POTEAUX

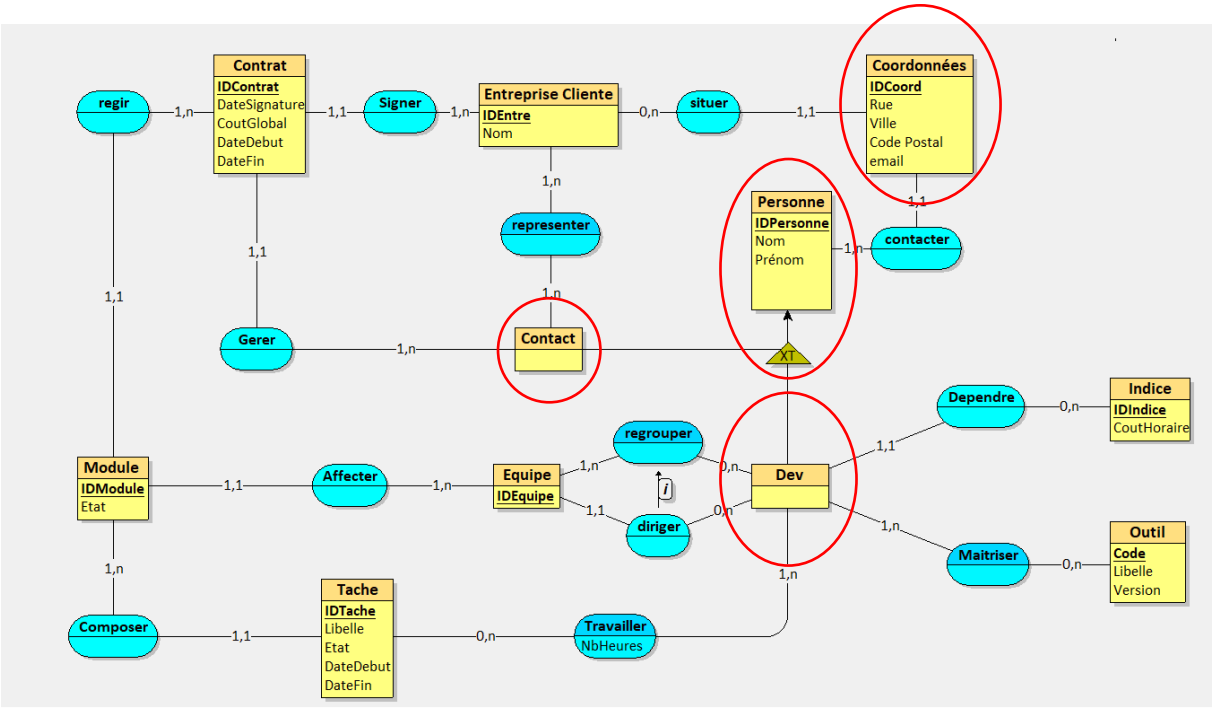
Agathe

Compte rendu Simpleduc SLAM

Sommaire :

I) Base de données

A) MCD



Mes parties

B) MLD

Entreprise_Cliente = (<u>**IDEntre**</u> *INT*, Nom *VARCHAR(50)*);
Personne = (<u>**IDPersonne**</u> *INT*, Nom *VARCHAR(50)*, Prénom *VARCHAR(50)*);
Contact = (*#IDPersonne***);**
Coordonnées = (<u>**IDCoord**</u> *INT*, Rue *VARCHAR(50)*, Ville *VARCHAR(50)*, Code_Postal *VARCHAR(50)*, email *VARCHAR(50)*, *#IDPersonne, #IDEntre*);
Contrat = (<u>**IDContrat**</u> *INT*, DateSignature *DATE*, CoutGlobal *CURRENCY*, DateDebut *DATE*, DateFin *DATE*, *#(#IDPersonne), #IDEntre*);
Outil = (<u>**Code**</u> *INT*, Libelle *VARCHAR(50)*, Version *VARCHAR(50)*);
Indice = (<u>**IDIndice**</u> *INT*, CoutHoraire *CURRENCY*);
Dev = (*#IDPersonne, #IDIndice***);**
Equipe = (<u>**IDEquipe**</u> *INT, #(#IDPersonne)*);
Module_ = (<u>**IDModule**</u> *INT*, Etat *VARCHAR(50)*, *#IDEquipe, #IDContrat*);
Tache = (<u>**IDTache**</u> *INT*, Libelle *VARCHAR(50)*, Etat *VARCHAR(50)*, DateDebut *DATE*, DateFin *DATE*, *#IDModule*);
regrouper = (*#(#IDPersonne), #IDEquipe***);**
representer = (*#IDEntre, #(#IDPersonne)***);**
Travailler = (*#(#IDPersonne), #IDTache*, NbHeures *DECIMAL(15,2)*);
Maitriser = (*#(#IDPersonne), #Code***);**

II) Ajout d'une personne dans la base de données

A) Ajout d'une personne

Pour ajouter une personne dans la base de données, on vient réaliser un formulaire. On commence par créer une fonction dans ProjetModel qui réalise une requête SQL permettant d'ajouter un enregistrement dans la table Personne de la BDD.

```php
function addPersonne($db, $nom, $prenom)
{
    $query = $db->prepare("INSERT INTO Personne (Nom, Prenom) VALUES (:Nom, :Prenom)");
    return $query->execute([
        'Nom' => $nom,
        'Prenom' => $prenom
    ]);
}
```

Pour réaliser cette requête, nous avons besoin de lui fournir un nom et un prénom (son identifiant est choisi automatiquement). Pour cela nous allons utiliser un controller :

```php
<?php
function addPersonneController($twig, $db) {
    include_once '../src/model/ProjetModel.php';


    if (isset($_POST['btnAddPersonne'])) {
        $nom = htmlspecialchars($_POST["nom"]);
        $prenom = htmlspecialchars($_POST["prenom"]);

        addPersonne($db, $nom, $prenom);

    }

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

Dans ce controller, on inclut le fichier ProjetModel qui contient notre fonction ainsi que notre BDD. Ce controller est e relation avec un fichier twig qui nous permet de gérer très facilement l'affichage de notre site.

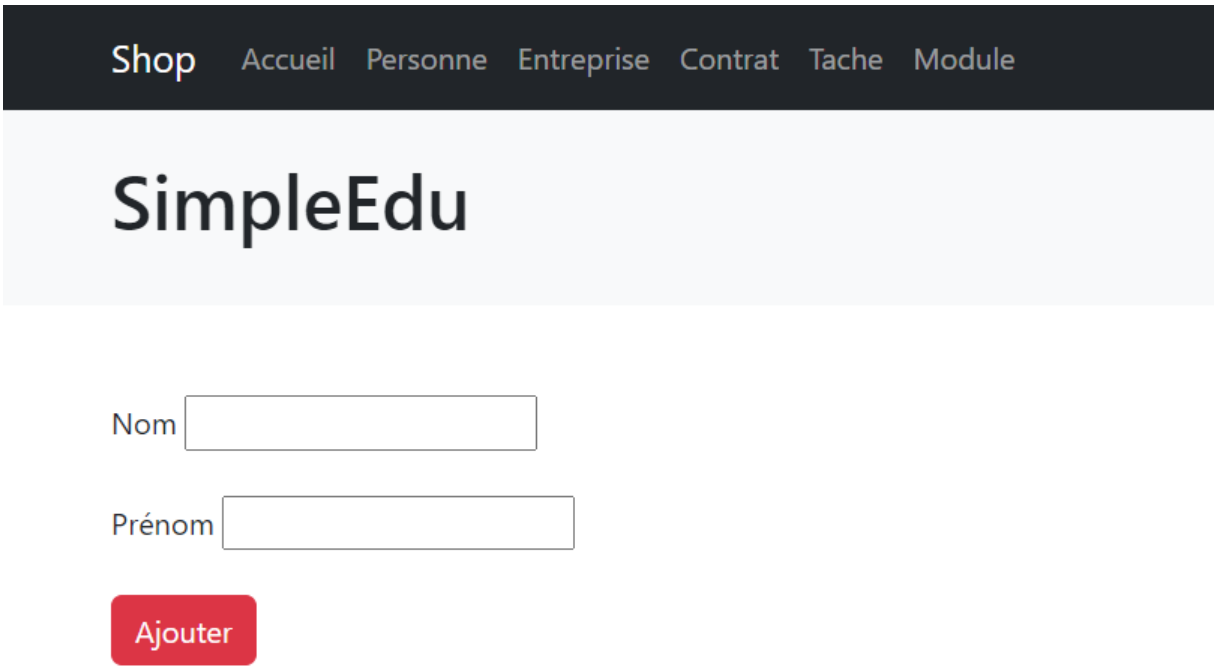
Avec if(isset($_POST['btnAddPersonne'])) on regarde si l'on a appuyé sur le bouton btnAddPersonne qui correspond au bouton d'ajout. Si c'est le cas, on place dans des variables $nom et $prenom les valeurs des inputs qui ont pour nom dans le twig 'nom' et 'prenom'. Enfin on appelle la fonction addPersonne avec les variables.

```twig
{% extends "layout.html.twig" %}
{% block head %}
<head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width,initial-scale=1">
    <title>
        Ajout d'une personne
    </title>

</head>
{% endblock %}
{% block content %}
<div class="container"></div>
    <form action="index.php?page=addPersonne" method="POST" class="">
        <div class="mb-3">
            <label for="nom" class="form-label">
                Nom
            </label>
            <input type="text" class="form-label" id="nom" name="nom">
        </div>
        <div class="mb-3">
            <label for="prenom" class="form-label">
                Prénom
            </label>
            <input  type="text" class="form-label" id="prenom" name="prenom">
        </div>
    <button type="submit" class="btn btn-danger" name="btnAddPersonne">
Ajouter</button></form>
    </div>
    {% endblock %}
```

On rajoute également la route correspondante :

```php
<?php

$routes = [
    'home' => 'homeController',
    'addPersonne' => 'addPersonneController',
    'dbError' => 'dbErrorController',
    'addProduct' => 'addProductController',
    'addEntreprise' => 'addEntrepriseController',
    'addModule' => 'addModuleController',
    'addTache' => 'addTacheController',
    'listePersonnes' => 'listePersonnesController',
    'modifModule' => 'modifModuleController'
];
```

Résultat :



B) Liste des personnes

Maintenant, nous souhaitons lister toutes les personnes inscrites dans la base de données ce qui nous permettra plus tard de faire la modification et l'ajout des coordonnées.

Pour cela, on vient de nouveau créer une fonction appelant une requête SQL dans ProjetModel, ainsi qu'un controller, un fichier twig et une route.

Pour la fonction, on vient réaliser une projection des enregistrements de la table Personne :

```php
function listePersonnes($db){
    $query = $db -> prepare("SELECT IDPersonne, Nom, Prenom FROM Personne");
    $query -> execute([]);
    $liste = $query->fetchAll();
    return $liste;
}
```

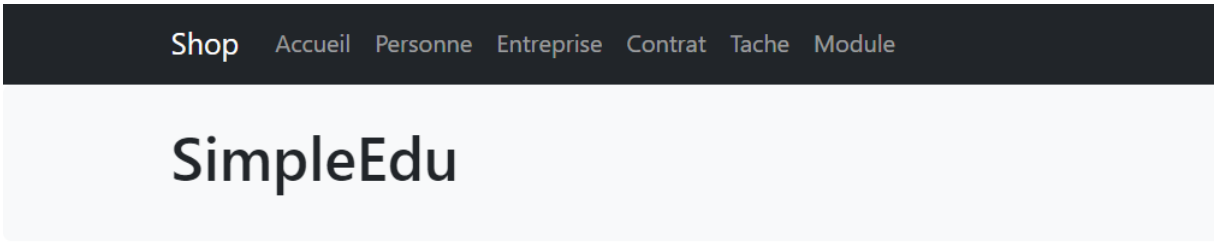
Cette fois-ci nous utilisons un fetchAll() car il y a plusieurs enregistrements à retourner. Dans le controller, on inclut et on appelle le résultat de la fonction dans une variable liste que l'on envoie vers le fichier twig.

```php
<?php
function listePersonnesController($twig, $db){
    include_once '../src/model/ProjetModel.php';

    echo $twig->render('listePersonnes.html.twig', ["liste"=> listePersonnes($db),]);
}
?>
```

Dans le fichier twig, on vient réaliser une boucle {% for personne in liste %} qui nous permet de passer par chaque valeur de liste (que l'on nomme tour à tour personne). Et à chaque itération de cette boucle, on affiche les attributs de la BDD, Nom et Prenom.

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

{% block title %}Liste des personnes !{% endblock %}

{% block content %}
  <h1>Liste des personnes</h1>
  <form method='POST' action="index.php?page=listePersonnes">
    {% for personne in liste %}
      <div>
        <input type="checkbox" name="liste[]" value="{{personne.IDPersonne}}" />
          <p> Prénom: {{personne.Nom}}</p>
          <p> Nom: {{personne.Prenom}}</p>
        </div>
        <br>
        {% endfor %}

      </form>


{% endblock %}
```

Résultat :



Liste des personnes

☐
Prénom: Moi

Nom: Coucou

☐
Prénom: fezjaolfezfez

Nom: fn ehjrbfrzkjhfkjerz h

☐
Prénom: Cac1

Nom: fc

III) Modification et suppression de personnes

A) Modification des personnes

On doit pouvoir modifier les informations des personnes de notre base de données :

Pour cela on vient créer une fonction dans ProjetModel qui nous permet de réaliser une requête SQL :

```php
function updatePersonne($db, $nom, $prenom, $IDPersonne){
    $query = $db -> prepare("UPDATE Personne SET Nom = :Nom, Prenom = :Prenom WHERE IDPersonne = :IDPersonne; ");
    return $query->execute([
        'Nom' => $nom,
        'Prenom' => $prenom,
        'IDPersonne' => $IDPersonne
    ]);
}
```

Cette fonction permet de modifier uniquement le nom et le prénom d'une personne.

Ensuite on vient créer un controller ainsi qu'une page twig qui nous permettent d'afficher ce formulaire et de l'appliquer :

```php
<?php
function ModifPersonneController($twig, $db){
    include_once '../src/model/ProjetModel.php';

    if (isset($_GET["id"])) {
        $IDPersonne = $_GET["id"];
    }
    if (isset($_POST['btnModifierPersonne'])) {
        updatePersonne(
            $db,
            $_POST["Nom"],
            $_POST["Prénom"],
            $IDPersonne,
        );
    }

    echo $twig->render('modifPersonne.html.twig', [
        "personne" => getPersonne($db, $IDPersonne),
    ]);
}

?>
```

On utilise $_GET qui nous permet de passer des informations dans l'url, ici l'identifiant de la personne que l'on souhaite modifier. Grâce à cet identifiant récupéré avec le formulaire, on peut afficher avec notre twig les informations de la personne choisie ainsi que modifier ce que l'on souhaite.

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

{% block title %}Modifier une personne{% endblock %}

{% block content %}
<div class="container">
  <h1>Modification :</h1>
  <div style="display:grid; grid-template-columns: 1fr 1fr;" class="m-5">
    <div style="display:grid; grid-template-columns: 1fr 1fr;">

      <div>
        <div class="container">
          <form method="post" id="{{personne.IDPersonne}}">
            <input type="text" value="{{personne.Nom}}" name="Nom"/>
            <input type="text" value="{{personne.Prenom}}" name="Prénom"/>
            <button type="submit" class="btn btn-dark" name="btnModifierPersonne"  value="{{personne.IDPersonne}}" >
              Modifier </button>
            </form>
            <form method="post" action="index.php?page=listePersonnes">
              <button type="submit" class="btn btn-dark" name="Revenir" action="index.php?page=listePersonnes" >
                  Revenir à la liste </button>
            </form>
        </div>
      </div>
  </div>
</div>
{% endblock %}
```

B) Suppression de personnes

Pour supprimer des personnes nous allons venir réaliser un affichage de toutes les personnes accompagnées d'une case à cocher, ce qui nous permettra de supprimer plusieurs personnes en même temps.

On fait une fonction dans ProjetModel avec DELETE FROM qui nous permet de supprimer un enregistrement dans une BDD :

```php
function deletePersonne($db, $IDPersonne){
    $query = $db -> prepare("DELETE FROM Personne WHERE IDPersonne = :IDPersonne ");
    return $query->execute([
        'IDPersonne' => $IDPersonne
    ]);
    }
```

Dans le fichier twig, on ajoute pour chaque itération de la boucle une checkbox qui fait partie d'une liste et si cette case est cochée, alors on ajoute comme valeur, l'identifiant de la personne dedans.

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

{% block title %}Liste des personnes !{% endblock %}

{% block content %}
  <h1>Liste des personnes</h1>

    {% for personne in liste %}
      <div>
          <input type="checkbox" name="liste[]" value="{{personne.IDPersonne}}" />
            <p> Prénom: {{personne.Nom}}</p>
            <p> Nom: {{personne.Prenom}}</p>
            <a class="link" href="?page=modifPersonne&id={{personne.IDPersonne}}">Modifier</a>
            <a class="link" href="?page=addCoPersonne&id={{personne.IDPersonne}}">Ajouter les coordonnees</a>
          </div>
          <br>
          <form method='POST' action="index.php?page=listePersonnes">
          </form>
          {% endfor %}
          <button type="submit" class="btn btn-dark"  name="btnDeletePersonne" value="{{personne.IDPersonne}}">Supprimer</button>



{% endblock %}
```

Dans le controller, on regarde si le bouton supprimer a été appuyé, si c'est le cas alors on regarde dans notre liste d'identifiants de personnes, chaque valeur afin de les supprimer.

```php
<?php
function listePersonnesController($twig, $db){
    include_once '../src/model/ProjetModel.php';



    if (isset($_POST['btnDeletePersonne'])) {
        if (!empty($_POST['liste'])){
            foreach($_POST['liste'] as $valeur){
                deletePersonne($db, $valeur);
            }
        }
    }



    echo $twig->render('listePersonnes.html.twig', ["liste"=> listePersonnes($db),]);

    if (isset($_POST['btnModifPersonne'])) {
        echo $twig->render('modifPersonne.html.twig', []);
}

}
?>
```

IV) Connexion, inscription au site et redirection

A) Inscription au site

Changement de la table Personne dans la BDD et ajout de la table rôle. Cela nous permet d'ajouter un email et un mot de passe à un utilisateur ainsi qu'un rôle.

Fonction qui nous permet d'ajouter une personne et en même temps un contact en liaison avec tous ces attributs :

```php
function addPersonneComplet($db, $nom, $prenom, $email, $password)
{
    $query = $db->prepare("INSERT INTO Personne (Nom, Prenom, Email, PasswordUser, idRole) VALUES (:Nom, :Prenom, :Email, :PasswordUser, :idRole)");
    $query2 = $db->prepare("INSERT INTO Contact(IDPersonne) VALUES((SELECT IDPersonne FROM Personne WHERE Email = :Email));");
    $query->execute([
        'Nom' => $nom,
        'Prenom' => $prenom,
        'Email' => $email,
        'PasswordUser' => $password,
        'idRole' => 1
    ]);
    return $query2->execute([
        'Email' =>$email
    ]);
}
```

Fonction qui nous permet de vérifier qu'il n'y a pas déjà une personne avec cette adresse mail.

```php
function testEmailExists($db, $email) {
    $query = $db->prepare("SELECT Nom FROM Personne WHERE :Email = Email");
    $query ->execute([
        'Email'=> $email,
    ]);
    $user = $query->fetch();

    if ($user == null) return false;
    else return true;
}
```

Controller register qui vérifie que toutes les informations sont entrées, que l'adresse email n'est pas déjà enregistrée et que le mot de passe n'est pas trop court (affichage erreur en fonction) :

Hachage du mot de passe avec password_hash() pour la sécurité.

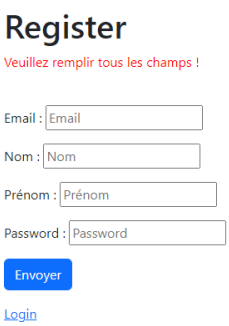
```php
function registerController($twig, $db)
{

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

    $erreur = 0;

    if (isset($_POST['registerBtn'])) {
        if (
            isset($_POST['registerEmail']) && isset($_POST['registerPassword'])  &&
            isset($_POST['registerNom']) && isset($_POST['registerPrenom'])  &&
            $_POST['registerEmail'] != "" && $_POST['registerPassword'] != "" &&
            $_POST['registerNom'] != "" && $_POST['registerPrenom'] != ""
        ) {
            if (strlen($_POST['registerPassword']) >= 3) {
                var_dump($_POST);

                $email = $_POST['registerEmail'];
                $password = $_POST['registerPassword'];
                $nom = $_POST['registerNom'];
                $prenom = $_POST['registerPrenom'];

                if (!testEmailExists($db, $email)) {
                    addPersonneComplet($db, $nom, $prenom, $email, password_hash($password, PASSWORD_DEFAULT));
                    $erreur = "compte cree";

                    $idUser = getOneUser($db, $email)['IDPersonne'];

                    envoyerVerification($db, $twig, $idUser, $email);

                } else $erreur = "existe deja";
            } else $erreur = "mdp trop court";
        } else $erreur = "remplir champs";
    }

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

Fichier twig en liaison :

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

{% block title %}Login{% endblock %}

{% block content %}
<div class="container">

  <div class="card bg-grey mx-auto" style="width: 50%;">
    <div class="m-5 ">
      <h1>Register</h1>

      {% if erreur == "existe deja" %}
      <p style="color: red;">Cette adresse mail existe déjà !</p>
      {% endif %}

      {% if erreur == "remplir champs" %}
      <p style="color: red;">Veuillez remplir tous les champs !</p>
      {% endif %}

      {% if erreur == "compte cree" %}
      <p style="color: green;">Compte créé !</p>
      {% endif %}

      <form action="?page=register" method="POST">
        <br>

        <p>
          <label for="registerEmail">Email :</label>
          <input type="email" name="registerEmail" placeholder="Email">
        </p>

        <p>
          <label for="registerNom">Nom :</label>
          <input type="text" name="registerNom" placeholder="Nom">
        </p>

        <p>
          <label for="registerPrenom">Prénom :</label>
          <input type="text" name="registerPrenom" placeholder="Prénom">
        </p>

        <p>
          <label for="registerPassword">Password :</label>
          <input type="password" name="registerPassword" placeholder="Password">
        </p>

        {#
        <p>
          <label for="password">Confirm Password :</label>
          <input type="password2" name="password2" placeholder="Confirm Password">
        </p>
        #}

        <input type="submit" name="registerBtn" class="btn btn-primary">
      </form>
      <a href="?page=login">Login</a>
    </div>
  </div>

</div>
{% endblock %}
```

On laisse un lien vers la page de connexion.

Résultat :



B) Connexion au site et redirection

Fonction qui regarde s'il y a un enregistrement dans personne qui existe avec cet adresse mail et ce mot de passe :

```php
function testOneUser($db, $email, $password) {
    $query = $db->prepare("SELECT Nom, Prenom, IDPersonne FROM Personne WHERE :Email = Email AND :PasswordUser = PasswordUser ");
    $query ->execute([
        'Email'=> $email,
        'PasswordUser' => $password
    ]);
    $user = $query->fetch();

    return $user;
}
```

Controller :

```php
function loginController($twig, $db)
{
    include_once '../src/model/ProjetModel.php';

    var_dump(" / POST : ");
    var_dump($_POST);
    var_dump(password_hash("yyy", PASSWORD_DEFAULT));
    $user = 1;
    if (isset($_POST['submitLogin'])) {
        if (
            isset($_POST['emailLogin']) && isset($_POST['passwordLogin']) &&
            $_POST['emailLogin'] != "" && $_POST['passwordLogin'] != ""
        ) {

            $email = $_POST['emailLogin'];
            $password = $_POST['passwordLogin'];


            $user = getOneUser($db, $email);
            var_dump($user);

            if ($user != null) {

                if (password_verify($password, $user['PasswordUser'])) {
                    if ($user['CompteVerifie'] == 1) {
                        $_SESSION['login'] = $email;
                        $_SESSION['role'] = $user['idRole'];
                        $user = "connecte";
                        header("Location: index.php");
                    } else $user = "pas verifie";
                } else $user = "faux";
            } else $user = "faux";
        } else $user = "remplir champs";
    }

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

Le controller renvoie les erreurs s'il y en a (il regarde si les champs ont été remplis, si le compte a été vérifié et si les identifiants et mots de passe correspondent).

Twig :

```twig
{% extends "layout.html.twig" %}
{% block title %}Login{% endblock %}
{% block content %}
<div class="container">
  <div class="card bg-grey mx-auto" style="width: 30%;">
    <div class="m-5 ">
      <h1>Login</h1>
      {% if user == "faux" %}
      <p style="color: red;">Mauvais email ou mot de passe</p>
      {% else %}
      {% if user == 'connecte' %}
      <p style="color: green;">Connecté !</p>
      {% endif %}
      {% endif %}
      {% if user == "remplir champs" %}
      <p style="color: red;">Veuillez remplir tous les champs !</p>
      {% endif %}
      {% if user == "pas verifie" %}
      <p style="color: red;">Votre compte n'est pas vérifié !</p>
      {% endif %}
      <form action="?page=login" method="post">
        <br>
        <p>
          <label for="emailLogin">Username :</label>
          <input type="email" name="emailLogin" placeholder="Username">
        </p>
        <p>
          <label for="passwordLogin">Password :</label>
          <input type="password" name="passwordLogin" placeholder="Password">
        </p>

        <input type="submit" name="submitLogin" class="btn btn-primary">
      </form>

      <a href="?page=register">Register</a>
    </div>
  </div>
</div>
{% endblock %}
```

C) Sécurisation des routes

On modifie notre fichier router.php pour qu'il devienne comme ceci :

```php
function initRouter($routes, $db)
{
    //if (isset($_SESSION['username']) || (isset($_GET['page']) && $_GET['page'] == 'register')) {
    //Si la variable page existe donc dans l'url on a mis ?page=truc alors $page devient ?page
    if (isset($_GET['page'])) {
        $page = $_GET['page'];
    } else $page = "home";
    //Si page est dans le tableau des routes alors $route devient la route (homeController)
    if (isset($routes[$page])) {
        $route = $routes[$page];
    } else {
        $route = $routes["home"];
    }
    if ($db == null) {
        $route = $routes["dbError"];
    }
    //test de si l'u a acces à la page
    $routeParameters = explode(':', $route);
    //$controller devient HomeController (nom du fichier)
    $controller = ucfirst($routeParameters[0]);
    $access = $routeParameters[1] ?? 0;
    if ($access != 0) {
        if (!isset($_SESSION['login']) || $_SESSION['role'] < $access) {
            $controller = "HomeController";
        }
    }
    //Require le fichier controller HomeController.php
    require_once 'controller/' . $controller . '.php';
    return $controller;
```

On peut alors rajouter dans le fichier routes.php à quels utilisateurs on souhaite laisser l'accès aux différentes pages. 0 étant pour un visiteur du site, 1 pour un utilisateur et 2 pour un administrateur. Ainsi si un visiteur entre l'url d'une page administrateur, il n'y aura pas accès.

```php
'home' => 'homeController:0',
'addPersonne' => 'addPersonneController:2',
'dbError' => 'dbErrorController:0',
'addProduct' => 'addProductController:2',
'addEntreprise' => 'addEntrepriseController:2',
'modifierEntreprise' => 'modifierEntrepriseController:2',
'addContrat' => 'addContratController:2',
'modifierContrat' => 'modifierContratController:2',
'addModule' => 'addModuleController:2',
'addTache' => 'addTacheController:2',
'listePersonnes' => 'listePersonnesController:1',
'modifModule' => 'modifModuleController:2',
'addTache' => 'addTacheController:2',
'modifTache' => 'modifTacheController:2',
'modifPersonne' => 'modifPersonneController:2',
'addCoPersonne' => 'addCoPersonneController:2',
'listeCo' => 'listeCoController:2',
'modifCoPersonne' => 'modifCoController:2',
'addOutil' => 'addOutilController:2',
'login' => 'loginController:0',
'register' => 'registerController:0',
'deconnexion' => 'deconnexionController:0',
'confirmRegister' => 'confirmRegisterController:0',
```

D) Confirmation par email

On rajoute un attribut CompteVerifie dans la table personne qui comprend un booléen, 0 si le compte n'a pas encore été vérifié et 1 s'il est. En fonction de ce booléen, l'utilisateur pourra se connecter ou non. On crée une table ConfirmationCompte qui va nous permettre d'ajouter les tables en cours

On crée une classe Mail dans un fichier MailModel.php qui va nous permettre de créer des fonctions qui envoient des emails. Cette classe comprend un destinataire, une adresse mail d'envoi (avec son mot de passe), un objet, un message et si l'on souhaite, une pièce jointe

```php
class Mail
{

    //put your code here
    public function __construct()
    {
    }

    public function envoyerMailer($destinataire, $sujet, $message, $piecejointe)
    {
        require_once 'confMail.php';

        $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 = $conf['email']; // Votre adresse email d'envoi
        $mail->Password = $conf['mdp']; // Le mot de passe de cette adresse email
        $mail->SMTPSecure = 'tls'; // Accepter SSL
        $mail->Port = 587;
        $mail->setFrom($conf['email'], 'Site PHP'); // 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 'Erreur, message non envoyé.';
            echo 'Mailer Error: ' . $mail->ErrorInfo;
        } else {
            echo 'Le message a bien été envoyé !';
        }
    }
}
```

Pour envoyer la confirmation de création de compte par email, voici la fonction :

```php
function envoyerVerification($db, $twig, $idUser, $email) {
    $mail2 = new Mail();

                //Création vérif
                $idRegister = addConfirmationCompte($db, $idUser);
                var_dump("ID REGISTER :");
                var_dump($idRegister);

                //Envoyer mail
                if ($idRegister != null) {
                    $mail2->envoyerMailer(
                        $email,
                        "Confirmer le compte PHP",
                        $twig->render("mail/register_message.html.twig", [
                            'email' => $email,
                            'idRegister' => $idRegister
                        ]),
                        null
                    );
                }
```

Pour valider la confirmation du compte :

```php
function confirmerCompte($db, $id)
{
    $query = $db->prepare("UPDATE Personne
                           SET CompteVerifie = 1
                           WHERE IDPersonne = :IDPersonne ");
    $query->execute([
        'IDPersonne' => $id
    ]);

    $query = $db->prepare("DELETE FROM ConfirmationCompte
                           WHERE IDPersonne = :IDPersonne ");
    $query->execute([
        'IDPersonne' => $id
    ]);
}
```

Ainsi on reçoit un mail qui nous permet de confirmer notre compte pour que l'on puisse se connecter. Cela nous permet d'éviter l'inscription de robots.

V) Différenciation entre les développeurs et les contacts

A) Différenciation dans l'ajout

Lorsque quelqu'un s'inscrit sur notre site, il est automatiquement placé en tant que personne et contact. Mais lorsqu'un administrateur ajoute une personne, il doit pouvoir décider de l'ajouter en tant que contact ou développeur :

Fonction pour ajouter un développeur :

```php
function addDev($db, $nom, $prenom, $email, $password)
{
    $query = $db->prepare("INSERT INTO Personne (Nom, Prenom, Email, PasswordUser, idRole, CompteVerifie) VALUES (:Nom, :Prenom, :Email, :PasswordUser, :idRole, :CompteVerifie);");
    $query2 = $db ->prepare("INSERT INTO Dev(IDPersonne, IDIndice) VALUES((SELECT IDPersonne FROM Personne WHERE Email = :Email), :IDIndice);");
    $query3 = $db -> prepare("INSERT INTO regrouper(IDPersonne, IDEquipe) VALUES((SELECT IDPersonne FROM Personne WHERE Email = :Email), :IDEquipe);");
    $query->execute([
        'Nom' => $nom,
        'Prenom' => $prenom,
        'Email' => $email,
        'PasswordUser' => $password,
        'idRole' => 1,
        'CompteVerifie' => 1,
    ]);
    $query2->execute([
        'Email' => $email,
        'IDIndice' => 1,

    ]);
    return $query3->execute([
        'Email' => $email,
        'IDEquipe' => 0

    ]);
}
```

On ajoute dans personne, dans développeur ainsi que dans l'équipe 0 (qui pourra être modifiée plus tard).

Controller :

On réalise notre choix grâce à 2 boutons qui permettent de distinguer la fonction que l'on va utiliser :

```php
function addPersonneController($twig, $db) {
    include_once '../src/model/ProjetModel.php';
    $erreur = "";
    if (isset($_POST['btnAddDev'])) {
        $nom = htmlspecialchars($_POST["nom"]);
        $prenom = htmlspecialchars($_POST["prenom"]);
        $email = htmlspecialchars($_POST["email"]);
        $password = htmlspecialchars($_POST["password"]);

        if (!testEmailExists($db, $email)) {
            addDev($db, $nom, $prenom, $email, password_hash($password, PASSWORD_DEFAULT));

        } else {
            $erreur = "existe deja";}

        if (isset($_POST['btnAddClient'])) {
            $nom = htmlspecialchars($_POST["nom"]);
            $prenom = htmlspecialchars($_POST["prenom"]);
            $email = htmlspecialchars($_POST["email"]);
            $password = htmlspecialchars($_POST["password"]);
            if (!testEmailExists($db, $email)) {
                addClient($db, $nom, $prenom, $email, password_hash($password, PASSWORD_DEFAULT));

            } else {
                $erreur = "existe deja";}
        }
    }


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

Twig :

```twig
<button type="submit" class="btn btn-danger" name="btnAddDev">
    Ajouter en tant que développeur</button>
    <button type="submit" class="btn btn-danger" name="btnAddClient">
        Ajouter en tant que contact d'entreprise</button>
```

Résultat :

B) Différenciation dans l'affichage

Fonctions pour savoir lorsqu'une personne est un développeur/ un contact :

```php
function isDev($db, $IDPersonne){
    $query = $db->prepare("SELECT IdIndice FROM Dev WHERE IDPersonne = :IDPersonne ;");
    $query ->execute([
        'IDPersonne'=> $IDPersonne
    ]);
    $user = $query->fetch();

    return $user;
}
function isContact($db, $IDPersonne){
    $query = $db->prepare("SELECT IDPersonne FROM Contact WHERE IDPersonne = :IDPersonne ;");
    $query ->execute([
        'IDPersonne'=> $IDPersonne
    ]);
    $user = $query->fetch();

    return $user;
}
```

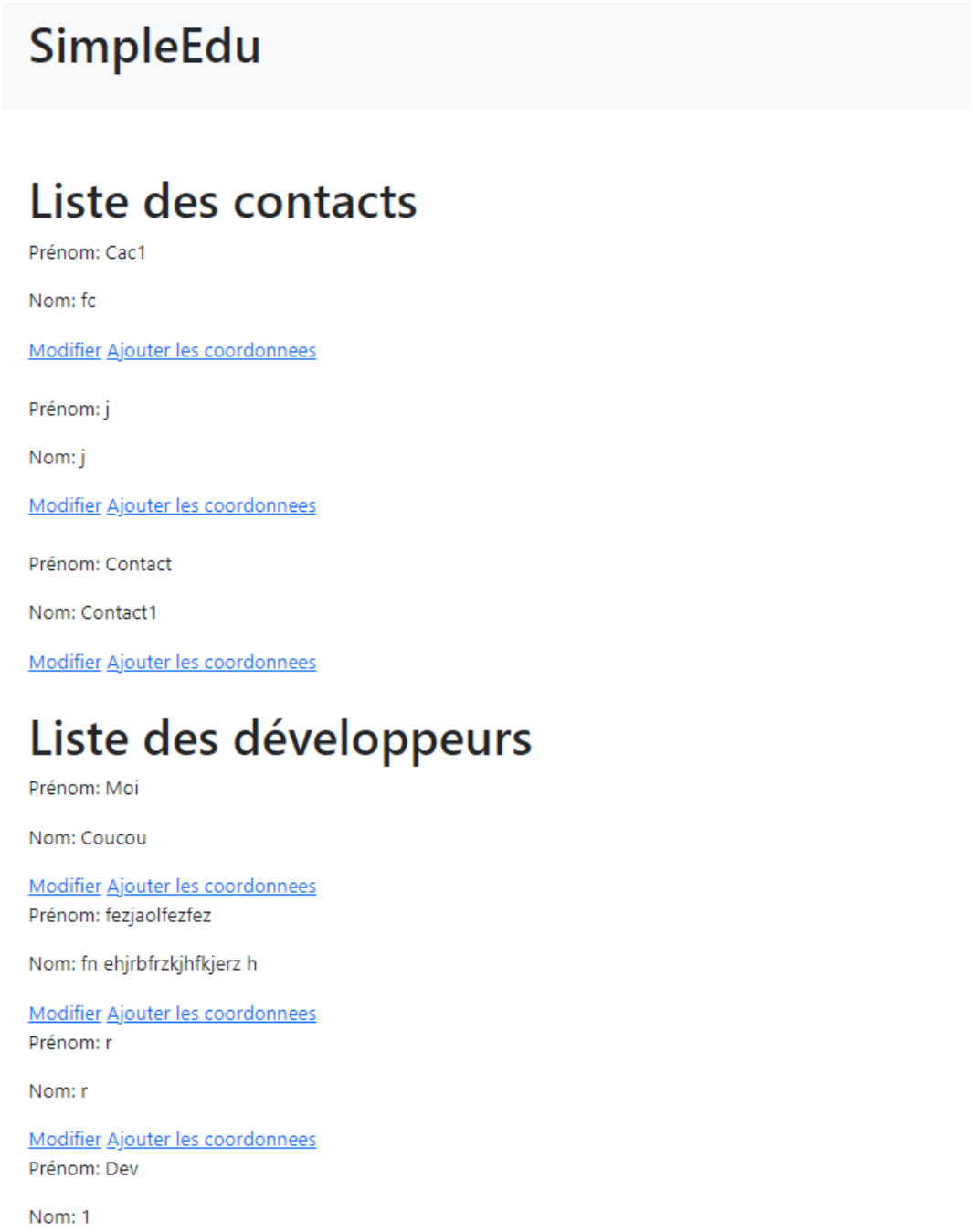
Modification du controller d'affichage des personnes pour séparer les contacts des développeurs. On ajoute leurs identifiants dans 2 listes différentes en fonction de ce qu'ils sont.

```php
function listePersonnesController($twig, $db){
    include_once '../src/model/ProjetModel.php';



    if (isset($_POST['btnDeletePersonne'])) {
        if (!empty($_POST['liste'])){
            foreach($_POST['liste'] as $valeur){
                deletePersonne($db, $valeur);
            }
        }
    }
    $liste = listePersonnes($db);
    $listeDev = array();
    $listeContact = array();
    foreach ($liste as $i){
        $var1 = isDev($db, $i[0]);
        #var_dump( $var);
        if ($var1 !== false){
            array_push($listeDev, $i);
        }
        $var2 = isContact($db, $i[0]);
        if ($var2 !== false){
            if ($var1 == false){
                array_push($listeContact, $i);
            }
        }

    }


    if (isset($_POST['btnModifPersonne'])) {
        echo $twig->render('modifPersonne.html.twig', []);
}
if (isset($_POST['btnModifDev'])) {
    echo $twig->render('modifDev.html.twig', []);
}
    echo $twig->render('listePersonnes.html.twig', ["liste"=> $liste, "listeDev"=>$listeDev, "listeContact"=>$listeContact,]);
```

Twig et affichage :

On parcourt les différentes listes :

```twig
{% for personne in listeContact %}
  <div>
        <p> Prénom: {{personne.Nom}}</p>
        <p> Nom: {{personne.Prenom}}</p>
        <a class="link" href="?page=modifPersonne&id={{personne.IDPersonne}}">Modifier</a>
        <a class="link" href="?page=addCoPersonne&id={{personne.IDPersonne}}">Ajouter les coordonnees</a>
      </div>
      <br>
      <form method='POST' action="index.php?page=listePersonnes">
      </form>
      {% endfor %}
      <h1>Liste des développeurs</h1>
      {% for dev in listeDev %}
      <div>
        <p> Prénom: {{dev.Nom}}</p>
        <p> Nom: {{dev.Prenom}}</p>
        <a class="link" href="?page=modifDev&id={{dev.IDPersonne}}">Modifier</a>
        <a class="link" href="?page=addCoPersonne&id={{dev.IDPersonne}}">Ajouter les coordonnees</a>
      </div>
      {% endfor %}
      <h1>Supprimer des personnes</h1>
      <hr>
      {% for personne in liste %}
      <form method='POST' action="index.php?page=listePersonnes">
   <div>
    <input type="checkbox" name="liste[]" value="{{personne.IDPersonne}}" />
        <p> Prénom: {{personne.Nom}}</p> <p> Nom: {{personne.Prenom}}</p>
        <hr>
      </div>
      {% endfor %}
      <button type="submit" class="btn btn-dark"  name="btnDeletePersonne" value="{{personne.IDPersonne}}">Supprimer</button>
    </form>



  </div>
    {% endblock %}
```



C) Différenciation dans la modification et la suppression

Modification et suppression d'un développeur :

Les fonctions avec les requêtes SQl (on utilise lorsque l'on peut d'avoir mis la suppression en cascade dans la base de données pour faciliter la suppression avec les clés étrangères) :

```php
function updateDev($db, $nom, $prenom, $IDEquipe, $IDPersonne){
    $query = $db -> prepare("UPDATE Personne SET Nom = :Nom, Prenom = :Prenom WHERE IDPersonne = :IDPersonne; ");
    $query2 = $db->prepare("UPDATE regrouper SET IDEquipe = :IDEquipe WHERE IDPersonne = :IDPersonne");
    $query2->execute([
        'IDEquipe' => $IDEquipe,
        'IDPersonne' => $IDPersonne
    ]);
    return $query->execute([
        'Nom' => $nom,
        'Prenom' => $prenom,
        'IDPersonne' => $IDPersonne
    ]);
}
```

```php
function deleteDev($db, $IDPersonne){
    $query = $db -> prepare("DELETE FROM regrouper WHERE IDPersonne = :IDPersonne");
    $query2 = $db -> prepare("DELETE FROM Equipe WHERE IDChef = :IDPersonne");
    $query3 = $db -> prepare("DELETE FROM Dev WHERE IDPersonne = :IDPersonne");
    $query->execute([
        'IDPersonne' => $IDPersonne]);
    $query2->execute([
            'IDPersonne' => $IDPersonne]);
    return $query3->execute([
        'IDPersonne' => $IDPersonne
    ]);
}
```

Dans le controller, on regarde si la personne est un développeur, si c'est un développeur on le supprime en tant que développeur puis en tant que personne :

```php
if (isset($_POST['btnDeletePersonne'])) {
    if (!empty($_POST['liste'])){
        foreach($_POST['liste'] as $valeur){
            $estundev = isDev($db, $valeur);
            if ($estundev !== false){
                deleteDev($db, $valeur);
            }
            deletePersonne($db, $valeur);
        }
    }
}
```

Pour la modification d'un développeur, on peut le supprimer des développeurs (mais pas des contacts et personnes) :

```php
<?php
function ModifDevController($twig, $db){
    include_once '../src/model/ProjetModel.php';

    if (isset($_GET["id"])) {
        $IDPersonne = $_GET["id"];
    }
    if (isset($_POST['btnModifierDev'])) {
        updateDev(
            $db,
            $_POST["Nom"],
            $_POST["Prénom"],
            $_POST["Equipe"],
            $IDPersonne,
        );
    }
    if (isset($_POST['btnDeleteDev'])){
        deleteDev($db, $IDPersonne);
    }

    $Equipe = getEquipeIDPersonne($db, $IDPersonne);


    echo $twig->render('modifDev.html.twig', [
        "personne" => getPersonne($db, $IDPersonne),"Equipe" => $Equipe
    ]);
}

?>
```

Résultat :

Pour la modification des contacts, on peut les transformer en développeur :

```php
function ModifPersonneController($twig, $db, $nbNotifs){
    include_once '../src/model/ProjetModel.php';

    if (isset($_GET["id"])) {
        $IDPersonne = $_GET["id"];
    }
    $email = getPersonne($db, $IDPersonne);

    if (isset($_POST['btnModifierPersonne'])) {
        updatePersonne(
            $db,
            $_POST["Nom"],
            $_POST["Prénom"],
            $IDPersonne,
        );
    }
    if (isset($_POST['btnDev'])) {
        transformerenDev(
            $db,
            $IDPersonne,
            $email[2]
        );
    }

    echo $twig->render('modifPersonne.html.twig', [
        "personne" => $email,
        'nbNotifs' => $nbNotifs
    ]);
```

Résultat :

Résultat de la suppression peu importe que ce soit un développeur ou un contact :

VII) Gestion des coordonnées pour les personnes

A) Ajout des coordonnées

On vient créer cette fonction qui permet d'ajouter des coordonnées dans ProjetModel :

```php
function addCoPersonne($db, $Rue, $Ville, $CodePostal, $Email, $IDPersonne)
{
    $query = $db->prepare("INSERT INTO Coordonnees(Rue, Ville, Code_Postal, email, IDPersonne) VALUES (:Rue, :Ville, :Code_Postal, :email, :IDPersonne)");
    return $query->execute([
        'IDPersonne' => $IDPersonne,
        'Rue' => $Rue,
        'Ville' => $Ville,
        'Code_Postal' => $CodePostal,
        'email' => $Email,
    ]);
}
```

On crée un controller d'ajout de coordonnées et on propose automatiquement l'email de la personne comme email pour les coordonnées (question de logique) :

```php
<?php
function addCoPersonneController($twig, $db, $nbNotifs) {
    include_once '../src/model/ProjetModel.php';


    if (isset($_GET["id"])) {
        $IDPersonne = $_GET["id"];
    }
    if (isset($_POST['btnAddCoPersonne'])) {
        $Rue = htmlspecialchars($_POST["Rue"]);
        $Ville = htmlspecialchars($_POST["Ville"]);
        $CodePostal = htmlspecialchars($_POST["CodePostal"]);
        $Email= htmlspecialchars($_POST["Email"]);

        addCoPersonne($db, $Rue, $Ville, $CodePostal, $Email, $IDPersonne);

    }

    echo $twig->render('addCoPersonne.html.twig', [
        "personne" => getPersonne($db, $IDPersonne),
        'nbNotifs' => $nbNotifs
        ]);
}
```

Twig :

```
<div class="container">
  <h1>Ajouter des coordonnées :</h1>
  <div style="display:grid; grid-template-columns: 1fr 1fr;" class="m-5">
    <div style="display:grid; grid-template-columns: 1fr 1fr;">

      <div>
        <div class="container">
            <p>Nom: {{personne.Nom}}</p>
            <p>Prenom: {{personne.Prenom}}</p>
          <form enctype="multipart/form-data" method="post" id="{{personne.IDPersonne}}" >
            <label for="nom" class="form-label">
        Rue
      </label>
            <input type="text" value="" name="Rue" id='Rue'/>

            <label for="nom" class="form-label">
        Code Postal
      </label>
            <input type="text" value="" name="CodePostal" id="CodePostal"/>

            <label for="nom" class="form-label">
        Ville
      </label>
            <input type="text" value="" name="Ville" id="Ville"/>
<br>
      <label for="nom" class="form-label">
        Email
      </label>
      <input type="text" value="{{personne.Email}}" name="Email" id="Email"/>

            <button type="submit" class="btn btn-dark" name="btnAddCoPersonne" value="{{personne.IDPersonne}}">
              Ajouter </button>
            </form>
            <form enctype="multipart/form-data" method="post" id="{{personne.IDPersonne}}" action="index.php?page=listePersonnes" >
              <button type="submit" class="btn btn-dark" name="Revenir" action="index.php?page=listePersonnes" >
                Revenir à la liste </button>
              </form>
      </div>
  </div>
```

Résultat :

B) Modification des coordonnées

On souhaite également pouvoir modifier ces coordonnées. Donc on crée une fonction qui permet de modifier :

```php
function updateCoPersonne($db, $Rue, $CodePostal, $Ville, $Email, $IDPersonne)
{
    $query = $db->prepare("UPDATE Coordonnees SET Rue = :Rue, Code_Postal = :CodePostal, Ville = :Ville, email = :email WHERE IDPersonne = :IDPersonne ");
    return $query->execute([
        'Rue' => $Rue,
        'CodePostal' => $CodePostal,
        'Ville' => $Ville,
        'email' => $Email,
        'IDPersonne' => $IDPersonne,
    ]);
}
```

Une fonction qui nous permet de récupérer les coordonnées déjà enregistrées :

```php
function hasCo($db, $IDPersonne){
    $query = $db -> prepare("SELECT Rue, Ville, Code_Postal, email FROM Coordonnees WHERE IDPersonne = :IDPersonne ");
    $query -> execute([
        'IDPersonne'=>$IDPersonne
    ]);
    $co = $query->fetchAll();
    return $co;
}
```

Un controller qui récupère les anciennes et les nouvelles coordonnées :

```php
<?php
function ModifCoController($twig, $db, $nbNotifs){
    include_once '../src/model/ProjetModel.php';

    if (isset($_GET["id"])) {
        $IDPersonne = $_GET["id"];
    }
    if (isset($_POST['btnModCo'])) {
        updateCoPersonne(
            $db,
            $_POST["Rue"],
            $_POST["CodePostal"],
            $_POST["Ville"],
            $_POST["email"],
            $IDPersonne,
        );
    }

    echo $twig->render('modifCoPersonne.html.twig', [
        "co" => getCo($db, $IDPersonne),
        'nbNotifs' => $nbNotifs
    ]);
}

?>
```

Et un twig :

```twig
{% block content %}
<div class="container">
  <h1>Modification des coordonnées :</h1>
  <div style="display:grid; grid-template-columns: 1fr 1fr;" class="m-5">
    <div style="display:grid; grid-template-columns: 1fr 1fr;">

      <div>
        <div class="container">
          <form method="post" id="{{co.IDPersonne}}" >
            <label for="nom" class="form-label">
              Rue
            </label>
            <input type="text" value="{{co.Rue}}" name="Rue"/>
            <label for="nom" class="form-label">
              Code Postal
            </label>
            <input type="text" value="{{co.Code_Postal}}" name="CodePostal"/>
            <label for="nom" class="form-label">
              Ville
            </label>
            <input type="text" value="{{co.Ville}}" name="Ville"/>
            <br>
            <label for="nom" class="form-label">
              Email
            </label>
            <input type="text" value="{{co.email}}" name="email"/>
            <button type="submit" class="btn btn-dark" name="btnModCo" value="{{co.IDPersonne}}" form="{{co.IDPersonne}}">

              Modifier </button>
            </form>
              <form method="post" id="{{co.IDPersonne}}" action="index.php?page=listePersonnes">
                <button type="submit" class="btn btn-dark" name="Revenir" action="index.php?page=listePersonnes" >
                  Revenir à la liste </button>
            </form>
        </div>
      </div>
  </div>
</div>
{% endblock %}
```

Résultat :

C) Affichage de la modification ou de l'ajout des coordonnées

Enfin quand on appuie sur notre bouton Voir les coordonnées, on souhaite afficher la page d'ajout de coordonnées lorsque la personne n'a pas de coordonnées et la page de modification lorsqu'elle en a.

Pour cela nous allons nous servir de notre fonction hasCo du dessus qui nous permet de récupérer des coordonnées. Si lorsque l'on appelle fonction, le résultat est vide, alors on renvoi avec un header vers la page d'ajout de coordonnées et vers la page de modification si c'est l'inverse avec toujours l'identifiant de la personne pour la BDD.

Le controller :

```php
if (isset($_POST['btnCo'])){
    $valeur = $_POST['btnCo'];
    $personne = getPersonne($db, $valeur);
    $co = hasco($db, $valeur);
    var_dump($valeur);
    var_dump($co);
    if (count($co) ==0){
        header("location: ?page=addCoPersonne&id=".$valeur."");
    }else{
        header("location: ?page=modifCoPersonne&id=".$valeur."");
    }
```

Le twig :

```
{% for personne in listeContact %}
  <div>
    <form method="POST">
      <p> Prénom: {{personne.Nom}}</p>
      <p> Nom: {{personne.Prenom}}</p>
      <a class="link" href="?page=modifPersonne&id={{personne.IDPersonne}}">Modifier</a>
      <br>
      <button type="submit" class="btn btn-dark" name="btnCo" value="{{personne.IDPersonne}}">Voir les coordonnees</button>
    </div>
    </form>
      <br>
      {% endfor %}
      <h1>Liste des développeurs</h1>
      {% for dev in listeDev %}
      <div>
        <form method='POST'>
          <p> Prénom: {{dev.Nom}}</p>
          <p> Nom: {{dev.Prenom}}</p>
          <a class="link" href="?page=modifDev&id={{dev.IDPersonne}}">Modifier</a>
          <br>
          <button type="submit" class="btn btn-dark" name="btnCo" value="{{dev.IDPersonne}}">Voir les coordonnees</button>
        </form>
      </div>
      {% endfor %}
```

Le résultat pour une personne qui n'a pas de coordonnées :

Prénom: Poteaux

Nom: Agathe

Modifier

Voir les coordonnees

Ajouter des coordonnées :

Nom: Poteaux

Prenom: Agathe

Rue

Code Postal

Ville

Email poteaux.agathe@gmail.cc

Ajouter

Revenir à la liste

Puis lorsqu'en lui en ajoute :

Prénom: Poteaux

Nom: Agathe

Modifier

Voir les coordonnees

Modification des coordonnées :

Rue avenue Jean

Code Postal

59599 Ville

Arras

Email a@a

Modifier

Revenir à la liste