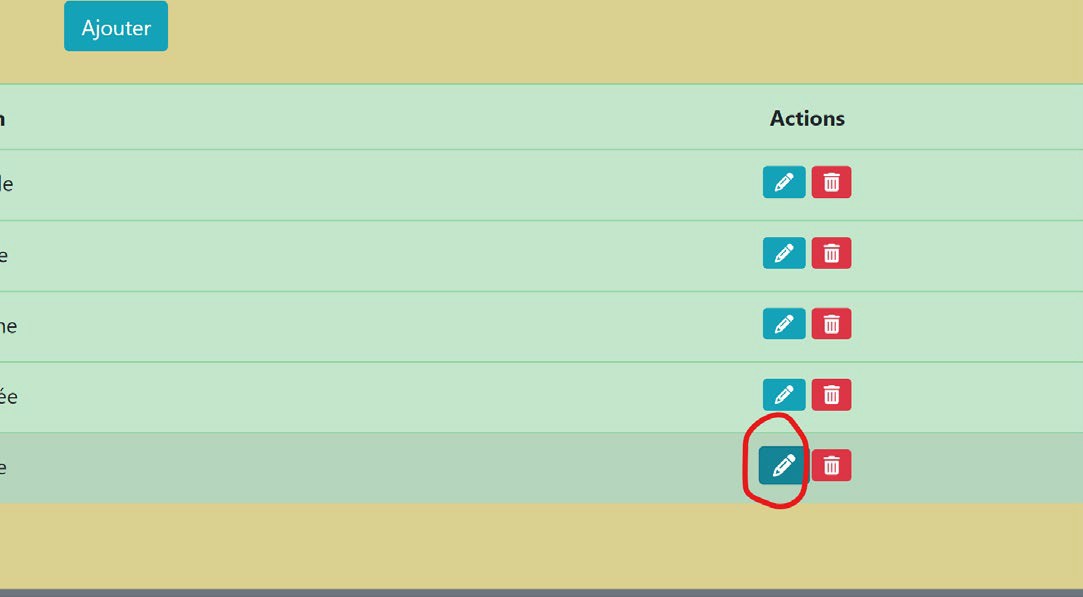
La clique sur le bouton «Ajouter» fera apparaître une formulaire d'ajout.

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



## 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 :





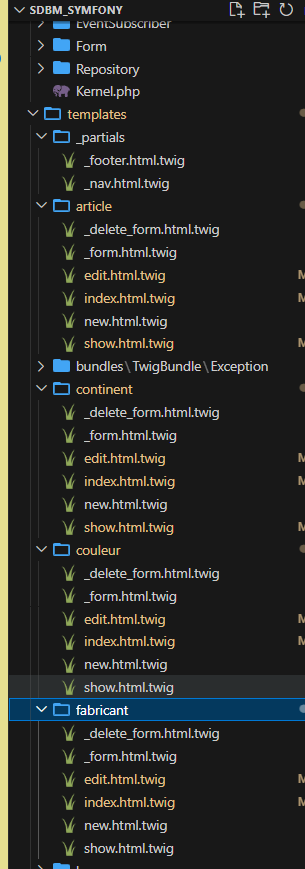
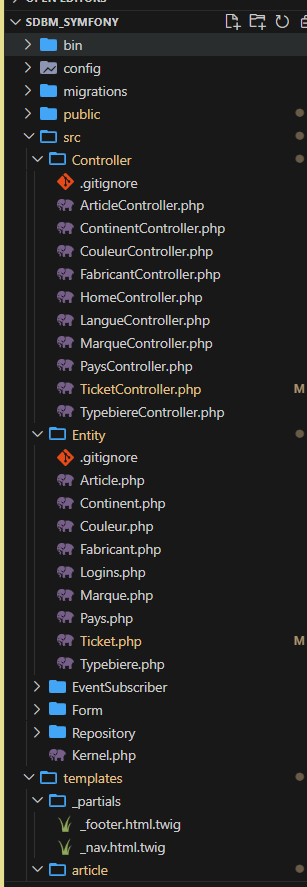
## FONCTIONNEMENT « SUPPRIMER »

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.



## 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.

Creation 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](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 génére 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 générationd’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. embeddedcontrollers)
  + 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 :

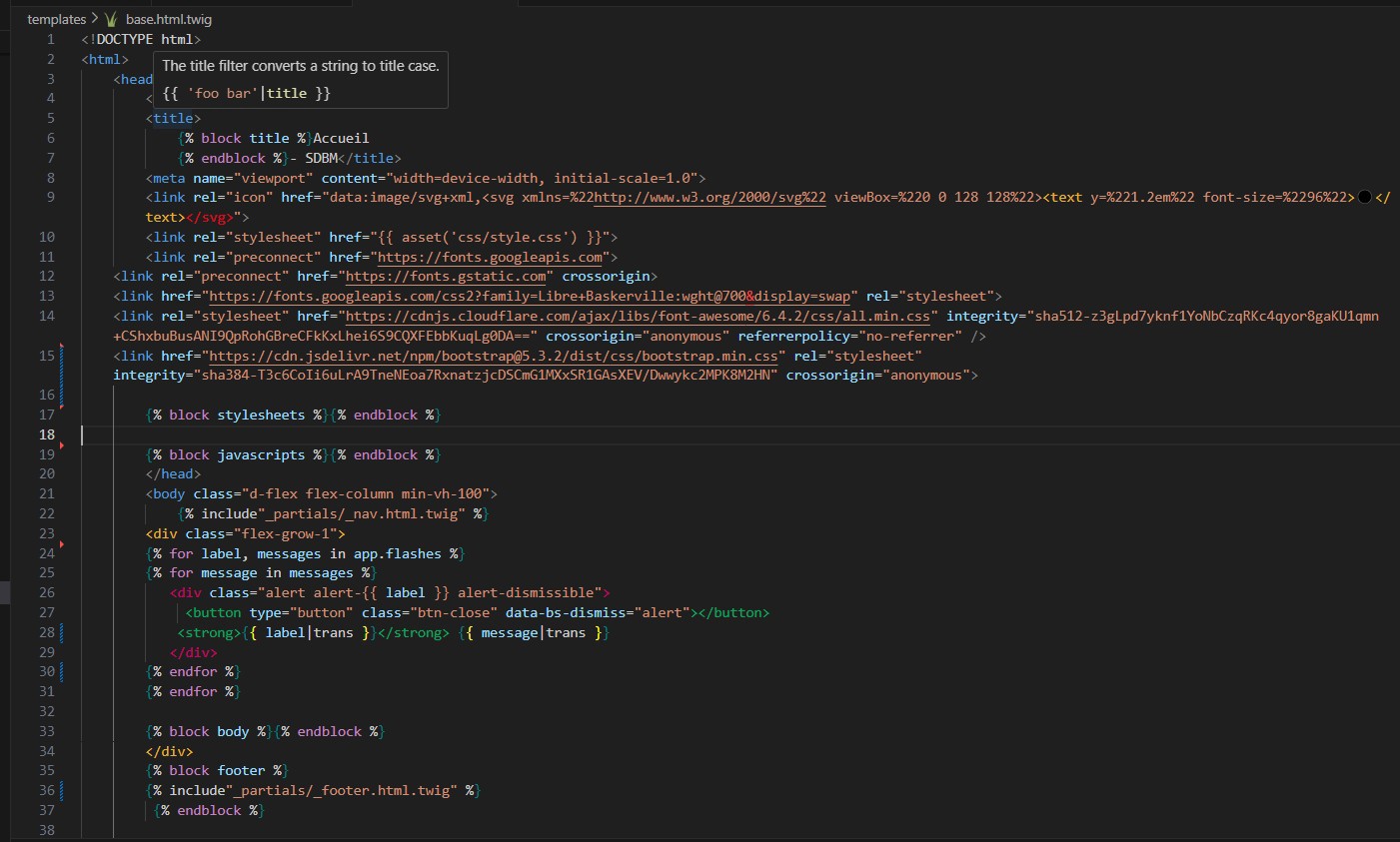
public : contient les fichiers publiquement accessibles (images, CSS, etc .) bin : contient les exécutables(notamment console)

var : cache, logs, etc

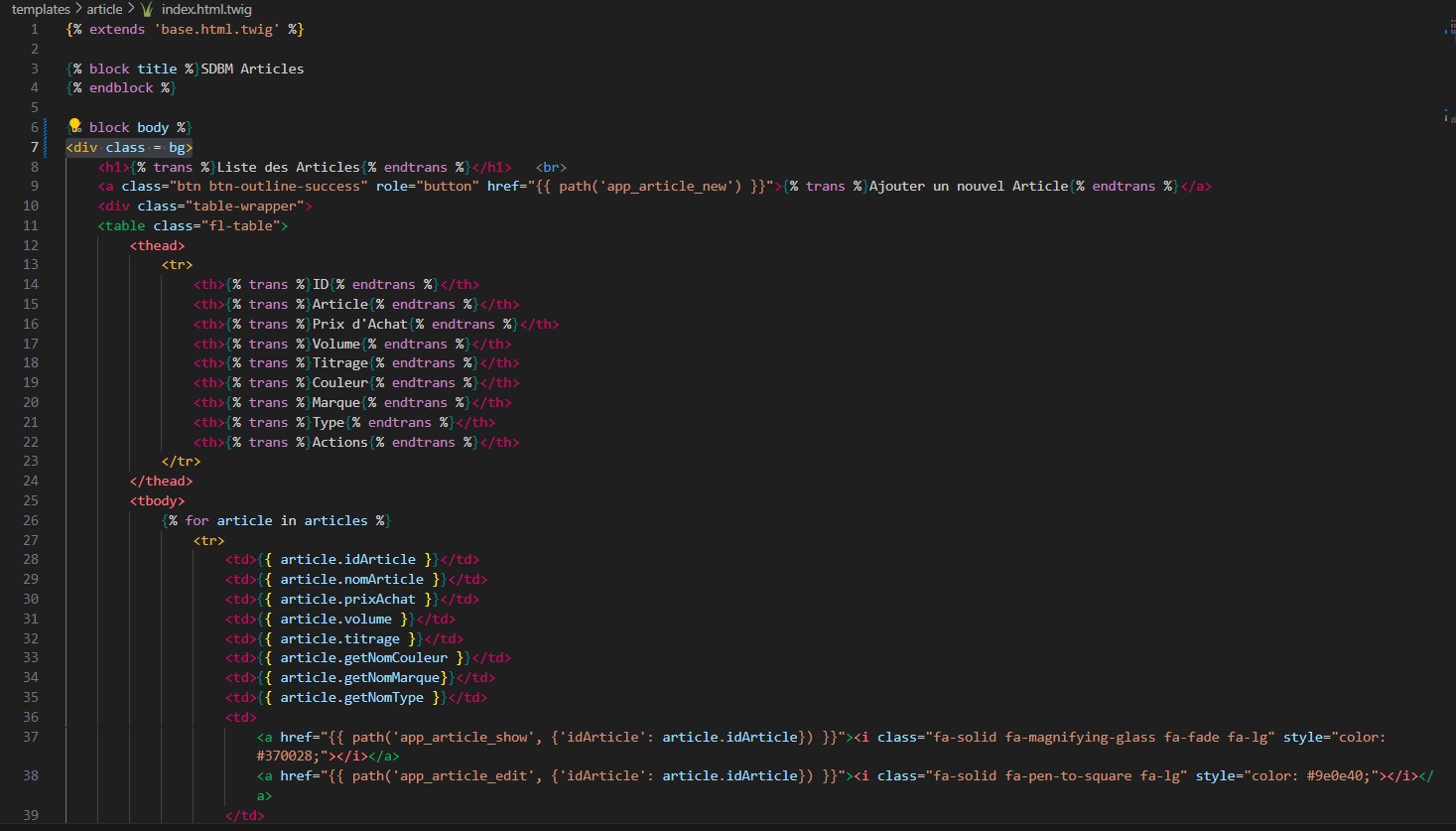
tests : pour les tests automatiques vendor : les modules additionnels translations : messages.en.xlf

*Base.html.twig*

**Le template de base, avec les blocs à remplir :**



# Template appelé avec l’objet articles en argument :

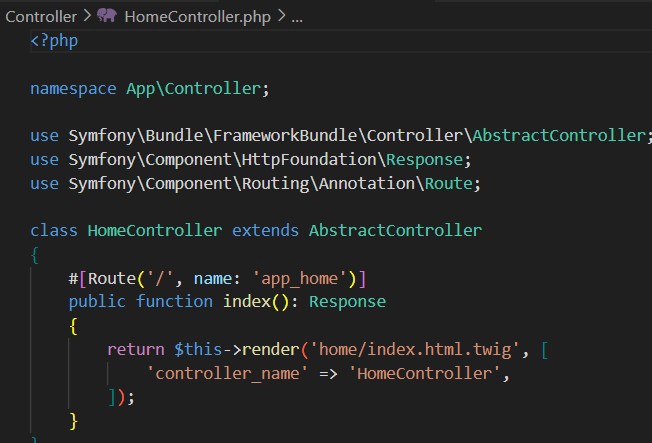




Dans le fichier **.env** , il faut relier le projet à la BDD en mettant les infos correctement afin de se connecter à la BDD MySQL de XAMPP.



## Routage et code simples (Home Controller) :



Le contrôleur est dans le **namespace ‘App\Controller**, ce qui signifie que la classe ‘HomeController’

est definie dans ce namespace.

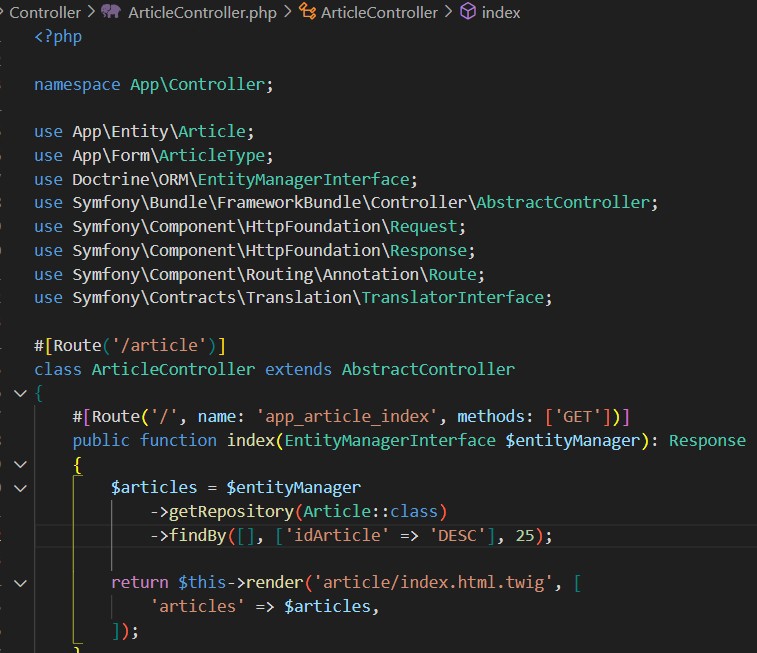
* **use Symfony\Bundle\FrameworkBundle\Controller\AbstractController** : Ce contrôleur hérite de la classe AbstractController fournie par Symfony. Cette classe abstraite offre des fonctionnalités de base pour la création de contrôleurs.
* **use Symfony\Component\HttpFoundation\Response :** Cette classe 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



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 dansl'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.

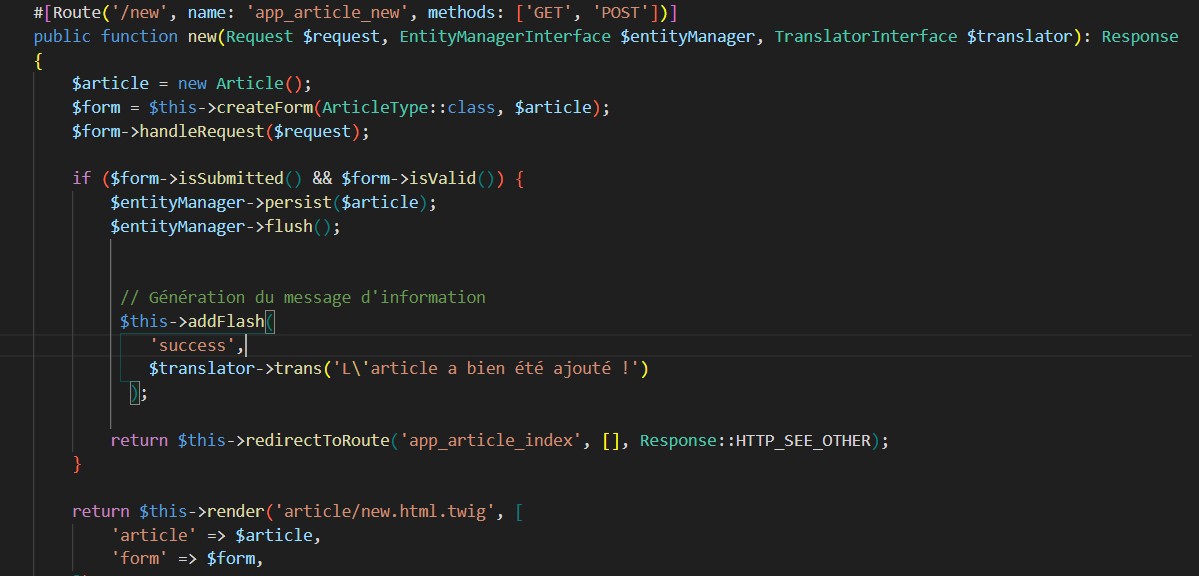
La méthode reçoit un paramètre $entityManager de type EntityManagerInterface, qui sera automatiquement injecté par Symfony grâce à l'injection de dépendance. Cet objet est utilisé pour interagir avec la base de données.

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.

* 1. *DEVELOPPEMENT D’UN CRUD COMPLEXE DE TABLE ARTICLE (en SYMFONY) CREATE (AJOUTER)*



**#[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 :*

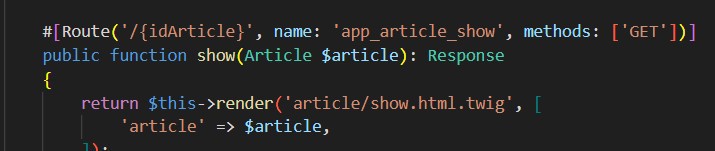
**$article = new Article()**; : Une nouvelle instance de la classe Article est créée. Cela prépare un objet pour stocker les informations d'un nouvel article.

*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)*



**Annotation de routage :**

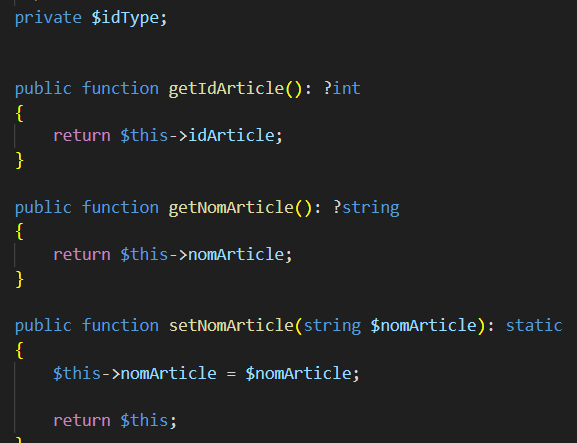
#[Route('/{idArticle}', name: 'app\_article\_show', methods: ['GET'])] : Cette annotation définit une route pour cette méthode. L'URL correspondante pour cette route est '/article/{idArticle}', où

{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 :*

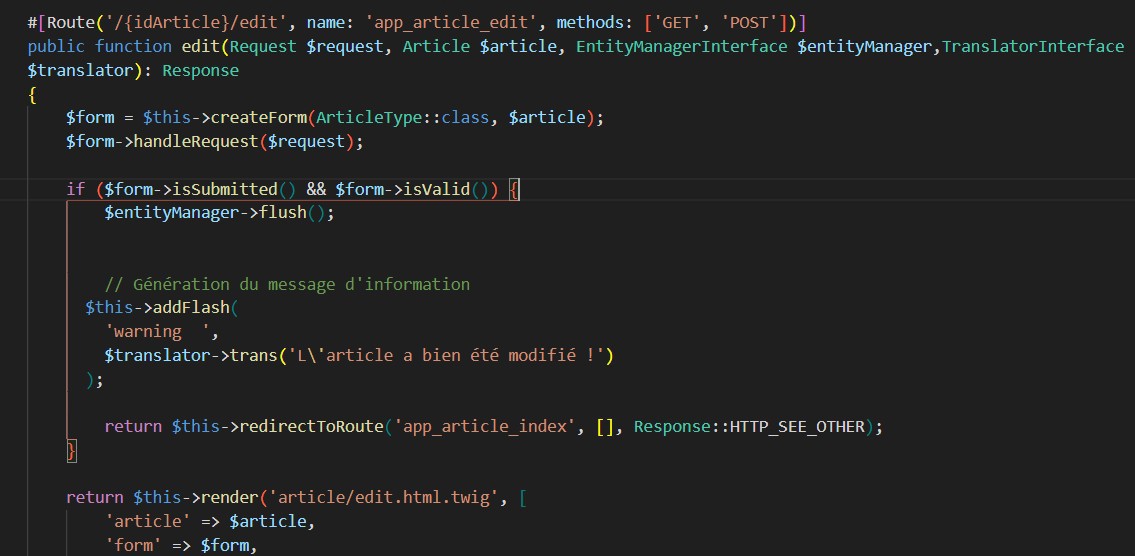
À l'intérieur de la méthode, elle renvoie une réponse HTTP en rendant un modèle Twig en utilisant return $this->render('article/show.html.twig', ['article' => $article]);.

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.



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)*



**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.

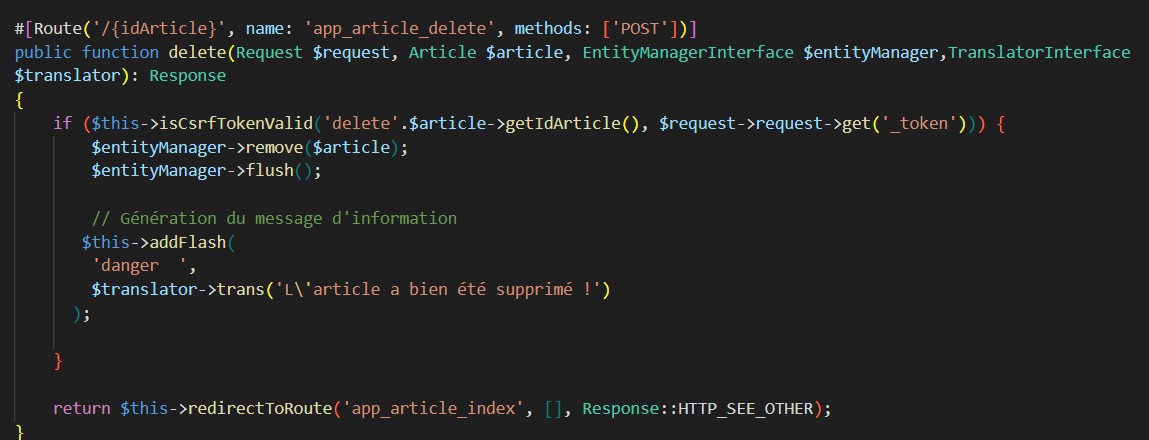
Mise à jour en base de données :

* **$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)



*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.

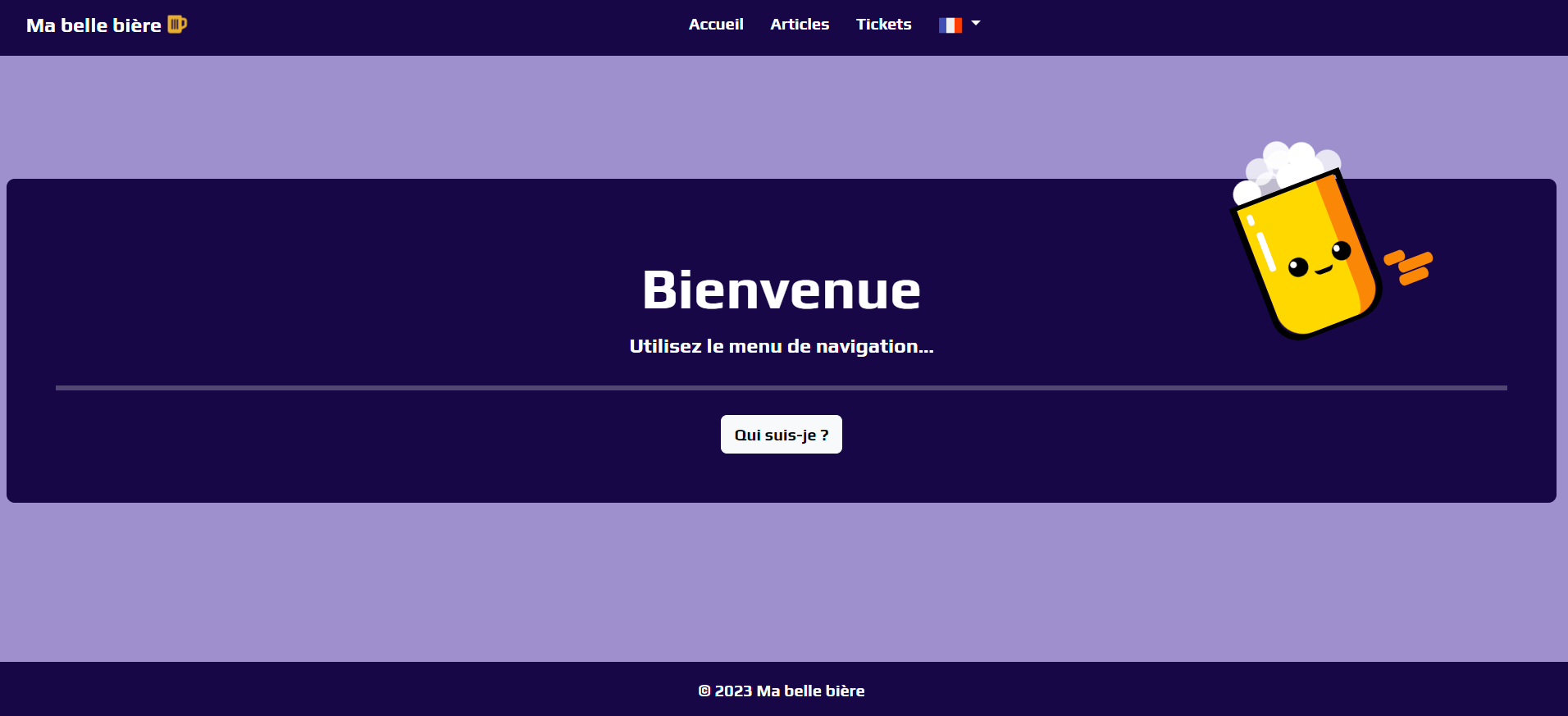
Suppression de l'article en base de données :

**$entityManager->remove($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'.

## 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.



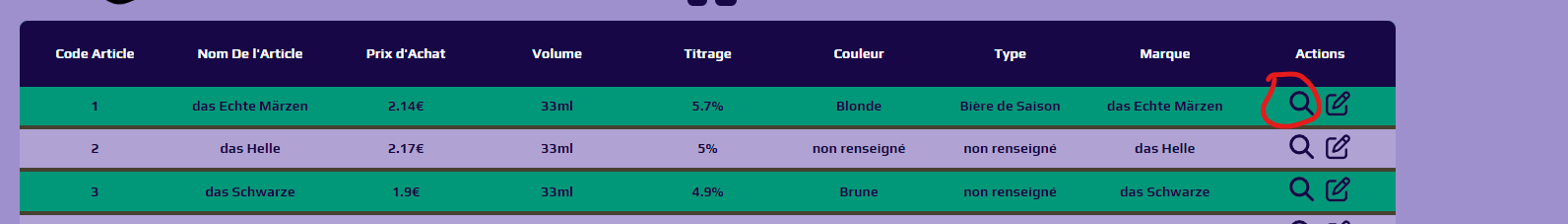
Un message de confirmation s’affichera à chaque fois que l’utilisateur enregistre de l’ajout d’un nouvel article de bière

.

*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 :

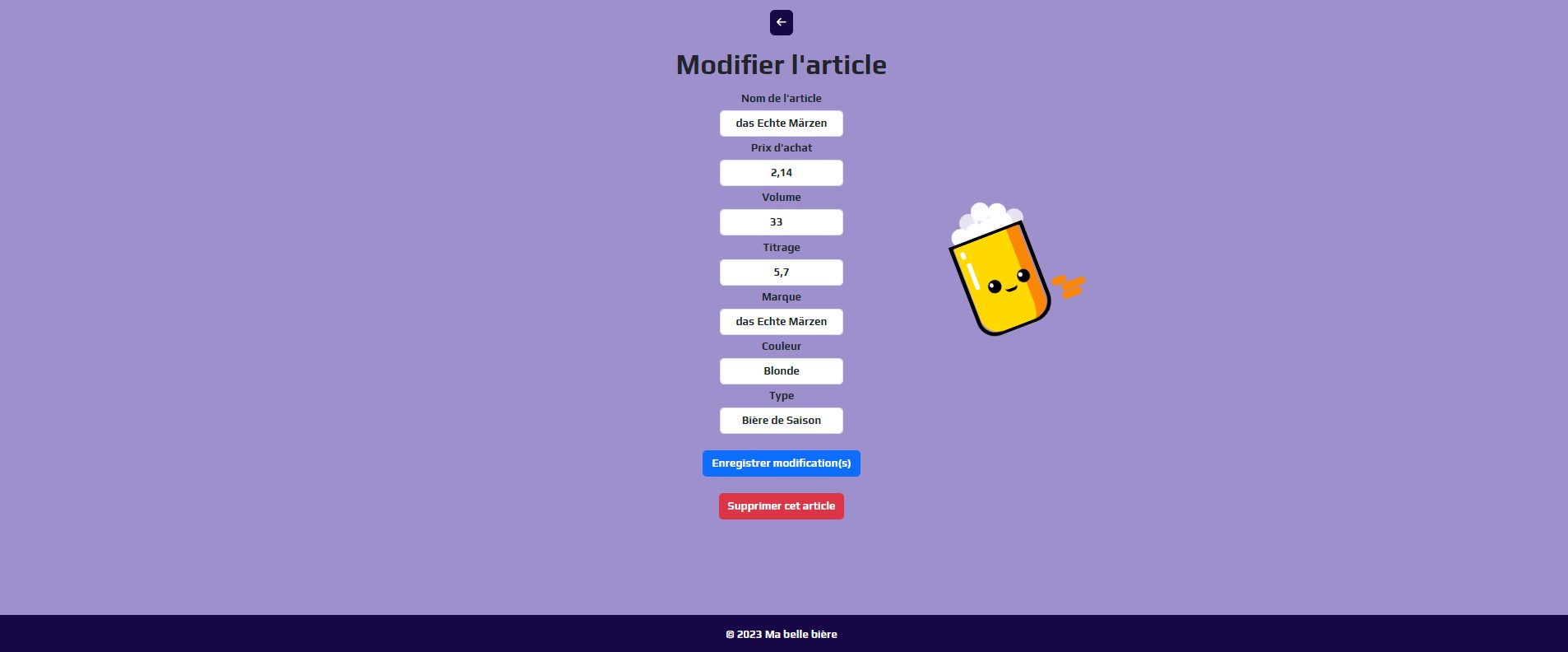
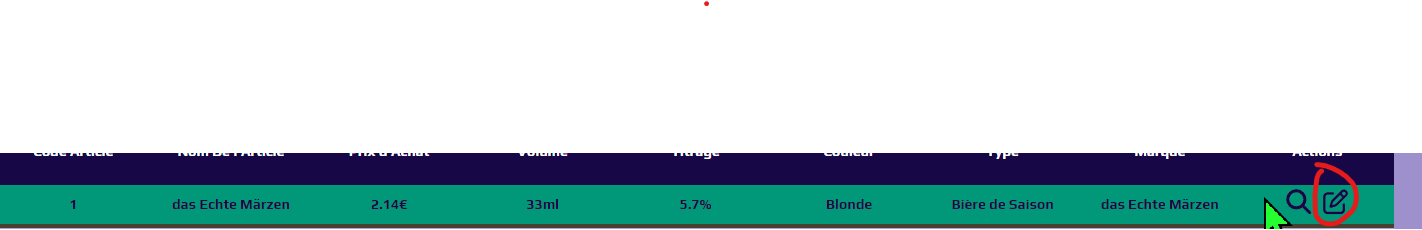




*FONCTIONNEMENT « MODIFIER »*

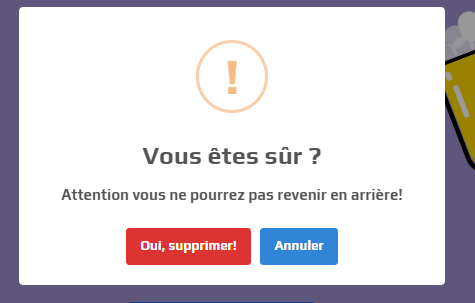
Il faut cliquer sur le bouton pour modifier : un autre page avec un formulaire pour modifier un article de bière, comme ci-dessous :





*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) :



# 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».



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)



**Chez Hostinger**



**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

## 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 commefiables.

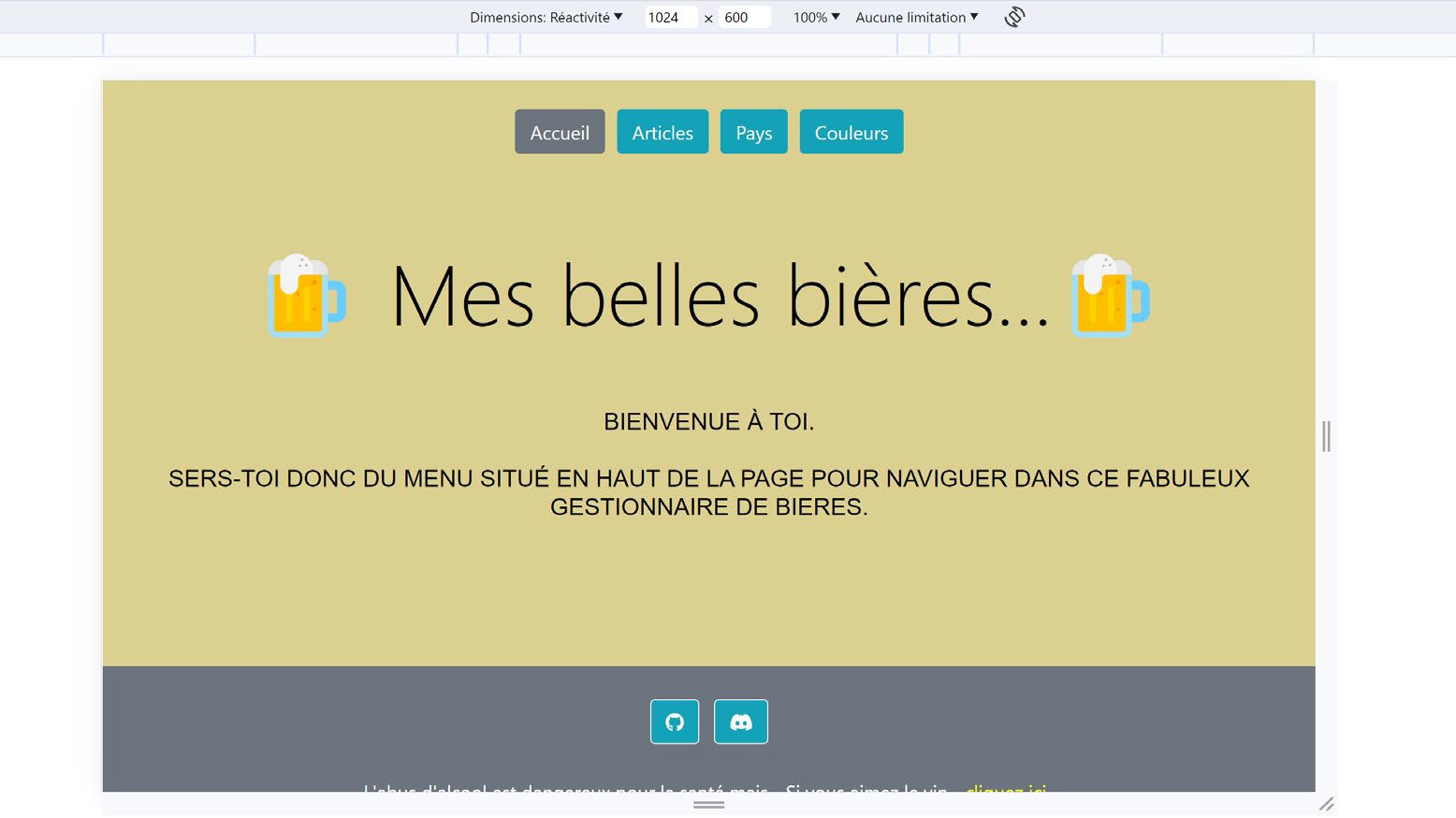
## Comment se protéger des injections SQL ?

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

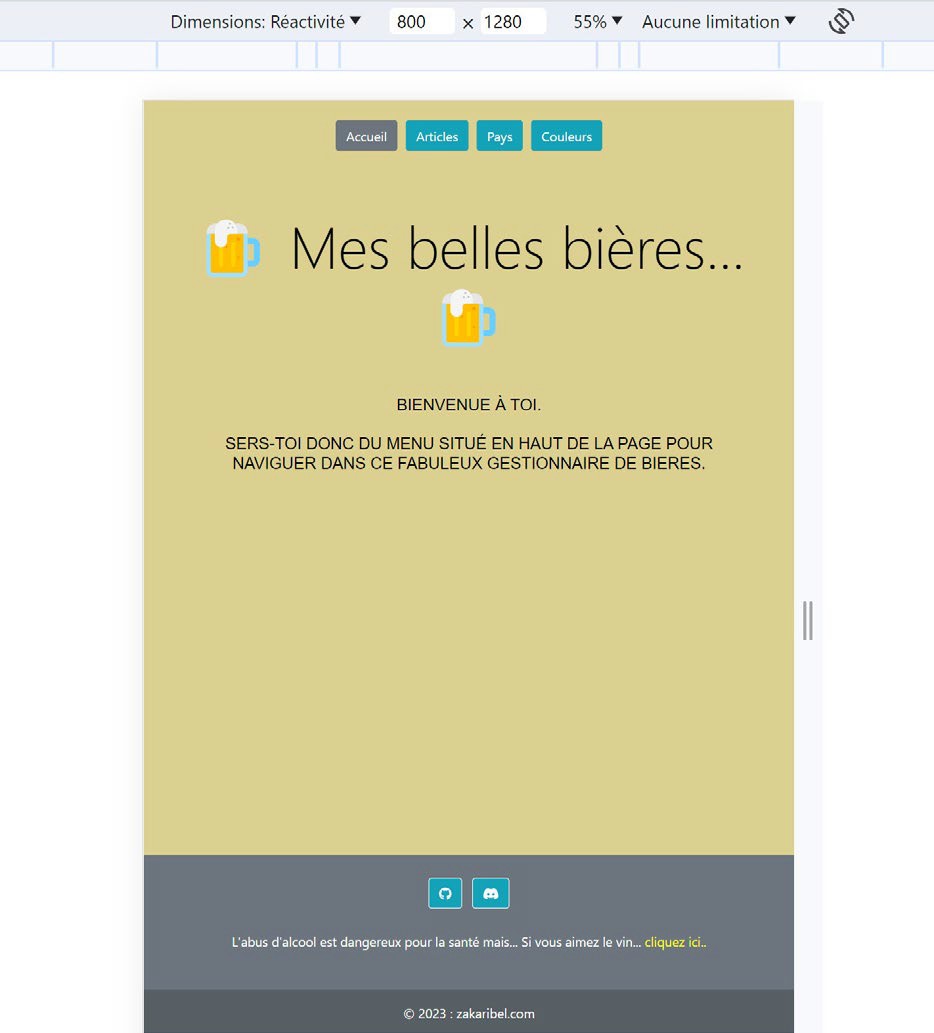
Cette méthode consiste à séparer la requête les valeurs envoyées. Il est possible de ne pas nommer ses clés et d’utiliser des points d’interrogation **« ?** » à la place. Dans ce cas, le tableau de valeurs n’est pas indexé et l’ordre des donnés doit correspondre à l’ordre d’apparition des points d’interrogation.

# Visuel de l’interface sur écrans Responsive Test

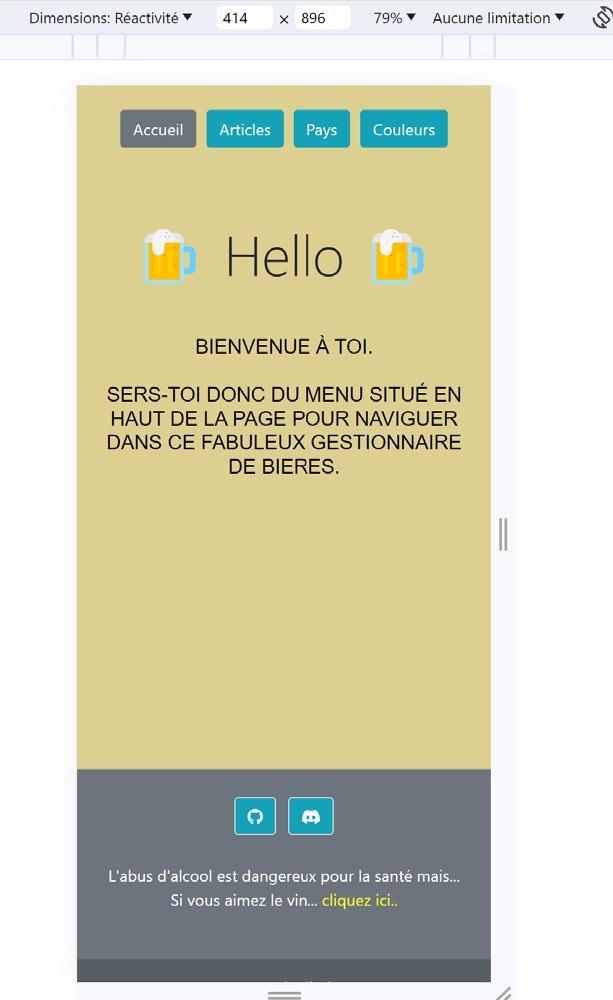
## Ordinateur portable



**Tablette**

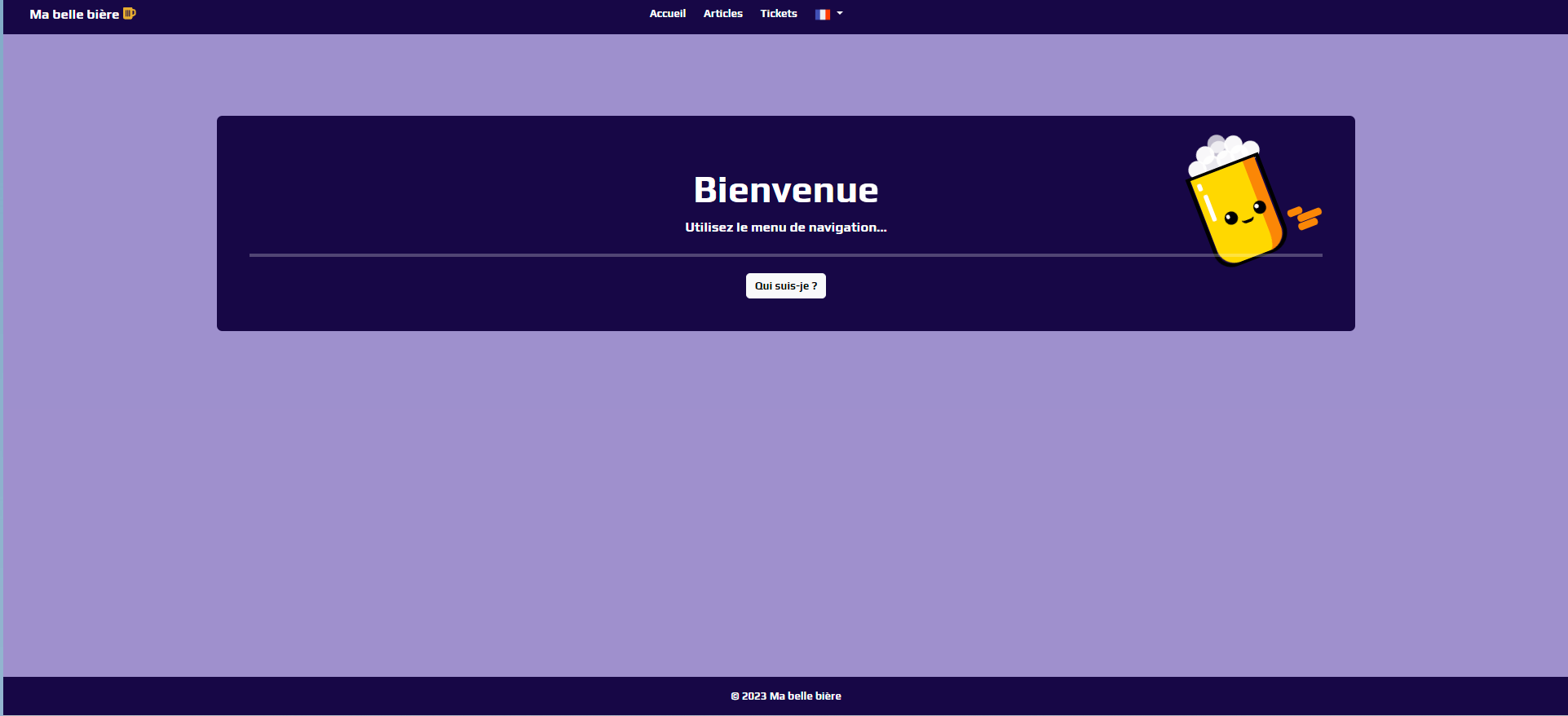


Mobile

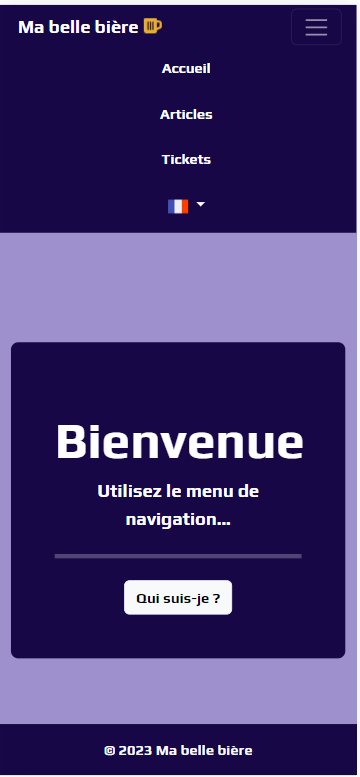


## RESPONSIVE TEST (Projet Symfony)

**Ordinateur portable**



**Mobile Tablette**



1. **LES TESTS PHP MVC**

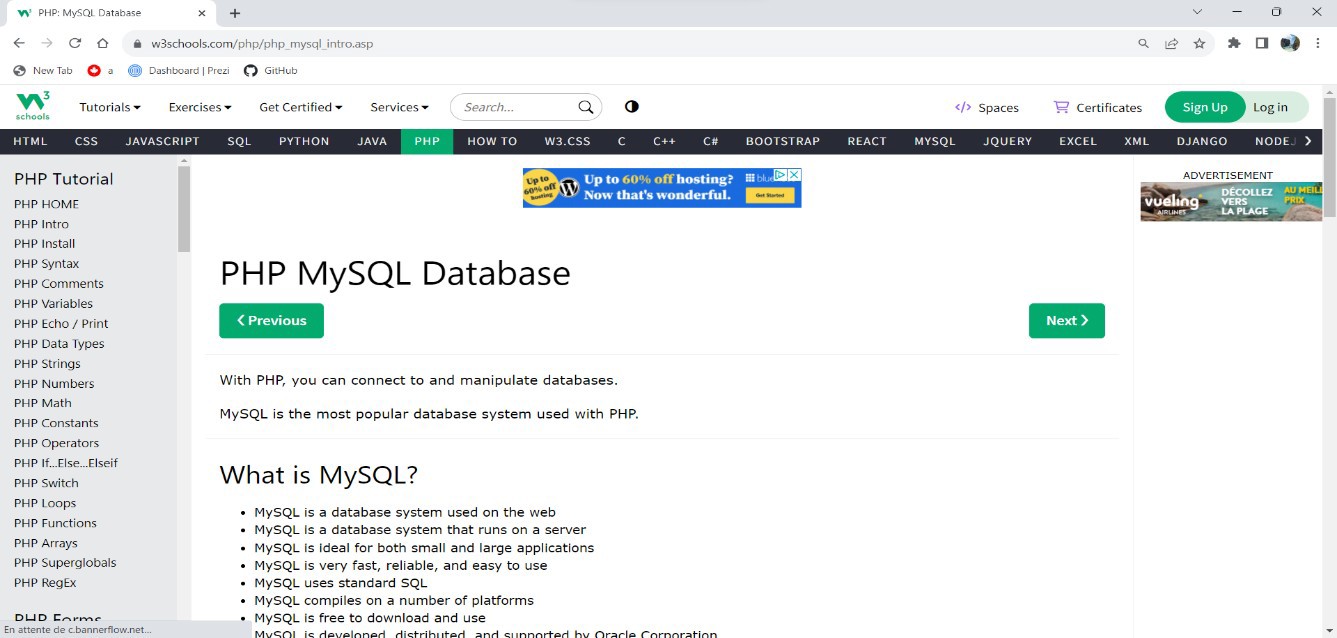
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test | Affichage | Nouveau | Modification | Suppression |
| Accueil | Ok |  |  |  |
| Gestion des Pays | OK | Ok | Ok | OK |
| Gestion des Articles | OK | OK | OK | OK |
| Gestion de Couleurs | OK | OK | OK | OK |

**SYMFONY**

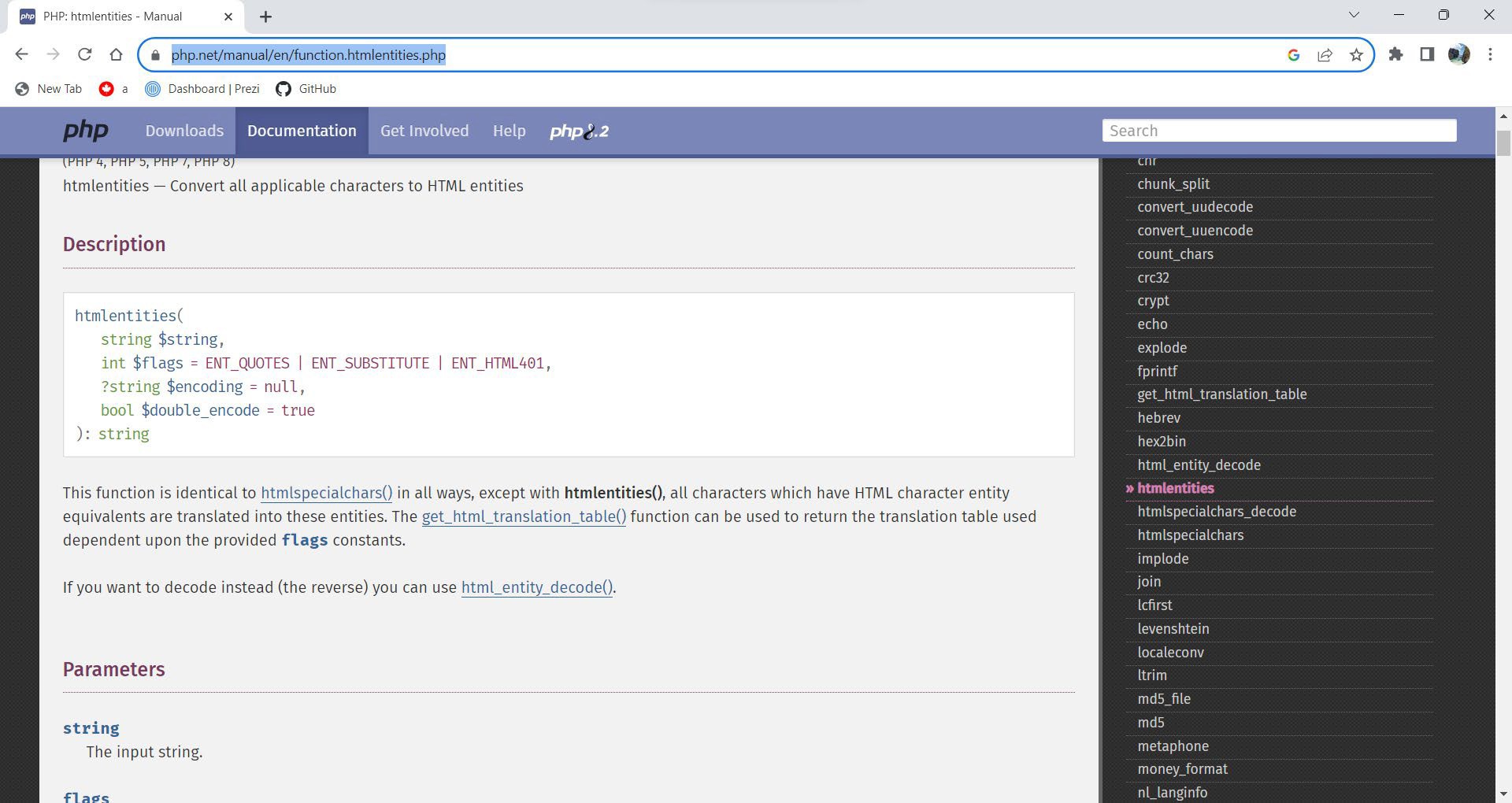
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test | Affichage | Nouveau | Modification | Suppression |
| Accueil | Ok |  |  |  |
| Gestion des Articles | OK | Ok | Ok | OK |
| Gestion des Tickets | OK | OK | OK | OK |
| Type de vente | OK | OK | OK | OK |

## 12 LA VEILLE TECHNIQUE

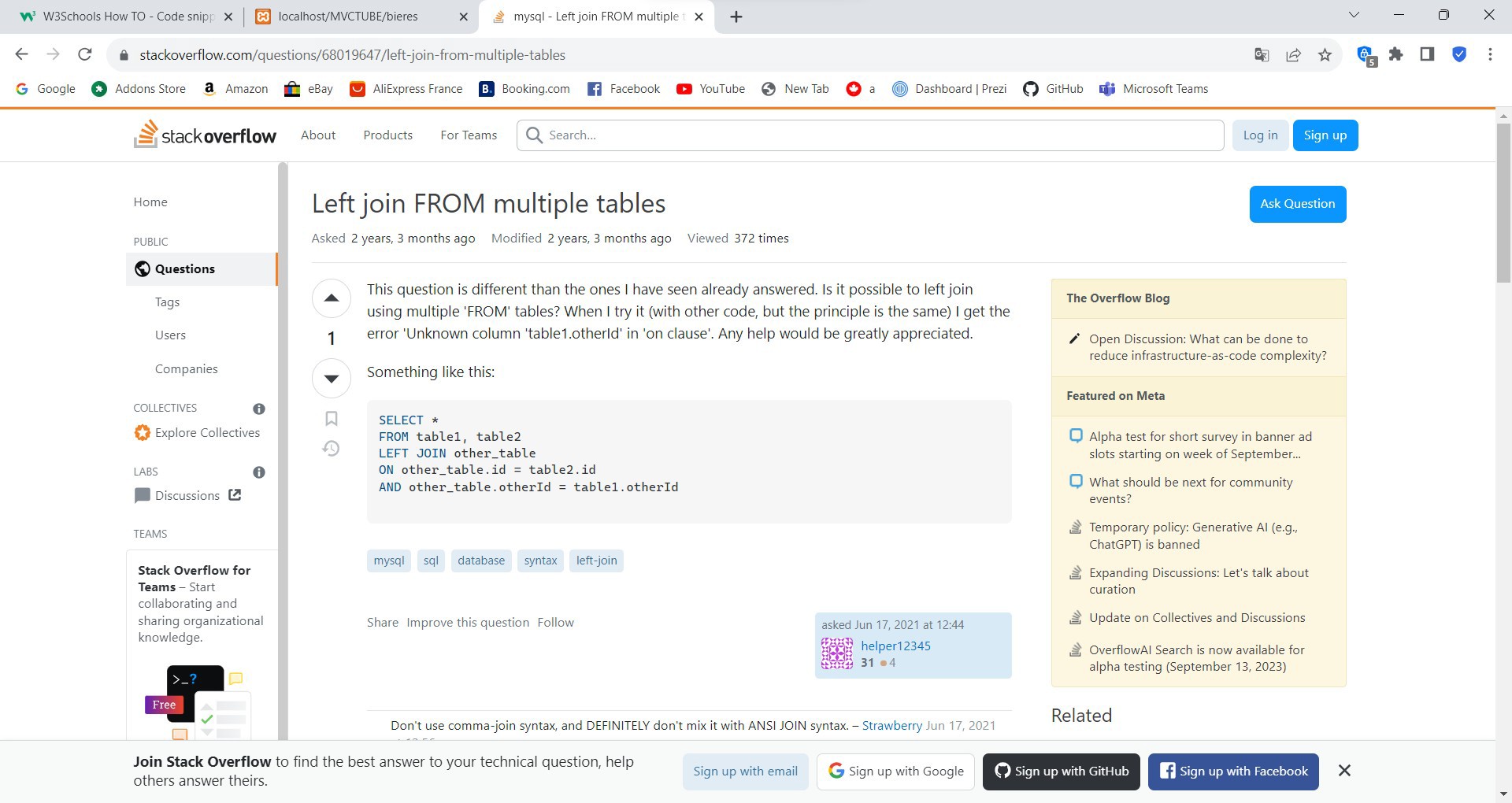
<https://www.w3schools.com/php/php_mysql_intro.asp>



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



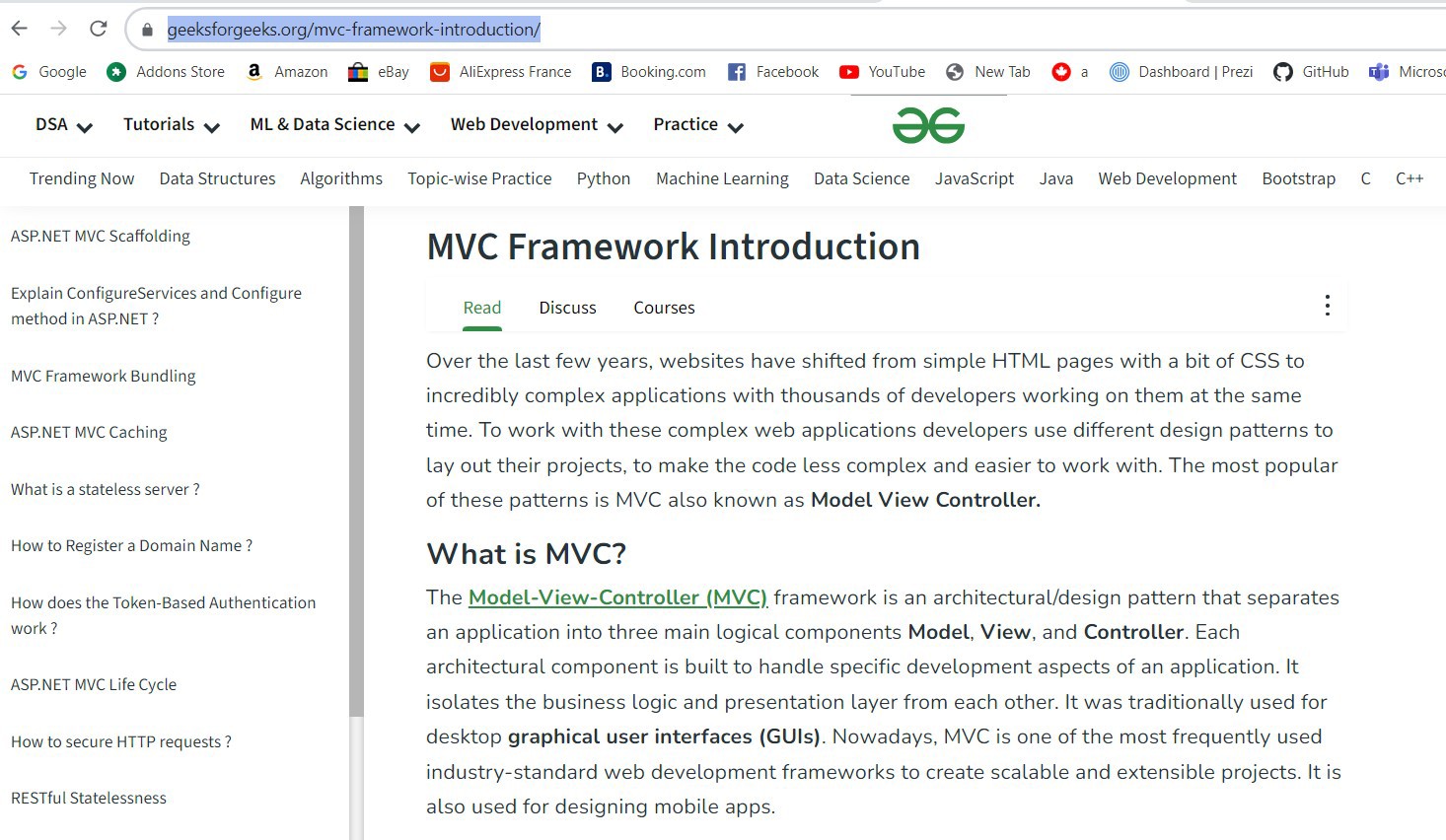
<https://stackoverflow.com/questions/68019647/left-join-from-multiple-tables>



## 13. RECHERCHE SUR DES SITES ANGLOPHONE

[**https://www.geeksforgeeks.org/mvc-framework-**](https://www.geeksforgeeks.org/mvc-framework-introduction/)

[**introduction/**](https://www.geeksforgeeks.org/mvc-framework-introduction/)



**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 unprojet.
  + 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 server.

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.