PARTIE PERSONNELLE

XXXXXXXXXXX E1

- Application gestion de permissions
- Base de données
- Serveur Apache

L'étudiant doit développer un module logiciel pour gérer l'ajout, la suppression et la modification des droits d'accès aux différentes salles pour les employés. Pour ce faire, j'ai opté pour une application qui permettra aux ressources humaines de gérer ces permissions.

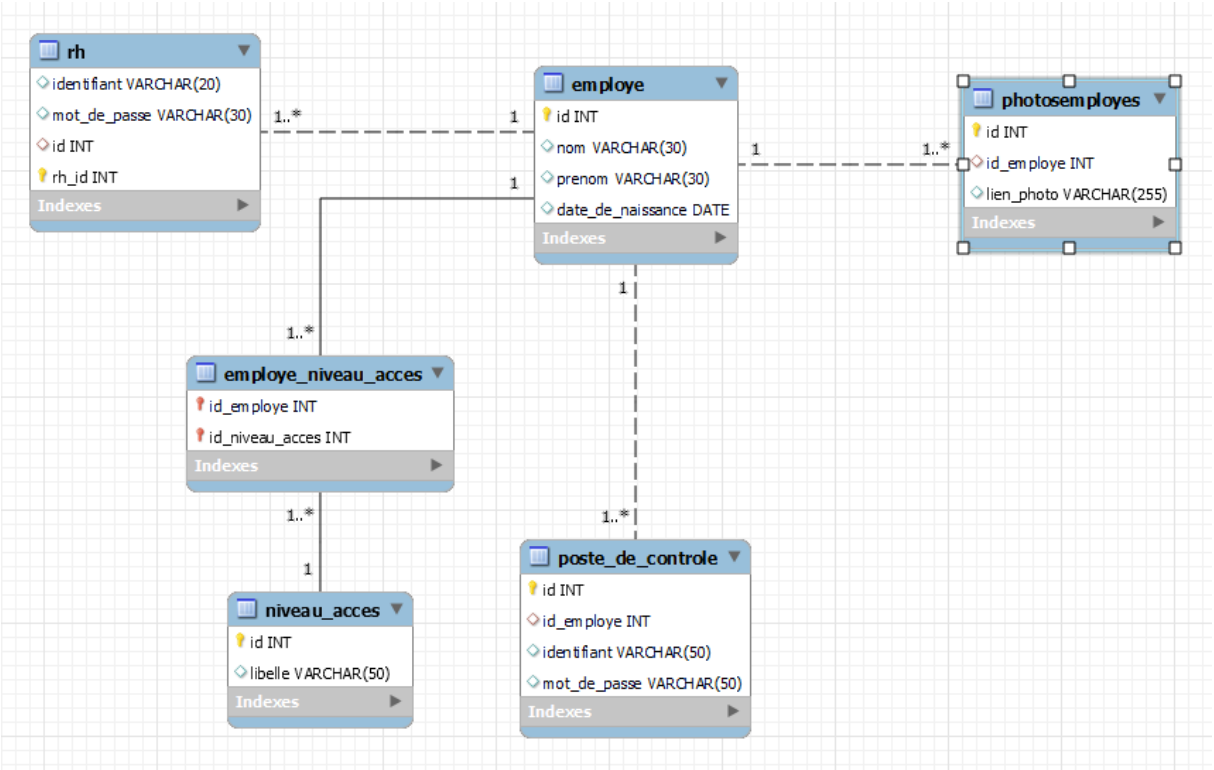
L'application de gestion des permissions sera conçue pour fonctionner sous Windows 10 et sera programmée en C++, conformément aux exigences du cahier des charges. Elle disposera d'une interface graphique pour offrir une meilleure expérience utilisateur. Il est important de noter que cette application sera connectée à une base de données et à un serveur Apache.

1. **Création de la base de données**

L'une de mes premières tâches a été de créer une base de données pour stocker les données de différents employés. Une des exigences pour mener à bien ce projet est que la base de données puisse communiquer avec le programme que je vais devoir développer ensuite. Il existe plusieurs options pour créer une base de données, mais dans mon cas, j'ai choisi de me tourner vers MySQL, qui offre plusieurs avantages sur différents plans tels que :

- **Stabilité et Performance** : MySQL est conçu pour gérer de grandes bases de données avec des millions d'enregistrements et pour supporter des charges de travail élevées.
- **Sécurité** : MySQL propose des options avancées de sécurité, incluant la gestion des utilisateurs avec des permissions, le chiffrement des données en transit et au repos, et la compatibilité avec des protocoles de sécurité réseau, garantissant ainsi la protection des données sensible.

Grâce à ces avantages, MySQL constitue une solution robuste et évolutive pour la création et la gestion de la base de données de ce projet, assurant à la fois performance, sécurité et facilité de gestion.

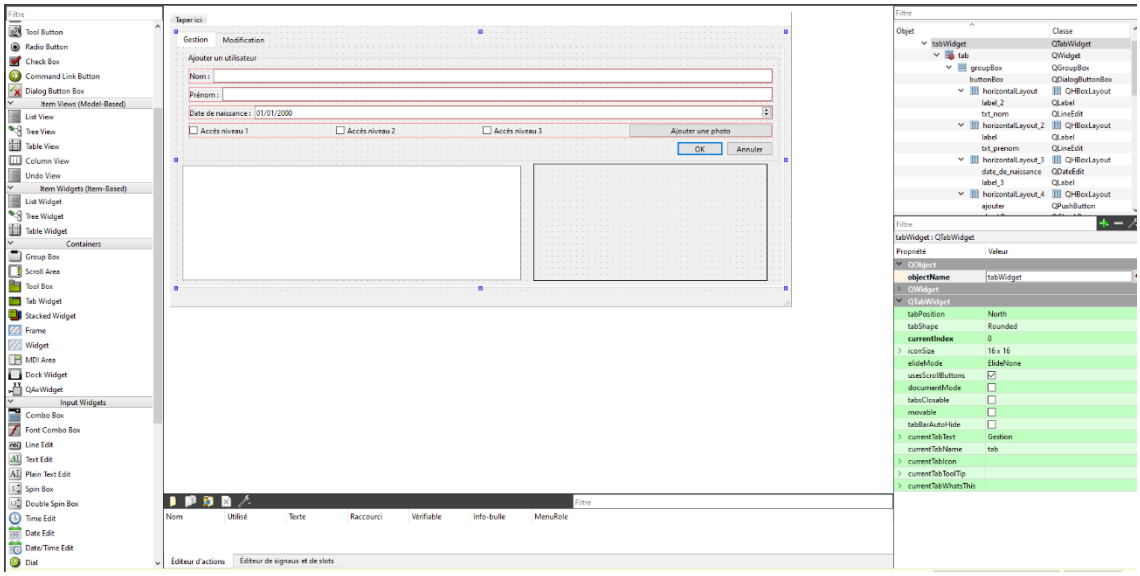


2. Création de l'application :

Tout au long du développement de l'application, j'ai utilisé un seul environnement de développement, Qt Creator est un environnement de développement intégré multiplate-forme faisant partie du framework Qt. Il est donc orienté pour la programmation en C++.

De plus Qt Creator, qui offre la possibilité de créer des applications à l'aide de widgets. En outre, Qt Creator permet la communication avec une base de données si celle-ci est prise en charge par Qt. Dans mon projet, j'utilise MySQL, qui est bien pris en charge par Qt.

Cependant, MySQL n'est pas intégré par défaut, j'ai donc dû installer les dépendances nécessaires pour permettre au programme de communiquer avec la base de données.



- Pourquoi un client lourd et non un client léger pour l'application utilisateur ?

Caractéristique de mon client lourd :

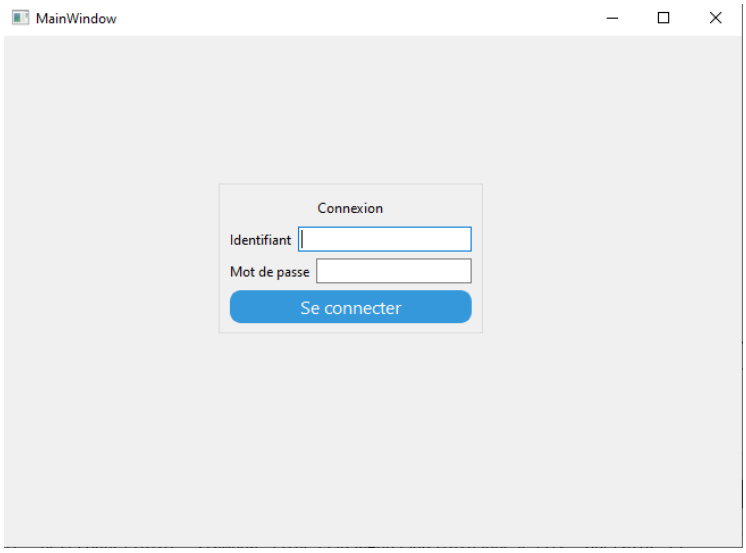
Mon programme est un client lourd car il effectue la majorité du traitement des données localement sur l'ordinateur de l'utilisateur, plutôt que de déléguer cette tâche à un serveur central. L'application envoie et reçoit des requêtes directement à la base de données pour obtenir les informations nécessaires, et une fois les données récupérées. Cela permet de construire les requêtes SQL directement au sein de l'application, le programme assume une partie significative du traitement et de la gestion des données. Cette architecture implique que l'application doit être installée sur chaque machine cliente et dépend des ressources matérielles et logicielles locales pour offrir une interface utilisateur riche et interactive.

Caractéristique d’un client léger :

L'utilisation d'une application en client léger implique que l'utilisateur n'a besoin que d'un navigateur Web. La principale raison du déploiement de clients légers est économique, visant à réduire le coût total de possession et de gestion des systèmes informatiques. De plus, il ne faut pas négliger l'aspect écologique, notamment lorsqu'il s'agit de recycler de vieux ordinateurs en tant que clients légers matériels.

Le développement de ce projet en client lourd est principalement dû aux contraintes du cahier des charges, qui exigeait l'utilisation du langage C++. De plus, le C++ est un langage que j'ai étudié durant mes années de formation en BTS. C’est grâce à cette expérience qui m’a poussé à choisir le développement d’un programme.

3. Connexion à la base de données via le programme :



Pour sécuriser l'accès à la base de données, nous utilisons une connexion directe avec des identifiants fournis par l'utilisateur. Avant de se connecter à la base de données, l'application demande à l'utilisateur de saisir un nom d'utilisateur et un mot de passe. Ces informations d'identification sont ensuite utilisées pour établir une connexion sécurisée à la base de données. En s'assurant que seuls les utilisateurs autorisés peuvent accéder aux données.

Tout cela est possible car dans le programme on vérifie si les identifiant saisis nous permet de nous connecter à la base de données. Nous pouvons voir que la base de données est en local. Il y a plusieurs raisons qui permettent d’expliquer cela :

- Lors de mon développement n’avaient pas besoin d’accéder à la base de données de ce fait héberger la base en local me permettait de réaliser mes tests sans problème.

- Je n’avais pas de serveur ou d’ordinateur qui aurait pu me permettre d’héberger la base de données.

```cpp
void MainWindow::VerifierLaConnexion()
{
    QString identifiant = ui->lineEdit_identifiant->text();
    QString password = ui->lineEdit_password->text();
    QSqlDatabase db = QSqlDatabase::addDatabase("QMYSQL");
    db.setUserName(identifiant);
    db.setPassword(password);
    db.setHostName("localhost");
    db.setDatabaseName("sas");
    db.setPort(3306);
    if(db.open()){
        this->hide();
        MainPage *mainpage = new MainPage();
        mainpage->show();
    }
    else {
        QMessageBox::about(nullptr, "Erreur de connexion", "Erreur de connexion à la base de données : " + db.lastError().text());
    }
}
```

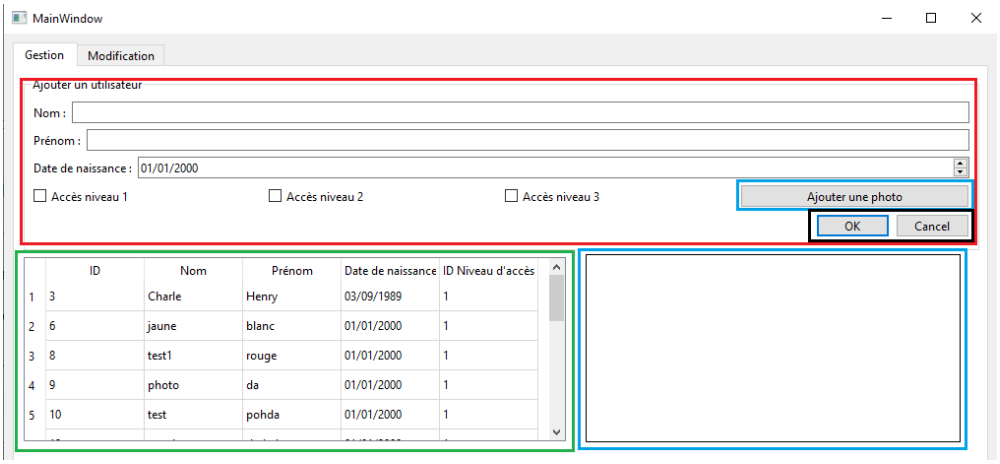
Les identifiants rentrés dans le programme sont en fait des utilisateurs même de la base de données. Bien évidemment il est possible de gérer les permissions de chaque utilisateur qui se connecte à ma base de données et de leur limiter certains accès. De plus les identifiants utilisés par les ressources humaines sont stockés dans la table « rh »

```
mysql> select * from rh;
+-------------+--------------+------+-------+
| identifiant | mot_de_passe | id   | rh_id |
+-------------+--------------+------+-------+
| manager     | root         |    1 |     1 |
+-------------+--------------+------+-------+
1 row in set (0.00 sec)
```

Comme ici présent nous pouvons voir qu'il y a un identifiant de connexion pour l'id employé 1.

La table « **rh »** est uniquement là pour permettre à l'administrateur de voir qui possède des droits d'accès à la base de données. La vérification des droits d'accès ne s'effectue pas à ce niveau. Cependant, un problème se pose tout de même dans notre cas car les mots de passe sont visibles en clair. Une des solutions envisagées est de hacher les mots de passe par précaution.

Lorsque la connexion est établie l'utilisateur se retrouve face à cette interface. Celle-ci se divise en deux parties, je vais détailler le fonctionnement de l'interface principal :



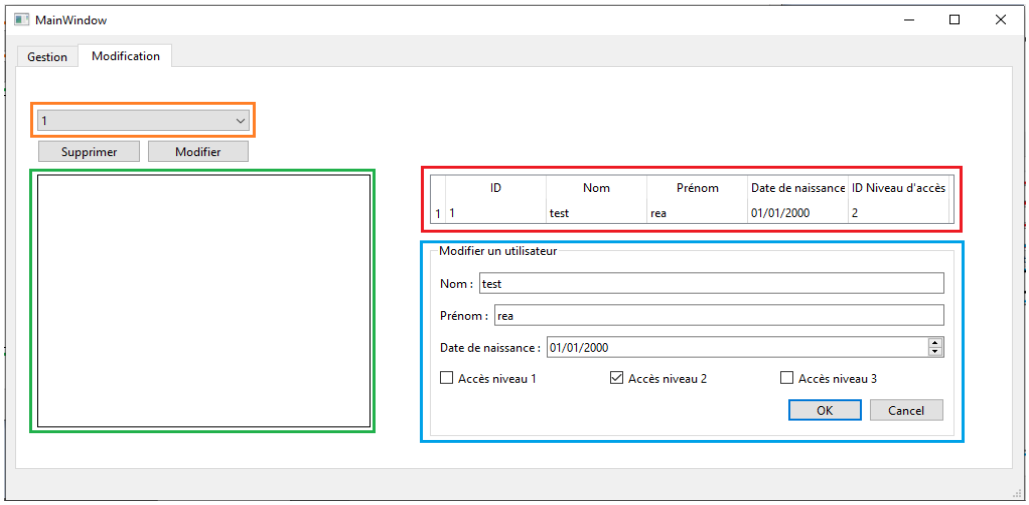
On saisit les informations de l'employé, et son niveau d'accès (on ne peut saisir qu'un seul niveau d'accès).

Ici on peut observer les données en direct présent dans la base de données.

Permet d'ajouter la photo de l'employé et le cadre permet de visualiser l'image présélectionnée.

Valide les données saisies et les rentre dans la base de données.

Voici la deuxième interface qui va permettre à l'utilisateur de modifier ou supprimer les données d'un employé.



Liste les ID des employés, quand un ID spécifique est sélectionné, cela nous permet de récupérer les données qui lui sont associées.

Ce cadre nous permet de visualiser la photo de l'employé si ce dernier en possède une dans la base de données.

Ce champ-là nous informe des informations au préalable de l'ID sélectionné.

Cela nous permet de changer les informations qui nous intéresse

4. **Comment ajouter un employé :**

Le programme communique directement avec la base de données et lui envoie des requêtes lorsque cela est nécessaire. Par exemple si on doit ajouter un employé, on fera appel au programme contenu dans le bouton « ok ».

```
QString nom = ui->txt_nom->text();
QString prenom = ui->txt_prenom->text();
QDate date_de_naissance = ui->date_de_naissance->date();
```

Dans un premier temps le programme va récupérer les données saisies dans les champs correspondants. Avant de faire une vérification si un élément n'est pas manquant. Avant de préparer la requête SQL permettant l'ajout de l'utilisateur. Cependant, il est nécessaire de convertir la date de naissance en chaîne de caractères pour que cela puisse être bien interprété par la base de données.

```
QString date_de_naissance_str = date_de_naissance.toString("yyyy-MM-dd");
```

Si tous les éléments sont correctement saisis alors on commence à préparer la requête SQL.

```
query.prepare("INSERT INTO employe (nom, prenom, date_de_naissance) VALUES (:nom, :prenom, :date_de_naissance)");
query.bindValue(":nom", nom);
query.bindValue(":prenom", prenom);
query.bindValue(":date_de_naissance", date_de_naissance_str);
```

Cette requête permet juste d'ajouter l'employé à la table employe, il est donc nécessaire d'élaborer une seconde requête SQL pour attribuer le niveau d'accès qu'on avait spécifié. Pour ce faire le programme vérifie quelle case est cochée afin de lui attribuer le bon niveau d'accès. Mais pour que cela soit possible, il est nécessaire de récupérer l'ID de l'employé qui vient d'être ajouté.

```
int idEmploye = query.lastInsertId().toInt();
```

```
if(ui->checkBox->isChecked()){
    QSqlQuery querycheck;
    querycheck.prepare("INSERT INTO employe_niveau_acces (id_employe, id_niveau_acces) VALUES (:idEmploye, 1)");
    querycheck.bindValue(":idEmploye", idEmploye);
    QMessageBox::information(this, "Succès", "L'employé a bien reçut l'accès 1");
    if (!querycheck.exec()) {
        qDebug() << "Erreur lors de l'insertion du niveau d'accès 1 :" << querycheck.lastError().text();
    }
}
```

```
ui->checkBox->setChecked(false);
ui->checkBox_2->setChecked(false);
ui->checkBox_3->setChecked(false);
ui->txt_nom->clear();
ui->txt_prenom->clear();
QDate dateSpecifique(2000, 1, 1);
ui->date_de_naissance->setDate(dateSpecifique);
ui->imageLabel_2->clear();
```

Puis enfin on efface les données saisies dans les champs afin qu'ils soient vides et permettent l'ajout d'une nouvelle personne.

5. Modifier un employé :

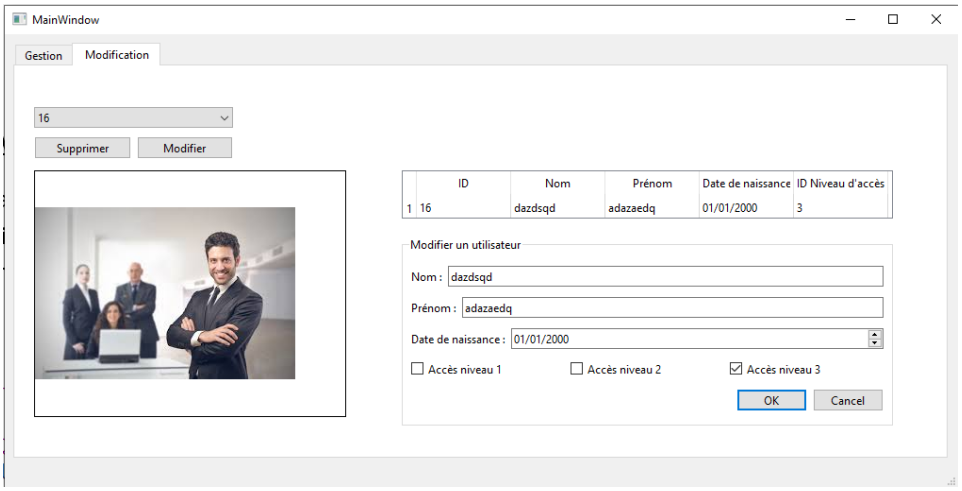
Le programme offre naturellement la possibilité de modifier un employé en cas d'erreur. La modification des informations relatives à un employé est relativement facile. Pour effectuer des modifications, il est d'abord nécessaire de lister les employés présents dans la base de données et de sélectionner celui concerné.

```
void MainPage::loadbox()
{
    QSqlQueryModel *model = new QSqlQueryModel();
    model->setQuery("SELECT id FROM employe");
    ui->comboBox->setModel(model);
}
```

Pour ce faire il nécessaire d'aller chercher tous les id dans la table « employe » présent dans la base de données. Puis d'associer les résultats dans une ComboBox.

Une ComboBox est un élément d'interface graphique qui réunit une zone de texte et une liste déroulante

Ici nous avons sélectionné l'employé qui possède l'id 16, on récupère bien les informations le concernant et nous avons la possibilité d'apporter des changements si nécessaire.



A l'heure actuelle malheureusement je n'ai pas encore pu trouver la solution pour pouvoir modifier la photo d'un employé. Car effectivement le principal problème est de pouvoir lister les photos présentes sur le serveur Apache de façon visuel.

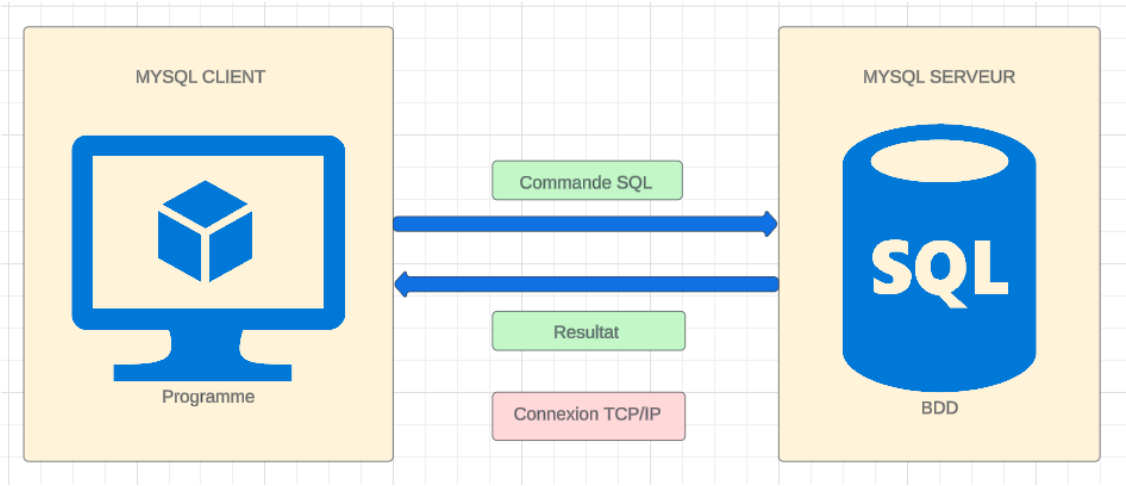
Lorsqu'une modification est appliquée on récupère les données dans les champs puis on prépare les requêtes SQL nécessaires.

```
    if(ui->checkBox_6->isChecked()){
        QSqlQuery querycheck;
        querycheck.prepare("UPDATE employe_niveau_acces SET id_niveau_acces = 3 WHERE id_employe = ?");
        querycheck.bindValue(0, idEmployeSelectionne); // Utiliser bindValue avec l'index 0
        QMessageBox::information(this, "Succès", "L'employé a bien reçu l'accès 3");
        if (!querycheck.exec()) {
            qDebug() << "Erreur lors de la mise à jour du niveau d'accès 3 :" << querycheck.lastError().text();
        }
    }

    // Préparer la requête SQL de mise à jour
    query.prepare("UPDATE employe SET nom = ?, prenom = ?, date_de_naissance = ? WHERE id = ?");
    query.bindValue(0, nouveauNom);
    query.bindValue(1, nouveauPrenom);
    query.bindValue(2, nouvelleDateNaissance);
    query.bindValue(3, idEmployeSelectionne.toInt()); // Convertir l'ID en entier

    // Exécuter la requête de mise à jour
    if (query.exec()) {
        qDebug() << "Les informations de l'employé ont été mises à jour avec succès.";
        QMessageBox::information(this, "Succès", "Les informations de l'employé ont été mises à jour avec succès.")
    } else {
        qDebug() << "Erreur lors de la mise à jour des informations de l'employé :" << query.lastError().text();
        QMessageBox::critical(this, "Erreur", "Erreur lors de la mise à jour des informations de l'employé : " + qu
    }
}
```

6. **<u>Communication entre le programme et la base de données :</u>**
   Voici un schéma qui explique comment le programme communique avec la base de données :



Le fonctionnement est assez simple, le programme est considéré comme un client MYSQL. Il peut donc envoyer des commandes SQL préparé grâce au programme. Puis le serveur lui renvoie une réponse naturellement.

<u>Pourquoi utiliser une connexion TCP/IP :</u>

L'utilisation d'une connexion TCP/IP pour faire communiquer un programme et une base de données présente plusieurs avantages :

Fiabilité de la Communication :

- **Transmission Fiable :** Le protocole TCP (Transmission Control Protocol) assure la transmission fiable des données entre le programme et la base de données, en garantissant que les paquets de données arrivent dans le bon ordre et sans perte.
- **Contrôle d'Erreur :** TCP inclut des mécanismes de détection et de correction des erreurs, ce qui augmente la fiabilité de la communication.

Compatibilité :

- **Standard Universel** : TCP/IP est un standard largement adopté et utilisé, ce qui signifie qu'il est compatible avec la plupart des systèmes d'exploitation et des environnements réseau.
- **Interopérabilité** : Grâce à sa standardisation, il permet une communication fluide entre différents types de systèmes et logiciels, favorisant l'interopérabilité entre les composants.

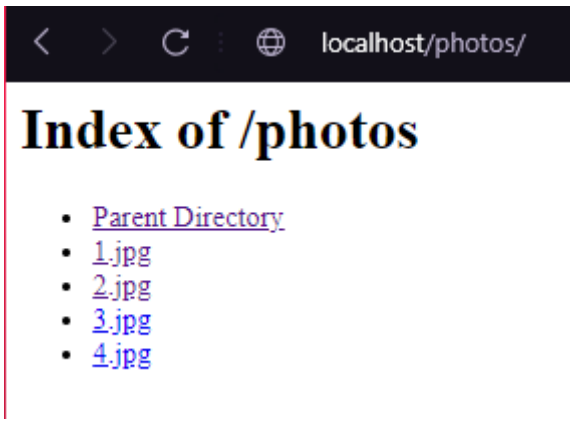
En résumé, utiliser une connexion TCP/IP pour la communication entre un programme et une base de données offre des avantages significatifs en termes de fiabilité, compatibilité, sécurité, flexibilité, performance et simplicité de configuration

7. Installation serveur Apache :

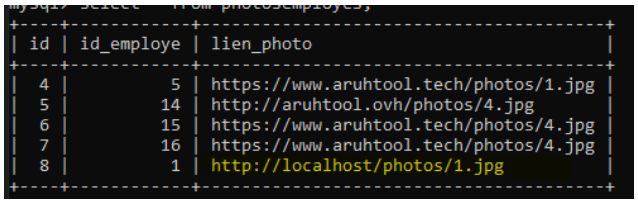
La mise en place du serveur apache est simple. Cependant il est crucial de bien choisir la version du serveur pour qu'il soit compatible avec d'autres outils en cas de nécessité dans notre cas. Ici dans notre projet nous utiliserons la version 2.4.59 d'Apache.

Durant le développement du projet le serveur Apache est héberger en local, le but principal du serveur Apache n'est pas de faire un site mais son rôle sera d'héberger et stocker les images que le programme lui envoie.

Afin de séparer les fichiers correctement et de bien organiser le répertoire, j'ai créé un fichier spécifique pour stocker les photos.



Et donc pour pouvoir avoir accès aux images via un lien cela se fait sous la forme localhost/photos/ « nom de l'image » et c'est ce type de lien qu'on va stocker dans la base de données.



id	id_employe	lien_photo
4	5	https://www.aruhtool.tech/photos/1.jpg
5	14	http://aruhtool.ovh/photos/4.jpg
6	15	https://www.aruhtool.tech/photos/4.jpg
7	16	https://www.aruhtool.tech/photos/4.jpg
8	1	http://localhost/photos/1.jpg

Puis le programme récupère le lien associé à l'employé associé avant de l'afficher dans le programme. Si l'employé n'a pas d'image associé on n'affiche rien et on avertit l'utilisateur.

```
QString idEmployeSelectionne = ui->comboBox->currentText();
QSqlQuery query;
qDebug() << "ID de l'employé sélectionné : " << idEmployeSelectionne; // Message pour afficher l'ID sélectionné
query.prepare("SELECT lien_photo FROM PhotosEmployes WHERE id_employe = ?");
query.bindValue(0, idEmployeSelectionne);

if(query.exec() && query.first()) {
    QString imageUrl = query.value(0).toString();
    qDebug() << "URL de l'image de l'employé : " << imageUrl; // Message pour afficher l'URL de l'image récupérée

    QNetworkAccessManager *manager = new QNetworkAccessManager(this);
    connect(manager, &QNetworkAccessManager::finished, this, &MainPage::replyFinished);
    manager->get(QNetworkRequest(QUrl(imageUrl)));

} else {
    QMessageBox::warning(this, "Erreur", "Impossible de récupérer les images des l'employés");
}
```

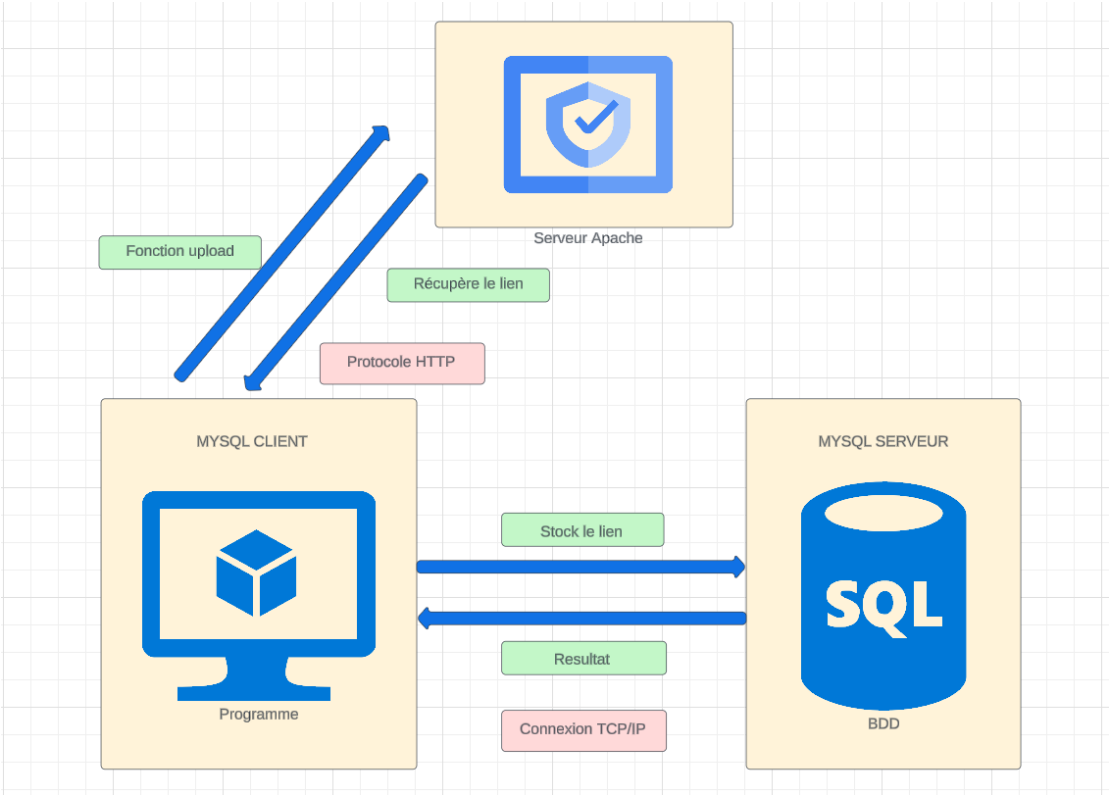
8. Ajouter une photo d'identité :

Mon programme offre aussi la possibilité d'ajouter une photo d'identité pour chaque utilisateur. Avant d'implémenter cette fonctionnalité, plusieurs questions cruciales ont dû être abordées, telles que le lieu de stockage des photos et les méthodes d'accès. Après avoir étudier le problème, j'ai opté pour l'installation d'un serveur Apache pour héberger les images en local dans un premier temps. Ce choix me permet de gérer efficacement les images et de garantir leur disponibilité.

Chaque photo téléchargée est stockée sur le serveur, et le lien correspondant est récupéré et enregistré dans la base de données. Cette approche assure une intégration fluide et sécurisée des photos d'identité dans mon système.

Apache est un serveur web open source largement utilisé pour héberger des sites web et des applications en ligne. En termes simples, il s'agit d'un logiciel qui reçoit des requêtes de navigateurs web (comme Chrome ou Firefox) et renvoie le contenu demandé, telle qu'une page web ou une image. Dans le cadre de mon projet, Apache joue réponds à mes besoins, en permettant d'héberger les photos d'identité des utilisateurs. Grâce à sa fiabilité et à sa flexibilité, Apache permet l'accès aux images facilement via des liens.

A travers le schéma suivant, je vais vous présenter la communication lors de l'association d'une photo d'identité à un employé :

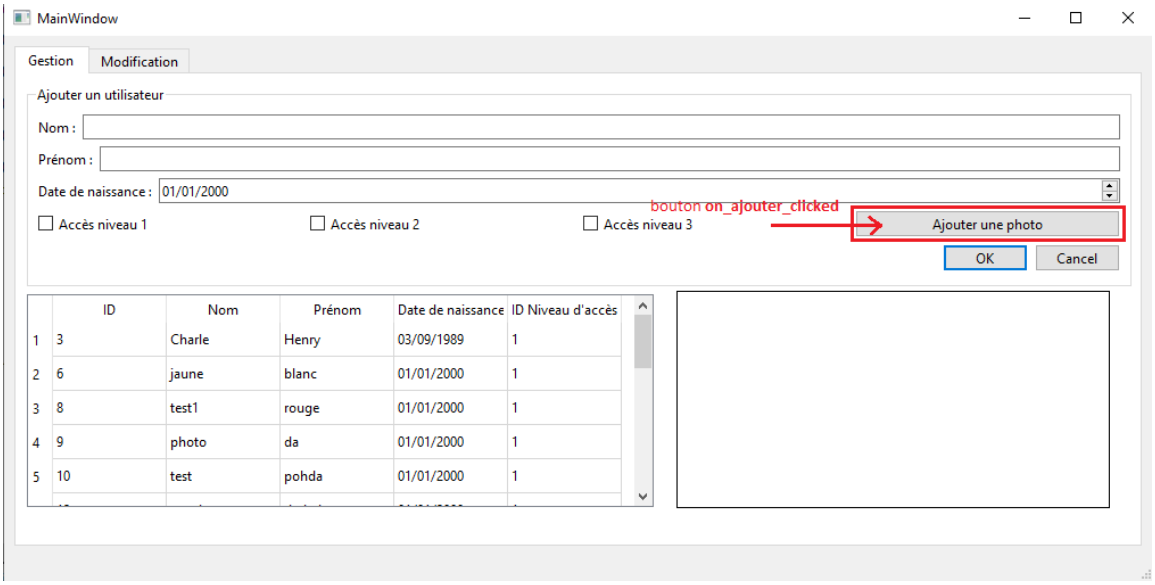


La méthode "upload" dans le programme permet d'acheminer l'image sélectionnée vers le serveur web. En échange, elle retourne le lien d'accès à l'image, et la stocker dans la base de données.

Cependant nous avons tout de même rencontré un gros problème lors du développement de cette phase et nous avons dû exploser plusieurs pistes afin de régler le problème. Car le programme envoyait les requêtes vers le serveur mais aucunes images envoyées arrivaient à être stocker. Mais tout d'abord je vais vous présenter la fonction principale qui permet d'envoyer les images vers le serveur Apache.

Fonction upload :

Lorsqu'on clique sur le bouton ajouter le programme se lance, nous utiliserons une image présélectionnée d'avance avant de facilité les tests. Quand le bouton **on_ajouter_clicked** est pressé il fait appel à la fonction **upload** sans problème, les tests de permissions sont passé avec succès ainsi nous avons un retour positif du programme.



Un autre test est réalisé on vérifie au préalable si le programme peut communiquer avec le serveur Apache, on a encore une fois un retour positif du serveur donc le problème ne peut pas venir de la connexion. L'image présélectionné s'affiche bien dans le programme cela signifie que le programme sélectionne bien la bonne image.

Bien que le programme précise que les tests de permissions sont passé avec succès nous allons tout de même vérifier cela.

Analyse des permissions :

Nous savons que le programme est lancé sur l'utilisateur principal, en regardant dans les sécurités du répertoire photos dans les fichiers Apache aucune anomalie à signaler. L'utilisateur principal possède bien les droits d'écrire et donc de modification. Par mesure de sécurité nous avons exécuter en plus quelques commande shell pour confirmer que l'utilisateur posséder bien les droits suffisants.

D'après la commande : icacls "C:\Apache24\htdocs\photos"

```
PS C:\Users\namex> icacls "C:\Apache24\htdocs\photos"
C:\Apache24\htdocs\photos BUILTIN\Utilisateurs:(OI)(CI)(W)
                          AUTORITE NT\Utilisateurs authentifiés:(I)(M)
                          AUTORITE NT\Utilisateurs authentifiés:(I)(OI)(CI)(IO)(M)
                          AUTORITE NT\Système:(I)(OI)(CI)(F)
                          BUILTIN\Administrateurs:(I)(OI)(CI)(F)
                          BUILTIN\Utilisateurs:(I)(OI)(CI)(RX)

1 fichiers correctement traités ; échec du traitement de 0 fichiers
PS C:\Users\namex>
```

L'utilisateur sur lequel le programme se lancer possède effectivement les droits nécessaires.

7.1 Retour des fichier log d'Apache :

Après examen des fichiers logs le fichier **error.log** ne spécifie aucune erreur après l'exécution du bouton. En revanche le fichier **access.log** nous retrouvons 2 lignes après l'exécution du programme.

```
::1 - - [02/May/2024:10:21:26 +0200] "POST /photos HTTP/1.1" 301 232
::1 - - [02/May/2024:10:21:26 +0200] "GET /photos/ HTTP/1.1" 200 328
```

Une requête POST avec le code 301 qui signifie une redirection permanente de la requête qu'on envoie, puis un retour GET avec le 200 qui correspond à un retour positif. Cela signifie que la requête envoyer par le programme est rediriger vers une autre page. Afin essayer de déterminer la source du problème nous allons examiner la configuration même du serveur Apache.

Il faut savoir qu'il peut y avoir plusieurs directives qui permette une potentielle redirection (**Directory, Redirect, Alias et RewriteRule).** Après étude du fichier **httpd.conf** nous n'avons rien trouver qu'il pourrait entrainer une quelconque redirection. Nous avons tout de même d'autre fichier de configuration présent dans deux autres fichiers : **extra** et **orignal**. Ces fichiers contiennent bien plusieurs autres configurations possibles du serveur Apache mais ce sont des configurations secondaires qui ne sont pas censé impacter la configuration principale.

Fontionnement de Upload :

Le programme commence par créer un instant QFile en se basant sur le chemin d'**imagePath.** Puis elle vérifie qu'elle existe bien, avant de l'ouvrir en lecture seule et de lire toutes les données à l'aide de **QbyteArray.** Il faut savoir que **QByteArray** est souvent utiliser pour lire le contenu binaire d'un fichier. Puis grâce à **QhttpMultiPart** on construit un ensemble de parties de formulaire qui peuvent contenir différents types de données.
Puis on crée un **QNetworkAccessManager** pour gérer la communication réseau. Elle envoie la requête HTTP POST avec les données du formulaire multipart via **manager.post()**. Cette méthode est bloquante, donc elle utilise un **QEventLoop** pour attendre que la requête soit terminée.

Une fois que la requête est terminée, on vérifie si on a réussi avec **reply->error()** est égal à **QNetworkReply::NoError**. Si la requête réussit, on affiche un message de débogage indiquant que le fichier a été envoyé avec succès et retourne **True**. Sinon, on affiche un message d'erreur contenant une description de l'erreur et retourne **False**.

7.2 Solution :

Nous avons bien une fonction dans notre programme qui envoie une requête HTTP POST grâce à la fonction Upload. Bien qu'on n'ait pas de retour d'erreur de la part du serveur nous allons utiliser une autre méthode pour envoyer des requête POST afin de nous assurer qu'on envoie des requêtes correctement structurées.
Pour ce faire nous allons utiliser l'outil **CURL** qui est principalement utilisée pour transférer des données à partir ou vers un serveur à l'aide de divers protocoles, notamment HTTP, HTTPS, FTP. Grâce à CURL nous savons qu'on envoie des requêtes configurées correctement vers le serveur Apache.

```
C:\Users\namex>curl -X POST -F "image=@C:/Users/namex/OneDrive/Bureau/4.jpg" http://localhost/photos/
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<html><head>
<title>404 Not Found</title>
</head><body>
<h1>Not Found</h1>
<p>The requested URL was not found on this server.</p>
</body></html>
```

Même après l'utilisation de CURL pour envoyer nos requêtes nous avons toujours le même problème, le serveur ne stock pas l'images qu'on lui envoie. Après m'être renseigner sur internet j'ai trouvé un début de solution.

En effet apache est incapable de traiter des requêtes correctement sans méthode de réception. Afin de remédier à ce problème il est possible de mettre en place un script PHP côté serveur. Pour permettre au serveur Apache d'être capable de gérer la réception des requêtes POST. L'installation de PHP n'est pas un problème néanmoins il est important de vérifier la compatibilité. Car effectivement certaines versions de PHP ou d'Apache ne sont pas compatible entre eux.
Dans mon cas j'ai choisi d'utilisé une version antérieur de PHP 8.1.28 car dans cette version on retrouve les fichiers nécessaires pour la compatibilité avec Apache (telle que le fichier php8apache2_4.dll qui n'est pas présent dans la dernière version de PHP)

En plus de la méthode de réception présent dans le script PHP nous avons mis en place quelques vérifications telle que : est-ce que le fichier existe-t-il déjà, la taille de l'image et si l'image correspond bien à certains formats imposés.

Avec les requêtes envoyées via CURL le serveur retourne une réponse positive est effectivement l'image a pu être stocké cette fois-ci sur serveur.

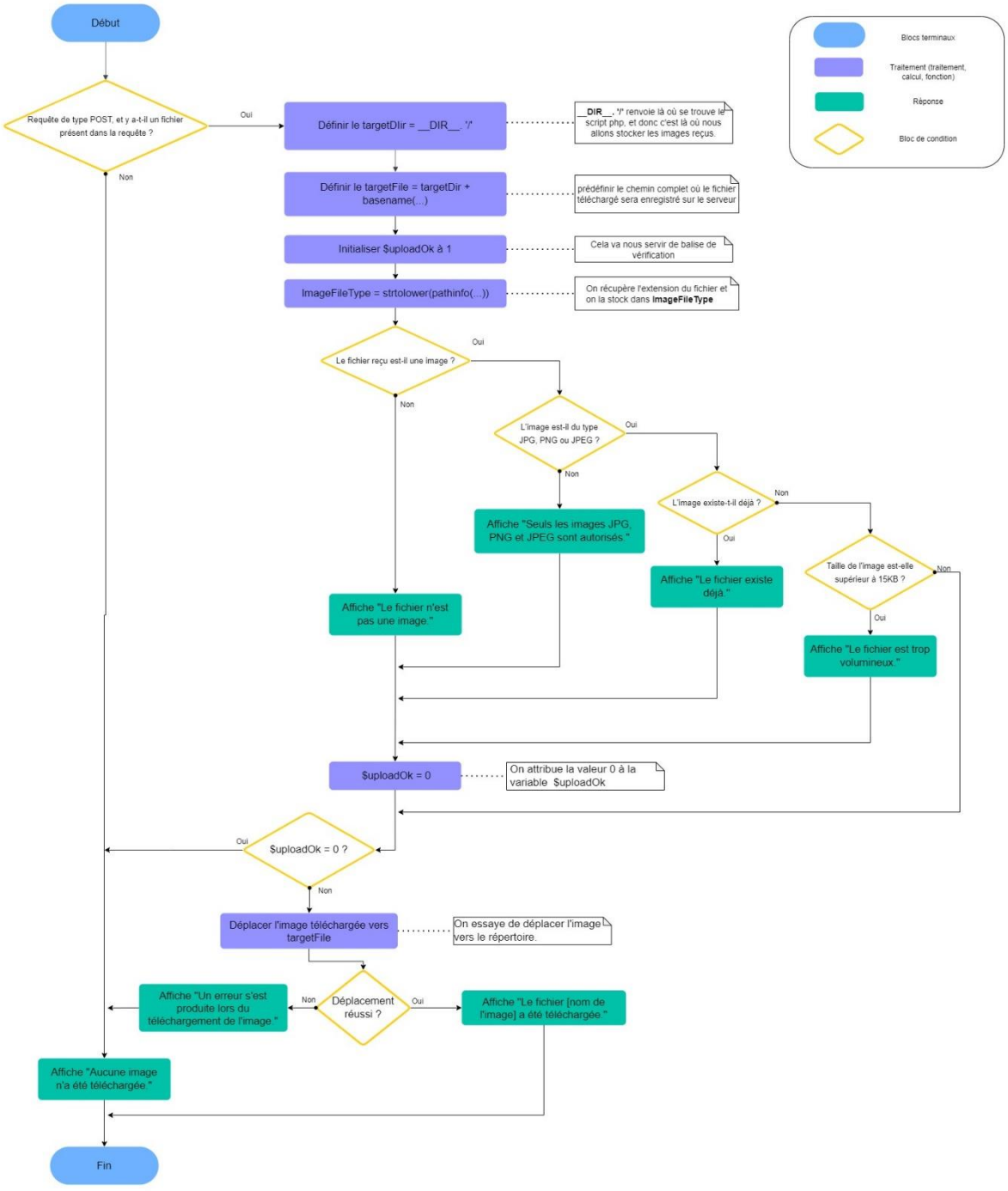
```
C:\Users\namex>curl -X POST -F "image=@C:/Users/namex/OneDrive/Bureau/4.jpg" http://localhost/photos/
Le fichier 4.jpg a ete telecharge.
C:\Users\namex>
```

En testant avec le programme cela fonctionne parfaitement. En conclusion pour résoudre ce problème nous avons donc dû mettre en place un script PHP côté serveur afin de pouvoir gérer les requêtes POST, cependant nous avons tout de même dû modifier la configuration du serveur Apache afin qu'il prenne en compte le script.

```
AddHandler application/x-httpd-php .php
AddType application/x-httpd-php .php .html
LoadModule php_module "C:/php/php8apache2_4.dll"
PHPInidir "C:/php"
DirectoryIndex index.php
```

Je vais tout de même vous détailler le fonctionnement du script PHP présent sur le serveur Apache. Vous trouverez plus bas le fonctionnement du script lorsqu'il reçoit une requête ainsi que les différentes méthodes de vérification.

Algorigramme du script PHP :

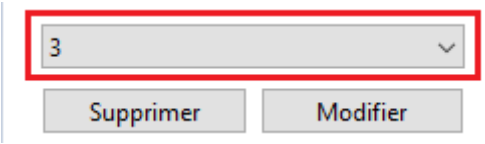


9. **Supprimer un employé :**

La fonction de suppression d'un employé a été intégrée tardivement dans le cycle de développement du projet. En effet, cette fonctionnalité est essentielle pour éliminer toutes les données associées à un employé, ce qui nécessite des fonctionnalités préexistantes bien établies pour être pleinement opérationnelle.

Pour supprimer un employé via le programme, un bouton dédié a été mis en place. Toutefois, la procédure complète pour effacer les données de l'employé de la base de données doit être effectuée à l'aide de requêtes SQL adaptées et spécifiques. Ces requêtes assurent que toutes les informations pertinentes, y compris les enregistrements liés dans d'autres tables, sont correctement supprimées.

La procédure de suppression d'un employé est semblable à celle pour en ajouter un, à quelques détailles près. Tout d'abord on doit récupérer l'id présélectionné dans la ComboBox qui va nous permettre de préparer les requêtes SQL



Dès que ceci est fait le programme prépare les requêtes en y spécifiant l'ID de l'employé concerné.

Les informations qu'on vise à supprimer sont les suivantes : le niveau d'accès, le lien d'accès vers sa photo d'identité, et ses informations personnelles. Mais avant de pouvoir supprimer le lien d'accès vers la photo d'identité il faut la supprimer la photo même qui stocké sur le serveur. Nous récupérons donc d'abord le lien d'accès de la photo d'abord.

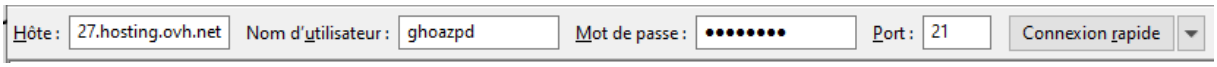
```
QSqlQuery querySelectImage;
querySelectImage.prepare("SELECT lien_photo FROM photosemployes WHERE id_employe = ?");
querySelectImage.bindValue(0, idEmployeSelectionne.toInt());
```

Il est donc à nouveau nécessaire de crée une requête HTTP POST a envoyé au serveur, cependant cette fois-ci en spécifiant que nous voulons supprimer la photo. Néanmoins il est aussi nécessaire d'adapter le code PHP pour que le serveur puisse gérer ce type de demande.

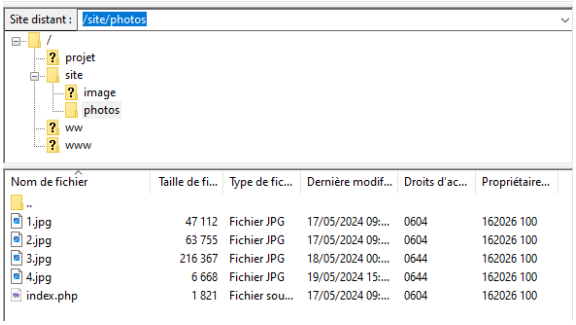
Et grâce à ceci quand un supprime un utilisateur cela se fait également sur le serveur web. Pour effacer le plus de données associées à un employé.

10. Partie commune :

Maintenant que le développement de ma partie personnelle est finalisé, il est nécessaire de mettre à disposition les données nécessaires à mes camarades. Concernant l'hébergement web, je dispose déjà d'un hébergeur chez OVH. OVH, une entreprise française spécialisée dans l'hébergement de serveurs, m'a permis de réorganiser mon espace d'hébergement pour accueillir les images et le script PHP. Pour pouvoir réorganiser les fichiers j'ai dû me connecter à l'espace de stockage en ligne via la méthode FTP.
Pour ce faire il est nécessaire de posséder le client FTP et de renseigner les informations requis à la connexion.



Quand la connexion est établie avec le serveur, on retrouve le répertoire avec les fichiers et ainsi son contenue. C'est grâce à cette méthode là on va pouvoir y déposer notre dossier PHP.



L'avantage avec ce genre d'hébergement c'est qu'une version de php est déjà préinstallé et nécessite donc pas d'une quelconque configuration.

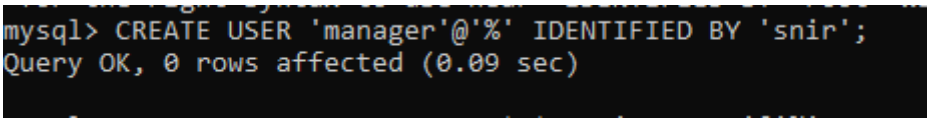
Cette réorganisation a été essentielle pour tester et vérifier si mon programme pouvait effectivement communiquer avec un serveur distant, et pas seulement avec un serveur local. Cela m'a permis de confirmer l'efficacité de mon programme sur un environnement réel.

Cependant je n'ai pas pu me permettre d'utiliser cet hébergement pour la mise en commun, cela n'allait pas résoudre second problème qui est de mettre à disposition aussi la base de données à mes camarades. Et que l'abonnement de l'hébergement allait prendre fin au début du mois de juin.
Je sais qu'actuellement mon programme et mon script php fonctionnent, j'ai donc demandé à professeur s'il était possible de monopoliser un ordinateur du lycée. Depuis cet ordinateur j'ai réinstallé les dépendances nécessaires pour mon projet. Et enfin sur le nouvel ordinateur j'ai importé la base de données qui était présente en local grâce à mysqldump qui me permet d'exporter une base de données d'une machine A vers une machine B, si la machine qui accueil la base de donnée est correctement configurer

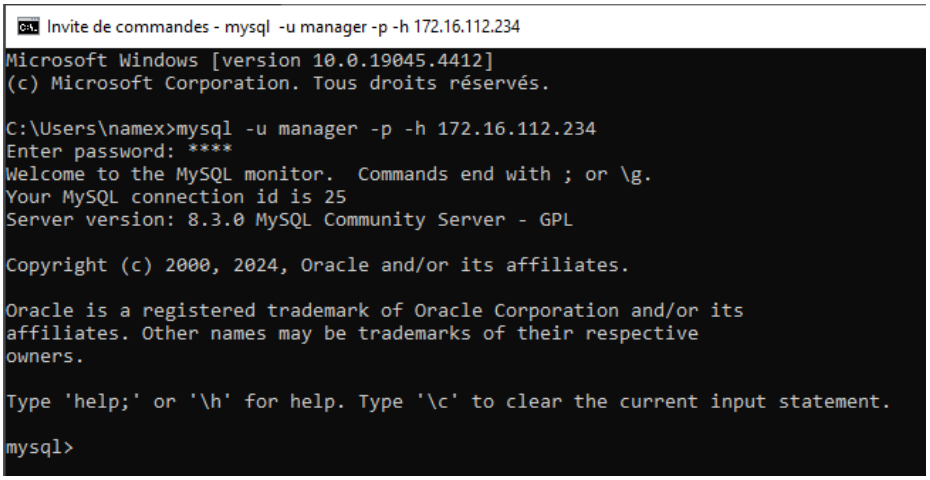
```
C:\Users\namex>mysqldump -u root -p sas >C:/Users/namex/OneDrive/Bureau/sas.sql
Enter password: ****

C:\Users\namex>
```

Maintenant que le serveur est correctement configuré il ne restait plus qu'à créer un utilisateur pour la connexion à la base de données.



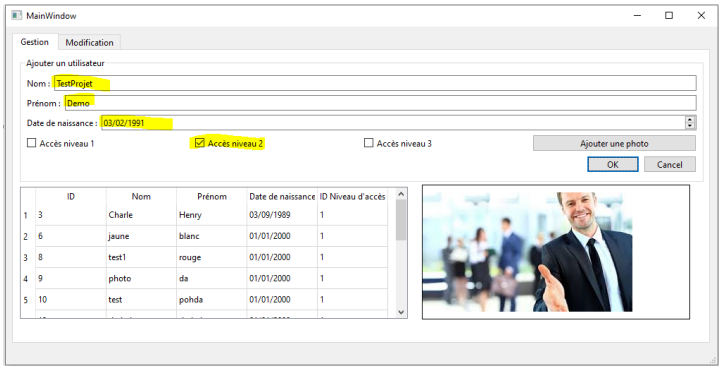
L'utilisateur est créé on lui attribue les permissions nécessaires pour modifier la base de données qui nous intéresse, mais avant d'essayer d'établir une connexion il nécessaire de réaliser quelques modifications au niveau de Windows même. J'ai dû désactiver pare-feu Windows pour pouvoir établir la connexion entre le programme vers la base de données.

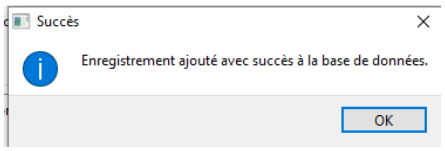


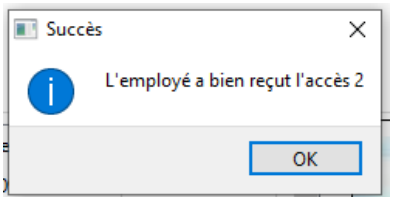
La connexion est bien établie, je peux donc dès à présent interagir avec la base de données depuis mon ordinateur.

11. Démonstration :

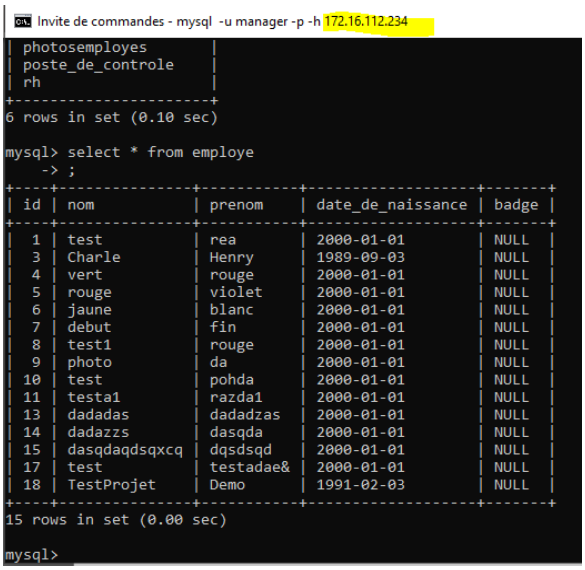
Une fois les informations saisies dans les différents champs du formulaire de notre programme, nous pouvons observer son fonctionnement en temps réel. Après la validation des données, le programme procède à leur enregistrement dans la base de données. Les messages de confirmation affichés à l'écran informent l'utilisateur que les ajouts ont été réalisés avec succès, assurant ainsi que les données ont été traitées et stockées correctement.

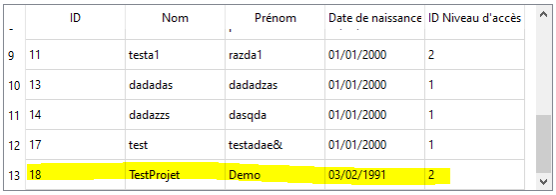




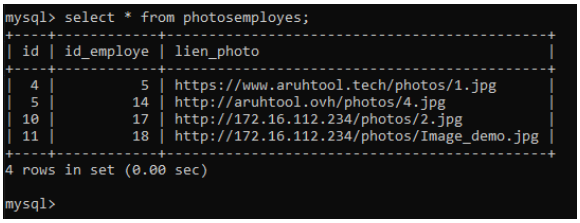


Pour confirmer l'enregistrement, nous effectuons deux vérifications distinctes : la première directement via le programme, où un message de confirmation valide l'ajout, et la seconde en consultant la base de données pour s'assurer que les nouvelles informations y sont correctement enregistrées.





Nous constatons que la colonne "badge" est vide. Cela s'explique par le fait que le module responsable de l'enregistrement des badges ne fait pas partie de mon périmètre de travail et qu'il n'est pas encore opérationnel en raison de problèmes de matériel.



On peut regarder dans la table qui contient les photos des employés on observe aussi la présence du lien menant à l'image que nous avons sélectionné.

Pour vérifier la modification des informations d'un employé, nous allons utiliser le dernier employé enregistré. Nous procéderons à la modification de toutes ses informations afin de confirmer que le processus de mise à jour fonctionne correctement.

Cette méthode nous permet de tester la fonctionnalité de modification, en s'assurant que toutes les données de l'employé peuvent être modifiées sans erreur et que les changements sont bien enregistrés dans la base de données.

Nous recevons à nouveau des messages de confirmation pour chaque modification effectuée. Pour garantir que les changements ont bien pris effet, nous vérifions une fois de plus les informations mises à jour.

```
mysql> select * from employe;
ERROR 2013 (HY000): Lost connection to MySQL server during query
No connection. Trying to reconnect...
Connection id:    23
Current database: sas

+----+------------+-----------+------------------+-------+
| id | nom        | prenom    | date_de_naissance | badge |
+----+------------+-----------+------------------+-------+
|  1 | test       | rea       | 2000-01-01       | NULL  |
|  3 | Charle     | Henry     | 1989-09-03       | NULL  |
|  4 | vert       | rouge     | 2000-01-01       | NULL  |
|  5 | rouge      | violet    | 2000-01-01       | NULL  |
|  6 | jaune      | blanc     | 2000-01-01       | NULL  |
|  7 | debut      | fin       | 2000-01-01       | NULL  |
|  8 | test1      | rouge     | 2000-01-01       | NULL  |
|  9 | photo      | da        | 2000-01-01       | NULL  |
| 10 | test       | pohda     | 2000-01-01       | NULL  |
| 11 | testa1     | razda1    | 2000-01-01       | NULL  |
| 13 | dadadas    | dadadzas  | 2000-01-01       | NULL  |
| 14 | dadazzs    | dasqda    | 2000-01-01       | NULL  |
| 17 | test       | testadae& | 2000-01-01       | NULL  |
| 18 | TestProjet1 | Demo1    | 1991-12-03       | NULL  |
+----+------------+-----------+------------------+-------+
14 rows in set (0.03 sec)

mysql>
```

12. Conclusion :

Le développement de ce projet a débuté à la mi-mars et, jusqu'à aujourd'hui, le 23 mai, nous avons réussi à répondre aux exigences du cahier des charges pour ma partie. Toutes les fonctionnalités demandées ont été implémentées avec succès, assurant ainsi que les besoins du projet.

Cependant, il y a quelques fonctionnalités supplémentaires que j'aurais souhaité mettre en place. Par exemple, un système de journalisation (log) pour suivre les actions et les modifications effectuées aurait été bénéfique la sécurité. De plus, une méthode de connexion sécurisée au site web, afin de restreindre l'accès qu'aux utilisateurs autorisés, aurait renforcé la protection des données. Bien que nous ayons retiré le répertoire général pour éviter l'affichage de toutes les images présentes, ces améliorations auraient apporté une couche supplémentaire de sécurité et de fonctionnalité au projet.

Le principal obstacle rencontré a été la méthode de publication des images sur le serveur Apache. Cette difficulté a engendré une perte de temps d'environ une semaine. Malgré mes efforts pour résoudre ce problème, j'ai dû solliciter l'aide de mes professeurs. Bien qu'ils aient pu me fournir des pistes.

En somme, le travail qui m'avait été confié a atteint ses objectifs principaux, mais il reste des pistes d'amélioration qui pourraient être explorées à l'avenir pour renforcer et enrichir le système développé.

Annexe :

```
#include "mainwindow.h"
#include "ui_mainwindow.h"
#include "mainpage.h"
#include <QMessageBox>
#include <QString>
#include <QtSql>
#include <QSqlDatabase>
#include <QSqlQuery>
#include <QSqlError>

#include <QtWidgets>
MainWindow::MainWindow(QWidget *parent)
   : QMainWindow(parent)
   , ui(new Ui::MainWindow)
{
   ui->setupUi(this);
   QPushButton *pushButton_singin = ui->pushButton_singin;
   connect(pushButton_singin, &QPushButton::clicked, this,
&MainWindow::VerifierLaConnexion);
   QShortcut *shortcut = new QShortcut(QKeySequence(Qt::Key_Return), this);
   connect(shortcut, &QShortcut::activated, this, &MainWindow::VerifierLaConnexion);


}

MainWindow::~MainWindow()
{
   delete ui;
}
```

```
void MainWindow::VerifierLaConnexion()
{
    QString identifiant = ui->lineEdit_identifiant->text();
    QString password = ui->lineEdit_password->text();
    QSqlDatabase db = QSqlDatabase::addDatabase("QMYSQL");
    db.setUserName(identifiant);
    db.setPassword(password);
    db.setHostName("172.16.112.234");
    db.setDatabaseName("sas");
    db.setPort(3306);
    if(db.open()){
        this->hide();
        MainPage *mainpage = new MainPage();
        mainpage->show();
    }
    else {
        QMessageBox::about(nullptr, "Erreur de connexion", "Erreur de connexion à la base de
données : " + db.lastError().text());
    }
}
```

```
#include "mainpage.h"
#include "ui_mainpage.h"
#include <QSqlDatabase>
#include <QSqlQuery>
#include <QMessageBox>
#include <QSqlError>
#include <QSqlQueryModel>
#include <QSqlRecord>
#include <QComboBox>
#include <QNetworkAccessManager>
#include <QNetworkReply>
#include <QFileDialog>
#include <QHttpMultiPart>
#include <QNetworkRequest>
#include <QFile>
#include <QDate>



MainPage::MainPage(QWidget *parent)
    : QMainWindow(parent)
    , ui(new Ui::MainPage)
{
    ui->setupUi(this);
    QNetworkAccessManager *manager;
    qDebug() << QSqlDatabase::drivers();
    manager = new QNetworkAccessManager(this);
    connect(manager, &QNetworkAccessManager::finished,
            this, &MainPage::replyFinished);
    connect(ui->comboBox, QOverload<int>::of(&QComboBox::currentIndexChanged), this,
&MainPage::chargerImageEmployeSelectionne);
```

```
    loadUserData();
    loadbox();
    networkManager = new QNetworkAccessManager(this);

}

MainPage::~MainPage()
{
    delete ui;
}

void MainPage::on_ajouter_clicked()
{
    imagePath = QFileDialog::getOpenFileName(
        this,
        tr("Ouvrir"),
        "",
        tr("Images (*.png *.jpg *.jpeg)")
        );
    QPixmap pixmap(imagePath);
    if (!pixmap.isNull()) {
        QFileInfo fileInfo(imagePath);
        fileName = fileInfo.fileName();
        qDebug() << "Image chargée avec succès";
        // Affichez l'image dans votre QLabel
        ui->imageLabel_2->setPixmap(pixmap);
    } else {
        // Affichez un message d'erreur si le chargement de l'image a échoué
        qDebug() << "Impossible de charger l'image depuis le chemin :" << imagePath;
```

```
    }
}

void MainPage::on_buttonBox_accepted()
{
    QString nom = ui->txt_nom->text();
    QString prenom = ui->txt_prenom->text();
    QDate date_de_naissance = ui->date_de_naissance->date();
    QStringList champsManquants;
    QString serverUrl = "http://172.16.112.234/photos/";
    if (upload(serverUrl, imagePath)) {
        qDebug() << "Votre application a la permission d'écrire dans le répertoire du serveur.";
    } else {
        qDebug() << "Votre application n'a pas la permission d'écrire dans le répertoire du
serveur.";
    }

    if (nom.isEmpty()) {
        champsManquants << "Nom";
    }
    if (prenom.isEmpty()) {
        champsManquants << "Prénom";
    }
    if (!date_de_naissance.isValid()) {
        champsManquants << "Date de naissance";
    }

    if (champsManquants.isEmpty()) {
        // Condition si tous les champs sont ok on continue
        QSqlDatabase db = QSqlDatabase::database();
```

```
QString date_de_naissance_str = date_de_naissance.toString("yyyy-MM-dd");


if (db.isOpen()) {
    // requête sql
    QSqlQuery query;
    query.prepare("INSERT INTO employe (nom, prenom, date_de_naissance) VALUES (:nom, :prenom, :date_de_naissance)");
    query.bindValue(":nom", nom);
    query.bindValue(":prenom", prenom);
    query.bindValue(":date_de_naissance", date_de_naissance_str);
    // Exécutez la requête
    if (query.exec()) {
        qDebug() << "Enregistrement ajouté avec succès à la base de données.";
        // message de succès
        QMessageBox::information(this, "Succès", "Enregistrement ajouté avec succès à la base de données.");
        int idEmploye = query.lastInsertId().toInt();
        qDebug() << "ID de l'employé ajouté : " << idEmploye;
        QString lien = serverUrl + fileName;
        QPixmap pixmap(imagePath);
        if (!pixmap.isNull()){
            QSqlQuery upload;
            upload.prepare("INSERT INTO photosemployes (id_employe, lien_photo) VALUES (:idEmploye, :lien_photo)");
            upload.bindValue(":idEmploye", idEmploye);
            upload.bindValue(":lien_photo", lien);
            if (!upload.exec()) {
                qDebug() << "Erreur: " << upload.lastError().text().toStdString();
            } else {
                qDebug() << "Ajouté avec succès";
```

```
            }
        }
        if(ui->checkBox->isChecked()){
            QSqlQuery querycheck;
            querycheck.prepare("INSERT INTO employe_niveau_acces (id_employe,
id_niveau_acces) VALUES (:idEmploye, 1)");
            querycheck.bindValue(":idEmploye", idEmploye);
            QMessageBox::information(this, "Succès", "L'employé a bien reçut l'accès 1");
            if (!querycheck.exec()) {
                qDebug() << "Erreur lors de l'insertion du niveau d'accès 1 :" <<
querycheck.lastError().text();
            }
        }
        if(ui->checkBox_2->isChecked()){
            QSqlQuery querycheck;
            querycheck.prepare("INSERT INTO employe_niveau_acces (id_employe,
id_niveau_acces) VALUES (:idEmploye, 2)");
            querycheck.bindValue(":idEmploye", idEmploye);
            QMessageBox::information(this, "Succès", "L'employé a bien reçut l'accès 2");
            if (!querycheck.exec()) {
                qDebug() << "Erreur lors de l'insertion du niveau d'accès 2 :" <<
querycheck.lastError().text();
            }
        }
        if(ui->checkBox_3->isChecked()){
            QSqlQuery querycheck;
            querycheck.prepare("INSERT INTO employe_niveau_acces (id_employe,
id_niveau_acces) VALUES (:idEmploye, 3)");
            querycheck.bindValue(":idEmploye", idEmploye);
            QMessageBox::information(this, "Succès", "L'employé a bien reçut l'accès 3");
            if (!querycheck.exec()) {
```

```
                qDebug() << "Erreur lors de l'insertion du niveau d'accès 3 :" << querycheck.lastError().text();
            }
        }
        loadUserData();
        loadbox();
    } else {
        qDebug() << "Erreur lors de l'ajout de l'enregistrement à la base de données:" << query.lastError().text();
        // message d'erreur
        QMessageBox::critical(this, "Erreur", "Erreur lors de l'ajout de l'enregistrement à la base de données:\n" + query.lastError().text());
    }
  } else {
    qDebug() << "Erreur: La connexion à la base de données n'est pas ouverte.";
    // message erreur connexion bdd
    QMessageBox::critical(this, "Erreur", "La connexion à la base de données n'est pas ouverte.");
  }
 } else {
  // message d'erreur détaillé
  QString message = "Les champs suivants sont manquants :";
  foreach (const QString &champ, champsManquants) {
    message += "\n- " + champ;
  }
  qDebug() << message;
  QMessageBox::warning(this, "Erreur", message);
 }
 ui->checkBox->setChecked(false);
 ui->checkBox_2->setChecked(false);
 ui->checkBox_3->setChecked(false);
```

```
    ui->txt_nom->clear();
    ui->txt_prenom->clear();
    QDate dateSpecifique(2000, 1, 1);
    ui->date_de_naissance->setDate(dateSpecifique);
    ui->imageLabel_2->clear();
}

void MainPage::loadUserData()
{
    QSqlQueryModel *model = new QSqlQueryModel();
    QString query = "SELECT employe.id, employe.nom, employe.prenom,
employe.date_de_naissance, employe_niveau_acces.id_niveau_acces"
                    " FROM employe"
                    " INNER JOIN employe_niveau_acces ON employe.id =
employe_niveau_acces.id_employe"
                    " ORDER BY employe.id";

    // Exécuter la requête SQL
    model->setQuery(query);

    // Définir les en-têtes du modèle
    model->setHeaderData(0, Qt::Horizontal, "ID");
    model->setHeaderData(1, Qt::Horizontal, "Nom");
    model->setHeaderData(2, Qt::Horizontal, "Prénom");
    model->setHeaderData(3, Qt::Horizontal, "Date de naissance");
    model->setHeaderData(4, Qt::Horizontal, "ID Niveau d'accès");

    // Afficher le modèle dans la vue
    ui->user_list->setModel(model);
}
```

```
void MainPage::loadbox()
{
   QSqlQueryModel *model = new QSqlQueryModel();
   model->setQuery("SELECT id FROM employe");
   ui->comboBox->setModel(model);
}
void MainPage::replyFinished(QNetworkReply *reply)
{
   if (reply->error() == QNetworkReply::NoError) {
      QByteArray data = reply->readAll();
      QPixmap pixmap;
      pixmap.loadFromData(data);
      ui->imageLabel->setPixmap(pixmap);
   } else {
      // Afficher un message d'erreur en cas de problème lors du chargement de l'image
      QMessageBox::warning(this, "Erreur", "Impossible de charger l'image : " + reply->errorString());
   }

   // Nettoyer la réponse
   reply->deleteLater();
}

void MainPage::chargerImageEmployeSelectionne()
{
   ui->imageLabel->clear();
   ui->checkBox_4->setChecked(false);
   ui->checkBox_5->setChecked(false);
   ui->checkBox_6->setChecked(false);
```

```
    ui->txt_nom_2->clear();
    ui->txt_prenom_2->clear();
    QDate dateSpecifique(2000, 1, 1);
    ui->date_de_naissance_2->setDate(dateSpecifique);
    QString idEmployeSelectionne = ui->comboBox->currentText();
    QSqlQuery query;
    qDebug() << "ID de l'employé sélectionné : " << idEmployeSelectionne; // Message pour afficher l'ID sélectionné
    query.prepare("SELECT lien_photo FROM PhotosEmployes WHERE id_employe = ?");
    query.bindValue(0, idEmployeSelectionne);

    if(query.exec() && query.first()) {
        QString imageUrl = query.value(0).toString();
        qDebug() << "URL de l'image de l'employé : " << imageUrl; // Message pour afficher l'URL de l'image récupérée

        QNetworkAccessManager *manager = new QNetworkAccessManager(this);
        connect(manager, &QNetworkAccessManager::finished, this, &MainPage::replyFinished);
        manager->get(QNetworkRequest(QUrl(imageUrl)));

    } else {
        QMessageBox::warning(this, "Erreur", "Impossible de récupérer les images des l'employés");
    }
    QSqlQueryModel *model = new QSqlQueryModel();
    QString queryString  = "SELECT employe.id, employe.nom, employe.prenom, employe.date_de_naissance, employe_niveau_acces.id_niveau_acces"
                           " FROM employe"
                           " INNER JOIN employe_niveau_acces ON employe.id = employe_niveau_acces.id_employe"
                           " WHERE employe.id = '" + idEmployeSelectionne  + "'";
```

```
    model->setQuery(queryString);
    // Définir les en-têtes du modèle
    model->setHeaderData(0, Qt::Horizontal, "ID");
    model->setHeaderData(1, Qt::Horizontal, "Nom");
    model->setHeaderData(2, Qt::Horizontal, "Prénom");
    model->setHeaderData(3, Qt::Horizontal, "Date de naissance");
    model->setHeaderData(4, Qt::Horizontal, "ID Niveau d'accès");

    // Afficher le modèle dans la vue
    ui->user_list_2->setModel(model);
}

void MainPage::on_load_button_clicked() // bouton reload pour la page modification
{
    loadUserData();
    loadbox();
}
void MainPage::chargerInfosEmployePourModification()
{
    ui->checkBox_4->setChecked(false);
    ui->checkBox_5->setChecked(false);
    ui->checkBox_6->setChecked(false);
    QString idEmployeSelectionne = ui->comboBox->currentText();
    qDebug() << "ID de l'employé sélectionné pour la modification : " << idEmployeSelectionne;

    QSqlQueryModel model;
    QString queryString = "SELECT employe.id, employe.nom, employe.prenom,
employe.date_de_naissance, employe_niveau_acces.id_niveau_acces"
                          " FROM employe"
```

```
                " INNER JOIN employe_niveau_acces ON employe.id =
employe_niveau_acces.id_employe"
                " WHERE employe.id = '" + idEmployeSelectionne + "'";
    model.setQuery(queryString);

    // Vérifier si des données sont disponibles
    if (model.rowCount() > 0) {
        // Récupérer les informations de la première ligne (seule ligne)
        QString nom = model.record(0).value("nom").toString();
        QString prenom = model.record(0).value("prenom").toString();
        QDate dateNaissance = model.record(0).value("date_de_naissance").toDate();
        int idAcces = model.record(0).value("id_niveau_acces").toInt();

        // Remplir les champs avec les informations récupérées
        ui->txt_nom_2->setText(nom);
        ui->txt_prenom_2->setText(prenom);
        ui->date_de_naissance_2->setDate(dateNaissance);
        if (idAcces == 1) {
            ui->checkBox_4->setChecked(true);
        } else if (idAcces == 2) {
            ui->checkBox_5->setChecked(true);
        } else if (idAcces == 3) {
            ui->checkBox_6->setChecked(true);
        }
        qDebug() << "Informations de l'employé récupérées pour la modification : " << nom <<
prenom << dateNaissance << idAcces;
    } else {
        qDebug() << "Aucune information trouvée pour l'employé sélectionné";
    }
}
```

```
void MainPage::on_pushButton_clicked()
{
   chargerInfosEmployePourModification();
}
void MainPage::modifierInfosEmploye()
{
   QString idEmployeSelectionne = ui->comboBox->currentText();
   QString nouveauNom = ui->txt_nom_2->text();
   QString nouveauPrenom = ui->txt_prenom_2->text();
   QDate nouvelleDateNaissance = ui->date_de_naissance_2->date();
   QSqlQuery query;

   // Vérifier les droits d'accès sélectionnés et mettre à jour l'ID d'accès en conséquence
   if(ui->checkBox_4->isChecked()){
      QSqlQuery querycheck;
      querycheck.prepare("UPDATE employe_niveau_acces SET id_niveau_acces = 1 WHERE
id_employe = ?");
      querycheck.bindValue(0, idEmployeSelectionne); // Utiliser bindValue avec l'index 0
      QMessageBox::information(this, "Succès", "L'employé a bien reçu l'accès 1");
      if (!querycheck.exec()) {
         qDebug() << "Erreur lors de la mise à jour du niveau d'accès 1 :" <<
querycheck.lastError().text();
      }
   }
   if(ui->checkBox_5->isChecked()){
      QSqlQuery querycheck;
      querycheck.prepare("UPDATE employe_niveau_acces SET id_niveau_acces = 2 WHERE
id_employe = ?");
      querycheck.bindValue(0, idEmployeSelectionne); // Utiliser bindValue avec l'index 0
```

```
        QMessageBox::information(this, "Succès", "L'employé a bien reçu l'accès 2");
        if (!querycheck.exec()) {
            qDebug() << "Erreur lors de la mise à jour du niveau d'accès 2 :" << querycheck.lastError().text();
        }
    }
    if(ui->checkBox_6->isChecked()){
        QSqlQuery querycheck;
        querycheck.prepare("UPDATE employe_niveau_acces SET id_niveau_acces = 3 WHERE id_employe = ?");
        querycheck.bindValue(0, idEmployeSelectionne); // Utiliser bindValue avec l'index 0
        QMessageBox::information(this, "Succès", "L'employé a bien reçu l'accès 3");
        if (!querycheck.exec()) {
            qDebug() << "Erreur lors de la mise à jour du niveau d'accès 3 :" << querycheck.lastError().text();
        }
    }


    // Préparer la requête SQL de mise à jour
    query.prepare("UPDATE employe SET nom = ?, prenom = ?, date_de_naissance = ? WHERE id = ?");
    query.bindValue(0, nouveauNom);
    query.bindValue(1, nouveauPrenom);
    query.bindValue(2, nouvelleDateNaissance);
    query.bindValue(3, idEmployeSelectionne.toInt()); // Convertir l'ID en entier


    // Exécuter la requête de mise à jour
    if (query.exec()) {
        qDebug() << "Les informations de l'employé ont été mises à jour avec succès.";
        QMessageBox::information(this, "Succès", "Les informations de l'employé ont été mises à jour avec succès.");
```

```
    } else {
        qDebug() << "Erreur lors de la mise à jour des informations de l'employé :" << query.lastError().text();
        QMessageBox::critical(this, "Erreur", "Erreur lors de la mise à jour des informations de l'employé : " + query.lastError().text());
    }
}



void MainPage::on_buttonBox_2_accepted()
{
    modifierInfosEmploye();
    loadUserData();
    loadbox();
}

void MainPage::supprimerUtilisateur() {
    QString idEmployeSelectionne = ui->comboBox->currentText();
    QSqlQuery queryDelete1, queryDelete2;

    queryDelete1.prepare("DELETE FROM employe_niveau_acces WHERE id_employe = ?");
    queryDelete1.bindValue(0, idEmployeSelectionne.toInt());
    if (!queryDelete1.exec()) {
        qDebug() << "Erreur lors de la suppression des données associées dans employe_niveau_acces :" << queryDelete1.lastError().text();
        QMessageBox::critical(this, "Erreur", "Erreur lors de la suppression des données associées dans employe_niveau_acces : " + queryDelete1.lastError().text());
        return; // Sortie de la fonction en cas d'erreur
    }
    QSqlQuery querySelectImage;
```

```
    querySelectImage.prepare("SELECT lien_photo FROM photosemployes WHERE id_employe = ?");
    querySelectImage.bindValue(0, idEmployeSelectionne.toInt());
    if (!querySelectImage.exec()) {
        qDebug() << "Erreur lors de la récupération de l'URL de l'image :" << querySelectImage.lastError().text();
        QMessageBox::critical(this, "Erreur", "Erreur lors de la récupération de l'URL de l'image : " + querySelectImage.lastError().text());
        return; // Sortie de la fonction en cas d'erreur
    }
    if (querySelectImage.next()) {
        QString imageUrl = querySelectImage.value(0).toString();
        if (deleteImage("http://172.16.112.234/photos/", imageUrl)) {
            qDebug() << "L'image a été supprimée avec succès.";
        } else {
            qDebug() << "La suppression de l'image a échoué.";
        }
    } else {
        qDebug() << "Aucune image n'a été trouvée pour l'ID de l'employé :" << idEmployeSelectionne;
    }
    queryDelete2.prepare("DELETE FROM photosemployes WHERE id_employe = ?");
    queryDelete2.bindValue(0, idEmployeSelectionne.toInt());
    if (!queryDelete2.exec()) {
        qDebug() << "Erreur lors de la suppression des données associées dans niveau_acces :" << queryDelete2.lastError().text();
        QMessageBox::critical(this, "Erreur", "Erreur lors de la suppression des données associées dans niveau_acces : " + queryDelete2.lastError().text());
        return; // Sortie de la fonction en cas d'erreur
    }
```

```
    QSqlQuery queryDeleteEmploye;
    queryDeleteEmploye.prepare("DELETE FROM employe WHERE id = ?");
    queryDeleteEmploye.bindValue(0, idEmployeSelectionne.toInt());
    if (queryDeleteEmploye.exec()) {
        qDebug() << "L'utilisateur a été supprimé avec succès.";
        QMessageBox::information(this, "Succès", "L'utilisateur a été supprimé avec succès.");
    } else {
        qDebug() << "Erreur lors de la suppression de l'utilisateur :" <<
queryDeleteEmploye.lastError().text();
        QMessageBox::critical(this, "Erreur", "Erreur lors de la suppression de l'utilisateur : " +
queryDeleteEmploye.lastError().text());
    }
}

void MainPage::on_btn_supprimer_clicked()
{
    connect(ui->btn_supprimer, &QPushButton::clicked, this,
&MainPage::supprimerUtilisateur);
    supprimerUtilisateur();
    loadUserData();
    loadbox();
}


bool MainPage::upload(const QString& serverUrl, const QString& imagePath) {
    QFile file(imagePath);
    if (!file.exists()) {
        qDebug() << "L'image spécifiée n'existe pas.";
        return false;
    }
```

```
    if (file.open(QIODevice::ReadOnly)) {
        QByteArray imageData = file.readAll();

        QNetworkRequest request(serverUrl);
        QHttpMultiPart multiPart(QHttpMultiPart::FormDataType);

        QHttpPart imagePart;
        imagePart.setHeader(QNetworkRequest::ContentTypeHeader,
QVariant("multipart/form-data"));
        imagePart.setHeader(QNetworkRequest::ContentDispositionHeader, QVariant("form-
data; name=\"image\"; filename=\"" + QFileInfo(imagePath).fileName() + "\""));
        imagePart.setBody(imageData);
        multiPart.append(imagePart);

        QNetworkAccessManager manager;
        QNetworkReply* reply = manager.post(request, &multiPart);
        QEventLoop loop;
        QObject::connect(reply, &QNetworkReply::finished, &loop, &QEventLoop::quit);
        loop.exec();

        if (reply->error() == QNetworkReply::NoError) {
            qDebug() << "Réponse du serveur Apache : " << reply->readAll();
            reply->deleteLater();
            return true;
        } else {
            qDebug() << "Erreur lors de l'envoi du fichier :" << reply->errorString();
            reply->deleteLater();
            return false;
        }
```

```
    } else {
        qDebug() << "Impossible de lire l'image spécifiée.";
        return false;
    }
}
bool MainPage::deleteImage(const QString& serverUrl, const QString& imageUrl) {
    QNetworkRequest request(serverUrl);
    request.setHeader(QNetworkRequest::ContentTypeHeader, "application/x-www-form-urlencoded");

    QByteArray postData;
    postData.append("delete=").append(imageUrl.toUtf8());

    QNetworkAccessManager manager;
    QNetworkReply* reply = manager.post(request, postData);
    QEventLoop loop;
    QObject::connect(reply, &QNetworkReply::finished, &loop, &QEventLoop::quit);
    loop.exec();

    if (reply->error() == QNetworkReply::NoError) {
        qDebug() << "Réponse du serveur Apache : " << reply->readAll();
        reply->deleteLater();
        return true;
    } else {
        qDebug() << "Erreur lors de la suppression de l'image :" << reply->errorString();
        reply->deleteLater();
        return false;
    }
}
```