Documentation du projet Air Tany

Table des matières

Définition du projet

Ce projet est un jeu d'entreprise simulant une salle de marchés. Il s'agit de modéliser le fonctionnement d'une salle de marché traitant des opérations de bourse et de réaliser une application permettant de simuler le fonctionnement d'une salle de marché (de façon élémentaire).

Dossier de documentation technique

Ce dossier de documentation vise à décrire les solutions techniques mises en place dans le cadre de la réalisation du projet, en vue de permettre le déploiement, la maintenance corrective et la maintenance évolutive du projet par une autre équipe de développement.

Solutions matérielles et logicielles de déploiement du projet

Expression des besoins

Le déploiement du projet nécessite les mises en place suivantes :

- Serveur de bases de données MySQL
  - Version approuvée : 10.6.5-MariaDB-1:10.6.5+maria~focal *(on considère les versions ultérieures compatibles)*
  - Espace de stockage : 100 Mégaoctets minimum recommandés

- Base de données préexistante
  - Nom de la base de données : au choix *(“airtany” utilisé pour le développement)*
  - Moteur de stockage : InnoDB *(permet la prise en charge de relations entre les tables)*

- Client web et / ou logiciel permettant la manipulation d’un serveur de bases de données MySQL *(client web “phpMyAdmin” utilisé pour le développement)*

- Serveur web Apache hébergeant l’API de cours des actions du projet
  - Version approuvée : Apache/2.4.38 (Debian) *(on considère les versions ultérieures compatibles)*
  - Prise en charge du langage de programmation PHP indispensable *(PHP 8 minimum)*
  - Extensions PHP :
    - “mbstring” *(manipulation de chaînes de caractères contenant des caractères UTF-8 spéciaux)*
    - “mysqli” *(connexion au serveur de bases de données MySQL*

- Application Windows Forms (.NET Framework C#)
  - Version approuvée : .NET Framework 4.7.2 *(on considère les versions ultérieures compatibles)*
  - Modules complémentaires :
    - “MySql.Data.MySqlClient” *(client de connexion au serveur de bases de données MySQL*

Matériel

Dans le cadre du développement et du déploiement initial du projet, l'infrastructure matérielle suivante a été mise en place :

- Serveur privé virtuel (ci-après désigné **VPS**)
    - Hébergeur : entreprise spécialisée OVHcloud
    - Système d'exploitation : Debian GNU/Linux 10 (Buster)
    - Configuration matérielle :
        - Cœurs de processeur virtuels (vCores) : 1
        - Mémoire (RAM) : 2 Gigaoctets
        - Espace de stockage : 40 Gigaoctets (SSD)

- Poste de travail
    - Système d'exploitation : Windows 10
    - Configuration matérielle :
        - Processeur (CPU) : Intel Core i7
        - Cœurs de processeur : 6
        - Fréquence de rafraîchissement du processeur : 2,6 GHz
        - Mémoire (RAM) : 16 Gigaoctets
        - Espace de stockage : 512 Gigaoctets (SSD)

Logiciel

Dans le cadre du développement et du déploiement initial du projet, l'infrastructure logicielle suivante a été mise en place :

- Serveur de bases de données MySQL MariaDB
  - Hébergement matériel : VPS
  - Version : 10.6.5-MariaDB-1:10.6.5+maria~focal

- Base de données "airtany"
  - Hébergement logiciel : serveur de bases de données MySQL MariaDB
  - Moteur de stockage : InnoDB

- Serveur web Apache
  - Hébergement matériel : VPS
  - Version : Apache/2.4.38 (Debian)
  - Prise en charge du langage de programmation PHP (version 7.4.20)
  - Extensions PHP :
    - "mbstring"
    - "mysqli"

- Client web "phpMyAdmin" permettant la manipulation du serveur de bases de données MySQL MariaDB
  - Hébergement logiciel : serveur web Apache
  - Version : 5.1.1

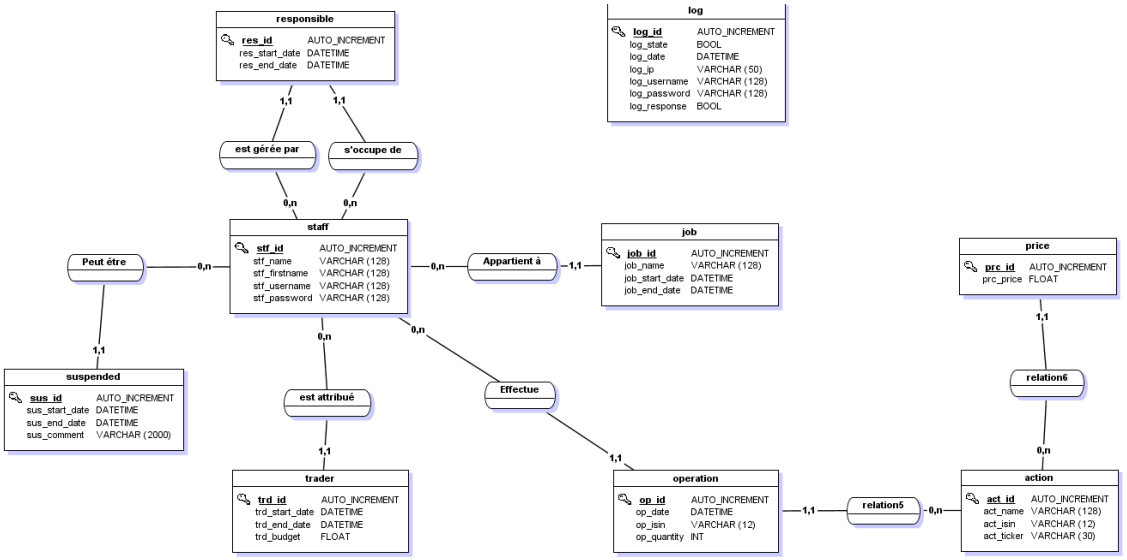
- Application Windows Forms (.NET Framework C#)
  - Hébergement matériel : poste de travail
  - Version : .NET Framework 4.7.2

- Code source de l'API de cours des actions du projet
  - Hébergement logiciel : serveur web Apache

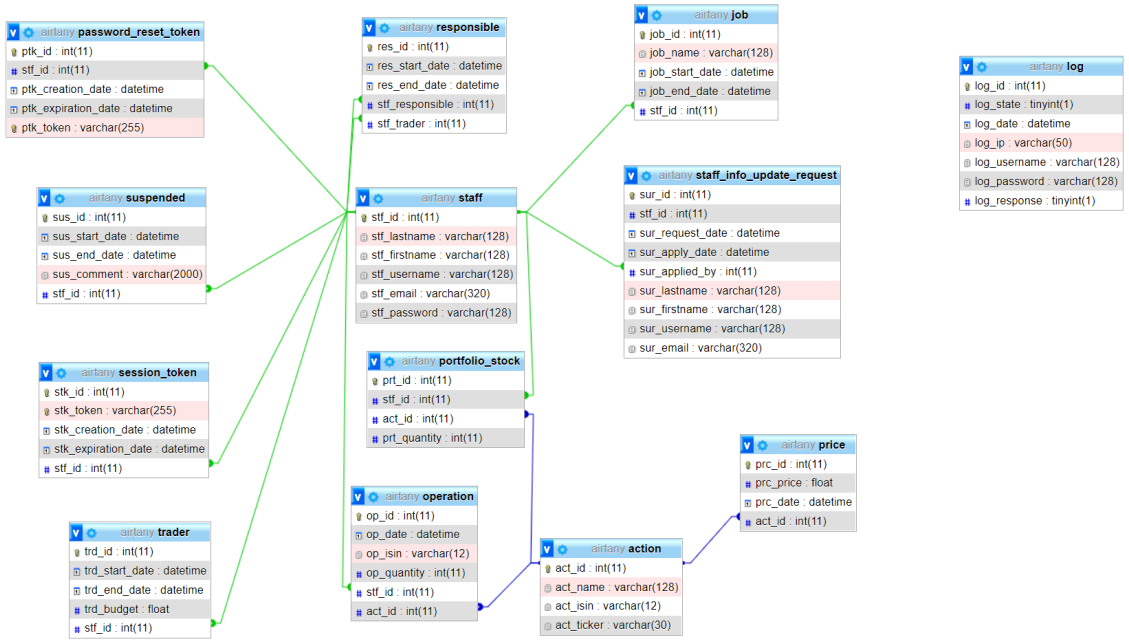
Base de données

Structure

Modèle conceptuel de données initial



Modèle logique de données initial



Description des tables

La base de données de l'application comporte les tables suivantes :

- "staff" : contient les informations personnelles fixes des utilisateurs de l'application, comme son nom et son mot de passe ;

- "job" : contient un historique des postes occupés par chaque utilisateur, en reliant l'identifiant de l'utilisateur à son poste (Trader, Responsable, Administrateur), avec les dates de début et de fin d'occupation du poste ;

- "suspended" : sauvegarde un historique des suspensions de personnel, en précisant l'employé, le motif et les dates de début et de fin de suspension ;

- "trader" : comporte l'historique des budgets alloués aux traders, en enregistrant les dates associées à ces budgets ;

- "responsible" : garde un historique des attributions des responsables aux traders, en précisant les dates associées ;
-
- "staff_info_update_request" : contient les demandes de modification des informations personnelles d'un employé, faites par lui-même, ainsi que les informations de l'administrateur qui a confirmé la demande, le cas échéant ;

- "session_token" : contient les jetons de sessions utilisateurs et les données associées (date de création, d'expiration…) ;

- "password_reset_token" : stocke les jetons de demandes de réinitialisation de mots de passe ;

- “portfolio_stock” : correspond aux portefeuilles d’actions des traders de l’application ;

- “operation” : enregistre les opérations d’achat et vente d’actions effectuées par les traders ;

- “action” : stocke les informations fixes concernant les actions (nom, numéro ISIN, ticker) ;

- “price” :

Script de sauvegarde automatisée

```
#! /bin/bash

backup_dir='/Volumes/STOREVA/Sauvegardes/ppe/db'
tmp_file="$backup_dir/.tmp"

DB_HOST='ADRESSE IP'
DB_PORT=PORT'
DB_USER='airtany_dump'
DB_PASSWORD='MOT DE PASSE'
DB_DATABASE='airtany'

DUMP_CMD='/usr/local/bin/mysqldump'
DUMP_OPTIONS='--skip-dump-date --column-statistics=0'
```

On déclare les variables nécessaires à la sauvegarde et à la connexion à la base de données. Il faut donc renseigner les identifiants de connexion au serveur de base de données MySQL. Dans notre cas, le compte d'utilisateur "airtany_dump" a été créé spécifiquement pour cet usage, avec uniquement les permissions nécessaires.

"DUMP_CMD" représente le chemin vers l'exécutable qui permet l'export de la base de données.

Dans la variable "DUMP_OPTIONS", on définit des options supplémentaires à passer à l'exécutable. L'option "--skip-dump-date" demande au programme de ne pas insérer la date et l'heure dans le fichier de sauvegarde, ce qui empêcherait la comparaison de deux sauvegardes. L'option "--column-statistics=0" désactive la récupération des données de statistiques de MySQL, qui nécessite des droits particuliers sur la base de données et qui ne nous est pas utile.

```
if [[ ! -d "$backup_dir" ]]; then exit; fi
```

Cette condition permet de vérifier si le dossier où on stocke la sauvegarde existe. S'il n'existe pas, le programme est arrêté.

```
"$DUMP_CMD" $DUMP_OPTIONS -h "$DB_HOST" -p"$DB_PASSWORD" -u "$DB_USER" -P \
"$DB_PORT" "$DB_DATABASE">"$tmp_file"
if [[ "$?" != "0" ]]; then exit; fi
```

On exécute la commande de sauvegarde avec les paramètres de connexion à la base de données, et on stocke la sauvegarde dans un fichier temporaire. La dernière ligne arrête le programme si la commande de sauvegarde a retourné une erreur.

```
last_file=$(ls "$backup_dir"|tail -1)
if [[ -f "$backup_dir/$last_file" ]]
then
   if [[ $(diff "$tmp_file" "$backup_dir/$last_file"|wc -l|xargs -L1 echo) == 0 ]]
```

```
   then
       is_new=0
   else
       is_new=1
   fi
else
   is_new=1
fi
```

On récupère le nom de la dernière sauvegarde. Si le fichier portant ce nom existe, et qu'il est identique au nouveau fichier temporaire, on stocke le résultat "0" (signifiant "la sauvegarde n'est pas nouvelle") dans une variable. Sinon, on stocke "1".

```
if [[ "$is_new" == "0" ]]
then
   rm "$tmp_file"
else
   new_file="$backup_dir/$(date +'%Y-%m-%d_%H-%M').sql"
   mv "$tmp_file" "$new_file"
fi
```

Cette portion de code vérifie le résultat retourné par la condition précédente.

Si le résultat est "0", la nouvelle sauvegarde est identique à l'ancienne donc on supprime le fichier temporaire.

Si le résultat est "1", la sauvegarde est différente donc on la déplace dans le dossier de sauvegarde "backup_dir" et on la renomme en incluant la date et l'heure actuelles, permettant de trier les sauvegardes par ordre chronologique.

```
$ crontab -l
@hourly ~/script/ppe-db-dump.sh
```

On utilise le système de tâches régulières "cron" intégré aux distributions GNU/Linux et à macOS pour exécuter le script précédemment décrit à intervalles réguliers : toutes les heures à heure pile (15h00, 16h00, 17h00 et ainsi de suite).

Application Windows Forms (.NET Framework C#)

Air-Tany-App : ce fichier contient l'ensemble des parties fonctionnelles.
Air-Tany-Lib : contient les dépendances.

Air-Tany-lib

Common.cs

Le fichier contient toutes les fonctions utiles au développement du projet.

```
2 références
static public string rdmString(int length = 16, string charList = "0123456789abcdefghijklmnopqrstuvwxyz")
{
    char[] result = new char[length];
    Random rand = new Random();
    for (int i = 0; i < length; i++)
    {
        int pos = rand.Next(charList.Length);
        result[i] = charList[pos];
    }
    return new String(result);
```

Cette fonction permet de générer une chaîne de caractères aléatoire. Elle prend en paramètre une longueur ainsi qu'une liste de caractères.

```
1 référence
static public int? checkUserCredentials(string UserName, string PasswordHash, DBConn Conn) // le ? peut etre nul
{
    MySqlCommand cmd = new MySqlCommand($"SELECT `stf_id` FROM `staff` WHERE `stf_username` = '{UserName}' AND `stf_password` = '{PasswordHash}'", Conn.Connection);
    try
    {
        int? Uid = (int?)cmd.ExecuteScalar();
        return Uid;
    }
    catch
    {
        return null;
    }
}
```

La fonction "checkUserCredentials" vérifie si l'utilisateur existe et retourne son identifiant si c'est le cas.

```
static public string createSessionToken(int? Uid, DBConn Conn)
{

    if (Uid != null)
    {
        string rdmToken = rdmString(255, "0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ");
        MySqlCommand cmd = new MySqlCommand($"INSERT INTO `session_token`( `stk_token`, `stk_creation_date`, " +
            $"`stf_id`) VALUES ('{rdmToken}',NOW(),'{Uid}'); SELECT LAST_INSERT_ID();", Conn.Connection);
        int res = (int)(UInt64) cmd.ExecuteScalar();
        if (res != -1)
        {
            return rdmToken;
        }
        else
        {
            return null;
        }
    }
    else
    {
        return null;
    }
}
```

Cette méthode permet de générer un jeton (“token”) aléatoire de 255 caractères, à partir de la méthode “rdmString” que nous avions défini auparavant. Elle va par la suite vérifier si le token est bien inséré dans la base de données en récupérant l’identifiant de la dernière ligne insérée en base de données. Si c’est le cas, on retourne le token généré.

```
public static string SHA512(string input)
{
    var bytes = System.Text.Encoding.UTF8.GetBytes(input);
    using (var hash = System.Security.Cryptography.SHA512.Create())
    {
        var hashedInputBytes = hash.ComputeHash(bytes);

        var hashedInputStringBuilder = new System.Text.StringBuilder(128);
        foreach (var b in hashedInputBytes)
            hashedInputStringBuilder.Append(b.ToString("X2"));
        return hashedInputStringBuilder.ToString();
    }
}
```

La méthode “SHA512” est une fonction de hachage. Elle permet de rendre n’importe quelle chaîne de caractère méconnaissable de ce qu'elle est initialement. Elle nous sert à rendre les mots de passe illisibles avant de les insérer en bases de données mais aussi, lors de la connexion utilisateur, de hacher le mot de passe pour le faire correspondre à celui de la base de données.

```
2 references
public static UserInfo GetUserInfo(DBConn Conn, string TK)
{
    MySqlCommand cmd = new MySqlCommand($"SELECT `stf_id`, `stf_lastname`, `stf_firstname`, " +
        $"`stf_username`, `stf_email` FROM `session_token` NATURAL JOIN `staff` WHERE session_token.stk_token = '{TK}'", Conn.Connection);
    using (MySqlDataReader reader = cmd.ExecuteReader())
    {
        while (reader.Read())
        {
            return new UserInfo((int)reader["stf_id"], reader["stf_lastname"].ToString(), reader["stf_firstname"].ToString(),
                reader["stf_username"].ToString(), reader["stf_email"].ToString());
        }
        return null;
    }
}
```

La méthode “GetUserInfo” permet de retourner les informations de l’utilisateur stockées en base de données, en fonction de son token de session. On crée ensuite une instance de la classe “UserInfo” à partir de ces informations.

```
public static string GetJob(int id, DBConn Conn)
{
    MySqlCommand req = new MySqlCommand($"SELECT `job_name` FROM `job` WHERE `job`.`stf_id` = '{id}' AND " +
        $"(`job`.`job_start_date` <= NOW()) AND (`job`.`job_end_date` >= NOW() OR `job`.`job_end_date` IS NULL) " +
        $"ORDER BY `job`.`job_start_date` DESC LIMIT 1", Conn.Connection);
    string res = (string)req.ExecuteScalar();
    if (!string.IsNullOrEmpty(res))
    {
        return res;
    }
    else
    {
        return null;
    }
}
```

Cette fonction nous permet de récupérer le poste actuellement occupé, ou le dernier occupé par un utilisateur, en passant son identifiant en paramètre.

UserInfo.cs

```
public class UserInfo
{
    private int _id;
    private string _lastname;
    private string _firstname;
    private string _username;
    private string _email;

    1 référence
    public UserInfo(int id, string lastanme, string firstname, string username, string email)
    {
        _id = id;
        _lastname = lastanme;
        _firstname = firstname;
        _username = username;
        _email = email;
    }

    1 référence
    public int id
    {
        get => _id;
    }

    1 référence
    public string lastname
    {
        get => _lastname;
    }

    1 référence
    public string firstname
    {
        get => _firstname;
    }

    0 références
    public string username
    {
        get => _username;
    }

    0 références
    public string email
    {
        get => _email;
    }
}
```

Cette classe stocke les informations d'un utilisateur. Elle est définie par un constructeur et des accesseurs qui permettent d'utiliser ou de modifier les informations privées de cette classe.

DBConn.cs

```
public class DBConn
{
    private MySqlConnection _connection;

    1 référence
    public DBConn(string port, string host, string dataBase, string userName, string pwd)
    {
        MySqlConnectionStringBuilder connBuilder = new MySqlConnectionStringBuilder();

        connBuilder.Port = uint.Parse(port);
        connBuilder.Server = host;
        connBuilder.Database = dataBase;
        connBuilder.UserID = userName;
        connBuilder.Password = pwd;
        connBuilder.OldGuids = true;

        _connection = new MySqlConnection(connBuilder.ConnectionString);
    }
    9 références
    public MySqlConnection Connection
    {
        get => _connection;
    }

    1 référence
    public bool Open()
    {
        try
        {
            _connection.Open();
            return true;
        }
        catch(Exception exc)
        {
            Console.WriteLine(exc.Message);
            return false;
        }
    }
}
```

La classe “DBConn” représente une connexion à une base de données MySQL. Son constructeur appelle une instance de la classe ”MySqlConnectionStringBuilder”, contenue dans le module importé MySql.Data.MySqlClient.63 v

-)/7 En lui passant les paramètres de la connexion à la base de données, à savoir l’hôte, le port, le nom de la base de données, le nom d’utilisateur et son mot de passe, la classe “MySqlConnectionStringBuilder” retourne la chaîne de connexion à la base de données correspondante, qu’on utilise ensuite pour créer cette connexion.

Air-Tany-App

Connexion à la base de données

Fichier DBConn.cs

```
public class DBConn
{
    private MySqlConnection _connection;

    1 référence
    public DBConn(string port, string host, string dataBase, string userName, string pwd)
    {
        MySqlConnectionStringBuilder connBuilder = new MySqlConnectionStringBuilder();

        connBuilder.Port = uint.Parse(port);
        connBuilder.Server = host;
        connBuilder.Database = dataBase;
        connBuilder.UserID = userName;
        connBuilder.Password = pwd;
        connBuilder.OldGuids = true;

        _connection = new MySqlConnection(connBuilder.ConnectionString);
    }
    9 références
    public MySqlConnection Connection
    {
        get => _connection;
    }

    1 référence
    public bool Open()
    {
        try
        {
            _connection.Open();
            return true;
        }
        catch(Exception exc)
        {
            Console.WriteLine(exc.Message);
            return false;
        }
    }
```

Cette classe définit une connexion à un serveur de base de données MySQL. A son constructeur on passe les informations de connexion.
on utilise la classe MySqlConnectionStringBuilder pour générer la chaîne de connexion à partir de ces informations.
La classe contient une méthode Open permettant d'ouvrir la connexion.

fichier MainLayout.cs

```
public bool CreateConnection()
{
    string host = ConfigurationManager.ConnectionStrings["DBHost"].ConnectionString;
    string port = ConfigurationManager.ConnectionStrings["DBPort"].ConnectionString;
    string dataBase = ConfigurationManager.ConnectionStrings["DBDatabase"].ConnectionString;
    string userName = ConfigurationManager.ConnectionStrings["DBUsername"].ConnectionString;
    string pwd = ConfigurationManager.ConnectionStrings["DBPassword"].ConnectionString;
    Program.connection = new DBConn(port, host, dataBase, userName, pwd); // insère la connexio
    return Program.connection.Open();

}
```

Cette méthode permet de déterminer les informations qui se trouvent dans le fichier de configuration de l'application (App.config).
Ses informations seront passées au constructeur de la classe DBConn pour cree la connexion a la base de données.

fichier App.config

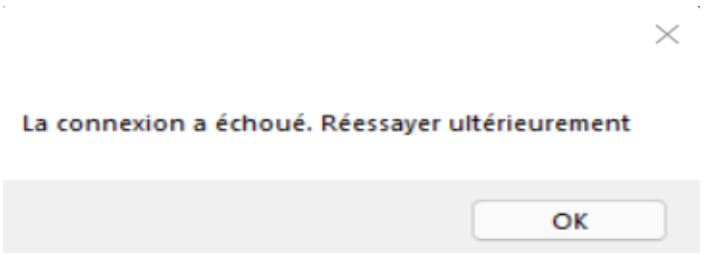
```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
    <startup>
        <supportedRuntime version="v4.0" sku=".NETFramework,Version=v4.7.2" />
    </startup>
    <connectionStrings>
        <add name="DBHost" connectionString="46.105.29.54"/>
        <add name="DBPort" connectionString="1632"/>
        <add name="DBDatabase" connectionString="airtany"/>
        <add name="DBUsername" connectionString="atwf001"/>
        <add name="DBPassword" connectionString="JMqsIpC1muazbgC4pVHMe6QRvL9YKzN6"/>
    </connectionStrings>
</configuration>
```

fichier Program.cs

```
    }
    static public string sessionToken;
    static public DBConn connection;
    static public string execPath = Application.ExecutablePath;
    static private IEnumerable<string> _tmp = execPath.Split('\\').Take(execPath.Length - 2);
    static public string appPath = string.Join("\\", _tmp.Skip(1));
}
```

La connexion est stockée dans une variable du contexte du programme entier, ce qui la rend accessible depuis tous les formulaires de l'application.

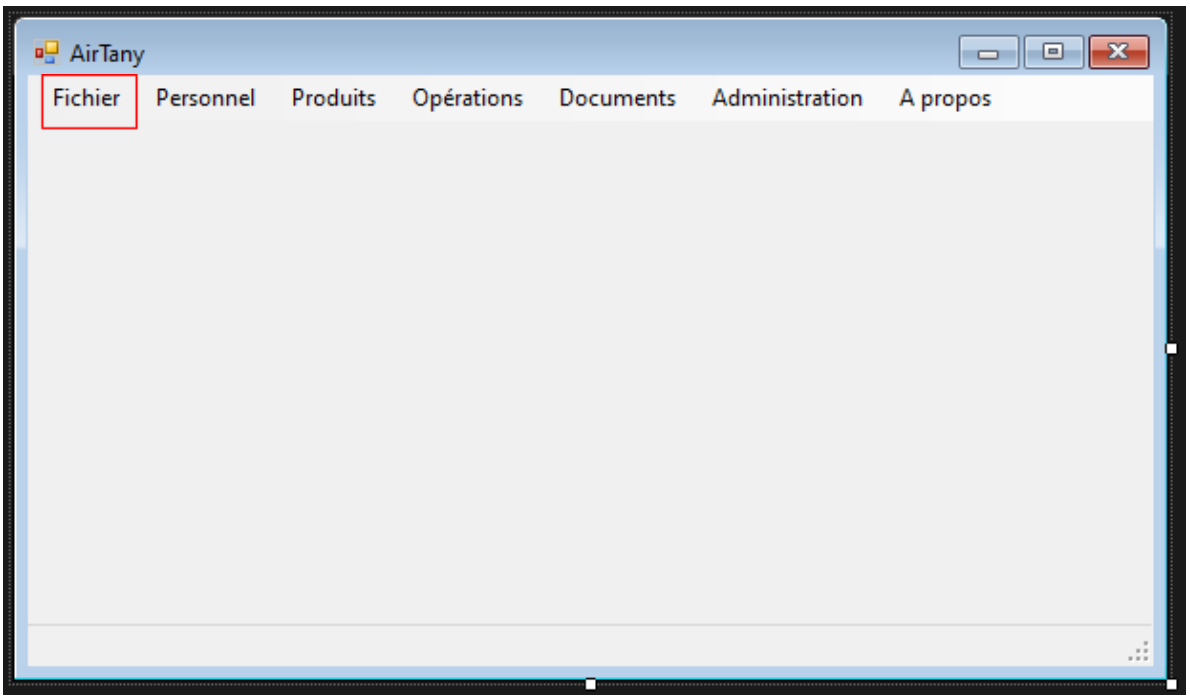
Si la connexion a échoué ce message est affiché. soit le serveur de base de donnée est hors service, soit les informations sont incorrectes, soit vous n'êtes pas connecté à internet.

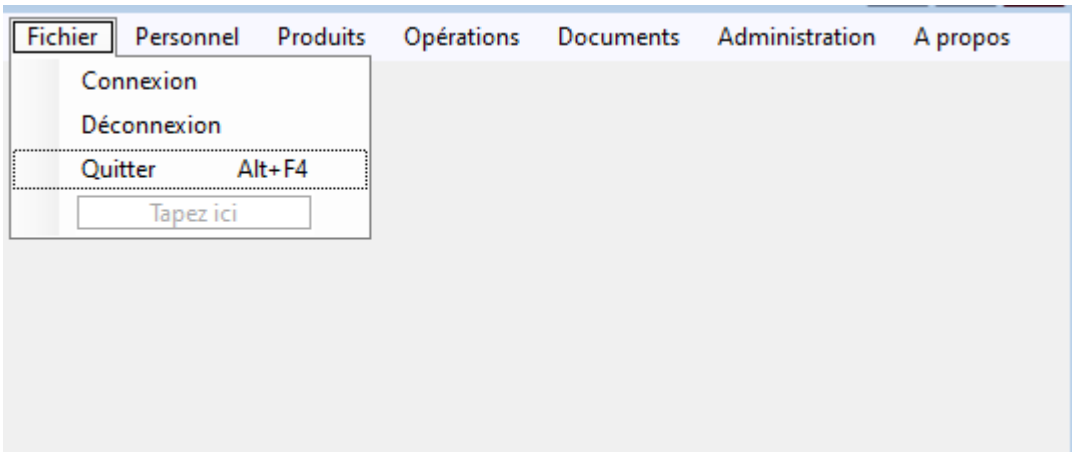


MainLayout.cs(page d'accueil)

le menu est déterminer par un objet menuStrip

Menu Fichier

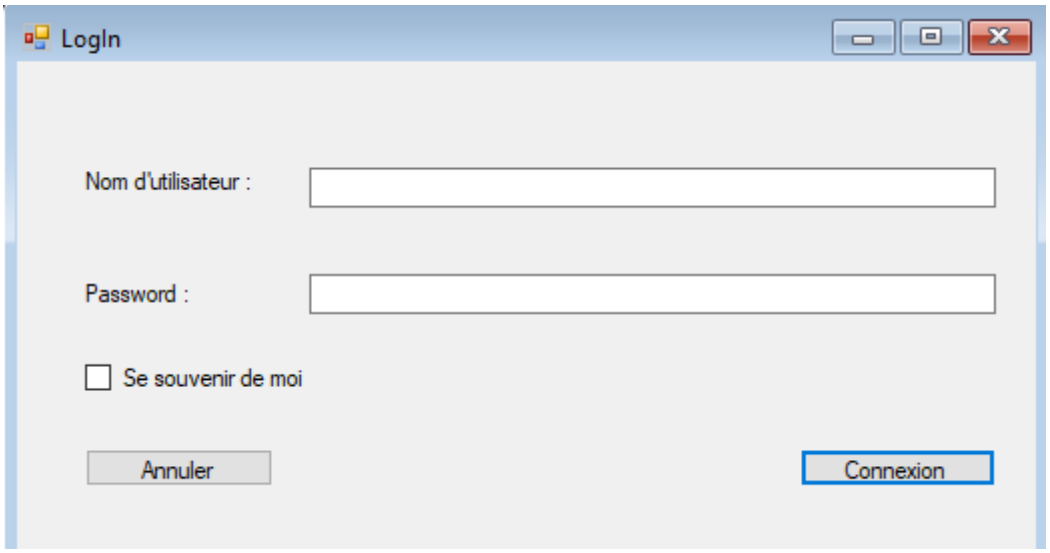




il contient 3 sous menu il y a tout d'abord le menu de connexion utilisateur

```
private void OptConnect_Click(object sender, EventArgs e)
{
    LogIn connectionForm = new LogIn();
    connectionForm.ShowDialog();
    if (!string.IsNullOrEmpty(Program.sessionToken))
    {
        UserInfo userInfo = Common.GetUserInfo(Program.connection, Program.sessionToken);
        tsslLogIn.Text = $"Bonjour {userInfo.firstname} {userInfo.lastname}";
        ShowAdminMnu();
    }
}
```

On appelle une instance de la classe login qui nous renvois à une autre fenêtre




```
using Air_Tany_Lib;
```

```
1 reference
public void submitForm()
{
    _userName = txbUsername.Text;
    _passwordHash = Common.SHA512(txbPassword.Text);
    _stayLogIn = cbxStayLogIn.Checked;

    int? uid = Common.checkUserCredentials(_userName, _passwordHash, Program.connection);
    if (uid.HasValue)
    {
        string TK = Common.createSessionToken(uid, Program.connection);
        if (TK != null)
        {
            Program.sessionToken = TK;

        }
    }
    Close();
}
```

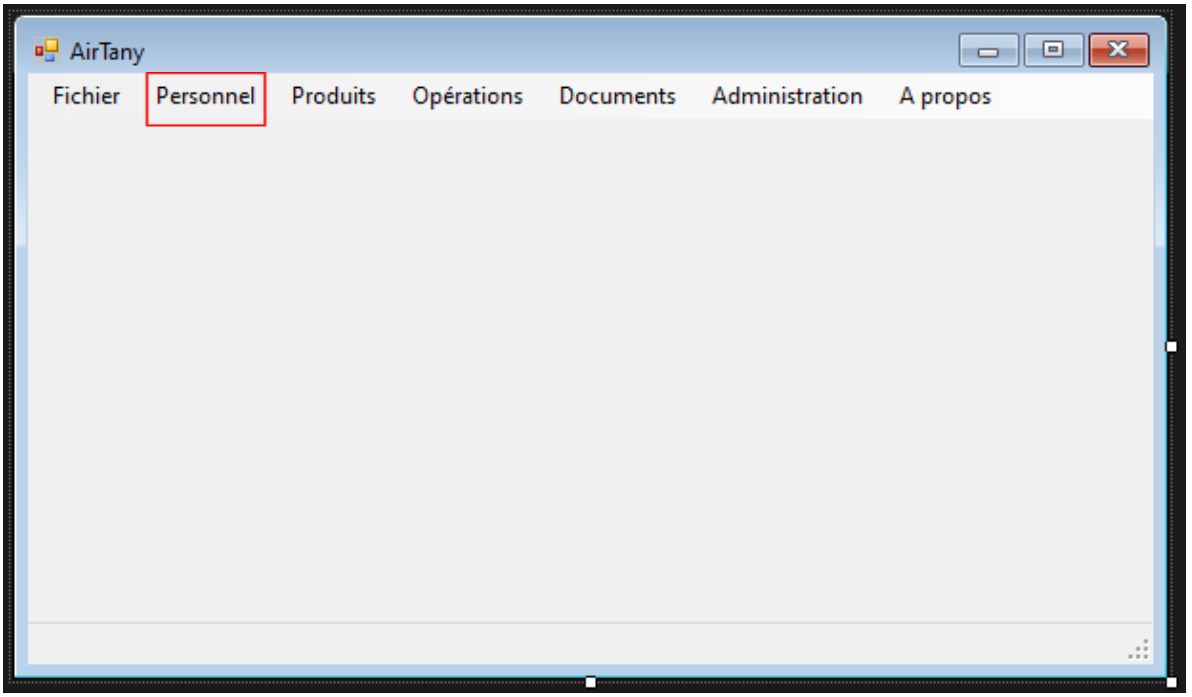
Tout d'abord il est nécessaire de using air_tany_lib pour utiliser nos méthodes préalablement créées.
On crée une fonction submitForm qui récupère les information entrer le mot de passe est haché par la méthode SHA512 que nous avons préalablement créé dans common.cs.
On vérifie si l'utilisateur existe via la fonction checkUseCredentials qui nous renvoie son id que l'on stock dans uid. Il nous permet de créer un token de session que l'on associe a Program.sessionToken.

On a donc la session token on peut retourner dans MainLayout

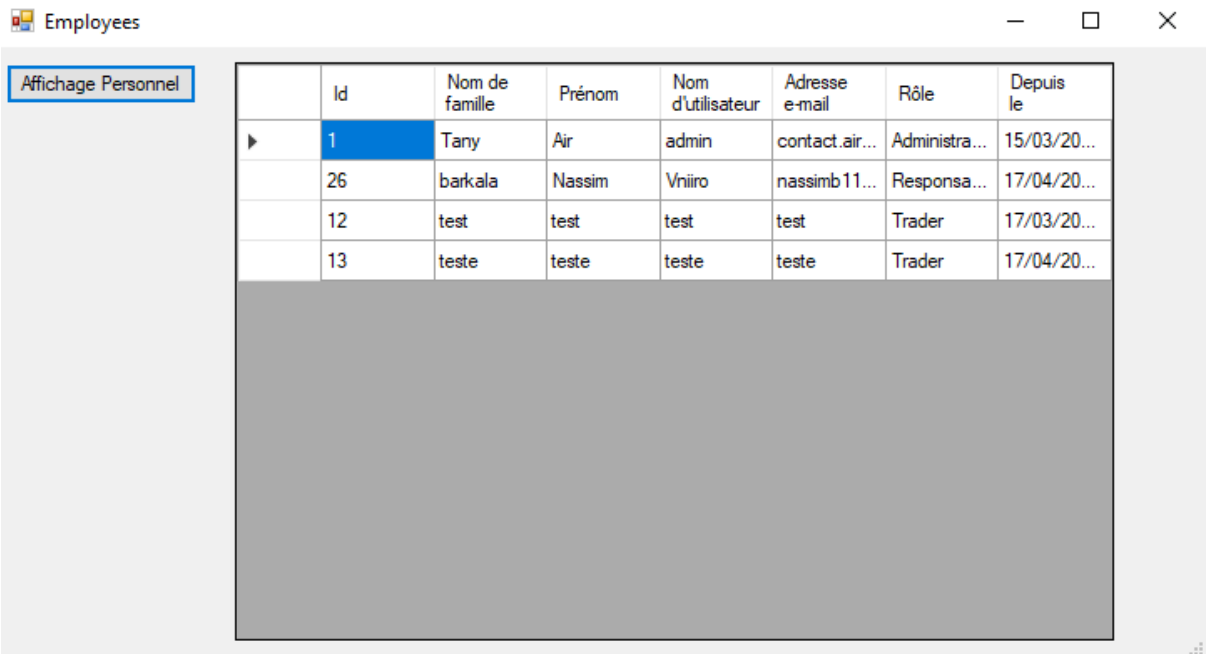
```
if (!string.IsNullOrEmpty(Program.sessionToken))
{
    UserInfo userInfo = Common.GetUserInfo(Program.connection, Program.sessionToken);
    tsslLogIn.Text = $"Bonjour {userInfo.firstname} {userInfo.lastname}";
    ShowAdminMnu();
}
```

A partir du Token préalablement sauvegarder dans Program. on stocke les information de l'utilisateur dans une instance de la classe UserInfo a partir de la méthode getUserInfo.

Menu Personnel

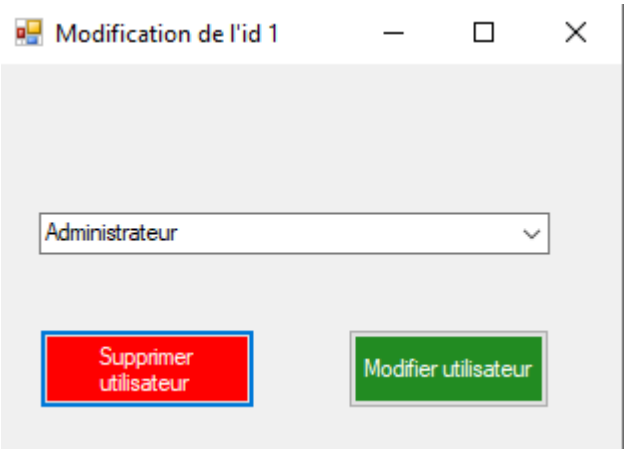


Le menu "Personnel" permet d'ouvrir une nouvelle fenêtre contenant une table listant tous les utilisateurs contenu en base de données. Il est accessible que par les administrateurs de l'application.

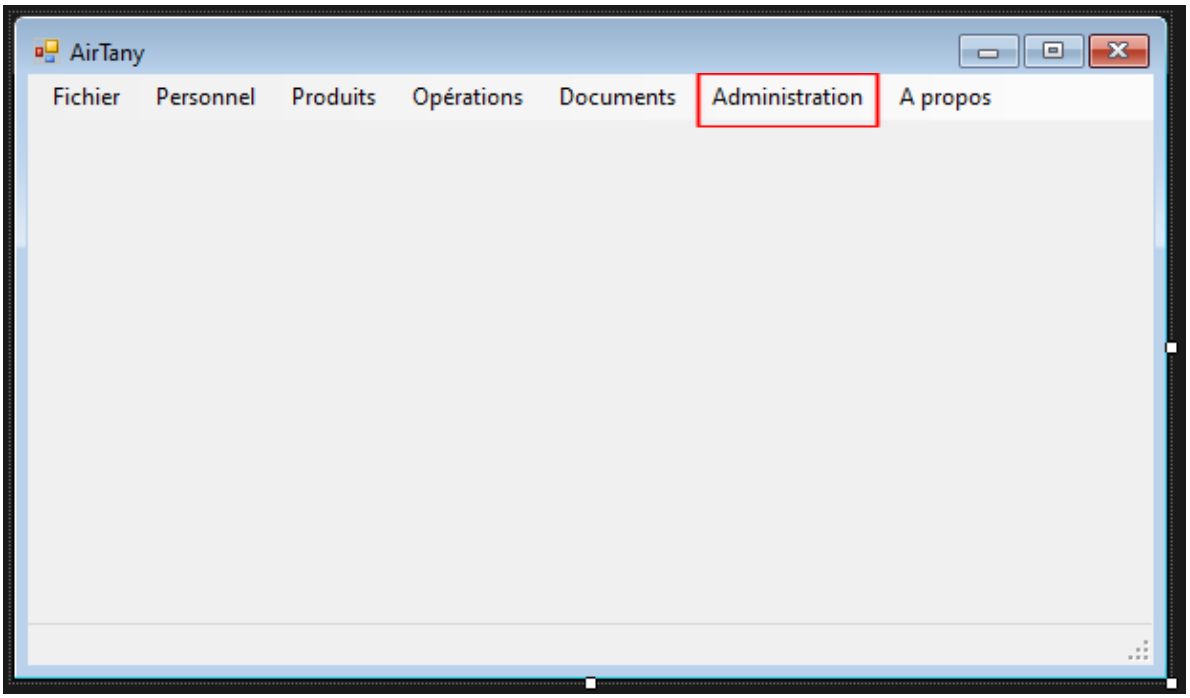



```
1 référence
private void dgvPersonnel_RowHeaderMouseDoubleClick(object sender, DataGridViewCellMouseEventArgs e)
{
    int id = int.Parse(dgvPersonnel.Rows[e.RowIndex].Cells[0].Value.ToString());//converti en entier
    ModifStaff staff = new ModifStaff(id);
    staff.ShowDialog();
    staff.Dispose();
}
```

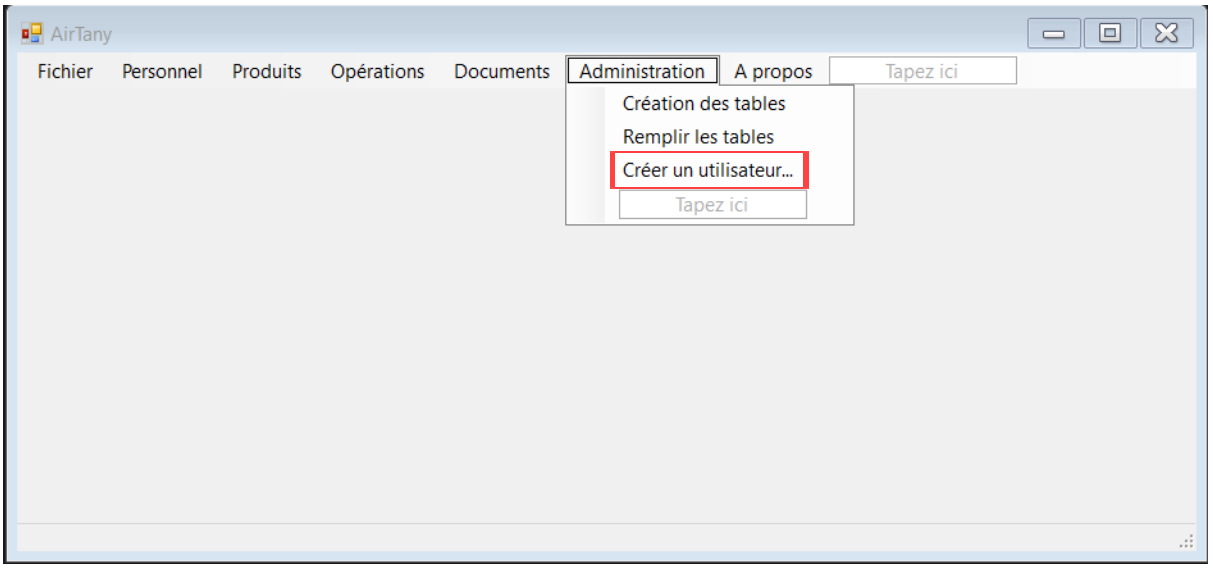
Il est possible de double-clic sur l'id d'un utilisateur pour ouvrir une nouvelle fenêtre permettant de modifier le rôle d'un utlisateur.

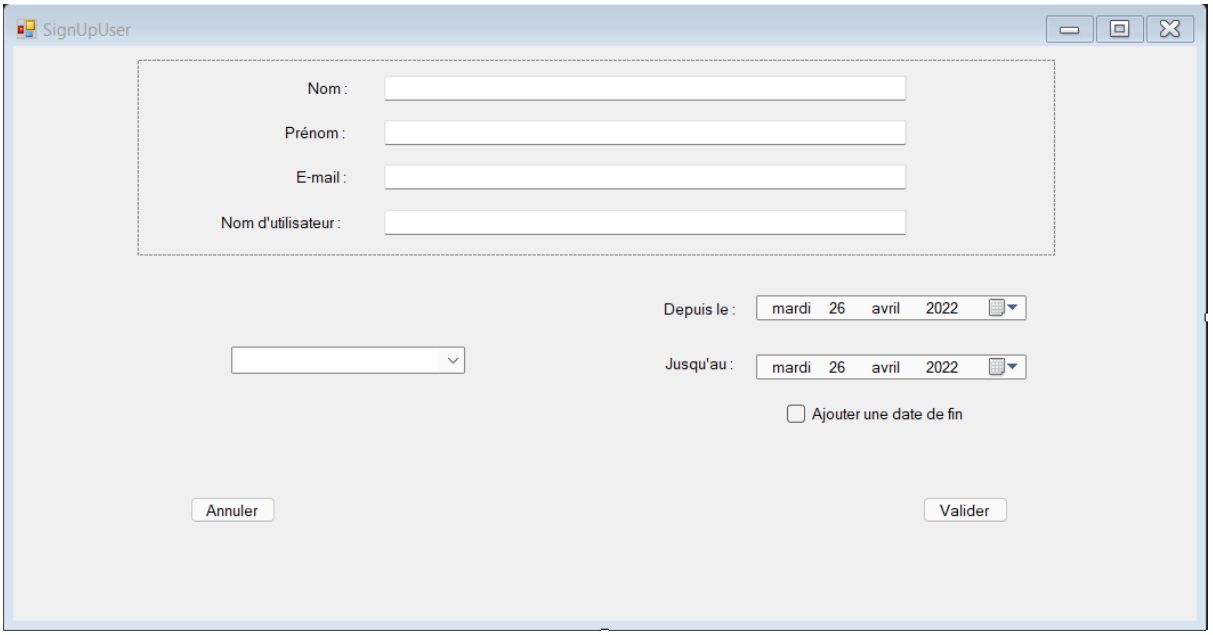


Menu Administration



Le Menu “Administration” offre aux utilisateurs de l’application possédant le rôle Administrateur des options de gestion de la base de données et des membres du personnel de l’application.





L'option "Créer un utilisateur…" ouvre un formulaire d'inscription d'utilisateur. L'administrateur saisit les informations du nouveau compte : nom et nom d'utilisateur, adresse e-mail, poste et dates de début et de fin du poste.

```
private void btnValidate_Click(object sender, EventArgs e)
{
    string lastname = txbName.Text;
    string firstname = txbFirstname.Text;
    string username = txbUsername.Text;
    string mail = txbMail.Text;
    string job = cbxJob.Text;
    DateTime start_date = dtpStartDate.Value;
    DateTime end_date = dtpEndDate.Value;
    string password = Common.rdmString(16, "0123456789abcdefghijklmnopqrstuvwxyz");
    MySqlCommand signup = new MySqlCommand($"INSERT INTO staff (`stf_lastname`, " +
        $"`stf_firstname`, `stf_username`, `stf_email`, `stf_password`) " +
        $"VALUES ('{lastname}', '{firstname}', '{username}', '{mail}', " +
        $"'{Common.SHA512(password)}'); SELECT LAST_INSERT_ID()", Program.connection.Connection);
    int res = (int)(UInt64)signup.ExecuteScalar();
    if (res != -1)
    {
        MySqlCommand q = new MySqlCommand($"INSERT INTO job (`job_name`, " +
            $"`job_start_date`, `job_end_date`, `stf_id`) VALUES ('{job}', '{start_date.ToString("s")}'" +
            $", {(cbxEndDate.Checked ? $"'{end_date.ToString("s")}'" : "NULL")}, '{res}')",
            Program.connection.Connection);
        q.ExecuteNonQuery();

    }
    Close();
}
```

Une fois le formulaire validé, les informations saisies sont récupérées. Le mot de passe est une chaîne de 16 caractères alphanumériques générée aléatoirement. Une requête SQL tente d'ajouter le nouvel utilisateur à la table "staff" de la base de données. Si l'opération réussit, on effectue une deuxième requête sur la base "job" pour lier le poste sélectionné à l'utilisateur, à partir de l'identifiant inséré lors de la requête précédente.

Dossier de documentation utilisateur