Documentation Technique

GESTION DES REFERENCES OU SOURCES

3 juin 2024

Pour Chatea9

Créé par : Fatima ABOUDA (stagiaire)

1 BASE DE DONNEES

Faire une base de données à partir d'un fichier word transformer dans un fichier xml ensuite sur Excel pour mieux voir afin de faire la bdd et à la fin mettre toutes les données envoyées dans la bdd qui est à créer

Pour créer une base de données à partir d'un fichier Word, le processus implique plusieurs étapes, notamment la conversion du fichier Word en fichier XML, son importation dans Excel pour une meilleure visualisation, et enfin, l'insertion des données dans la base de données.

1.1 ÉTAPE 1 : CONVERSION DU FICHIER WORD EN XML

Le premier pas consiste à extraire les données du fichier Word et les convertir en un format exploitable comme le XML. Vous pouvez utiliser des outils dédiés ou écrire un script qui parcourt le document Word, extrait les informations nécessaires, et les structure en XML.

1.2 ÉTAPE 2 : IMPORTATION DANS EXCEL

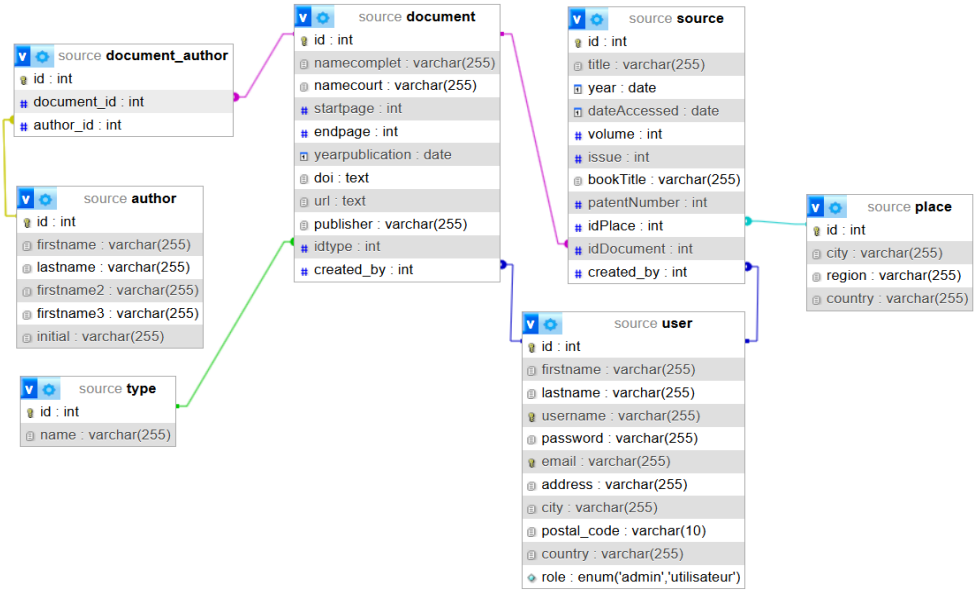
Une fois le fichier XML obtenu, il peut être importé dans Microsoft Excel pour une meilleure visualisation et manipulation des données. Excel offre des fonctionnalités puissantes pour organiser les données sous forme de tableaux, ce qui facilitera la création de la base de données.

Exemple de format XML :

Après la vérification de l'auteur du livre du contenu de toutes les informations

```xml
<?xml version="1.0" encoding="ISO-8859-1"?>
<?xml-stylesheet type="text/css" href="style.css"?>
<carnet>
    <parent>
        <nom id="famille">Adam Bernard</nom>
        <fonction>père</fonction>
        <addresse>3 rue Descartes</addresse>
    </parent>
    <medecin>
        <nom id="professionnel">Borel Annie</nom>
        <fonction>Ophtalmo</fonction>
        <addresse>12 rue de la Fontaine</addresse>
    </medecin>
    <ami>
        <nom id="ami">Desmarais Jacques</nom>
        <addresse>3 rue de la Forge</addresse>
    </ami>
    <parent>
        <nom id="famille">Garcin Paule</nom>
        <fonction>mère</fonction>
        <addresse>16 avenue de la Palace Fabienne</addresse>
    </parent>
    <medecin>
        <nom id="professionnel">Lamard Jean</nom>
        <fonction>Généraliste</fonction>
        <addresse>12 rue de l'Hopital</addresse>
    </medecin>
    <ami>
        <nom id="ami">Rouvert Julien</nom>
        <addresse>34 rue de l'Université</addresse>
    </ami>
</carnet>
```

1.3 ÉTAPE 3 : CREATION DE LA BASE DE DONNEES



Avec les données maintenant dans Excel, concevoir la structure de la base de données. Identifier les entités, les relations entre elles, et déterminez les types de données appropriés pour chaque champ. Créez ensuite les tables nécessaires avec les clés primaires et étrangères appropriées pour assurer l'intégrité des données

- Author : contient des informations sur les auteurs, y compris leur prénom, nom de famille, initiales, etc.
- Document : contient des informations sur les documents, y compris leur titre complet, titre court, nombre de pages, date de publication, DOI, URL, éditeur, auteurs et type de document.
- Document_author : relie les documents aux auteurs.
- Place : contient des informations sur les lieux, y compris la ville, la région et le pays.
- Source : contient des informations sur les sources des documents, y compris le titre, l'année, la date d'accès, le volume, le numéro, le titre du livre, le numéro de brevet, le lieu et le document associé.
- Type : contient des informations sur les types de documents, par exemple "Livre", "Site Web", etc.
- User : Contient des utilisateurs avec des informations personnelles, des rôles et des identifiants uniques.

2 DOSSIER SOURCE

Pour commencer j'ai fait le choix de faire ce projet sous format MVC

3 ANALYSE DES FICHIERS DU PROJET MVC

4 STRUCTURE DU PROJET :

Le projet est organisé selon le pattern MVC (Model-View-Controller) :

- **Contrôler :** Contient les scripts PHP qui gèrent la logique métier et l'interaction avec l'utilisateur.
- **Css :** Contient les styles CSS pour la mise en forme de l'interface.
- **Model :** Contient les classes PHP qui définissent les modèles de données (ex: Author, Document).
- **View :** Contient les fichiers HTML et PHP qui définissent l'interface utilisateur.

4.1 FICHIERS IMPORTANTS :

- **index.php :** Point d'entrée principal du projet, il charge le contrôleur approprié en fonction de l'action demandée.
- **controllerPrincipale.php :** Contrôleur principal qui gère les actions générales (ex: affichage de la page d'accueil).
- **defaut.php :** Contrôleur par défaut si aucune action n'est spécifiée.
- **Fichiers .php dans le contrôle r:* Gèrent les actions spécifiques à chaque fonctionnalité (ex: ajout d'un auteur, suppression d'un document).
- **Fichiers .php dans le model :* Définissent les classes et les méthodes pour manipuler les données (ex: classe Author avec des méthodes pour ajouter, modifier et supprimer des auteurs).
- **Fichiers .php dans le view :* Définissent l'interface utilisateur pour chaque fonctionnalité (ex: vueAuthor.php pour afficher la liste des auteurs).

4.2 FONCTIONNALITES IMPLEMENTEES :

- Ajout, modification et suppression d'auteurs, de documents, de sources , de lieux et de types de documents.
- Affichage des listes d'auteurs, de documents, de lieux et de types de documents.
- Insertion et mise à jour des sources des documents.

5 LE CONTENU DES FICHIERS

5.1 DOSSIER CONTROLEUR

5.1.1 Explication du fichier controllerPrincipale.php :

5.1.1.1 Rôle :

- Contrôleur principal de l'application.
- Dirige les requêtes vers les contrôleurs appropriés en fonction de l'action demandée.

5.1.1.2 Contenu :

1. **Fonction** controllerPrincipale($action)**:**
    - Prend en paramètre l'action à effectuer (ex: "document", "author", etc.).

- Définit un tableau $lesActions associant actions à contrôleurs :
  - "defaut" : Contrôleur par défaut.
  - "document" : Contrôleur pour gérer les documents.
  - "place" : Contrôleur pour gérer les lieux.
  - "source" : Contrôleur pour gérer les sources.
  - "type" : Contrôleur pour gérer les types de documents.
  - "author" : Contrôleur pour gérer les auteurs.
  - Autres actions liées à la modification, l'ajout ou la suppression de données.
- Vérifie si l'action existe dans le tableau.
  - Si oui, retourne le nom du fichier contrôleur correspondant à l'action.
  - Sinon, retourne le contrôleur par défaut ("defaut.php").

5.1.1.3 Fonctionnement :

1. Une requête arrive dans l'application.
2. Le contrôleur principal est appelé avec l'action à effectuer.
3. La fonction controllerPrincipale du contrôleur principal détermine le contrôleur approprié à appeler en fonction de l'action.
4. Le contrôleur approprié est exécuté, ce qui permet de réaliser les traitements nécessaires et de générer la réponse à envoyer au client.

5.1.2 Les autres contrôleurs :

```php
<?php
// Démarrer la session uniquement si elle n'est pas déjà active
if (session_status() == PHP_SESSION_NONE) {
    session_start();
}

// Fonction pour diriger vers le bon contrôleur en fonction de l'action
function controllerPrincipale($action)
{
    $lesActions = array(
        "defaut" => "defaut.php",
        "document" => "document.php",
        "place" => "place.php",
        "source" => "source.php",
        "type" => "type.php",
        "author" => "author.php",
        "updateSource" => "updateSource.php",
        "modifdocument" => "modifdocument.php",
        "ajoutdocument" => "ajoutdocument.php",
        "deleteSource" => "deleteSource.php",
        "insertSource" => "insertSource.php",
        "deletedocument" => "deletedocument.php",
```

```
        "ajoutAuthor" => "ajoutAuthor.php",
        "modifAuthor" => "modifAuthor.php",
        "deleteAuthor" => "deleteAuthor.php",
        "suppPlace" => "suppPlace.php",
        "ajoutPlace" => "ajoutPlace.php",
        "modifPlace" => "modifPlace.php",
        "ajoutType" => "ajoutType.php",
        "modifType" => "modifType.php",
        "deleteType" => "deleteType.php",
        "inscription" => "incUser.php",
        "connexion" => "connexion.php",
        "deconnexion" => "deconnexion.php",
        "profil" => "profil.php",
        "modifierProfil" => "modifierProfil.php",
        "supprimerCompte" => "supprimerProfil.php",
        "modifierMdp" => "modifierMdp.php",
        "voirUtilisateurs" => "voirUtilisateurs.php",
        "modifierutilisateurparadmin" => "modifierutilisateurparadmin.php",
        "afficherInfoUser" => "afficherInfoUser.php"
    );

    // Si l'action existe dans le tableau des actions on retourne le fichier
correspondant
    if (array_key_exists($action, $lesActions)) {
        return $lesActions[$action];
    } else {
        return $lesActions["defaut"];
    }
}
?>
```

5.1.2.1 1. Inclusion des classes du modèle

- include_once "model/classeDocument.php";
- include_once "model/classeType.php";
- include_once "model/classeSource.php";
- include_once "model/classeAuthor.php";
- include_once "model/DataBase.php";

Ces lignes utilisent l'instruction include_once pour inclure les fichiers PHP du répertoire model. Chaque fichier définit probablement une classe liée aux données gérées par le contrôleur. Par exemple, classeDocument.php définit la classe Document, classeAuthor.php définit la classe Auteur, etc. L'instruction include_once garantit que le fichier n'est inclus qu'une seule fois, même s'il est référencé plusieurs fois dans le script.

5.1.2.2 2. Récupération des données (potentiellement)

- // Récupération des données (commenté)
- // include "view/vueDocument.php"; (commenté)

Exemple :

```
<?php

// Inclusion du modèle
```

```php
include_once "model/classeDocument.php";
include_once "model/classeType.php";
include_once "model/classeSource.php";
include_once "model/classeAuthor.php";
include_once "model/DataBase.php";

// Récupération des données
include "view/vueDocument.php";
?>
```

5.2 DOSSIER MODELE

5.2.1 Structure de la classe User :

La classe User gère les informations et les opérations liées aux utilisateurs d'une base de données. Elle contient des attributs privés pour les informations de l'utilisateur, des méthodes pour récupérer et modifier ces informations, et plusieurs méthodes statiques pour interagir avec la base de données.

6 Attributs

- **$id**
- **$firstname**
- **$lastname**
- **$username**
- **$password**
- **$email**
- **$address**
- **$city**
- **$postalCode**
- **$country**
- **$role**

7 Méthodes

- **Constructeur** : Initialise les attributs de l'utilisateur.
- **Getters** : Récupèrent les valeurs des attributs.
- **Setters** : Modifient les valeurs des attributs (par exemple, setRole).

8 Méthodes statiques

- getUsernameById($userId) : Récupère le nom d'utilisateur à partir de l'ID.
- getIdByUsername($username) : Récupère l'ID à partir du nom d'utilisateur.
- isAdmin($id) : Vérifie si l'utilisateur est administrateur.
- createUser(...) : Crée un nouvel utilisateur avec des validations et hachage du mot de passe.
- authenticate($username, $password) : Authentifie un utilisateur avec vérification du mot de passe.

- getUser($username) : Récupère les informations de l'utilisateur par nom d'utilisateur.
- updateUser(...) : Met à jour les informations de l'utilisateur.
- updateRole($username, $role) : Met à jour le rôle de l'utilisateur.
- getAllUsers() : Récupère tous les utilisateurs.
- deleteUser($id) : Supprime un utilisateur par ID.
- isUsernameAvailable($newUsername) : Vérifie si un nom d'utilisateur est disponible.
- updateUsername($oldUsername, $username) : Met à jour le nom d'utilisateur.
- updatePassword($username, $password) : Met à jour le mot de passe.
- getUserById($id) : Récupère un utilisateur par ID.
- isUserInDocument($id) : Vérifie si un utilisateur est présent dans un document.
- isUserInSource($id) : Vérifie si un utilisateur est présent dans une source.
- updateUserWithoutRole(...) : Met à jour les informations de l'utilisateur sans modifier le rôle.

```
    public static function getUsernameById($userId) {
        $database = new Database();
        $db = $database->getConnection();
        $sql = "SELECT username FROM user WHERE id = ?";
        $stmt = $db->prepare($sql);
        $stmt->bindParam(1, $userId);
        $stmt->execute();
        $user = $stmt->fetch();
        return $user ? $user['username'] : null;
    }

    public static function getIdByUsername($username) {
        $database = new Database();
        $db = $database->getConnection();
        $sql = "SELECT id FROM user WHERE username = ?";
        $stmt = $db->prepare($sql);
        $stmt->bindParam(1, $username);
        $stmt->execute();
        $user = $stmt->fetch();
        return $user ? $user['id'] : null;
    }

    public static function isAdmin($id) {
        $database = new Database();
        $db = $database->getConnection();
        $sql = "SELECT role FROM user WHERE id = ?";
        $stmt = $db->prepare($sql);
        $stmt->bindParam(1, $id);
        $stmt->execute();
        $user = $stmt->fetch();
        return $user && $user['role'] === 'admin';
    }
    public static function createUser($firstname, $lastname, $username,
$password, $email, $address, $city, $postalCode, $country) {
        try {

            if (empty($firstname) || empty($lastname) || empty($username) ||
```

```
empty($password) || empty($email) || empty($address) || empty($city) ||
empty($postalCode) || empty($country)) {
                throw new Exception("Tous les champs sont requis.");
            }
            if (!filter_var($email, FILTER_VALIDATE_EMAIL)) {
                throw new Exception("L'email n'est pas valide.");
            }
            $database = new Database();
            $connection = $database->getConnection();
            $hashedPassword = password_hash($password, PASSWORD_DEFAULT);

            $query = "INSERT INTO user (firstname, lastname, username,
password, email, address, city, postal_code, country) VALUES (?, ?, ?, ?, ?,
?, ?, ?, ?)";
            $statement = $connection->prepare($query);
            $statement->bindParam(1, $firstname);
            $statement->bindParam(2, $lastname);
            $statement->bindParam(3, $username);
            $statement->bindParam(4, $hashedPassword);
            $statement->bindParam(5, $email);
            $statement->bindParam(6, $address);
            $statement->bindParam(7, $city);
            $statement->bindParam(8, $postalCode);
            $statement->bindParam(9, $country);
            $statement->execute();
            $connection = null;
            return true;
        } catch (Exception $e) {
            echo "Erreur : " . $e->getMessage();
            return false;
        }
    }

    public static function authenticate($username, $password) {
        try {
            $database = new Database();
            $connection = $database->getConnection();

            $query = "SELECT * FROM user WHERE username = ?";
            $statement = $connection->prepare($query);
            $statement->bindParam(1, $username);
            $statement->execute();

            $user = $statement->fetch(PDO::FETCH_ASSOC);
            if ($user && password_verify($password, $user['password'])) {
                $_SESSION['user'] = $user;
                $_SESSION['authenticated'] = true;
                $_SESSION['username'] = $user['username'];
                $_SESSION['firstname'] = $user['firstname'];
                $_SESSION['lastname'] = $user['lastname'];
                $_SESSION['password'] = $user['password'];
                $_SESSION['email'] = $user['email'];
                $_SESSION['address'] = $user['address'];
                $_SESSION['city'] = $user['city'];
                $_SESSION['postal_code'] = $user['postal_code'];
                $_SESSION['country'] = $user['country'];
                return true;
            } else {
                                return false;
            }
```

```
        } catch (PDOException $e) {
            echo "Erreur : " . $e->getMessage();
            return false;
        }
    }
    public static function getUser($username) {
        try {
            $database = new Database();
            $connection = $database->getConnection();

            $query = "SELECT * FROM user WHERE username = ?";
            $statement = $connection->prepare($query);
            $statement->bindParam(1, $username);
            $statement->execute();

            $user = $statement->fetch(PDO::FETCH_ASSOC);

            return $user;
        } catch (PDOException $e) {
            echo "Erreur : " . $e->getMessage();
            return false;
        }
    }

    public static function updateUser($id, $username, $firstname, $lastname,
$email, $address, $city, $postal_code, $country, $role){
        try {
            $database = new Database();
            $connection = $database->getConnection();
            $query = "UPDATE user SET username = ?, firstname = ?, lastname =
?, email = ?, address = ?, city = ?, postal_code = ?, country = ?, role = ?
WHERE id = ?";
            $statement = $connection->prepare($query);
            $statement->bindParam(1, $username);
            $statement->bindParam(2, $firstname);
            $statement->bindParam(3, $lastname);
            $statement->bindParam(4, $email);
            $statement->bindParam(5, $address);
            $statement->bindParam(6, $city);
            $statement->bindParam(7, $postal_code);
            $statement->bindParam(8, $country);
            $statement->bindParam(9, $role);
            $statement->bindParam(10, $id);
            $statement->execute();
            $connection = null;

            return true;
        } catch (PDOException $e) {
            echo "Erreur : " . $e->getMessage();
            return false;
        }
    }
    public static function updateRole($username, $role) {
            try {
                $database = new Database();
                $connection = $database->getConnection();
                $query = "UPDATE user SET role = ? WHERE username = ?";
                $statement = $connection->prepare($query);
                $statement->bindParam(1, $role);
                $statement->bindParam(2, $username);
```

```
            $statement->execute();
            $connection = null;
            return true;
        } catch (PDOException $e) {
            echo "Erreur : " . $e->getMessage();
            return false;
        }
    }
    public static function getAllUsers(){
        try {
            $database = new Database();
            $connection = $database->getConnection();

            $query = "SELECT * FROM user";
            $statement = $connection->prepare($query);
            $statement->execute();

            $users = $statement->fetchAll(PDO::FETCH_ASSOC);

            return $users;
        } catch (PDOException $e) {
            echo "Erreur : " . $e->getMessage();
            return false;
        }
    }
    public static function deleteUser($id) {
        try {
            $database = new Database();
            $connection = $database->getConnection();
            $query = "DELETE FROM user WHERE id = ?";
            $statement = $connection->prepare($query);
            $statement->bindParam(1, $id);
            $statement->execute();
            $connection = null;
            return true;
        } catch (PDOException $e) {
            echo "Erreur : " . $e->getMessage();
            return false;
        }
    }

    public static function isUsernameAvailable($newUsername){
        try {
            $database = new Database();
            $connection = $database->getConnection();
            $query = "SELECT * FROM user WHERE username = ?";
            $statement = $connection->prepare($query);
            $statement->bindParam(1, $newUsername);
            $statement->execute();
            $user = $statement->fetch(PDO::FETCH_ASSOC);
            if ($user) {
                return false;
            } else {
                return true;
            }
        } catch (PDOException $e) {
            echo "Erreur : " . $e->getMessage();
            return false;
        }
        }
```

```
    public static function updateUsername($oldUsername, $username) {
    try {
        $database = new Database();
        $connection = $database->getConnection();
        $query = "UPDATE user SET username = ? WHERE username = ?";
        $statement = $connection->prepare($query);
        $statement->bindParam(1, $username);
        $statement->bindParam(2, $oldUsername);
        $statement->execute();
        $connection = null;

        return true;
    } catch (PDOException $e) {
        echo "Erreur : " . $e->getMessage();
        return false;
    }
    }
    public static function updatePassword($username, $password) {
    try {
        $database = new Database();
        $connection = $database->getConnection();
        $query = "UPDATE user SET password = ? WHERE username = ?";
        $statement = $connection->prepare($query);
        $statement->bindParam(1, $password);
        $statement->bindParam(2, $username);
        $statement->execute();
        $connection = null;
        return true;
    } catch (PDOException $e) {
        echo "Erreur : " . $e->getMessage();
        return false;
    }

}

public static function getUserById($id) {
    try {
        $database = new Database();
        $connection = $database->getConnection();
        $query = "SELECT * FROM user WHERE id = ?";
        $statement = $connection->prepare($query);
        $statement->bindParam(1, $id);
        $statement->execute();
        $user = $statement->fetch(PDO::FETCH_ASSOC);
        $connection = null;
        return $user;
    } catch (PDOException $e) {
        echo "Erreur : " . $e->getMessage();
        return false;
    }
}

public static function isUserInDocument($id) {
    try {
        $database = new Database();
        $connection = $database->getConnection();
        $query = "SELECT * FROM document WHERE id = ?";
        $statement = $connection->prepare($query);
        $statement->bindParam(1, $id);
```

```
            $statement->execute();
            $document = $statement->fetch(PDO::FETCH_ASSOC);
            $connection = null;
            return $document;
        } catch (PDOException $e) {
            echo "Erreur : " . $e->getMessage();
            return false;
        }
    }

    public static function isUserInSource($id) {
        try {
            $database = new Database();
            $connection = $database->getConnection();
            $query = "SELECT * FROM source WHERE id = ?";
            $statement = $connection->prepare($query);
            $statement->bindParam(1, $id);
            $statement->execute();
            $source = $statement->fetch(PDO::FETCH_ASSOC);
            $connection = null;
            return $source;
        } catch (PDOException $e) {
            echo "Erreur : " . $e->getMessage();
            return false;
        }
    }

    public static function updateUserWithoutRole($newUsername, $firstname,
$lastname, $email, $address, $city, $postal_code, $country)
    {
        try {
            $database = new Database();
            $connection = $database->getConnection();
            $query = "UPDATE user SET username = ?, firstname = ?, lastname =
?, email = ?, address = ?, city = ?, postal_code = ?, country = ? WHERE
username = ?";
            $statement = $connection->prepare($query);
            $statement->bindParam(1, $newUsername);
            $statement->bindParam(2, $firstname);
            $statement->bindParam(3, $lastname);
            $statement->bindParam(4, $email);
            $statement->bindParam(5, $address);
            $statement->bindParam(6, $city);
            $statement->bindParam(7, $postal_code);
            $statement->bindParam(8, $country);
            $statement->bindParam(9, $newUsername);
            $statement->execute();
            $connection = null;

            return true;
        } catch (PDOException $e) {
            echo "Erreur : " . $e->getMessage();
            return false;
        }
    }
}
```

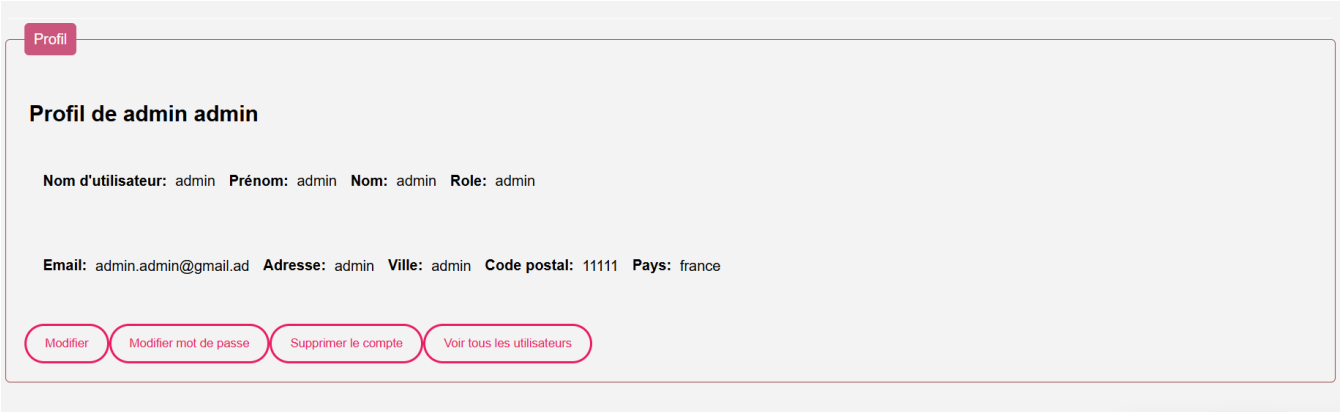
8.1.1.1 Page de connexion

```php
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Connexion</title>
    <style>
        .error {
            color: red;
        }
    </style>
</head>
<body>
    <?php

        if(isset($_POST['login'])){
            $username = $_POST['username'];
            $password = $_POST['password'];

            // Vérifiez les informations de connexion
            if(User::authenticate($username, $password)) {

                echo " Connexion réussie";

            } else {

                echo "Nom d'utilisateur ou mot de passe incorrect";
            }
        }
    ?>

    <h2>Connexion</h2>
    <form method="post" action="">

        <label for="username">Nom d'utilisateur:</label>
        <input type="text" id="username"
name="username">
        <span id="usernameError"
class="error"></span>

        <label for="password">Mot de
passe:</label>
        <input type="password"
id="password" name="password">
        <span id="passwordError"
class="error"></span>

        <input type="submit" name="login"
value="Se connecter">

    </form>
</body>
</html>
```

La page de connexion permet aux utilisateurs de s'authentifier en utilisant leur nom d'utilisateur et leur mot de passe. Lorsque le formulaire est soumis via la méthode POST, les informations d'identification sont vérifiées à l'aide de la méthode `User::authenticate($username, $password)`. Si l'authentification réussit, un message de succès est affiché. Sinon, un message d'erreur indiquant que le nom d'utilisateur ou le mot de passe

est incorrect est montré. Le formulaire est stylisé avec du CSS intégré, et les erreurs sont affichées en rouge grâce à la classe `.error`. Cette page de connexion est une composante essentielle pour sécuriser l'accès aux fonctionnalités réservées aux utilisateurs authentifiés sur le site.

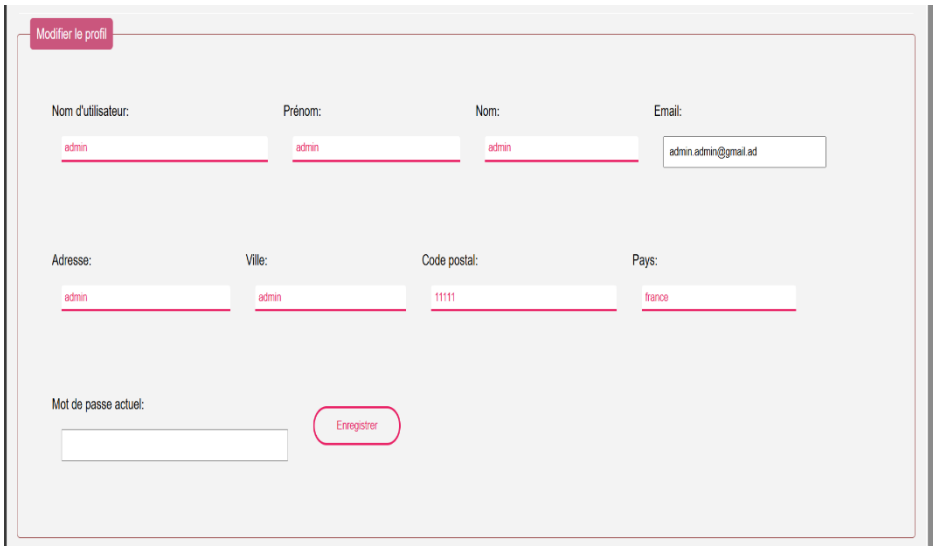
8.1.1.2 Affichage profil

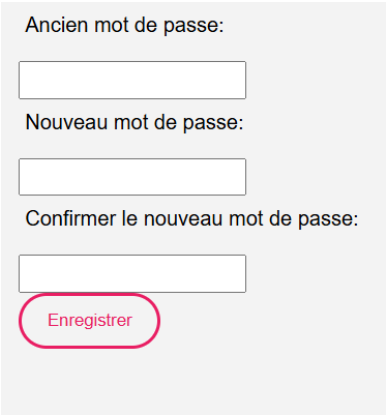


La page "Informations sur l'utilisateur", qui affiche les informations d'un utilisateur connecté en utilisant PHP. Après avoir vérifié la connexion de l'utilisateur via $_SESSION['username'], elle récupère l'ID de l'utilisateur à l'aide de User::getIdByUsername($username) et ses informations complètes avec User::getUserById($user_id). Si les données de l'utilisateur sont trouvées, elles sont affichées dans un tableau HTML. En cas d'utilisateur non trouvé, un message d'erreur s'affiche. Si l'utilisateur n'est pas connecté, des liens vers les pages de connexion et d'inscription sont proposés. La page inclut une référence à une feuille de style externe style.css pour le design. La sécurité est assurée en échappant correctement les données et en utilisant des sessions sécurisées.

8.1.1.3 Modification du profil

la modification du profil utilisateur vérifie si l'utilisateur est connecté et authentifié. Ensuite, il récupère les informations de l'utilisateur à partir de la session. Lorsque le formulaire de modification est soumis, il vérifie d'abord si le mot de passe actuel est correct en utilisant la fonction `password_verify()`. Si le mot de passe est correct, il met à jour les informations du profil telles que le nom, le prénom, l'adresse e-mail, etc. Il vérifie également si le nom d'utilisateur est modifié et s'il est disponible. Si tout se passe bien, il affiche un message de succès, sinon, il affiche un message d'erreur approprié.



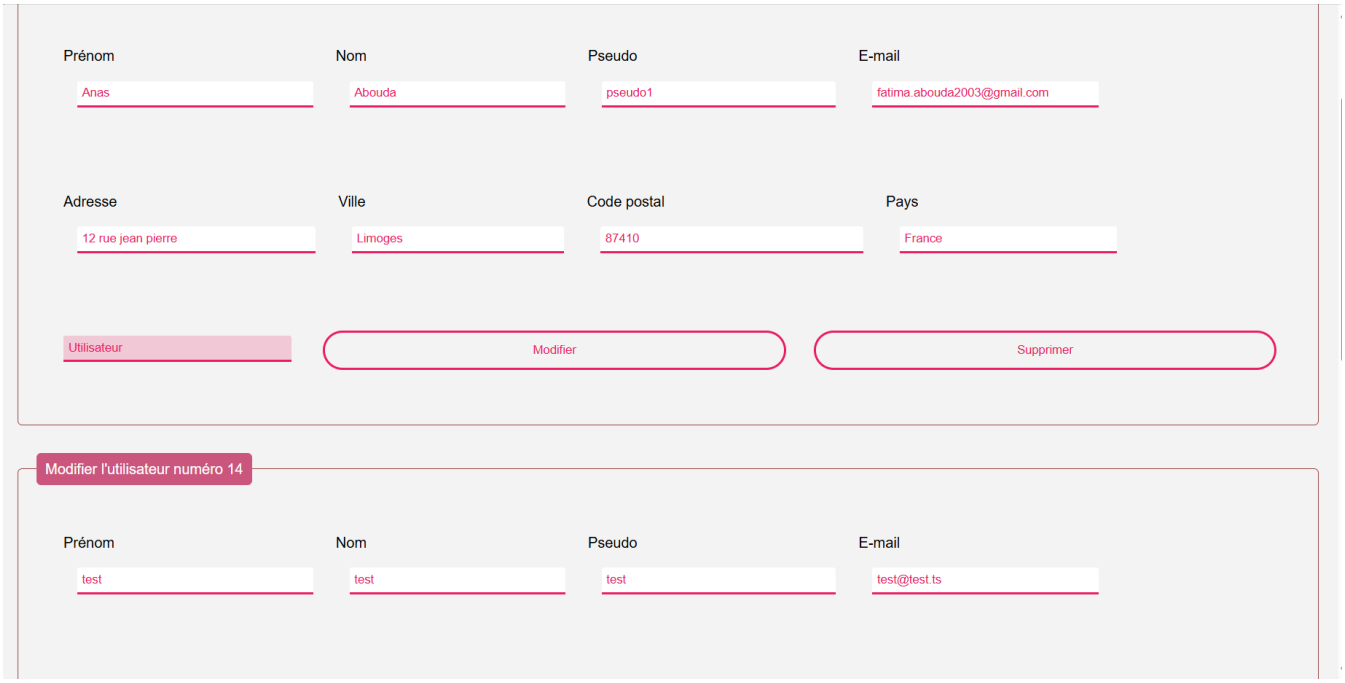


Modifier le mot de passe

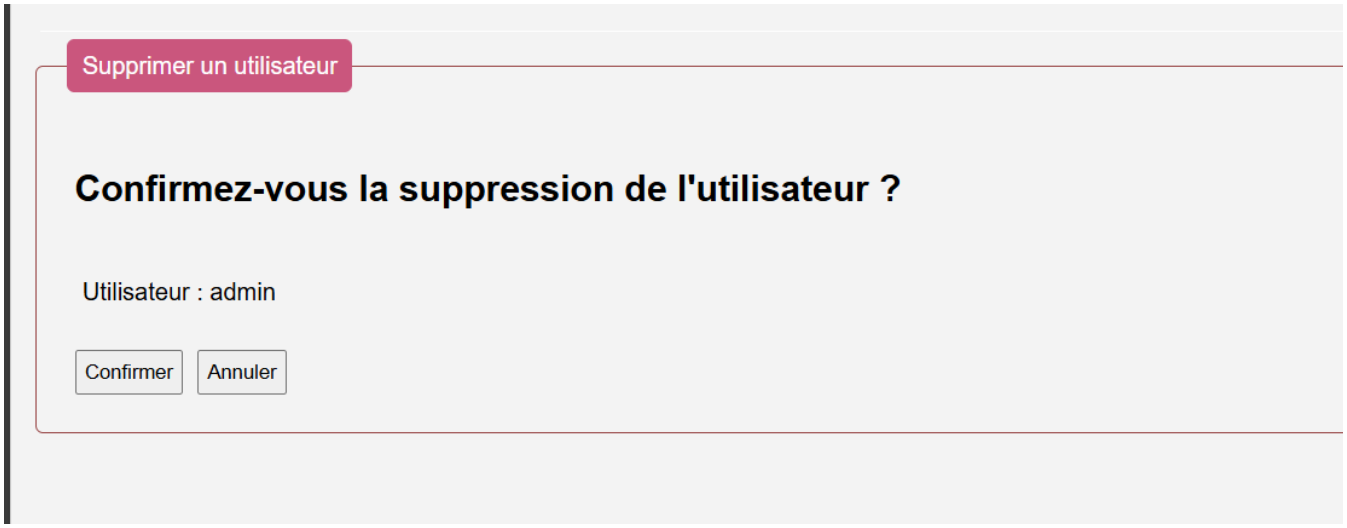
Récupère les informations de l'utilisateur à partir de la session. Lorsque le formulaire de modification est soumis, il vérifie d'abord si les champs du formulaire sont remplis. Ensuite, il vérifie si le mot de passe actuel est correct en utilisant la fonction password_verify(). Si le mot de passe actuel est correct, il vérifie si le nouveau mot de passe et sa confirmation correspondent. Si tout est valide, il hache le nouveau mot de passe avec password_hash() et met à jour le mot de passe de l'utilisateur avec la méthode updatePassword() de la classe User. Il affiche ensuite un message de succès ou d'erreur selon le résultat de la mise à jour.

Voir tous les utilisateurs pour admin

La modification et la suppression des utilisateurs dans un système. Tout d'abord, il vérifie si l'utilisateur est connecté. Ensuite, il récupère tous les utilisateurs à partir de la base de données. Lorsqu'un formulaire est soumis, il met à jour ou supprime l'utilisateur correspondant en fonction de l'action spécifiée. Pour la mise à jour, il récupère les informations du formulaire et utilise la méthode updateUser() de la classe User. Pour la suppression, il récupère l'ID de l'utilisateur à supprimer et utilise la méthode deleteUser() de la classe User.

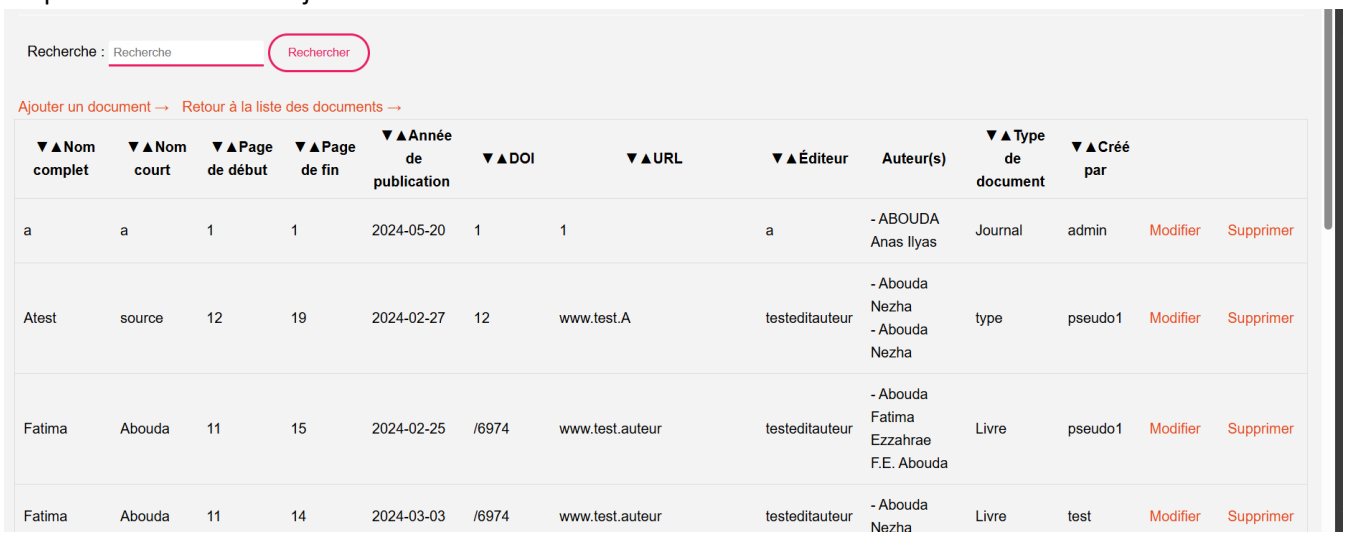


supprimer



pour les autres pages on est obligé de s'inscrire ou de se connecter pour y accéder

Pour la page document et source sa récupérer seulement ceux des utilisateur sauf pour l'admin il peut voir tout et quelle utilisateur l'as ajouté



▼▲Nom complet	▼▲Nom court	▼▲Page de début	▼▲Page de fin	▼▲Année de publication	▼▲DOI	▼▲URL	▼▲Éditeur	Auteur(s)	▼▲Type de document	▼▲Créé par		
a	a	1	1	2024-05-20	1	1	a	- ABOUDA Anas Ilyas	Journal	admin	Modifier	Supprimer
Atest	source	12	19	2024-02-27	12	www.test.A	testeditauteur	- Abouda Nezha - Abouda Nezha	type	pseudo1	Modifier	Supprimer
Fatima	Abouda	11	15	2024-02-25	/6974	www.test.auteur	testeditauteur	- Abouda Fatima Ezzahrae F.E. Abouda	Livre	pseudo1	Modifier	Supprimer
Fatima	Abouda	11	14	2024-03-03	/6974	www.test.auteur	testeditauteur	- Abouda Nezha	Livre	test	Modifier	Supprimer

8.1.2 Structure de la classe Author:

Cette classe correspond à la table author dans la base de données

8.1.3 Attributs :

- **$id:** Identifiant unique de l'auteur dans la base de données.
- **$lastname:** Nom de famille de l'auteur.
- **$firstname:** Prénom de l'auteur.
- **$firstname2:** Deuxième prénom de l'auteur (optionnel).
- **$firstname3:** Troisième prénom de l'auteur (optionnel).
- **$initial:** Initiale de l'auteur (optionnel).

```
public $id;
    public $lastname;
    public $firstname;
    public $firstname2;
    public $firstname3;
    public $initial;
```

8.1.4 Méthodes :

- **__construct():** Constructeur de la classe, utilisé pour initialiser les propriétés d'un objet Author.
- **getAll():** Retourne un tableau contenant tous les auteurs à partir de la base de données.

```
public static function getAll()
    {
        $database = new Database();
        $connection = $database->getConnection();
        $query = "SELECT * FROM author";
        $stmt = $connection->prepare($query);
        $stmt->execute();
        $authorsData = $stmt->fetchAll(PDO::FETCH_ASSOC);
        $authors = array();
        foreach ($authorsData as $authorData) {
            $author = new Author($authorData['firstname'], $authorData['lastname'],
$authorData['firstname2'], $authorData['firstname3'], $authorData['initial']);
            $author->id = $authorData['id'];
            array_push($authors, $author);
        }
        return $authors;    }
```

- **create():** Crée un nouvel auteur dans la base de données.

```
public function create($lastname, $firstname, $firstname2, $firstname3, $initial)
    {
        $database = new Database();
        $connection = $database->getConnection();
        // Vérifier si l'auteur existe déjà
        // Check if the author already exists
        if (self::isAuthorExists($lastname, $firstname, $firstname2, $firstname3,
$initial)) {
            // L'auteur existe déjà, ne pas l'ajouter
            echo "L'auteur existe déjà dans la base de données.";
            // Arrêter l'exécution de la méthode ici
```

```
            // Stop the execution of the method here
            return;
        }
        // Ajouter l'auteur
        $query = "INSERT INTO author (firstname, lastname, firstname2, firstname3,
initial)
              VALUES (:firstname, :lastname, :firstname2, :firstname3, :initial)";
        $statement = $connection->prepare($query);
        $statement->bindParam(':firstname', $firstname);
        $statement->bindParam(':lastname', $lastname);
        $statement->bindParam(':firstname2', $firstname2);
        $statement->bindParam(':firstname3', $firstname3);
        $statement->bindParam(':initial', $initial);
        $statement->execute();
        return "L'auteur a été ajouté avec succès.";
    }
```

- **isAuthorExists():** Vérifie si un auteur existe déjà dans la base de données.

```
public static function isAuthorExists($lastname, $firstname, $firstname2, $firstname3,
$initial)
    {
        $database = new Database();
        $connection = $database->getConnection();
        // Vérifiez si l'auteur existe déjà dans la base de données
        // Check if the author already exists in the database
        $query = "SELECT COUNT(*) FROM author WHERE lastname = :lastname  AND
firstname = :firstname  AND firstname2 = :firstname2  AND firstname3 = :firstname3 AND
initial = :initial";
        $statement = $connection->prepare($query);
        $statement->bindParam(':lastname', $lastname);
        $statement->bindParam(':firstname', $firstname);
        $statement->bindParam(':firstname2', $firstname2);
        $statement->bindParam(':firstname3', $firstname3);
```

```
        $statement->bindParam(':initial', $initial);
        $statement->execute();
        $count = $statement->fetchColumn();
        return $count > 0;
    }
```

- **update():** Met à jour les informations d'un auteur existant.

```
public static function update($id, $lastname, $firstname, $firstname2 = null,
$firstname3 = null, $initial = null)
    {
        try {
            $database = new Database();
            $connection = $database->getConnection();
            // Utilisez une requête préparée pour éviter les injections SQL
            // Use a prepared query to avoid SQL injections
            $sql = "UPDATE `author` SET
            `lastname` = :lastname,
            `firstname` = :firstname,
            `firstname2` = :firstname2,
            `firstname3` = :firstname3,
            `initial` = :initial
            WHERE `id` = :id";

            $stmt = $connection->prepare($sql);
            $stmt->bindParam(':id', $id);
            $stmt->bindParam(':lastname', $lastname);
            $stmt->bindParam(':firstname', $firstname);
            $stmt->bindParam(':firstname2', $firstname2);
            $stmt->bindParam(':firstname3', $firstname3);
            $stmt->bindParam(':initial', $initial);
            $stmt->execute();
            // Fermeture de la connexion après l'exécution de la requête
```

```
            // Close the connection after the query has been executed
            $connection = null;
            echo "Mise à jour de l'auteur réussie.";
        } catch (PDOException $e) {
            echo "Erreur lors de la mise à jour de l'auteur : " . $e->getMessage();
        }
    }
```

- **getById():** Récupère un auteur spécifique à partir de la base de données en utilisant son identifiant.

```
public static function getById($id)
    {
        $database = new Database();
        $connection = $database->getConnection();
        // Utilisez une requête préparée pour éviter les injections SQL
        // Use a prepared query to avoid SQL injections
        $query = "SELECT * FROM author WHERE id = :id";
        $stmt = $connection->prepare($query);
        $stmt->bindParam(':id', $id);
        $stmt->execute();
        // Récupérez les données de l'auteur
        // Get the author's data
        $authorData = $stmt->fetch(PDO::FETCH_ASSOC);
        // Si l'auteur n'existe pas, retournez null
        // If the author does not exist, return null
        if (!$authorData) {
            return null;
        }
        // Créez un nouvel objet Author en supposant un constructeur approprié
        // Create a new Author object assuming an appropriate constructor
        $author = new Author($authorData['lastname'], $authorData['firstname'],
$authorData['firstname2'], $authorData['firstname3'], $authorData['initial']);
```

```
        $author->id = $authorData['id'];
        return $author;
    }
```

- **delete():** Supprime un auteur de la base de données.

```
public static function delete($id)
    {
        $database = new Database();
        $connection = $database->getConnection();
        // Vérifiez d'abord s'il y a des documents associés à cet auteur
        // First check if there are any documents associated with this author
        $queryCheck = "SELECT COUNT(*) FROM document_author WHERE author_id = :id";
        $stmtCheck = $connection->prepare($queryCheck);
        $stmtCheck->bindParam(':id', $id);
        $stmtCheck->execute();
        $countDocuments = $stmtCheck->fetchColumn();

        // Si des documents sont associés à cet auteur, affichez un message d'erreur
        // If there are documents associated with this author, display an error
message
        if ($countDocuments > 0) {
            echo "Cet auteur est associé à des documents et ne peut pas être
supprimé.";
        }
        // Sinon, supprimez l'auteur
        // Otherwise, delete the author
        else {
            $queryDelete = "DELETE FROM author WHERE id = :id";
            $stmtDelete = $connection->prepare($queryDelete);
            $stmtDelete->bindParam(':id', $id);
            $stmtDelete->execute();
            echo "L'auteur a été supprimé avec succès.";
```

```
        }
    }
```

- **getAllSorted():** Retourne tous les auteurs triés selon une colonne et un ordre spécifiés.

```
public static function getAllSorted($order, $sort)
    {
        $database = new Database();
        $connection = $database->getConnection();
        // Si la colonne de tri est vide ou invalide, utilisez la colonne par défaut
        // If the sort column is empty or invalid, use the default column
        $allowedColumns = ['firstname', 'lastname', 'firstname2', 'firstname3',
'initial'];
        $order = in_array($order, $allowedColumns) ? $order : 'id';
        // Assurez-vous que le tri est soit 'asc' soit 'desc', sinon utilisez 'asc'
par défaut
        // Make sure the sort is either 'asc' or 'desc', otherwise use 'asc' by
default
        $sort = in_array(strtolower($sort), ['asc', 'desc']) ? strtolower($sort) :
'asc';
        // Utilisez une requête préparée pour éviter les injections SQL
        // Use a prepared query to avoid SQL injections
        $query = "SELECT * FROM author ORDER BY CASE WHEN $order = '' THEN 1 ELSE 0
END, $order $sort";
        $stmt = $connection->prepare($query);
        $stmt->execute();
        $authorsData = $stmt->fetchAll(PDO::FETCH_ASSOC);
        $authors = array();
        foreach ($authorsData as $authorData) {
            // Créez un nouvel objet Author en supposant un constructeur approprié
            // Create a new Author object assuming an appropriate constructor
            $author = new Author(
                $authorData['firstname'],
                $authorData['lastname'],
                $authorData['firstname2'],
```

```
                $authorData['firstname3'],
                $authorData['initial']
            );
            // Assurez-vous que l'ID existe avant de l'assigner
            // Make sure the ID exists before assigning it
            if (isset($authorData['id'])) {
                $author->id = $authorData['id'];
            }
            // Ajoutez l'auteur à la liste des auteurs
            // Add the author to the list of authors
            $authors[] = $author;
        }
        return $authors;    }
```

- **search():** Recherche des auteurs en fonction d'une requête de recherche.

```
public static function search($searchQuery)
    {
        $database = new Database();
        $connection = $database->getConnection();
        // Utilisez une requête préparée pour éviter les injections SQL
        // Use a prepared query to avoid SQL injections
        $query = "SELECT * FROM author WHERE lastname LIKE :searchQuery OR firstname
LIKE :searchQuery OR firstname2 LIKE :searchQuery OR firstname3 LIKE :searchQuery OR
initial LIKE :searchQuery";
        $stmt = $connection->prepare($query);
        $stmt->bindValue(':searchQuery', "%$searchQuery%", PDO::PARAM_STR);
        $stmt->execute();
        // Récupérez les données de l'auteur
        // Get the author's data
        $authorsData = $stmt->fetchAll(PDO::FETCH_ASSOC);
        $authors = array();
        // Créez un nouvel objet Author en supposant un constructeur approprié
```

```
        // Create a new Author object assuming an appropriate constructor
        foreach ($authorsData as $authorData) {
            $author = new Author(
                $authorData['firstname'],
                $authorData['lastname'],
                $authorData['firstname2'],
                $authorData['firstname3'],
                $authorData['initial']
            );
            // Permet de vérifier si l'ID existe avant de l'assigner
            // Make sure the ID exists before assigning it
            if (isset($authorData['id'])) {
                $author->id = $authorData['id'];            }
            $authors[] = $author;        }
        return $authors;    }
```

8.1.5 Rôle de la classe dans l'application :

- Représente les auteurs au sein de l'application.
- Fournit des méthodes pour la gestion des auteurs, notamment :
    - La récupération des informations des auteurs.
    - La création de nouveaux auteurs.
    - La mise à jour des auteurs existants.
    - La suppression des auteurs.
    - La recherche d'auteurs.

8.1.6 Structure de la classe document :

Cette classe gère les documents dans une base de données. Voici une explication détaillée de ses propriétés et méthodes :

8.1.6.1 Propriétés

- **$id :** Identifiant unique du document (nombre entier).
- **$namecomplet :** Titre complet du document (chaîne de caractères).
- **$namecourt :** Titre court du document (chaîne de caractères).
- **$startpage :** Page de début du document (nombre entier).

- **$endpage :** Page de fin du document (nombre entier).
- **$yearpublication :** Année de publication du document (nombre entier).
- **$doi :** Identifiant DOI du document (chaîne de caractères).
- **$url :** URL du document en ligne (chaîne de caractères).
- **$publisher :** Éditeur du document (chaîne de caractères).
- **$idauthor :** Identifiant de l'auteur associé au document (nombre entier).
- **$idtype :** Identifiant du type de document (nombre entier).

```
public $id;
    public $namecomplet;
    public $namecourt;
    public $startpage;
    public $endpage;
    public $yearpublication;
    public $doi;
    public $url;
    public $publisher;
    public $idauthor;
    public $idtype;
```

Constructeur

- **__construct($id, $namecomplet, $namecourt, $startpage, $endpage, $yearpublication, $doi, $url, $publisher, $idauthor, $idtype) :** Cette méthode permet de créer un nouvel objet Document en initialisant ses propriétés avec les valeurs fournies en paramètres.

Méthodes statiques

- **getAll():** Récupère tous les documents de la base de données et renvoie un tableau d'objets Document.

```
public static function getAll()
    {
        $database = new Database();
        $connection = $database->getConnection();
        $query = "SELECT * FROM document";
        $stmt = $connection->prepare($query);
        $stmt->execute();
        $documents = array();
        while ($row = $stmt->fetch(PDO::FETCH_ASSOC)) {
            $document = new Document($row['id'],$row['namecomplet'],
$row['namecourt'], $row['startpage'], $row['endpage'], $row['yearpublication'],
$row['doi'], $row['url'], $row['publisher'], $row['idauthor'], $row['idtype']);
            $documents[] = $document;
        }
```

```
        return $documents;
    }
```

- **getById($id):** Renvoie un objet Document spécifique en fonction de son identifiant, ou null si le document n'est pas trouvé.

```
public static function getById($id)
    {
        $database = new Database();
        $connection = $database->getConnection();
        $query = "SELECT * FROM document WHERE id = :id";
        $stmt = $connection->prepare($query);
        $stmt->bindParam(':id', $id);
        $stmt->execute();
        $row = $stmt->fetch(PDO::FETCH_ASSOC);
        if ($row) {$document = new Document($row['id'], $row['namecomplet'],
$row['namecourt'], $row['startpage'], $row['endpage'], $row['yearpublication'],
$row['doi'], $row['url'], $row['publisher'], $row['idauthor'], row['idtype']
            );
            return $document;
        }
        return null;
    }
```

Création et modification de documents

- **createWithAuthors($namecomplet, $namecourt, $startpage, $endpage, $yearpublication, $doi, $url, $publisher, $authors, $idtype) :** Crée un nouveau document dans la base de données et l'associe aux auteurs fournis dans le tableau $authors. La méthode vérifie si le document existe déjà et lève une exception en cas d'erreur.

```
public static function createWithAuthors($namecomplet, $namecourt, $startpage,
$endpage, $yearpublication, $doi, $url, $publisher, $authors, $idtype)
    {
        $database = new Database();
        $connection = $database->getConnection();
        // Vérifier si le document existe déjà
        // Check if the document already exists
```

```
        if (self::isDocumentExists($namecomplet, $yearpublication, $idtype)) {
            // Le document existe déjà, ne pas l'ajouter
            // The document already exists, do not add it
            echo "Le document existe déjà dans la base de données.";
            // Arrêter l'exécution de la fonction
            // Stop the execution of the function
            return;
        }
        // Créer le document dans la base de données
        // Create the document in the database
        $query = "INSERT INTO document (namecomplet, namecourt, startpage, endpage,
yearpublication, doi, url, publisher, idtype)
          VALUES (:namecomplet, :namecourt, :startpage, :endpage, :yearpublication,
:doi, :url, :publisher, :idtype)";
        $stmt = $connection->prepare($query);
        $stmt->bindParam(':namecomplet', $namecomplet);
        $stmt->bindParam(':namecourt', $namecourt);
        $stmt->bindParam(':startpage', $startpage);
        $stmt->bindParam(':endpage', $endpage);
        $stmt->bindParam(':yearpublication', $yearpublication);
        $stmt->bindParam(':doi', $doi);
        $stmt->bindParam(':url', $url);
        $stmt->bindParam(':publisher', $publisher);
        $stmt->bindParam(':idtype', $idtype);
        // Exécuter la requête
        // Execute the query
        try {
            $stmt->execute();
        } catch (PDOException $e) {
            throw new Exception("Error creating document: " . $e->getMessage());
        }
        // Récupérer l'ID du document nouvellement créé
```

```
        // Get the ID of the newly created document
        $documentId = $connection->lastInsertId();
        // Si $authors est une chaîne, la diviser en un tableau
        // If $authors is a string, split it into an array
        if (is_string($authors)) {
            $authors = explode(',', $authors);
        }
        // Parcourir les auteurs et les ajouter au document
        // Loop through the authors and add them to the document
        foreach ($authors as $authorId) {
            try {
                self::addAuthor($documentId, $authorId);
            } catch (Exception $e) {
                throw new Exception("Error adding author to document: " . $e-
>getMessage());
            }
        }
       // Vérifier si des auteurs ont été ajoutés au document
        // Check if any authors were added to the document
        if (empty($authors)) {
            echo "Vous n'avez pas ajouté d'auteur au document.";
        }
        return $documentId;
    }
```

- **isDocumentExists($namecomplet, $yearpublication, $idtype) :** Vérifie si un document avec le nom complet, l'année de publication et le type spécifiés existe déjà dans la base de données. Renvoie true si le document existe, false sinon.

```
public static function isDocumentExists($namecomplet, $yearpublication, $idtype)
    {
        $database = new Database();
        $connection = $database->getConnection();
        $query = "SELECT COUNT(*) FROM document
```

```
            WHERE namecomplet = :namecomplet AND yearpublication = :yearpublication
AND idtype = :idtype";
        $stmt = $connection->prepare($query);
        $stmt->bindParam(':namecomplet', $namecomplet);
        $stmt->bindParam(':yearpublication', $yearpublication);
        $stmt->bindParam(':idtype', $idtype);
        $stmt->execute();
        $count = $stmt->fetchColumn();
        return $count > 0;
    }
```

- **addAuthor($documentId, $authors) :** Ajoute un ou plusieurs auteurs (identifiés par leur ID) à un document existant. La méthode effectue des vérifications pour s'assurer que les auteurs existent et gère les transactions dans la base de données.

```
public static function addAuthor($documentId, $authors)
    {
        $database = new Database();
        $connection = $database->getConnection();
        // Commencer une transaction
        try {
            // Commencer une transaction
            // Start a transaction
            $connection->beginTransaction();
            // Vérifier si $authors est un tableau
            // Ensure $authors is an array
            if (!is_array($authors)) {
                throw new Exception("Invalid argument type for \$authors. Expected
array.");
            }
            // Parcourir les auteurs et les ajouter au document
            // Loop through the authors and add them to the document
```

```
            foreach ($authors as $authorId) {
                // Ignorer l'insertion si l'ID de l'auteur est '0'
                // Skip the insertion if the author ID is '0'
                if ($authorId === '0') {
                    // Déclaration de débogage pour identifier quand l'ID de l'auteur
est '0'
                    // Debugging statement to identify when author ID is '0'
                    echo "Skipped author with ID 0.<br>";
                    continue;
                }
                // Vérifier si l'auteur existe dans la table des auteurs
                // Check if the author exists in the author table
                if (!self::authorExists($authorId)) {
                    // Annuler la transaction si l'auteur n'existe pas
                    // Rollback the transaction if the author does not exist
                    $connection->rollBack();
                    echo "Author with ID $authorId does not exist.<br>"; // Debugging
statement
                    throw new Exception("Author with ID $authorId does not exist.");
                }
                // Continuer avec l'insertion
                // Continue with the insertion
                $query = "INSERT INTO document_author (document_id, author_id) VALUES
(:documentId, :authorId)";
                $stmt = $connection->prepare($query);
                // Convertir explicitement $documentId et $authorId en entiers s'ils
sont censés être des entiers
                // Explicitly cast $documentId and $authorId to integers if they are
supposed to be integers
                $documentId = (int)$documentId;
                $authorId = (int)$authorId;
                $stmt->bindParam(':documentId', $documentId, PDO::PARAM_INT);
                $stmt->bindParam(':authorId', $authorId, PDO::PARAM_INT);
```

```
                $stmt->execute();
            }
            // Valider la transaction
            // Commit the transaction
            $connection->commit();
        } catch (PDOException $e) {
            // Annuler la transaction en cas d'erreur
            // Rollback the transaction on error
            $connection->rollBack();
            // Journaliser ou gérer l'exception PDO
            // Log or handle the PDO exception
            echo "PDO Error: " . $e->getMessage() . "<br>"; // Debugging statement
            throw new Exception("Error adding author to document: " . $e-
>getMessage());
        } catch (Exception $ex) {
            // Journaliser l'exception pour un débogage supplémentaire
            // Log the exception for additional debugging
            echo "Exception: " . $ex->getMessage() . "<br>"; // Debugging statement
            throw $ex; // Re-throw the exception after logging
        }
    }
private static function authorExists($authorId)
    {
        // Si $authorId est '0', considérez-le comme une condition valide (aucun
auteur sélectionné)
        // If $authorId is '0', consider it as a valid condition (no author selected)
        if ($authorId === '0') {
            return true;
        }
        $database = new Database();
        $connection = $database->getConnection();
        // Vérifier si l'auteur existe dans la table des auteurs
```

```
        // Check if the author exists in the author table
        $query = "SELECT COUNT(*) FROM author WHERE id = :authorId";
        $stmt = $connection->prepare($query);
        $stmt->bindParam(':authorId', $authorId, PDO::PARAM_INT);
        $stmt->execute();
        return ($stmt->fetchColumn() > 0);
```

- **removeAuthor($documentId, $authorId) :** Supprime un auteur spécifique d'un document.

```
public static function removeAuthor($documentId, $authorId)
    {
        $database = new Database();
        $connection = $database->getConnection();
        $query = "DELETE FROM document_author WHERE document_id = :documentId AND
author_id = :authorId";
        // Convertir explicitement $documentId et $authorId en entiers s'ils sont
censés être des entiers
        // Explicitly cast $documentId and $authorId to integers if they are supposed
to be integers
        try {
            $stmt = $connection->prepare($query);
            $stmt->bindParam(':documentId', $documentId, PDO::PARAM_INT);
            $stmt->bindParam(':authorId', $authorId, PDO::PARAM_INT);
            $stmt->execute();
        } catch (PDOException $e) {
            echo "PDO Error: " . $e->getMessage();
            throw new Exception("Error removing author from document: " . $e-
>getMessage());
        }
    }
```

- **update($id, $namecomplet, $namecourt, $startpage, $endpage, $yearpublication, $doi, $url, $publisher, $idtype) :** Met à jour les propriétés d'un document existant dans la base de données.

```
 public static function update($id, $namecomplet, $namecourt, $startpage, $endpage,
$yearpublication, $doi, $url, $publisher, $idtype)
    {
```

```
        $database = new Database();
        $connection = $database->getConnection();
        $query = "UPDATE document SET namecomplet = :namecomplet, namecourt =
:namecourt, startpage = :startpage, endpage = :endpage
yearpublication = :yearpublication,  doi = :doi,  url = :url,  publisher = :publisher,
idtype = :idtype WHERE id = :id";
        $stmt = $connection->prepare($query);
        $stmt->bindParam(':id', $id);
        $stmt->bindParam(':namecomplet', $namecomplet);
        $stmt->bindParam(':namecourt', $namecourt);
        $stmt->bindParam(':startpage', $startpage);
        $stmt->bindParam(':endpage', $endpage);
        $stmt->bindParam(':yearpublication', $yearpublication);
        $stmt->bindParam(':doi', $doi);
        $stmt->bindParam(':url', $url);
        $stmt->bindParam(':publisher', $publisher);
        $stmt->bindParam(':idtype', $idtype);
        $stmt->execute();
    }
```

- **updateAuthors($documentId, $authors) :** Met à jour la liste des auteurs associés à un document. La méthode identifie les auteurs à supprimer et à ajouter en fonction de la nouvelle liste fournie.

```
public static function updateAuthors($documentId, $authors)
    {
        $database = new Database();
        $connection = $database->getConnection();
        // Recupérer les auteurs existants associés au document
        // Get the existing authors associated with the document
        $existingAuthorsQuery = "SELECT author_id FROM document_author WHERE
document_id = :documentId";
        $stmtExistingAuthors = $connection->prepare($existingAuthorsQuery);
        $stmtExistingAuthors->bindParam(':documentId', $documentId);
```

```
        $stmtExistingAuthors->execute();
        $existingAuthors = $stmtExistingAuthors->fetchAll(PDO::FETCH_COLUMN);
        // Identifier les auteurs à supprimer
        // Identify authors to be removed
        $authorsToRemove = array_diff($existingAuthors, $authors);
        // Supprimer les auteurs qui ne sont pas dans la nouvelle liste
        // Remove authors that are not in the new list
        if (!empty($authorsToRemove)) {
            $placeholders = implode(',', array_fill(0, count($authorsToRemove), '?'));
            $removeAuthorsQuery = "DELETE FROM document_author WHERE document_id =
:documentId AND author_id IN ($placeholders)";
            $stmtRemoveAuthors = $connection->prepare($removeAuthorsQuery);
            $stmtRemoveAuthors->bindParam(':documentId', $documentId);
            foreach ($authorsToRemove as $index => $authorIdToRemove) {
                $stmtRemoveAuthors->bindValue(($index + 1), $authorIdToRemove,
PDO::PARAM_INT);
            }
            $stmtRemoveAuthors->execute();
        }
        // Ajouter de nouveaux auteurs
        // Add new authors
        foreach ($authors as $authorId) {
            if (!empty($authorId)) {
                $query = "INSERT INTO document_author (document_id, author_id) VALUES
(:documentId, :authorId)";
                $stmt = $connection->prepare($query);
                $stmt->bindParam(':documentId', $documentId);
                $stmt->bindParam(':authorId', $authorId);
                $stmt->execute();
            }
        }
        echo "Le document a bien été modifié";
```

```
    }
```

- **removeAuthors($documentId) :** Supprime tous les auteurs associés à un document.

```
 public static function removeAuthors($documentId)
    {
        $database = new Database();
        $connection = $database->getConnection();
        $query = "DELETE FROM document_author WHERE document_id = :documentId";
        $stmt = $connection->prepare($query);
        $stmt->bindParam(':documentId', $documentId);
        $stmt->execute();
    }
```

Suppression de documents

- **delete($id) :** Supprime un document de la base de données après avoir supprimé les auteurs qui lui sont associés. La méthode vérifie également si le document est lié à une source

```
ublic static function delete($id)
    {
        // Vérifier si le document est lié à une source
        // Check if the document is linked to a source
        if (self::isDocumentLinkedToSource($id)) {
            // Le document est lié à une source, afficher un message ou prendre toute
autre action nécessaire.
            // The document is linked to a source, display a message or take any other
necessary action.
            echo "Le document est lié à une source et ne peut pas être supprimé.";
            return;
        }

        // Supprimer les auteurs liés au document
        // Remove the authors linked to the document
        self::deleteAuthorsForDocument($id);
```

```
        // Supprimer le document
        // Remove the document
        $database = new Database();
        $connection = $database->getConnection();
        $query = "DELETE FROM document WHERE id = :id";
        $stmt = $connection->prepare($query);
        $stmt->bindParam(':id', $id);
        $stmt->execute();

        echo "Le document a été supprimé avec succès.";
    }
```

- **isDocumentLinkedToSource($documentId) :** Vérifie si un document est lié à une source

```
public static function isDocumentLinkedToSource($documentId)
    {
        $database = new Database();
        $connection = $database->getConnection();
        // Vérifier si le document est lié à une source
        // Check if the document is linked to a source
        $query = "SELECT COUNT(*) FROM source WHERE idDocument = :documentId";
        $stmt = $connection->prepare($query);
        $stmt->bindParam(':documentId', $documentId);
        $stmt->execute();
        // Récupérer le nombre de lignes
        // Get the number of rows
        $count = $stmt->fetchColumn();
        return $count > 0;
    }
```

Affichage et recherche de documents

- **getAuthors():** Récupère un tableau d'objets Author associés à l'objet document courant.

```php
public function getAuthors()
    {
        $database = new Database();
        $connection = $database->getConnection();
        $query = "SELECT author.* FROM author
              JOIN document_author ON author.id = document_author.author_id
              WHERE document_author.document_id = :documentId";
        $stmt = $connection->prepare($query);
        $stmt->bindParam(':documentId', $this->id);
        $stmt->execute();
        // Créer un tableau pour stocker les auteurs
        // Create an array to store the authors
        $authors = array();
        // Parcourir les résultats et créer des objets auteur
        // Loop through the results and create author objects
        while ($row = $stmt->fetch(PDO::FETCH_ASSOC)) {
            $author = new Author(
                $row['firstname'],
                $row['lastname'],
                $row['firstname2'],
                $row['firstname3'],
                $row['initial']
            );
            $authors[] = $author;
        }
        return $authors;
    }
```

- **getAuthorsWithAssociation():** Récupère un tableau d'objets Author associés au document courant. La méthode inclut un champ supplémentaire "associated_document" dans chaque objet auteur, indiquant le document auquel il est lié.

```php
public function getAuthorsWithAssociation()
```

```
    {
        $database = new Database();
        $connection = $database->getConnection();
        // Récupérer les auteurs associés au document
        // Get the authors associated with the document
        $query = "SELECT author.*, document_author.document_id AS associated_document
              FROM author
              LEFT JOIN document_author ON author.id = document_author.author_id
                                        AND document_author.document_id =
:documentId";
        $stmt = $connection->prepare($query);
        $stmt->bindParam(':documentId', $this->id);
        $stmt->execute();
        // Créer un tableau pour stocker les auteurs
        // Create an array to store the authors
        $authors = array();
        // Parcourir les résultats et créer des objets auteur
        // Loop through the results and create author objects
        while ($row = $stmt->fetch(PDO::FETCH_ASSOC)) {
            $author = new Author(
                $row['firstname'],
                $row['lastname'],
                $row['firstname2'],
                $row['firstname3'],
                $row['initial']
            );
            // Ajouter l'ID du document associé à l'auteur s'il est associé
            // Add the ID of the document associated with the author if it's
associated
            $author->associatedDocument = isset($row['associated_document']) ?
$row['associated_document'] : null;
            $authors[] = $author;
```

```
        }
        return $authors;
    }
```

- **getAllSorted($order, $sortOrder) :** Renvoie un tableau d'objets Document triés selon une colonne et un ordre spécifiés (croissant ou décroissant).

```
public static function getAllSorted($order, $sortOrder)
    {
        $database = new Database();
        $connection = $database->getConnection();
        // Colonnes valides pour trier les documents
        // Valid columns to sort documents
        $validColumns = ['namecomplet', 'namecourt', 'startpage', 'endpage',
'yearpublication', 'doi', 'url', 'publisher', 'idtype'];
        // Vérifiez si la colonne spécifiée est valide
        // Check if the specified column is valid
        if (!in_array($order, $validColumns)) {
            $order = 'namecomplet'; // Colonne par défaut si la colonne spécifiée
n'est pas valide // Default column if the specified column is not valid
        }
        $validSortOrders = ['asc', 'desc'];
        // Vérifiez si le sens de tri spécifié est valide
        // Check if the specified sort order is valid
        $sortOrder = in_array($sortOrder, $validSortOrders) ? $sortOrder : 'asc';
        // Récupérer tous les documents triés par colonne et ordre
        // Get all documents sorted by column and order
        $query = "SELECT * FROM document ORDER BY $order $sortOrder";
        $stmt = $connection->prepare($query);
        $stmt->execute();

        // Créer un tableau pour stocker les documents
        // Create an array to store the documents
```

```
        $documents = array();


        // Parcourir les résultats et créer des objets document
        // Loop through the results and create document objects
        while ($row = $stmt->fetch(PDO::FETCH_ASSOC)) {
            $document = new Document(
                $row['id'], $row['namecomplet'], $row['namecourt'], $row['startpage'], $row['endpage'],
       $row['yearpublication'], $row['doi'], $row['url'], $row['publisher'], $row['idauthor'], $row['idtype']
            );
            // Ajouter l'objet document au tableau
            // Add the document object to the array
            array_push($documents, $document);
        }
        return $documents;
    }
```

- **search($searchTerm) :** Recherche des documents en fonction d'un terme de recherche dans le titre complet, le titre court ou d'autres champs. Renvoie un tableau d'objets Document trouvés.

```
public static function search($searchTerm)
    {
        $database = new Database();
        $connection = $database->getConnection();


        $query = "SELECT * FROM document WHERE namecomplet LIKE :searchTerm OR namecourt LIKE :searchTerm";
        $stmt = $connection->prepare($query);
        $searchTerm = '%' . $searchTerm . '%';
        $stmt->bindParam(':searchTerm', $searchTerm);
        $stmt->execute();
        // Créer un tableau pour stocker les documents
```

```
        // Create an array to store the documents
        $documents = array();
        // Parcourir les résultats et créer des objets document
        // Loop through the results and create document objects
        while ($row = $stmt->fetch(PDO::FETCH_ASSOC)) {
            $document = new Document(
                $row['id'],
                $row['namecomplet'],
                $row['namecourt'],
                $row['startpage'],
                $row['endpage'],
                $row['yearpublication'],
                $row['doi'],
                $row['url'],
                $row['publisher'],
                $row['idauthor'],
                $row['idtype']
            );
            // Ajouter l'objet document au tableau
            // Add the document object to the array
            array_push($documents, $document);
        }
        return $documents;
    }
```

- **getTypes():** Récupère la liste des types de documents disponibles.

```
public static function getTypes()
    {
        $database = new Database();
        $connection = $database->getConnection();
        $query = "SELECT * FROM type";
        $stmt = $connection->prepare($query);
```

```
        $stmt->execute();
        // Créer un tableau pour stocker les types de documents
        // Create an array to store the document types
        $types = array();
        // Parcourir les résultats et créer des objets type
        // Loop through the results and create type objects
        while ($row = $stmt->fetch(PDO::FETCH_ASSOC)) {
            $type = new Type($row['id'], $row['name']);
            array_push($types, $type);
        }
        return $types;
    }
```

8.1.7 Structure de la classe Lieux

Cette classe gère les lieux dans une base de données. Voici un résumé de ses fonctionnalités :

8.1.7.1 Propriétés

- **$id:** Identifiant unique du lieu (nombre entier).
- **$city:** Nom de la ville (chaîne de caractères).
- **$region:** Région (chaîne de caractères).
- **$country:** Pays (chaîne de caractères).

```
public $id;
    public $city;
    public $region;
    public $country;
```

8.1.7.2 Méthodes statiques

- **getAll():** Récupère tous les lieux de la base de données et renvoie un tableau d'objets Place.

```
public static function getAll()
    {
        $database = new Database();
        $connection = $database->getConnection();
        $query = "SELECT * FROM place";
        $stmt = $connection->prepare($query);
        $stmt->execute();
```

```
        $places = array();
        while ($row = $stmt->fetch(PDO::FETCH_ASSOC)) {
            $place = new Place($row['id'], $row['city'], $row['region'],
$row['country']);
            array_push($places, $place);
        }
        return $places;
    }
```

- **getById($id):** Renvoie un objet Place spécifique en fonction de son identifiant, ou null si le lieu n'est pas trouvé.

```
public static function getById($id)
    {
        $database = new Database();
        $connection = $database->getConnection();
        $query = "SELECT * FROM place WHERE id = :id";
        $stmt = $connection->prepare($query);
        $stmt->bindParam(':id', $id);
        $stmt->execute();

        // Récupérer la ligne de résultat
        // Get the result row
        $row = $stmt->fetch(PDO::FETCH_ASSOC);

        //si la ligne existe on retourne un objet lieu sinon on retourne null
        //if the row exists we return a place object otherwise we return null
        if ($row) {
            $place = new Place($row['id'], $row['city'], $row['region'],
$row['country']);
            return $place;
        } else {
            // Gérer le cas où aucun résultat n'a été trouvé
            // Handle the case where no result is found
```

```
            return null;
        }
    }
```

- **create($city, $region, $country):** Crée un nouveau lieu dans la base de données en vérifiant s'il existe déjà.

```
public static function create($city, $region, $country)
    {
        // Vérifier si le lieu existe déjà
        // Check if the place already exists
        if (self::isPlaceExists($city, $region, $country)) {
            // Le lieu existe déjà, ne pas l'ajouter
            // The place already exists, do not add it
            echo "Le lieu existe déjà dans la base de données.";
            return;
        }

        // Ajouter le lieu
        // Add the place
        $database = new Database();
        $connection = $database->getConnection();
        $query = "INSERT INTO place (city, region, country) VALUES (:city, :region,
:country)";
        $stmt = $connection->prepare($query);
        $stmt->bindParam(':city', $city);
        $stmt->bindParam(':region', $region);
        $stmt->bindParam(':country', $country);
        $stmt->execute();
    }
```

- **isPlaceExists($city, $region, $country):** Vérifie si un lieu avec la ville, la région et le pays spécifiés existe déjà dans la base de données. Renvoie true si le lieu existe, false sinon.

```
public static function isPlaceExists($city, $region, $country)
    {
```

```
        $database = new Database();
        $connection = $database->getConnection();
        // Vérifier si le lieu existe déjà dans la base de données en fonction de la ville, de la région et du pays
        // Check if the place already exists in the database based on the city, region, and country
        $query = "SELECT COUNT(*) FROM place WHERE city = :city AND region = :region AND country = :country";
        $stmt = $connection->prepare($query);
        $stmt->bindParam(':city', $city);
        $stmt->bindParam(':region', $region);
        $stmt->bindParam(':country', $country);
        $stmt->execute();
        // Récupérer le nombre de lignes trouvées
        // Get the number of found rows
        $count = $stmt->fetchColumn();
        // Retourner vrai si le lieu existe déjà, faux sinon
        // Return true if the place already exists, false otherwise
        return $count > 0;
    }
```

- **update($id, $city, $region, $country):** Met à jour les propriétés d'un lieu existant dans la base de données.

```
public static function update($id, $city, $region, $country)
    {
        $database = new Database();
        $connection = $database->getConnection();
        $query = "UPDATE place SET city = :city, region = :region, country = :country WHERE id = :id";
        $stmt = $connection->prepare($query);
        $stmt->bindParam(':id', $id);
        $stmt->bindParam(':city', $city);
        $stmt->bindParam(':region', $region);
        $stmt->bindParam(':country', $country);
```

```
        $stmt->execute();
    }
```

- **delete($id):** Supprime un lieu de la base de données après avoir vérifié s'il est lié à une source.

```
public static function delete($id)
    {
        // Vérifier si le lieu est lié à une source
        // Check if the place is linked to a source
        if (self::isPlaceLinkedToSource($id)) {
            // Le lieu est lié à une source, afficher un message ou prendre toute autre action nécessaire.
            // The place is linked to a source, display a message or take any other necessary action.
            echo "Le lieu est lié à une source et ne peut pas être supprimé.";
            return;
        }

        // Supprimer le lieu
        // Delete the place
        $database = new Database();
        $connection = $database->getConnection();
        $query = "DELETE FROM place WHERE id = :id";
        $stmt = $connection->prepare($query);
        $stmt->bindParam(':id', $id);
        $stmt->execute();
    }
```

- **isPlaceLinkedToSource($placeId):** Vérifie si un lieu est lié à une source

```
public static function isPlaceLinkedToSource($placeId)
    {
        $database = new Database();
        $connection = $database->getConnection();
        // Vérifier si le lieu est lié à une source
        // Check if the place is linked to a source
```

```
        $query = "SELECT COUNT(*) FROM source WHERE idPlace = :placeId";
        $stmt = $connection->prepare($query);
        $stmt->bindParam(':placeId', $placeId);
        $stmt->execute();
        // Récupérer le nombre de lignes trouvées
        // Get the number of found rows
        $count = $stmt->fetchColumn();
        // Retourner vrai si le lieu est lié à une source, faux sinon
        // Return true if the place is linked to a source, false otherwise
        return $count > 0;
    }
```

- **getAllSorted($order, $column):** Renvoie un tableau d'objets Place triés selon une colonne et un ordre spécifiés (croissant ou décroissant).

```
public static function getAllSorted($order, $column)
    {
        $database = new Database();
        $connection = $database->getConnection();

        // Assurez-vous que les valeurs de tri et de colonne sont sécurisées
        // Ensure that the sort and column values are secured
        $allowedColumns = ['city', 'region', 'country'];
        $allowedOrders = ['asc', 'desc'];

        // Assurez-vous que les valeurs de tri et de colonne sont sécurisées
        // Ensure that the sort and column values are secured
        $column = in_array($column, $allowedColumns) ? $column : 'city';
        $order = in_array($order, $allowedOrders) ? $order : 'asc';

        // Récupérer tous les lieux triés
        // Get all places sorted
        $query = "SELECT * FROM place ORDER BY $column $order";
```

```
        $stmt = $connection->prepare($query);
        $stmt->execute();

        // Récupérer les lieux triés sous forme de tableau
        // Get the sorted places as an array
        $places = array();

        // Parcourir les lignes de résultats et créer des objets lieu
        // Loop through the result rows and create place objects
        while ($row = $stmt->fetch(PDO::FETCH_ASSOC)) {
            $place = new Place($row['id'], $row['city'], $row['region'],
$row['country']);
            array_push($places, $place);
        }

        return $places;
    }
```

- **search($searchTerm):** Recherche des lieux en fonction d'un terme de recherche dans la ville, la région ou le pays. Renvoie un tableau d'objets Place trouvés.

```
public static function search($searchTerm)
    {
        $database = new Database();
        $connection = $database->getConnection();
        // Requête pour rechercher un lieu par un terme de recherche
        // Query to search for a place by a search term
        $query = "SELECT * FROM place WHERE city LIKE :searchTerm OR region LIKE
:searchTerm OR country LIKE :searchTerm";

        $stmt = $connection->prepare($query);
        $searchTerm = '%' . $searchTerm . '%';
        $stmt->bindParam(':searchTerm', $searchTerm);
```

```
        $stmt->execute();


        // Récupérer les lieux trouvés sous forme de tableau
        // Get the found places as an array
        $places = array();


        // Parcourir les lignes de résultats et créer des objets lieu
        // Loop through the result rows and create place objects
        while ($row = $stmt->fetch(PDO::FETCH_ASSOC)) {
            $place = new Place($row['id'], $row['city'], $row['region'],
$row['country']);
            array_push($places, $place);
        }


        return $places;
    }
```

8.1.8 Structure de la classe Source

Cette classe gère les sources dans une base de données. Voici un résumé de ses fonctionnalités :

8.1.8.1 Attributs

- **$id:** Identifiant unique de la source (nombre entier).
- **$title:** Titre de la source (chaîne de caractères).
- **$year**: Année de publication (entier).
- **$dateAccessed**: Date d'accès à la source (date).
- **$volume:** Volume (chaîne de caractères).
- **$issue:** Numéro de parution (chaîne de caractères).
- **$bookTitle**: Titre du livre (si la source est un livre) (chaîne de caractères).
- **$patentNumber:** Numéro de brevet (chaîne de caractères).
- **$idPlace:** Identifiant du lieu associé à la source (entier). (relation possible avec la classe Place)
- **$idDocument:** Identifiant du document associé à la source (entier). (relation possible avec une autre classe Document)

```
public $id;
    public $title;
    public $year;
    public $dateAccessed;
    public $volume;
    public $issue;
    public $bookTitle;
    public $patentNumber;
    public $idPlace;
    public $idDocument;
```

8.1.8.2 Méthodes statiques

- **getAll():** Récupère toutes les sources de la base de données et renvoie un tableau d'objets Source.

```
public static function getAll()
    {
        $database = new Database();
        $connection = $database->getConnection();
        $query = "SELECT * FROM source";
        $stmt = $connection->prepare($query);
        $stmt->execute();
        //recuperer les sources dans un tableau
        //get the sources in an array
        $sources = array();
        //parcourir les lignes de resultat et les stocker dans un tableau
        //browse the result rows and store them in an array
        while ($row = $stmt->fetch(PDO::FETCH_ASSOC)) {
            $source = new Source($row['id'], $row['title'], $row['year'],
$row['dateAccessed'], $row['volume'], $row['issue'], $row['bookTitle'],
$row['patentNumber'], $row['idPlace'], $row['idDocument']);
            array_push($sources, $source);
        }
        return $sources;
    }
```

- **getById($id):** Renvoie un objet Source spécifique en fonction de son identifiant, ou null si la source n'est pas trouvée.

```
public static function getById($id)
    {
        $database = new Database();
        $connection = $database->getConnection();
        $query = "SELECT * FROM source WHERE id = :id";
        $stmt = $connection->prepare($query);
        $stmt->bindParam(':id', $id);
```

```
        $stmt->execute();


        // Vérifier si la requête a renvoyé des résultats
        // Check if the query returned any results
        if ($row = $stmt->fetch(PDO::FETCH_ASSOC)) {
            // Récupérer la ligne de résultat
            // Get the result row
            $source = new Source($row['id'], $row['title'], $row['year'],
$row['dateAccessed'], $row['volume'], $row['issue'], $row['bookTitle'],
$row['patentNumber'], $row['idPlace'], $row['idDocument']);
            return $source;
        } else {
            // Aucun résultat trouvé, retourner null ou une valeur par défaut selon
vos besoins
            // No result found, return null or a default value as needed
            return null;
        }
    }
```

- **create($title, $year, $dateAccessed, $volume, $issue, $bookTitle, $patentNumber, $idPlace, $idDocument):** Crée une nouvelle source dans la base de données en vérifiant si elle existe déjà.

```
public static function create($title, $year, $dateAccessed, $volume, $issue,
$bookTitle, $patentNumber, $idPlace, $idDocument)
    {
        $database = new Database();
        $connection = $database->getConnection();


        // Vérifier si la source existe déjà
        // Check if the source already exists
        if (self::isSourceExists($title, $year, $idDocument)) {
            // La source existe déjà, ne pas l'ajouter
            // The source already exists, do not add it
            echo "La source existe déjà dans la base de données.";
```

```
            return; // Arrêter l'exécution de la méthode si la source existe déjà // Stop the execution of the method if the source already exists
        }

        // Préparer la requête d'insertion SQL
        // Prepare the SQL insertion query
        $query = "INSERT INTO source (title, year, dateAccessed, volume, issue, bookTitle, patentNumber, idPlace, idDocument) VALUES (:title, :year, :dateAccessed, :volume, :issue, :bookTitle, :patentNumber, :idPlace, :idDocument)";
        $stmt = $connection->prepare($query);
        $stmt->bindParam(':title', $title);
        $stmt->bindParam(':year', $year);
        $stmt->bindParam(':dateAccessed', $dateAccessed);
        $stmt->bindParam(':volume', $volume);
        $stmt->bindParam(':issue', $issue);
        $stmt->bindParam(':bookTitle', $bookTitle);
        $stmt->bindParam(':patentNumber', $patentNumber);
        $stmt->bindParam(':idPlace', $idPlace);
        $stmt->bindParam(':idDocument', $idDocument);

        // Exécuter la requête
        // Execute the query
        try {
            $stmt->execute();
        } catch (PDOException $e) {
            throw new Exception("Error creating source: " . $e->getMessage());
        }
    }
```

- **isSourceExists($title, $year, $idDocument):** Vérifie si une source avec le titre, l'année et l'identifiant de document spécifiés existe déjà dans la base de données. Renvoie true si la source existe, false sinon.

```
public static function isSourceExists($title, $year, $idDocument)
    {
```

```
        $database = new Database();
        $connection = $database->getConnection();

        // Préparer la requête de recherche SQL
        // Prepare the SQL search query
        $query = "SELECT COUNT(*) FROM source
              WHERE title = :title
              AND year = :year
              AND idDocument = :idDocument";

        $stmt = $connection->prepare($query);
        $stmt->bindParam(':title', $title);
        $stmt->bindParam(':year', $year);
        $stmt->bindParam(':idDocument', $idDocument);

        $stmt->execute();

        // Récupérer le nombre de lignes trouvées et le retourner
        // Get the number of found rows and return it
        $count = $stmt->fetchColumn();
        return $count > 0;
    }
```

- **update($id, $title, $year, $dateAccessed, $volume, $issue, $bookTitle, $patentNumber, $idPlace, $idDocument):** Met à jour les propriétés d'une source existante dans la base de données.

```
public static function update($id, $title, $year, $dateAccessed, $volume, $issue,
$bookTitle, $patentNumber, $idPlace, $idDocument)
    {
        $database = new Database();
        $connection = $database->getConnection();
        $query = "UPDATE source SET title = :title, year = :year, dateAccessed =
:dateAccessed, volume = :volume, issue = :issue, bookTitle = :bookTitle, patentNumber
= :patentNumber, idPlace = :idPlace, idDocument = :idDocument WHERE id = :id";
```

```
        $stmt = $connection->prepare($query);
        $stmt->bindParam(':id', $id);
        $stmt->bindParam(':title', $title);
        $stmt->bindParam(':year', $year);
        $stmt->bindParam(':dateAccessed', $dateAccessed);
        $stmt->bindParam(':volume', $volume);
        $stmt->bindParam(':issue', $issue);
        $stmt->bindParam(':bookTitle', $bookTitle);
        $stmt->bindParam(':patentNumber', $patentNumber);
        $stmt->bindParam(':idPlace', $idPlace);
        $stmt->bindParam(':idDocument', $idDocument);
        $stmt->execute();
```

- **delete($id):** Supprime une source de la base de données.

```
public static function delete($id)
    {
        $database = new Database();
        $connection = $database->getConnection();
        $query = "DELETE FROM source WHERE id = :id";
        $stmt = $connection->prepare($query);
        $stmt->bindParam(':id', $id);
        $stmt->execute();
    }
```

- **getByDocument($idDocument):** Récupère toutes les sources associées à un document spécifique (identifié par idDocument). Renvoie un tableau d'objets Source.

```
function getByDocument($idDocument){
        $database = new Database();
        $connection = $database->getConnection();
        $query = "SELECT * FROM source WHERE idDocument = :idDocument";
        $stmt = $connection->prepare($query);
        $stmt->bindParam(':idDocument', $idDocument);
        $stmt->execute();
```

```
        //recuperer les sources dans un tableau
        //get the sources in an array
        $sources = array();
        //parcourir les lignes de resultat et les stocker dans un tableau
        //browse the result rows and store them in an array
        while ($row = $stmt->fetch(PDO::FETCH_ASSOC)) {
            $source = new Source($row['id'], $row['title'], $row['year'],
$row['dateAccessed'], $row['volume'], $row['issue'], $row['bookTitle'],
$row['patentNumber'], $row['idPlace'], $row['idDocument']);
            array_push($sources, $source);
        }
        return $sources;
    }
```

- **getAllSorted($columnName, $sortOrder):** Trie toutes les sources selon une colonne et un ordre spécifiés. La colonne peut être un attribut de la source (title, year, etc.) ou l'identifiant d'une entité liée (idDocument, idPlace).

```
public static function getAllSorted($columnName, $sortOrder)
    {
        $database = new Database();
        $connection = $database->getConnection();
        // Assurez-vous que les valeurs de tri et de colonne sont sécurisées
        // Ensure that the sort and column values are secured
        switch ($columnName) {
            case 'title':
            case 'year':
            case 'dateAccessed':
            case 'volume':
            case 'issue':
            case 'bookTitle':
            case 'patentNumber':
                // requête pour trier les sources par colonne et ordre
                // query to sort sources by column and order
```

```
                $query = "SELECT * FROM source ORDER BY $columnName $sortOrder";
                break;
            case 'idDocument':
                // requête pour trier les sources par nom de document
                // query to sort sources by document name
                $query = "SELECT source.*, document.namecomplet AS documentTitle FROM source LEFT JOIN document ON source.idDocument = document.id ORDER BY documentTitle $sortOrder";
                break;
            case 'idPlace':
                // requête pour trier les sources par nom de lieu
                // query to sort sources by place name
                $query = "SELECT source.*, place.city, place.region, place.country FROM source LEFT JOIN place ON source.idPlace = place.id ORDER BY place.city $sortOrder, place.region $sortOrder, place.country $sortOrder";
                break;
            default:
                // requête pour trier les sources par titre
                // query to sort sources by title
                $query = "SELECT * FROM source ORDER BY title $sortOrder";
                break;
        }
        $stmt = $connection->prepare($query);
        $stmt->execute();
        //recuperer les sources dans un tableau
        //get the sources in an array
        $sources = array();
        //parcourir les lignes de resultat et les stocker dans un tableau
        //browse the result rows and store them in an array
        while ($row = $stmt->fetch(PDO::FETCH_ASSOC)) {
            $source = new Source($row['id'], $row['title'], $row['year'], $row['dateAccessed'], $row['volume'], $row['issue'], $row['bookTitle'], $row['patentNumber'], $row['idPlace'], $row['idDocument']);
```

```
            array_push($sources, $source);
        }
        return $sources;
    }
```

- **search($searchQuery):** Recherche des sources en fonction d'un terme de recherche dans divers champs de la source et des entités liées. Renvoie un tableau d'objets Source trouvés.

```
public static function search($searchQuery){
        $database = new Database();
        $connection = $database->getConnection();
        $query = "SELECT * FROM source WHERE title LIKE :searchQuery OR year LIKE
:searchQuery OR dateAccessed LIKE :searchQuery OR volume LIKE :searchQuery OR issue
LIKE :searchQuery OR bookTitle LIKE :searchQuery OR patentNumber LIKE :searchQuery OR
idPlace LIKE :searchQuery OR idDocument LIKE :searchQuery";
        $stmt = $connection->prepare($query);
        $searchQuery = "%$searchQuery%";
        $stmt->bindParam(':searchQuery', $searchQuery);
        $stmt->execute();
        $sources = array();
        while ($row = $stmt->fetch(PDO::FETCH_ASSOC)) {
            $source = new Source($row['id'], $row['title'], $row['year'],
$row['dateAccessed'], $row['volume'], $row['issue'], $row['bookTitle'],
$row['patentNumber'], $row['idPlace'], $row['idDocument']);
            array_push($sources, $source);
        }
        return $sources;
    }
```

8.1.9 Structure de classe type

8.1.9.1 Propriétés

- **$id :** Identifiant unique du type (nombre entier).
- **$name :** Nom du type (chaîne de caractères).
-

```
public $id;
    public $name;
```

8.1.9.2 Méthodes

- **getAll() :** Récupère tous les types de la base de données et renvoie un tableau d'objets Type.

```
public static function getAll()
    {
        $database = new Database();
        $connection = $database->getConnection();
        $query = "SELECT * FROM type";
        $stmt = $connection->prepare($query);
        $stmt->execute();
        $types = array();
        while ($row = $stmt->fetch(PDO::FETCH_ASSOC)) {
            $type = new Type($row['id'], $row['name']);
            array_push($types, $type);
        }
        return $types;
    }
```

- **getById($id) :** Renvoie un objet Type spécifique en fonction de son identifiant, ou null si le type n'est pas trouvé.
- **create($name) :** Crée un nouveau type dans la base de données en vérifiant si un type avec le même nom existe déjà.
- **isTypeExists($name) :** Vérifie si un type avec le nom spécifié existe déjà dans la base de données. Renvoie true si le type existe, false sinon.

```
public static function isTypeExists($name)
    {        $database = new Database();
        $connection = $database->getConnection();
        $query = "SELECT COUNT(*) FROM type WHERE name = :name";
        $stmt = $connection->prepare($query);
        $stmt->bindParam(':name', $name);
        $stmt->execute();
        $count = $stmt->fetchColumn();        return $count > 0    }
```

- **update($id, $name) :** Met à jour le nom d'un type existant dans la base de données.
- **delete($id) :** Supprime un type de la base de données.
- **isTypeLinkedToDocument($typeId) :** Vérifie si un type est lié à un document. Renvoie true si au moins un document est associé au type spécifié, false sinon.

```
public static function isTypeLinkedToDocument($typeId)
    {
        $database = new Database();
        $connection = $database->getConnection();
        // Définit la requête SQL pour compter le nombre d'enregistrements dans la
table "document" avec l'ID de type spécifié.
        // Set the SQL query to count the number of records in the "document" table
with the specified type ID.
```

```
        $query = "SELECT COUNT(*) FROM document WHERE idtype = :typeId";
        $stmt = $connection->prepare($query);
        $stmt->bindParam(':typeId', $typeId);
        $stmt->execute();
        // Récupère le nombre de lignes trouvées.
        // Get the number of found rows.
        $count = $stmt->fetchColumn();
        // Retourne vrai si le type est lié à un document, faux sinon.
        // Return true if the type is linked to a document, false otherwise.
        return $count > 0;
    }
```

- **getNameById($typeId) :** Récupère le nom d'un type en fonction de son identifiant.
- **getAllOrder($order, $sort) :** Récupère tous les types triés par une colonne spécifique (par défaut : name) et un ordre de tri croissant ou décroissant (par défaut : croissant).

8.2 Dossier Vue

8.2.1 Vue Docuemnt

Gère l'affichage d'une liste de documents en tenant compte du tri et de la recherche.

8.2.1.1 Tri

- Le script récupère les paramètres de tri de l'URL (order et sort).
- Il s'assure que les valeurs de tri sont sécurisées pour éviter les injections SQL (in_array).
- Par défaut, le tri se fait par namecomplet (nom complet) en ordre croissant (asc).
- L'utilisateur peut modifier le tri en cliquant sur les en-têtes des colonnes du tableau.
- Le script appelle la méthode getAllSorted de la classe Document en lui fournissant les options de tri (order et sort).

```
$order = isset($_GET['order']) ? $_GET['order'] : 'namecomplet'; // Colonne par défaut
pour le tri // Default column for sorting
$sort = isset($_GET['sort']) && in_array($_GET['sort'], ['asc', 'desc']) ?
$_GET['sort'] : 'asc'; // Sens de tri par défaut // Default sorting direction
// Assurez-vous que les valeurs de tri sont sécurisées pour éviter les injections SQL
// Ensure that the sorting values are secured to avoid SQL injections
$order = in_array($order, ['namecomplet', 'namecourt', 'startpage', 'endpage',
'yearpublication', 'doi', 'url', 'publisher', 'idtype']) ? $order : 'namecomplet';
$sort = in_array($sort, ['asc', 'desc']) ? $sort : 'asc';
// Récupérez les documents triés en fonction des paramètres de tri
// Get the documents sorted based on the sort parameters
$documents = Document::getAllSorted($order, $sort);
$authors = Author::getAll();
$types = Type::getAll();
```

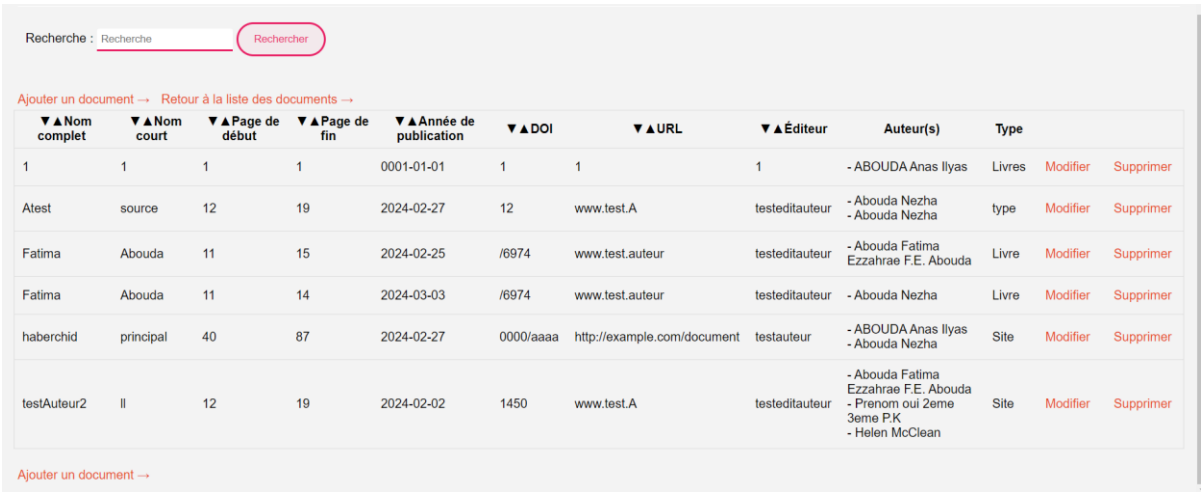
8.2.1.2 Recherche

- Le script vérifie si un terme de recherche (searchQuery) est fourni dans l'URL.
- Si un terme de recherche est présent, il appelle la méthode search de la classe Document pour récupérer les documents correspondants.
- Si aucun terme de recherche n'est fourni, il utilise la logique de tri par défaut ou une autre logique souhaitée.

```
if (isset($_GET['searchDocument'])) {
    $searchTerm = $_GET['searchQuery']; // Utilisez 'searchQuery' au lieu de 'search' pour récupérer la valeur de recherche // Use 'searchQuery' instead of 'search' to get the search value
    $documents = Document::search($searchTerm);
} else {
    // Si la recherche n'est pas effectuée, utilisez la logique de tri par défaut ou autre logique souhaitée.
    // If search is not performed, use the default sorting logic or other desired logic.
    $order = isset($_GET['order']) ? $_GET['order'] : 'default';
    $sort = isset($_GET['sort']) ? $_GET['sort'] : 'asc';
    $documents = Document::getAllSorted($order, $sort);
}
```

8.2.1.3 Affichage

- Le script affiche un tableau contenant les informations de chaque document, y compris : Nom complet, Nom court, Page de début, Page de fin, Année de publication, DOI, URL, Éditeur, Auteurs associés (noms complets), Type de document
- Des liens sont fournis pour ajouter un nouveau document, retourner à la liste des documents (si la recherche était active), ajouter un auteur et voir la liste des auteurs, et ajouter un type de document et voir la liste des types de documents.



▼▲Nom complet	▼▲Nom court	▼▲Page de début	▼▲Page de fin	▼▲Année de publication	▼▲DOI	▼▲URL	▼▲Éditeur	Auteur(s)	Type		
1	1	1	1	0001-01-01	1	1	1	- ABOUDA Anas Ilyas	Livres	Modifier	Supprimer
Atest	source	12	19	2024-02-27	12	www.test.A	testeditauteur	- Abouda Nezha - Abouda Nezha	type	Modifier	Supprimer
Fatima	Abouda	11	15	2024-02-25	/6974	www.test.auteur	testeditauteur	- Abouda Fatima Ezzahrae F.E. Abouda	Livre	Modifier	Supprimer
Fatima	Abouda	11	14	2024-03-03	/6974	www.test.auteur	testeditauteur	- Abouda Nezha	Livre	Modifier	Supprimer
haberchid	principal	40	87	2024-02-27	0000/aaaa	http://example.com/document	testauteur	- ABOUDA Anas Ilyas - Abouda Nezha	Site	Modifier	Supprimer
testAuteur2	II	12	19	2024-02-02	1450	www.test.A	testeditauteur	- Abouda Fatima Ezzahrae F.E. Abouda - Prenom oui 2eme 3eme P.K - Helen McClean	Site	Modifier	Supprimer

8.2.2 Vue ajout document

8.2.2.1 Formulaire

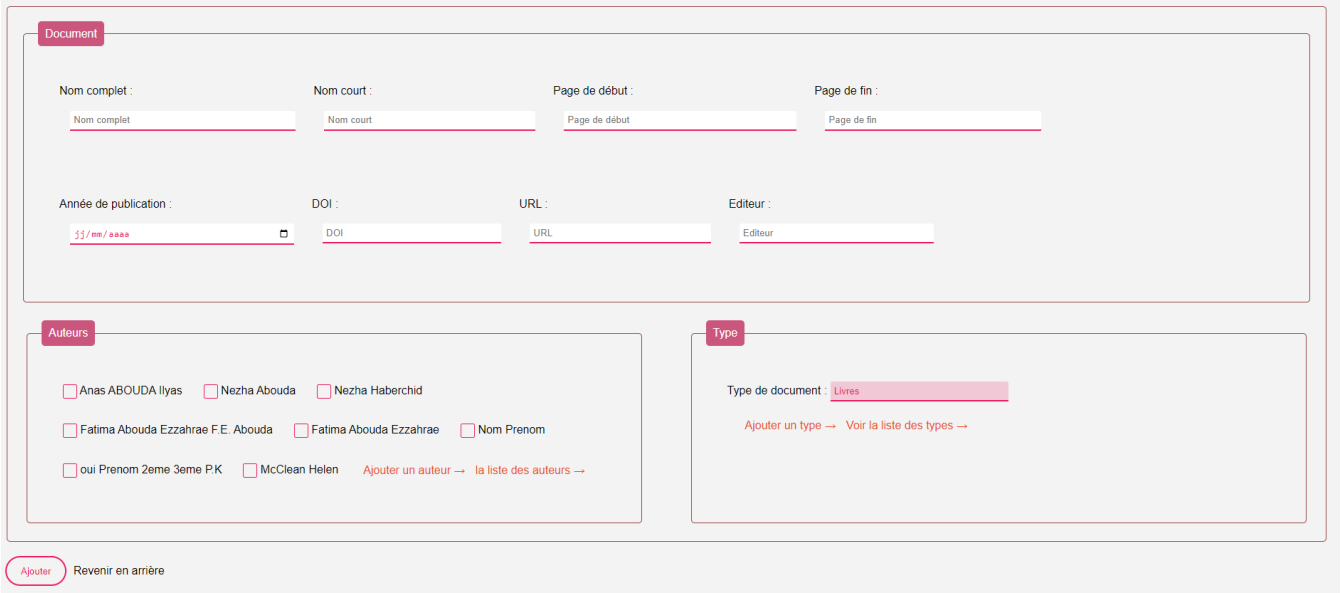
- Le script récupère les listes des documents, auteurs et types existants dans la base de données (Document :getAll(), Author::getAll(), Type::getAll()).
    - Le formulaire permet la saisie des informations du document : Nom complet, Nom court, Page de début, Page de fin, Année de publication, DOI, URL, Éditeur
- Le formulaire propose une sélection multiple des auteurs associés au document à l'aide de cases à cocher.
    - Il récupère la liste de tous les auteurs ($allAuthors).
    - Il vérifie si des auteurs ont été sélectionnés précédemment ($selectedAuthors).
    - Il affiche une case à cocher pour chaque auteur avec son nom complet.
    - Un lien permet d'ajouter un nouvel auteur ou de voir la liste des auteurs existants.
- Le formulaire propose la sélection d'un type de document existant à l'aide d'une liste déroulante.
    - Il récupère la liste de tous les types ($types).
    - Il affiche une option pour chaque type dans la liste déroulante.
    - Des liens permettent d'ajouter un nouveau type ou de voir la liste des types existants.

Soumission du formulaire

- Lorsque le bouton "Ajouter" est cliqué ($_POST['save']), le script récupère les données saisies dans le formulaire.
- Il appelle la méthode createWithAuthors de la classe Document pour créer un nouveau document en lui fournissant les informations du document et la liste des identifiants des auteurs sélectionnés.
- Un message de réussite ou d'erreur pourrait être affiché en fonction du résultat de la création du document (non implémenté dans le code fourni).

```php
$documents = Document::getAll();
$authors = Author::getAll();
$types = Type::getAll();
// Si le formulaire est soumis (c'est-à-dire si le bouton "Ajouter" est cliqué)
// If the form is submitted (i.e. if the "Ajouter" button is clicked)
if (isset($_POST['save'])) {
    // Récupérer les données du formulaire
    // Get the form data
    $namecomplet = $_POST['namecomplet'];
    $namecourt = $_POST['namecourt'];
    $startpage = $_POST['startpage'];
    $endpage = $_POST['endpage'];
    $yearpublication = $_POST['yearpublication'];
    $doi = $_POST['doi'];
    $url = $_POST['url'];
    $publisher = $_POST['publisher'];
```

```php
    $idtype = $_POST['idtype'];
    $authors = $_POST['authors'];

    // Créer un nouveau document avec les auteurs
    // Create a new document with the authors
    $documentId = Document::createWithAuthors($namecomplet, $namecourt, $startpage, $endpage, $yearpublication, $doi, $url, $publisher, $authors, $idtype);
    // afficher un message de succées ou d'erreur
    // display a success or error message
}

?>
```



8.2.3 Vue modif doc

8.2.3.1 Mise à jour des informations du document

- Nom complet, nom court, numéros de page, année de publication, DOI, URL, éditeur et type.

8.2.3.2 Gestion des auteurs associés

- Affichage des auteurs actuellement liés au document et possibilité de les modifier.
- Fourniture d'une liste à cases à cocher contenant tous les auteurs pour en sélectionner de nouveaux à associer.

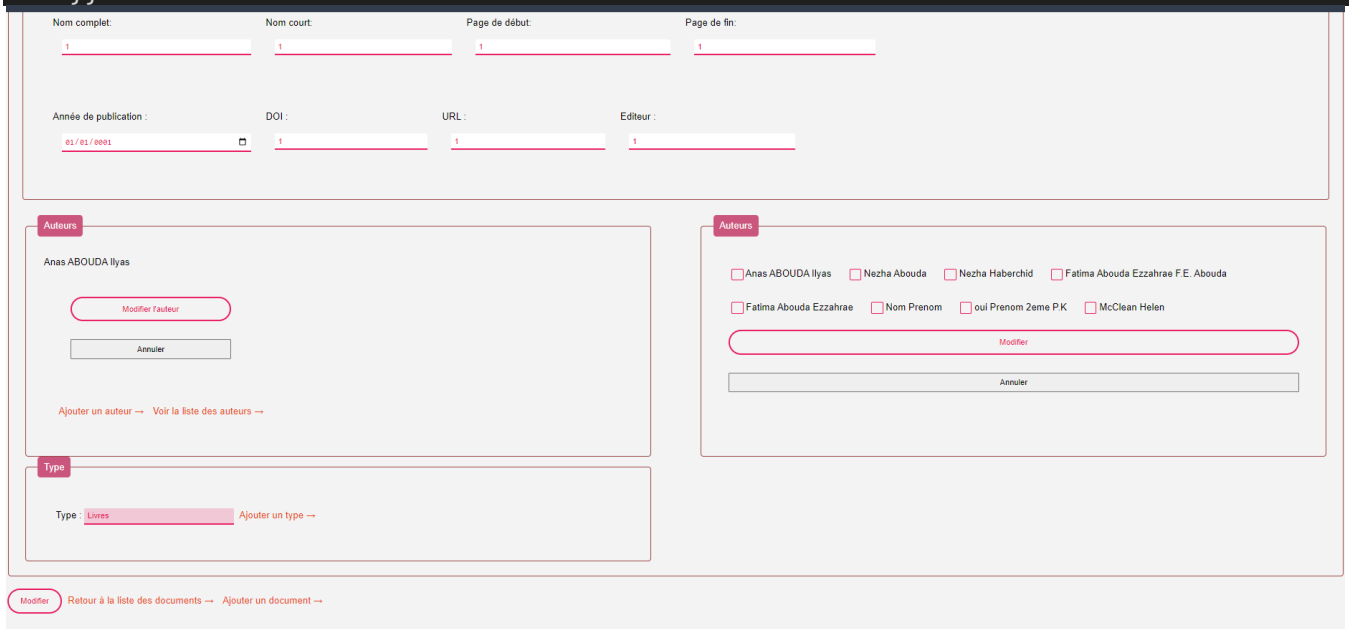
8.2.3.3 Fonctionnement détaillé

1. **Récupération des informations du document**
    - Le script récupère les détails du document à modifier en utilisant la fonction Document::getById($id).
2. **Gestion de la soumission de la modification du document**
    - Le code vérifie si le formulaire a été soumis en utilisant isset($_POST["update"]).
    - Si oui, il met à jour les informations du document dans la base de données avec la fonction document->update().

3. **Gestion de la modification des auteurs**
    - Le code vérifie si le bouton "Modifier l'auteur" a été soumis avec isset($_POST["mAuthor"]).
    - Si oui :
        - Il supprime les associations d'auteurs existantes avec document->removeAuthors($id).
        - Il ajoute les nouveaux auteurs sélectionnés à l'aide de document->addAuthor($id, $auteurs).
4. **Affichage des auteurs actuels**
    - Le script récupère la liste des auteurs associés au document avec document->getAuthors().
    - Il affiche ensuite leurs noms et prénoms.
5. **Options de modification des auteurs**
    - Un bouton "Modifier l'auteur" permet d'afficher la liste à cases à cocher de tous les auteurs.
    - Des liens supplémentaires sont fournis pour ajouter ou voir la liste complète des auteurs.
6. **Sélection des auteurs**
    - Le script récupère tous les auteurs de la base de données à l'aide de Author::getAll().
    - Il crée une liste à cases à cocher avec les noms des auteurs.
    - Les cases cochées correspondent aux auteurs déjà associés au document en cours de modification.
7. **Gestion du type de document**
    - Le code récupère tous les types de documents de la base de données avec Type::getAll().
    - Il affiche une liste déroulante permettant de sélectionner le type de document, avec l'option actuelle pré-sélectionnée.
8. **Soumission du formulaire**
    - Des boutons "Modifier" et "Mettre à jour" permettent d'enregistrer les modifications.
    - Des liens supplémentaires renvoient vers la liste des documents ou permettent d'en créer un nouveau.

```php
<?php
// si le formulaire est soumis pour la modification du document
// if the form is submitted for document update
if (isset($_GET['id'])) {
    $id = $_GET['id'];
    $document = Document::getById($id);
    $types = Type::getAll();
    $allAuthors = Author::getAll();
    $documentAuthors = $document->getAuthorsWithAssociation();
    // Récupérer les auteurs associés à ce document
    // Get the authors associated with this document
```

```
    $authors = array();
    // Si le formulaire est soumis pour la modification du document
    // If the form is submitted for document update
    if (isset($_POST["update"])) {
        $namecomplet = $_POST['namecomplet'];
        $namecourt = $_POST['namecourt'];
        $startpage = $_POST['startpage'];
        $endpage = $_POST['endpage'];
        $yearpublication = $_POST['yearpublication'];
        $doi = $_POST['doi'];
        $url = $_POST['url'];
        $publisher = $_POST['publisher'];
        $idtype = $_POST['idtype'];
        // Mettre à jour le document dans la base de données
        // Update the document in the database
        $document->update($id, $namecomplet, $namecourt, $startpage, $endpage,
$yearpublication, $doi, $url, $publisher, $idtype);
    }
    // Si le formulaire est soumis pour la modification des auteurs
    // If the form is submitted for author update
    if (isset($_POST["mAuthor"])) {
        // Si le bouton est cliqué, mettez à jour les auteurs associés
        // If the button is clicked, update the associated authors
        $authors = isset($_POST['authors']) ? $_POST['authors'] : array();
        // Supprimer les associations d'auteurs existantes pour ce document
        // Remove the existing author associations for this document
        $document->removeAuthors($id);
        // Ajouter les nouveaux auteurs
        // Add the new authors
        $document->addAuthor($id, $authors);
    }}
```



8.2.4 Vue supp document

8.2.4.1 Vérification de l'identifiant du document

- Le code vérifie si un identifiant de document numérique est passé via la variable GET.

8.2.4.2 Récupération du document

- Si l'identifiant est valide, le script utilise la fonction Document::getById($id) pour récupérer le document correspondant de la base de données.

8.2.4.3 Confirmation de la suppression

- Si le document est trouvé, le code affiche un message de confirmation avec les informations du document.
- Un formulaire est présenté avec un bouton "Confirmer la suppression".

8.2.4.4 Suppression du document

- Si l'utilisateur confirme la suppression, le code utilise la fonction Document::delete($id) pour supprimer le document de la base de données.

8.2.4.5 Redirection vers la page des documents

- Après la suppression, le code redirige l'utilisateur vers la page des documents.

```php
if (isset($_POST['confirmDelete'])) {
    $id = $_POST['id'];
    Document::delete($id);
}
```

8.2.5 Le reste des pages de vue

J'ai appliqué le même style de vue et les fonctionnalités de suppression pour le reste des pages, assurant une cohérence et une ergonomie optimale pour l'ensemble de l'interface.