Agathe POTEAUX SN1

Jordan MORITZ SN1

Batsien DEVIENNE SN1

Compte-rendu du projet : Conception et Développement d'une solution web

Administrateurs :

test.test@gmail.com

administrateur@epsi.fr

Elèves :

agathe.poteaux@epsi.fr

jordan.moritz@epsi.fr

bastien.devienne@epsi.fr

Sommaire :

Introduction :

I) Outils et base du site

Pour réaliser ce projet nous avons utilisé différents outils, comme GitHub, phpMyAdmin, le nuage pédagogique et nous sommes basé sur une arborescence assez simple pour le site :

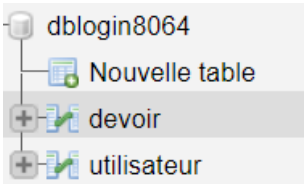
- Config --> contient les logs de la base de données ainsi que les routes qui permettent de se déplacer dans le site
- Public
  - Img
  - Css
  - Js
- Src --> contient les fonctions du site

    - Controller
    - Model
- Template --> contient les pages twig soit le visuel du site
- Vendor

Nous avons un fichier qui s'appelle rooter.php qui regarde le nom du Controller et si ce dernier est incorrect, il nous renvoie vers la page de connexion.

II) Base de données

La base de données dblogin8064 est constituée de deux tables, une table utilisateur qui nous permet d'enregistrer les différents utilisateurs du site et une table devoir qui nous permet d'ajouter des boîtes de rendu.



La table utilisateur :

#	Nom	Type	Interclassement	Attributs	Null	Valeur par défaut
1	**mail_utilisateur**	varchar(50)	utf8mb4_general_ci		Non	*Aucun(e)*
2	**nom_utilisateur**	varchar(50)	utf8mb4_general_ci		Non	*Aucun(e)*
3	**prenom_utilisateur**	varchar(50)	utf8mb4_general_ci		Non	*Aucun(e)*
4	**role_utilisateur**	int(11)			Non	*Aucun(e)*
5	**ecole**	int(50)			Non	*Aucun(e)*
6	**bts**	int(50)			Non	*Aucun(e)*
7	**classe**	int(50)			Non	*Aucun(e)*
8	**mdp**	varchar(255)	utf8mb4_general_ci		Non	*Aucun(e)*

La clé primaire de la table utilisateur est mail_utilisateur (de type varchar -> chaîne de caractère) car il n'est pas possible que deux personnes aient la même adresse email (deux homonymes peuvent être ajoutés en tant qu'utilisateurs). Les autres attributs de la table correspondent aux informations importantes d'un utilisateur. Il y a son nom, son prénom ((de type varchar -> chaîne de caractère), son rôle (0 ou 1 de type int) c'est-à-dire s'il est administrateur ou élève. Puis pour les autres attributs si le rôle est administrateur (0) alors ils auront tous pour valeur 0. Pour l'école, cela correspond à EPSI ou WIS (1, 2), pour le bts c'est l'option réseau ou développement (1, 2) et pour la classe c'est l'année (1, 2, 3, 4, 5). Aucun des attributs ne peut être NULL.

La table devoir :

#	Nom	Type	Interclassement	Attributs	Null	Valeur par défaut	Commentaires	Extra
1	**id_devoir**	int(50)			Non	*Aucun(e)*		AUTO_INCREMENT
2	**matiere**	varchar(50)	utf8mb4_general_ci		Oui	*NULL*		
3	**mail_utilisateur**	varchar(50)	utf8mb4_general_ci		Non	*Aucun(e)*		
4	**Date_fin**	date			Non	*Aucun(e)*		
5	**enonce**	varchar(255)	utf8mb4_general_ci		Non	*Aucun(e)*		
6	**ecole**	int(11)			Non	*Aucun(e)*		
7	**bts**	int(11)			Non	*Aucun(e)*		
8	**classe**	int(11)			Non	*Aucun(e)*		

La clé primaire de cette table est id_devoir qui est un nombre qui s'ajoute automatiquement (AUTO_INCREMENT). Il y a une clé étrangère mail_utilisateur qui est reliée à la clé primaire mail_utilisateur de la table utilisateur. Il y a un attribut Date_fin de type date (AAAA-MM-JJ) qui permet de définir la date limite pour rendre ce devoir. Il y a les attributs matière et énonce qui sont de type VARCHAR. Enfin les attributs ecole, bts et classe permettent de décider à quels élèves l'on souhaite donner ce devoir. On choisit la classe (1, 2, 3, 4, 5), l'école (1, 2) et si l'on souhaite que le devoir soit spécifique à une option de bts (1 ou 2) ou à l'ensemble des élèves de bts (3).

III) Ajout d'utilisateurs et connexion

Tout d'abord pour l'ajout d'utilisateur nous avons dû créer une fonction inscription :

Qui insère dans la base de données dans la table utilisateur les valeurs des colonnes, ensuite en fonction du nom des colonnes nous créons des variables leur correspondant, puis nous fermons la fonction. Pour chaque fonction qui appelle la base de données, on appelle une suite de variables qui sont les données que nous allons utiliser pour notre requête SQL. A chaque fois on prépare la requête dans une fonction $query puis on exécute cette requête en remplaçant les données nécessaires par les variables correspondantes. Ensuite nous allons créer un Controller qui permet de gérer les inscriptions, le voici :

```php
<?php
session_start();
use PHPMailer\PHPMailer\PHPMailer;
function addUserController($twig, $db)
{
    include_once '../src/model/ProjetModel.php';

    $erreur = "";
    if (isset($_POST['btnAdduser'])) {
        if ($_POST['mail_utilisateur'] != "" && $_POST['nom_utilisateur'] != "" && $_POST['prenom_utilisateur'] != "" && $_POST['mdp']) {


            $mail_utilisateur = htmlspecialchars($_POST["mail_utilisateur"]);
            $nom_utilisateur = htmlspecialchars($_POST["nom_utilisateur"]);
            $prenom_utilisateur = htmlspecialchars($_POST["prenom_utilisateur"]);
            $role_utilisateur = htmlspecialchars($_POST["role_utilisateur"]);
            $classe = htmlspecialchars($_POST["classe"]);
            $ecole = htmlspecialchars($_POST["ecole"]);
            $bts = htmlspecialchars($_POST["bts"]);
            $mdp = htmlspecialchars($_POST['mdp']);
            $mdp2 = htmlspecialchars($_POST['mdp2']);

            if ($mdp != $mdp2){
                $erreur = "mdp";
            }else{
                if (!testEmailExists($db, $mail_utilisateur)) {
                    inscription($db, $mail_utilisateur, $nom_utilisateur, $prenom_utilisateur, $role_utilisateur, $classe, $ecole, $bts, $mdp);
                    echo '<script>alert("Utilisateur correctement ajouté !");</script>';
                    $erreur = "bon";
                    $message = 'Bonjour,
                    voici vos identifiants de connexion pour votre espace de rendu.
                    Identifiant :'.$mail_utilisateur.'
                    Mot de passe : '.$mdp.'
                    Cordialement
                    Equipe EPSI/WIS';
                    $email = new Mail(); $email->envoyerMailer($mail_utilisateur, 'Vos identifiants de boite de rendu EPSI/WIS', $message, '');
                } else {
                    $erreur = "existe deja";
                }
            }

        } else {
            $erreur = "remplir champs";
        }
    }

    echo $twig->render('addUser.html.twig', ['erreur' => $erreur]);
}
```

Nous lançons une session puis importons le projet model contenant la fonction inscription, ensuite grâce à la méthode $_POST nous récupérons les valeurs des ‘name’ dans un formulaire comme ici le bouton : ‘btnAdduser’ puis on vérifie si le bouton est cliqué avec la méthode isset puis nous faisons en sorte de ne pas pouvoir inscrire d’utilisateur si un des champs ‘mail’, ‘prenom’, ‘nom’ et ‘mdp’ sont identiques.

Ensuite nous définissons les variables qui correspondent aux noms des input dans le formulaire twig, encore une fois grâce à la méthode $_POST.

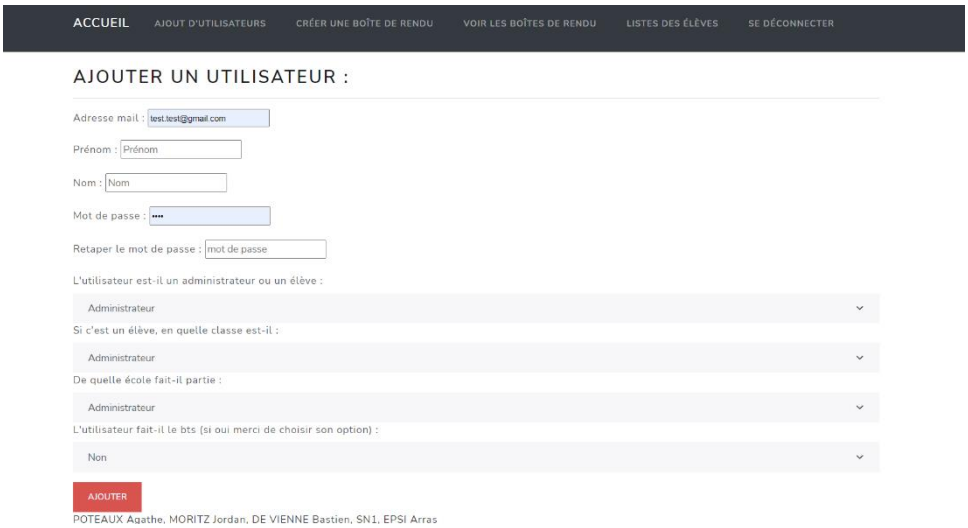
La ligne ‘if (!testEmailExists($db, $mail_utilisateur)) {‘ permet de tester si le mail existe et lui envoie un mail de confirmation de création de compte avec son login est son mot de passe qui d’ailleurs a été hash dans la fonction inscription afin que ce dernier soit crypté dans la base de données.

La variable erreur existe pour si jamais le mail est déjà dans la base de données ou si un des champs requis pour l’inscription est manquant.

La dernière ligne de ce Controller nous renvoie sur le fichier de la page twig, celle que l’on voit sur le site.

```
}
```

Cette fonction est là pour tester si le mail existe déjà dans la base de données, si le mail est déjà dans la base de données alors la fonction retourne True sinon elle retourne False



Pour la connexion :

```php
<?php
session_start();
function identificationController($twig, $db)
{
    $connexion = "";
    if (isset($_POST['btnConnexion'])) {
        if (isset($_POST['mailConnexion']) && isset($_POST['mdpConnexion'])) {
            if (filter_var($_POST['mailConnexion'], FILTER_VALIDATE_EMAIL)) {
                $email = $_POST['mailConnexion'];
                $loginUser = $db->prepare("SELECT * FROM utilisateur WHERE mail_utilisateur = :mail_utilisateur");
                $loginUser->execute([
                    "mail_utilisateur" => $email
                ]);
                $user = $loginUser->fetch();

                if ($user) {
                    $password = $_POST['mdpConnexion'];
                    if (password_verify($password, $user['mdp'])) {
                        $_SESSION['user']['mail'] = $user['mail_utilisateur'];
                        $_SESSION['user']['prenom'] = $user['prenom_utilisateur'];
                        $_SESSION['user']['role'] = $user['role_utilisateur'];
                        $_SESSION['user']['bts'] = $user['bts'];
                        print_r('bonjour ' . $_SESSION['user']['prenom'] . '!');
                        print_r('bonjour ' . $_SESSION['user']['mail'] . '!');
                        print_r($_SESSION['user']['role']);
                        $connexion = "bon";



                        if ($_SESSION['user']['role'] === 0) {
                            header('location: /projet/?page=homeadmin');
                        } else {
                            $_SESSION['user']['classe'] = $user['classe'];
                            $_SESSION['user']['ecole'] = $user['ecole'];
                            $_SESSION['user']['bts'] = $user['bts'];
                            header('location: /projet/?page=home');
                        }
                    }else{
                        $connexion = "erreur";
                    }
                }else{
                    $connexion = "erreur";
                }
            }else{
                $connexion = "erreur";


            }
        }
    }
    echo $twig->render('identification.html.twig', ["connexion" => $connexion]);
```

Donc c'est une succession de test avec le mot clé 'if', tout d'abord on regarde si le bouton 'btnConnexion' a été cliqué puis si les champs 'mailConnexion' et 'mdpConnexion' aussi.

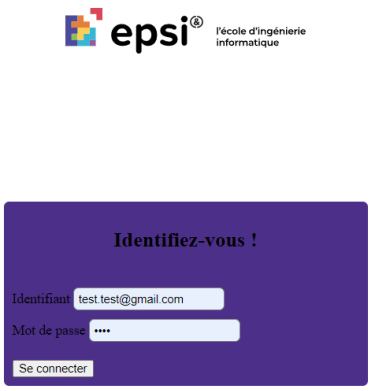
Ensuite on vérifie si l'adresse mail rentré est valide avec le filter_var(…, FILTER_VALIDATE_EMAIL)

On associe ensuite à la variable $email, le contenu de la case 'mailConnexion' puis en fonction de l'email on sélectionne toutes les valeurs correspondant à chaque colonne dans la base de données

Ensuite si la variable $user est valide, la variable $password est égal au mdp rentrer sur la page de connexion, si le mdp rentré correspond à celui qui est hash dans la base de données alors on donne à la session plusieurs valeurs de la base de données tels que mail, prenom, role et bts.

Ensuite, si le role de l'utilisateur est 0 soit administrateur alors il est redirigé sur la page d'accueil des admins sinon si les informations correspondent entre celle de la bdd et celle donné à la session, la personne est redirigée sur la page accueil des élèves.

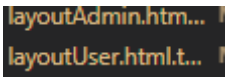
La dernière ligne nous renvoie vers la page twig du formulaire de connexion.



IV) Distinction des pages et déconnexion

A) distinction des pages :

Pour la distinction des pages nous avons dû créer des pages servant à la même chose mais en les appelant différemment pour qu'un élève ne se retrouve pas sur le site côté admins, pour organiser tout ça grâce à twig il nous faut créer plusieurs layout, un pour les élèves et un pour les admins.



Ces deux pages contiennent un menu navigable cependant les routes sont différentes.

LayoutAdmin:

```
<body class="d-flex flex-column min-vh-100">
    <nav class="navbar navbar-expand-lg navbar-dark bg-dark">
        <div class="container">
            <a class="navbar-brand" href="?page=homeadmin">Accueil</a>
            <button class="navbar-toggler" type="button" data-bs-toggle="collapse" data-bs-target="#navbarResponsive" aria-expanded="false" aria-label="toggle Navigation">
                <span class="navbar-toggler-icon"></span>
            </button>
            <div class="collapse navbar-collapse" id="navbarResponsive">
                <ul class="navbar-nav" mb-2 mb-lg-0>
                    <li class="nav-item">
                        <a href="?page=addUser" class="nav-link">
                            Ajout d'utilisateurs</a>
                    </li>
                    <li class="nav-item">
                        <a href="?page=creationBoite" class="nav-link">
                            Créer une boîte de rendu</a>
                    </li>
                    <li class="nav-item">
                        <a href="?page=voirDevoirsAdmin" class="nav-link">
                            Voir les boîtes de rendu</a>
                    </li>
                    <li class="nav-item">
                        <a href="?page=voireleves" class="nav-link">
                            Listes des élèves</a>
                    </li>
                    <li class="nav-item">
                            <a href="?page=profil" class="nav-link">
                                Profil</a>
                        </li>
                    <li class="nav-item">
                        <a href="?page=deconnexion" class="nav-link">
                            Se déconnecter</a>
                    </li>
                </ul>
            </div>
        </div>
    </nav>
```

LayoutUser :

```html
<!DOCTYPE html>
<html lang="en">
<head>

    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <meta name="description" content="Page utilisateur">
    <meta name="Authors" content="Agathe, Jordan, Bastien">

    <link rel="stylesheet" href="style3.css">
    <title>Page utilisateur</title>
</head>
<body>
    <nav class="navbar navbar-expand-lg navbar-dark bg-dark">
        <div class="container">
            <a class="navbar-brand" href="?page=home">Accueil</a>
            <button class="navbar-toggler" type="button" data-bs-toggle="collapse" data-bs-target="#navbarResponsive" aria-expanded="false" aria-label="toggle Navigation">
                <span class="navbar-toggler-icon"></span>
            </button>
            <div class="collapse navbar-collapse" id="navbarResponsive">
                <ul class="navbar-nav" mb-2 mb-lg-0>
                    <li class="nav-item">
                        <a href="?page=voirDevoirsEleve" class="nav-link">Devoirs</a>
                    </li>
                    <li class="nav-item">
                        <a href="?page=profil" class="nav-link">Profil</a>
                    </li>
                    <li class="nav-item">
                        <a href="?page=deconnexion" class="nav-link">Se déconnecter</a>
                    </li>
                </ul>
            </div>
        </div>
    </nav>
</body>
<footer class="card-footer  text-dark mt-auto">
    <p>POTEAUX Agathe, MORITZ Jordan, DEVIENNE Bastien, SN1, EPSI Arras
    <br />
    </p>
</footer>
</html>
```

Déconnexion :

```php
<?php

function deconnexioncontroller ($twig) {
        session_start();
        $_SESSION = array();
        session_destroy();
        unset($_SESSION);
        echo $twig->render('deconnexion.html.twig',[]);
        header('location: /projet/?page=connexion');

    }
```

Dans le Controller de la déconnexion qui est relié au bouton de déconnexion de la barre de navigation, on utiles les propriétés de la variable globale $_SESSION de PHP pour se déconnecter. On utilise la fonction session_destroy() qui permet d'arrêter la connexion à la session et la fonction unset qui permet de supprimer les informations contenues dans la variable SESSION. Ensuite on renvoie vers la page de connexion grâce au header.

V) Ajout et affichage des boites de rendu

A) Ajout d'un devoir dans la base de données

Il s'agit de la même fonction que pour ajouter des utilisateurs à 2-3 choses prêts, tout d'abord ce n'est pas la même table, les mêmes colonnes etc, voici la fonction :

```php
function creation($db, $enonce, $matiere, $mail_utilisateur, $Date_fin, $ecole, $bts, $classe)
{
    $query = $db->prepare("INSERT INTO devoir (enonce, matiere, mail_utilisateur, Date_fin, ecole, bts, classe) VALUES (:enonce, :matiere, :mail, :Date_fin, :ecole, :bts, :classe) ");
    return $query->execute([
        'enonce' => $enonce,
        'matiere' => $matiere,
        'mail' => $mail_utilisateur,
        'Date_fin' => $Date_fin,
        'ecole' => $ecole,
        'bts' =>$bts,
        'classe' => $classe,


    ]);
```

Cette fonction insère dans la bdd les informations nécessaires contenues dans des variables aux devoirs comme l'énonce, la matière, la classe visée etc...

Passons au Controller :

```php
<?php
session_start();
function creationboiteController($twig, $db) {

    include_once '../src/model/ProjetModel.php';


    if (isset($_POST['btnCreation'])) {
        $enonce = htmlspecialchars($_POST["enonce"]);
        $matiere = htmlspecialchars($_POST["matiere"]);
        $Date_fin = htmlspecialchars($_POST["Date_fin"]);
        $mail_utilisateur = $_SESSION['user']['mail'] ;
        $ecole = htmlspecialchars($_POST["ecole"]);
        $bts = htmlspecialchars($_POST["bts"]);
        $classe = htmlspecialchars($_POST["classe"]);

        creation($db, $enonce, $matiere, $mail_utilisateur, $Date_fin, $ecole, $bts, $classe);

    }
    echo $twig->render('creationBoite.html.twig', []);


}
```

Qui comme le Controller d'ajout d'utilisateur importe le fichier ou est contenu la fonction, regarde si le bouton 'btnCreation' a été cliqué puis associe à chaque variable les valeurs rentrées dans les champs du formulaire twig pour ensuite les insérer dans la bdd. On remarque également que pour la variable $mail_utilisateur on récupère une information de la variable globale PHP $_SESSION (ici le mail) ce qui est très utile car cela nous permet d'associer la création d'un devoir à un créateur et par conséquent de regrouper plus tard les devoirs par créateurs en plus de pouvoir les regrouper par classe, options, école, …

B) Affichage des devoirs pour les administrateurs

Il y a une distinction entre l'affichage des devoirs pour les administrateurs et les élèves. Les administrateurs peuvent voir les boîtes de rendu que ces derniers ont créées alors que les élèves voient les devoirs auxquels ils ont été assignés.

Pour l'affichage des boîtes de rendu des administrateurs on commence par une fonction qui exécute une requête SQL nous permettant d'aller chercher dans la base de données tous les devoirs qui ont comme mail_utilisateur (c'est-à-dire comme créateur), le même mail que l'administrateur connecté.

```php
function voirDevoirsAdmin($db, $mail_utilisateur)
{
    $query = $db->prepare("SELECT id_devoir, enonce, matiere, Date_fin, ecole, bts, classe FROM devoir WHERE mail_utilisateur = :mail_utilisateur");

    $query->execute([
        'mail_utilisateur' => $mail_utilisateur,
    ]);

    $DevoirsAdmin = $query->fetchAll();

    return $DevoirsAdmin;
}
```

On sélectionne toutes les informations des devoirs où le mail utilisateur est correspondant à celui enregistré dans la variable globale $_SESSION.

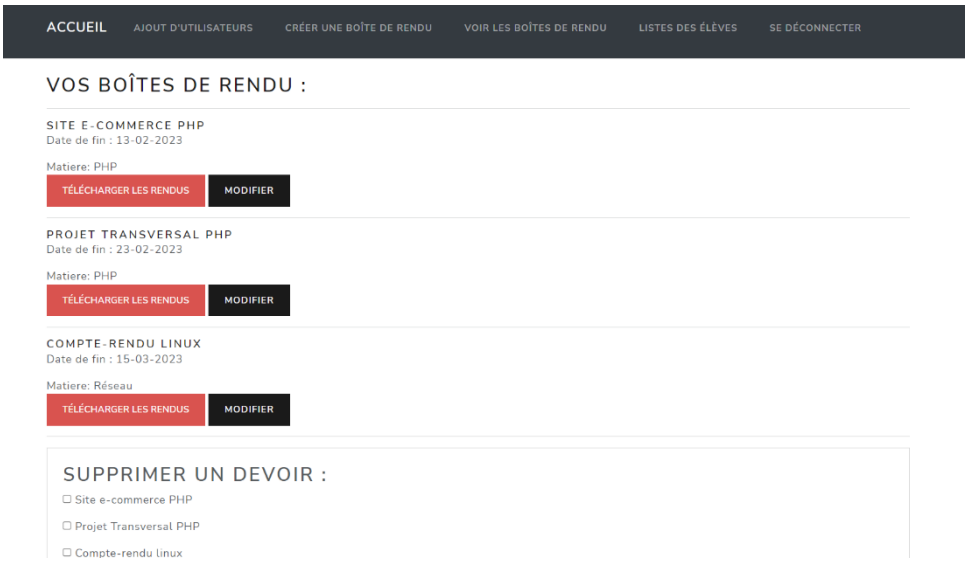
On crée un controller voirDevoirsAdminController.php qui définit la variable mail_utilisateur, qui met dans la variable $DevoirsAdmin, le résultat de la requête SQL puis qui renvoie cette variable dans un twig voirDevoirsAdmin.html.twig pour l'affichage.

```php
<?php
session_start();
include_once '../src/model/ProjetModel.php';
function voirDevoirsAdminController($twig, $db)
{
    $mail_utilisateur = $_SESSION['user']['mail'];
    $DevoirsAdmin = voirDevoirsAdmin($db, $mail_utilisateur);
```

```php
echo $twig->render('voirDevoirsAdmin.html.twig', ["Devoirs" => $DevoirsAdmin]);
```

Dans le fichier twig, on réalise une boucle pour sélectionner toutes les valeurs de DevoirsAdmin, c'est-à-dire tous les devoirs ({% for Devoir in Devoirs %}). Puis pour chaque valeur de Devoir, on affiche d'abord l'énoncé du devoir avec {{Devoir.enonce}} (après le . on marque le nom de l'attribut dans la base de données que l'on souhaite sélectionner). Puis on affiche la matière du devoir et la date de fin avec un pipe |date('d-m-Y') qui nous permet la date au format JJ/MM/AAAA au lieu de AAAA/MM/JJ comme enregistré dans la base de données (format de phpMyAdmin).

```twig
{% extends "layoutAdmin.html.twig" %}

{% block title %}
Voir les Boites de rendus
{% endblock %}

{% block content %}
<h2>Vos boîtes de rendu :</h2>
<hr>
    {% for Devoir in Devoirs %}
        <div>
            <h5 class="card-title">{{Devoir.enonce}}</h5>
            <p class="card-text">Date de fin :
                {{Devoir.Date_fin|date('d-m-Y')}}</p>
                <p class="card-text">Matiere:
                {{Devoir.matiere}}</p>
        </div>
        <form         method="post" id="{{Devoir.id_devoir}}">
            <button type="submit" class="btn btn-danger" name="telecharger" value="{{Devoir.id_devoir}}">Télécharger les rendus</button>
            <a href="?page=changeDevoirs&id={{ Devoir.id_devoir }}" target="blank" class="btn btn-primary">modifier</a>
        </form>
        <hr>
    {% endfor %}
```



C) Affichage des devoirs pour les élèves

Pour les élèves, le principe est le même, on vient simplement changer les critères de sélection des devoirs dans la bdd.

```php
function voirDevoirsEleve($db, $classe, $ecole, $date_auj, $bts)
{
    $query = $db->prepare("SELECT id_devoir, enonce, matiere, Date_fin, ecole, bts, classe FROM devoir WHERE classe = :classe AND ecole = :ecole AND
    bts = :bts OR bts =3 AND Date_fin >= :date_auj");

    $query->execute([
        'classe' => $classe,
        'ecole' => $ecole,
        'date_auj' => $date_auj,
        'bts' => $bts
    ]);
```

Dans cette fonction, on change les informations que l'on recherche dans le WHERE, en mettant cette fois-ci la classe, l'école, l'option de bts ou s'il n'y a pas d'option en particulier et si la date du devoir n'est pas dépassée.

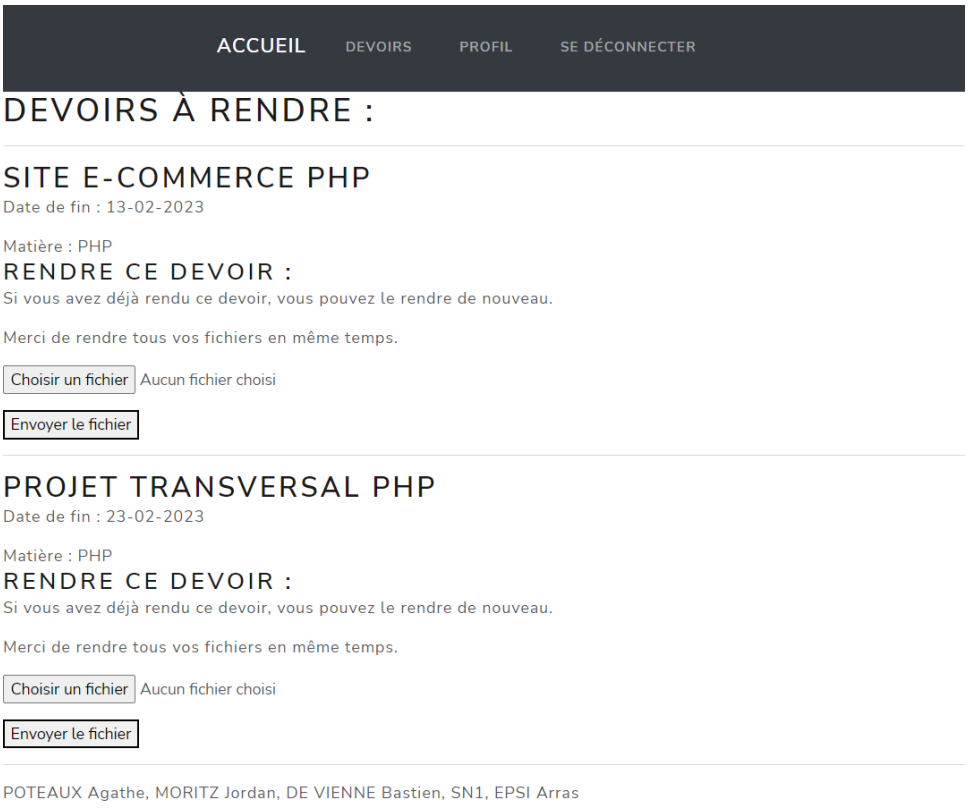
```php
<?php
session_start();
include_once '../src/model/ProjetModel.php';
function voirDevoirsEleveController($twig, $db) {
    $classe = $_SESSION['user']['classe'] ;
    $ecole = $_SESSION['user']['ecole'] ;
    $bts = $_SESSION['user']['bts'];
    $date_auj = date('Y/m/d');
    $DevoirsEleve = voirDevoirsEleve($db, $classe, $ecole, $date_auj, $bts);
echo $twig->render('voirDevoirsEleve.html.twig', ["Devoirs"=>$DevoirsEleve,]);
```

Dans le controller voirDevoirsEleveController, on définit les variables que l'on va appeler l'on de la requête SQL grâce encore une fois aux informations enregistrées dans la variable $_SESSION au moment de la connexion de l'élève. On vient également enregistrer la date d'aujourd'hui grâce à la fonction date de PHP qui nous permet de voir si la date du devoir est antérieure ou non à celle d'aujourd'hui. On rentre le résultat de la requête SQL dans la variable $DevoirsEleve que l'on dépose dans le echo.

Enfin dans le twig voirDevoirsEleve.html.twig :

```
{% block content %}
    <h1>Devoirs à rendre :
    </h1>
    <hr>
    {% for Devoir in Devoirs %}
        <div>

            <h2 class="card-title">{{Devoir.enonce}}</h2>
            <p type="date">Date de fin :
                {{Devoir.Date_fin|date('d-m-Y')}}</p>
            <p class="card-text">Matière :
                {{Devoir.matiere}}</p>
        </div>
        <h3 class="card-title">Rendre ce devoir :</h3>
        <p>Si vous avez déjà rendu ce devoir, vous pouvez le rendre de nouveau.</p>
        <p class="card-text">Merci de rendre tous vos fichiers en même temps. </p>
        <form enctype="multipart/form-data" method="post"  id="{{Devoir.id_devoir}}">
            <input type="file" name="fichier"/>
        </form>
        <button type="submit" name="rendre" value="{{Devoir.id_devoir}}" name="rendre" form="{{Devoir.id_devoir}}">Envoyer le fichier</button>
        <HR>


    {% endfor %}
```

On vient de nouveau afficher toutes les informations des devoirs avec une boucle for.



VI) Upload des fichiers

Après avoir affiché les boîtes de rendus pour les élèves, il faut qu'ils puissent envoyer leur(s) rendu(s). Tout d'abord, on crée dans le dossier public du projet, un dossier rendus qui permet de stocker dans des dossiers les rendus des devoirs. Dans le twig, voirDevoirsEleve.html.twig on vient ajouter un formulaire dans la boucle de chaque devoir qui permet d'enregistrer des fichiers sur le serveur. On vient relier ce formulaire au Controller voirDevoirsEleves controller :

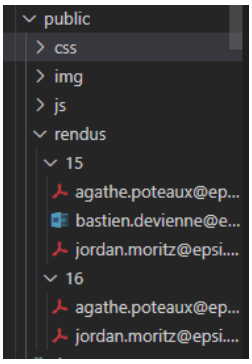
```php
function voirDevoirsEleveController($twig, $db) {
    $classe = $_SESSION['user']['classe'] ;
    $ecole = $_SESSION['user']['ecole'] ;
    $bts = $_SESSION['user']['bts'];
    $date_auj = date('Y/m/d');
    $DevoirsEleve = voirDevoirsEleve($db, $classe, $ecole, $date_auj, $bts);
    $id_devoir = $_POST["rendre"];
    if (is_uploaded_file($_FILES['fichier']['tmp_name'])){
            $nomOrigine = $_FILES['fichier']['name'];
            $elementsChemin = pathinfo($nomOrigine);
            $extensionFichier = $elementsChemin['extension'];
                $extensionsAutorisees = array("docx", "pdf", "zip", "7zip", "rar");
                if (!(in_array($extensionFichier, $extensionsAutorisees))) {
                    echo '<script>alert("Type de fichier invalide. Uniquement les fichiers Powerpoint (.ppt/.pptx), Zip (.zip) et PDF (.pdf) sont acceptés");</script>';
                } else {
                    $repertoireDestination = dirname(__FILE__)."/../../public/rendus/".$id_devoir."/";
                    if (file_exists($repertoireDestination)) {
                        echo '<script>alert("Transfert de vos fichiers en cours");</script>';
                    } else {
                        mkdir($repertoireDestination);
                    }
                    $nomDestination =$_SESSION['user']['mail']."_".$id_devoir.".".$extensionFichier;

                    if (move_uploaded_file($_FILES["fichier"]["tmp_name"],$repertoireDestination.$nomDestination)) {
                        echo '<script>alert("Fichier correctement transmis");</script>';
                    } else {
                        echo '<script>alert("Erreur de transmission");</script>';
                    }

                }
            }
            echo $twig->render('voirDevoirsEleve.html.twig', ["Devoirs"=>$DevoirsEleve,]);
        }
```

On commence par mettre dans la variable $id_devoir l'identifiant du devoir pour lequel on souhaite rendre un rendu, que l'on récupère grâce à la valeur du bouton rendre sur le formulaire. Ensuite on regarde si un fichier a été déposé dans la variable $_FILES grâce au formulaire, puis on regarde si l'extension du fichier correspond à ceux que l'on accepte (docx, pdf, zip, etc), si ce n'est pas le cas alors on renvoie une alerte qui explique l'extension est incorrecte. Ensuite on regarde si le répertoire de la boîte de rendu existe déjà (son nom correspond à son identifiant), s'il ne l'est pas on le crée avec mkdir et on ajoute dans ce dossier, le fichier récupéré avec comme nom, l'adresse mail de l'utilisateur et l'id du devoir pour que le fichier soit unique. Si tout se passe correctement alors on envoie une alerte de réussite, sinon une alerte d'erreur.

```twig
{% extends 'layoutUser.html.twig' %}


{% block content %}
    <h1>Devoirs à rendre :
    </h1>
    <hr>
    {% for Devoir in Devoirs %}
        <div>

            <h2 class="card-title">{{Devoir.enonce}}</h2>
            <p type="date">Date de fin :
                {{Devoir.Date_fin|date('d-m-Y')}}</p>
            <p class="card-text">Matière :
                {{Devoir.matiere}}</p>
        </div>
        <h3 class="card-title">Rendre ce devoir :</h3>
        <p>Si vous avez déjà rendu ce devoir, vous pouvez le rendre de nouveau.</p>
        <p class="card-text">Merci de rendre tous vos fichiers en même temps. </p>
        <form enctype="multipart/form-data" method="post"  id="{{Devoir.id_devoir}}">
            <input type="file" name="fichier"/>
        </form>
        <button type="submit" name="rendre" value="{{Devoir.id_devoir}}" name="rendre" form="{{Devoir.id_devoir}}">Envoyer le fichier</button>
        <HR>


    {% endfor %}
{% endblock %}
```

Dans le twig, on voit le formulaire qui contient un input de type « file » qui lance un explorateur de fichier pour sélectionner le fichier que l'on souhaite. Et le bouton « rendre » qui a pour valeur l'identifiant du devoir qui permet d'envoyer le fichier vers le bon dossier.

Résultat dans l'arborescence du site :



VII) Téléchargement des rendus pour les administrateurs

```php
if (isset($_POST['telecharger'])) {
    $id_devoir = $_POST["telecharger"];
    $zip = new ZipArchive();
    if ($zip->open('rendus/'.$id_devoir.'/zip.zip', ZipArchive::CREATE) === true) {
        echo '&amp;quot;' . $mail_utilisateur . '&amp;quot; ouvert<br/>';

        $zip->addEmptyDir('newDirectory');

        $pathdir='rendus/'.$id_devoir.'/';

        $dir = opendir($pathdir);

        while($file = readdir($dir)) {
            if(is_file($pathdir.$file)) {
                $zip -> addFile($pathdir.$file, $file);
            }
        }
        $zip->close();
        echo $twig->render('telecharger.html.twig',['lien'=> 'rendus/'.$id_devoir.'/zip.zip' ]);
    } else {
        echo 'Impossible d&amp;#039;ouvrir &amp;quot;Zip.zip<br/>';
    }

}
```

On vérifie si le bouton ‘telecharger’ est cliqué puis on stock dans $id_devoir la valeur du formulaire ‘telecharger’ puis on crée une nouvelle archive zip appelé $zip.

Puis on crée l’archive dans le répertoire rendus/ $id_devoir / nom_de_devoir

Ensuite on récupère le chemin du devoir puis on lit ce chemin et on le stocke dans $dir, on parcourt les fichiers du répertoire

```twig
<a href="{{ lien }}" target="_blank">Télécharger les fichiers. </a>
```

Cette page twig nous renvoie un lien cliquable qui si l’on clique dessus nous télécharge les rendus de tous les devoirs des élèves, bien évidemment les fichiers télécharger sont sous format zip.

Cependant pendant la phase de développement nous avons eu un problème de droit, le téléchargement ne nous était pas autorisé, nous avons donc dû rentrer une ligne de commande avec root : ‘chown -R www-data:www-data rendus’ qui change le propriétaire du dossier rendus au login8064.

```
root@HTTPS-8064:/var/www/html/projet/public# chown -R www-data:www-data rendus
```

VIII) Affichage en arborescence des devoirs pour les administrateurs

Sur la page d'accueil des administrateurs, on souhaite afficher en arborescence les boites de rendus et leurs contenus. Pour cela dans le homeAdminController.php :

```php
<?php
 session_start();
include_once '../src/model/ProjetModel.php';
 function homeadminController($twig, $db) {
  $prenom = $_SESSION['user']['prenom'];
   function mkmap($dir){
     if (file_exists($dir)) {
       $folder = opendir ($dir);
       $arbo ="";
       while ($file = readdir ($folder)) {
         if ($file != "." && $file != "..") {
           $pathfile = $dir.'/'.$file;
           $arbo .= "<li><a target='blank' href=$pathfile>$file</a></li>";

           if(filetype($pathfile) == 'dir'){
             mkmap($pathfile);
            }
          }
        }

        closedir ($folder);
        return $arbo;

    }

  }
```

On vient créer une fonction mkmap qui prend en argument un chemin de dossier et vient, tout à tour, donner le lien des fichiers contenus dans le dossier. Pour pouvoir afficher cette arborescence correctement dans le twig, on vient créer une variable $arbo en dehors de cette fonction qui va contenir du code html. On commence par mettre dans cette variable « Vos boites de rendus » puis on va venir réaliser des concaténations sur cette variable pour ajouter du code html au fur et à mesure. On ne peut pas mettre directement l'arborescence du dossier rendu car on ne doit afficher que les boites créées par l'administrateur.

```php
  $id = $_SESSION['user']['mail'];
  $listeDevoirs = voirIdDevoir($db, $id);
  $arbo = "<li> Boîtes de rendus: </li>";
  for ($i = 0; $i < count($listeDevoirs); $i++) {
    $id_devoir =  $listeDevoirs[$i]['id_devoir'];
    $nom_devoir =  $listeDevoirs[$i]['enonce'];
    $arbo .= "<ul> <li><a target='blank' href='/../../projet/public/rendus/$id_devoir/'>$nom_devoir</a></li><ul>";
    $arbo .= mkmap("../../projet/public/rendus/$id_devoir/");
    $arbo .= "</ul> </ul>";
  }
ho $twig->render('homeadmin.html.twig', ['arbo' => $arbo]);
```

C'est pourquoi on enregistre dans la variable $listeDevoirs, tous les identifiants des devoirs créés par l'administrateur grâce à la fonction voirIdDevoir qui prend en argument le courriel de l'administrateur (même principe que pour l'affichage des devoirs).

```php
function voirIdDevoir($db, $mail_utilisateur)
{
    $query = $db->prepare("SELECT id_devoir, enonce FROM devoir WHERE mail_utilisateur = :mail_utilisateur");

    $query->execute([
        'mail_utilisateur' => $mail_utilisateur,
    ]);

    return $query->fetchAll();
}
```

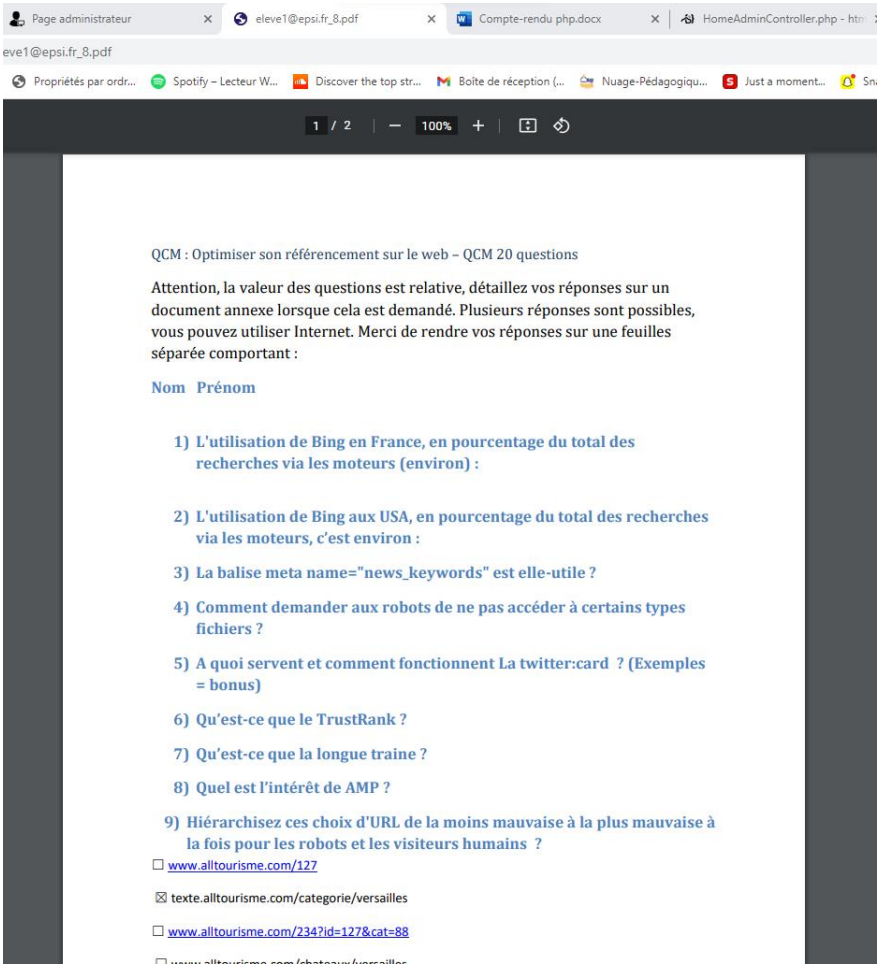
Ainsi on vient réaliser une boucle qui passe par tous les dossiers des rendus que l'administrateur a créé et dans chaque dossier passe par touts les documents. On utilise la balise <ul> pour faire un alinéa (et pour enlever un alinéa) et la balise <li> pour aller à la ligne et réaliser une liste. Une fois cette boucle terminée, on passe la variable $arbo dans le twig grâce au echo.

Dans le twig, on affiche le contenu de la variable $arbo en mettant un pipe et raw ce qui fait que le site lit le contenu de la variable en tant que langage html et non en tant que chaîne de caractères :

```twig
{% endblock %}

{% block content %}
    <div class='text text-primary'>
        <h1>Bienvenue sur votre espace de création de boîtes de rendu !</h1>
        <p>Bonjour {{prenom}} !
        </p>

    </div>
    <div id="content">
        {{arbo|raw}}
    </div>


{% endblock %}
```

Depuis les liens des devoirs par exemple ici 'eleve1@epsi.fr_8.pdf' on peut ouvrir les devoirs rendus des élèves directement dans une nouvelle fenêtre grâce au target='blank' contenu dans la variable arbo.

Exemple d'ouverture d'un fichier :



IX) Visuel et envoie de mail

A) Visuel

Pour le visuel, à part pour la page identification, on utilise un lien bootstrap qui ajoute les couleurs et la mise en page.

```
<link href="https://bootswatch.com/5/lux/bootstrap.min.css" rel="stylesheet">
```

Pour le visuel de la page d'identification, on fait appel dans le html de la page à un fichier css :

```
<!DOCTYPE html>
<html lang="en">
<head>

    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <meta name="description" content="Page de connexion pour personnel de l'EPSI et apprennant">
    <meta name="Authors" content="Agathe, Jordan, Bastien">

    <title>Page de connexion</title>

    <link rel="icon" type="image/gif" href="https://www.epsi.fr/wp-content/uploads/2017/04/Notre-futur-campus-en-video-!-101483_reference.png" />
    <link rel="stylesheet" href="/projet/css/style1.css">
```

```css
body {
    display: flex;
    align-items: center;
    justify-content: center;
    height: 100vh;
    background-color: white;
}

section {
    background-color: rgb(76, 47, 137);
    padding: 10px;
    display: flex;
    flex-direction: column;
    width: 400px;
    border-radius: 6px;
}
section h1 {
    text-align: center;
}
section p {
    color: red;
}
form {
    display: flex;
    flex-direction: column;
}
form input {
    margin: 5px 0;
    padding: 5px 5px;
    outline: 0;
    border: 1px solid grey;
    border-radius: 6px;
}
```

Dans la parie section, on vient créer un encadrement et mettre un fond violet. On vient mettre les titre (h1) au centre tout comme les éléments du body.

B) Envoie de mail

Lorsque l'on crée un utilisateur, on lui transmet par mail son identifiant ainsi que son mot de passe.

Pour cela, dans le terminal, en root on réalise :

apt install postfix

cd projet

composer require phpmailer/phpmailer

Ce qui nous permet d'installer la librairie phpmailer dans notre composer pour qu'on puisse envoyer des mails.

Dans ProjetModel on vient créer une class Mail (class étudiée en cours) qui va nous permettre d'envoyer un mail :

```php
class Mail {
    public function __construct() {}
    public function envoyerMailer($destinataire, $sujet, $message, $piecejointe){
        $mail = new PHPMailer\PHPMailer\PHPMailer();
                $mail->isSMTP(); // Paramétrer le Mailer pour utiliser SMTP
                $mail->Host = 'smtp.office365.com'; // Spécifier le serveur SMTP
                $mail->SMTPAuth = true; // Activer authentication SMTP
                $mail->Username = 'boitederendu@outlook.fr'; // Votre adresse email d'envoi
                $mail->Password = 'Epsi.2022'; // Le mot de passe de cette adresse email
                $mail->SMTPSecure = 'tls'; // Accepter SSL
                $mail->Port = 587;
                $mail->setFrom('boitederendu@outlook.fr', 'Rendu'); // Personnaliser l'envoyeur
                $mail->addAddress($destinataire);
                if(!empty($piecejointe)){
                    $mail->addAttachment($piecejointe); // Ajouter un attachement
                }
                    $mail->isHTML(true); // Paramétrer le format des emails en HTML ou non
                    $mail->Subject = $sujet;
                    $mail->Body = $message;
                    if(!$mail->send()) {
                        echo 'Mailer Error: ' . $mail->ErrorInfo;
                    } else {
                    }}}
```

Cette classe contient une fonction qui prend en paramètres, un destinataire, un sujet, un message et si l'on souhaite, une pièce jointe. On vient paramétrer le serveur puis on enregistre comme adresse email d'envoi notre adresse email (créée pour le projet) en tant que username. On met dans password le mot de passe de cette adresse mail, puis si la variable pièce jointe est présente alors on l'ajoute au mail, on définit l'objet et le contenu du mail dans subject et body. Enfin on envoie le mail et on affiche une erreur s'il y a un problème d'envoi.

Cette fonction est appelée lors de la création d'un utilisateur (dans addUserController.php) :

```php
}else{
    if (!testEmailExists($db, $mail_utilisateur)) {
        inscription($db, $mail_utilisateur, $nom_utilisateur, $prenom_utilisateur, $role_utilisateur, $classe, $ecole, $bts, $mdp);
        echo '<script>alert("Utilisateur correctement ajouté !");</script>';
        $erreur = "bon";
        $message = 'Bonjour,
        voici vos identifiants de connexion pour votre espace de rendu.
        Identifiant :'.$mail_utilisateur.'
        Mot de passe : '.$mdp.'
        Cordialement
        Equipe EPSI/WIS';
        $email = new Mail(); $email->envoyerMailer($mail_utilisateur, 'Vos identifiants de boite de rendu EPSI/WIS', $message, '');
    } else {
        $erreur = "existe deja";
    }
```

Dans le message du mail, on donne l'identifiant et le mot de passe du site. On définit le destinataire avec le mail_utilisateur et on l'envoie.

X) Améliorations futures

Plusieurs améliorations sont possibles pour ce projet :

- Nous pourrions par exemple utiliser cron qui nous permettrait d'envoyer des mails lorsqu'il ne reste plus que 24 heures à un projet.
- Nous pourrions envoyer des mails aux élèves lorsqu'ils ont été attribués à un devoir

- Il serait préférable d’afficher lorsque l’on a déjà rendu un devoir (pour cela il faudrait ajouter une table à la base de données)
- Nous pouvons améliorer le visuel