



Ecole Nationale des Sciences Appliquées de Tanger

RAPPORT : GESTION D'ÉCOLE

Fait par :

ALLAM ELARBI

DAHBI MOAD

Encadré par :

Pr. Ghailani Mohamed

Date : 13/03/2025

Table des matières

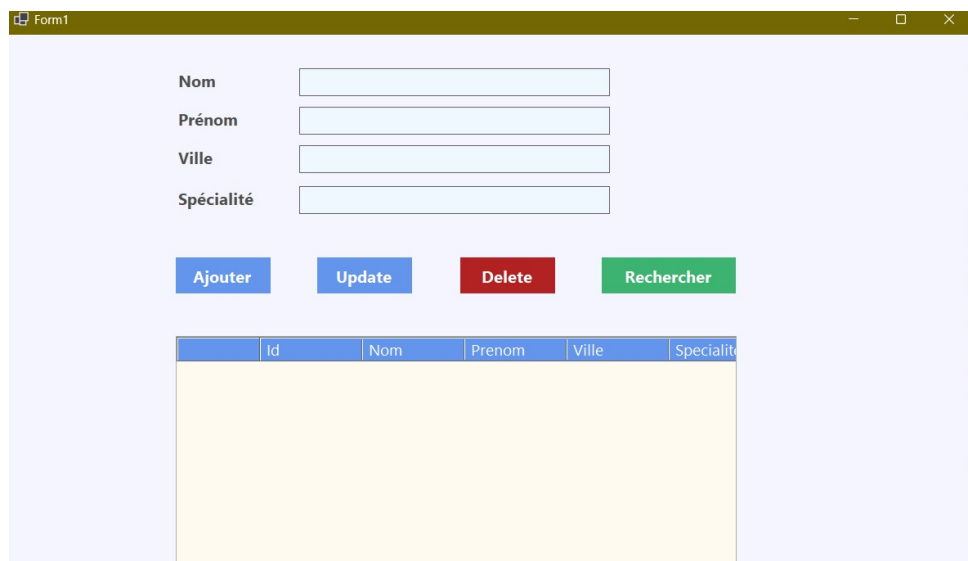
1	Introduction	1
2	Avantages	2
3	Inconvénients	3
4	Améliorations suggérées	4
5	Conclusion	7

Chapitre 1

Introduction

L'application *Gestion_Ecole* est un système de gestion des étudiants qui permet d'insérer, de mettre à jour, de supprimer et de rechercher des étudiants dans une base de données MySQL. L'application suit une approche orientée objet en utilisant des interfaces et un modèle d'accès aux données (*DAO*) pour les interactions avec la base de données.

L'interface utilisateur est développée avec Windows Forms (*WinForms*).



The screenshot shows a Windows Forms application window titled "Form1". The interface is light blue and contains the following elements:

- Four text input fields stacked vertically, labeled "Nom", "Prénom", "Ville", and "Spécialité".
- Four buttons below the input fields: "Ajouter" (blue), "Update" (blue), "Delete" (red), and "Rechercher" (green).
- A table below the buttons with a yellow background and a blue header. The header has six columns: "Id", "Nom", "Prenom", "Ville", and "Specialite". The table body is currently empty.

FIGURE 1.1 – Capture d'écran de l'application Gestion_Ecole (exemple).

Chapitre 2

Avantages

- **Conception modulaire** : L'utilisation des interfaces (IDAO et IConnexion) améliore la réutilisabilité et la maintenabilité du code.
- **Implémentation du modèle DAO** : La classe *DAOEleve* contient la logique métier des opérations de base de données, garantissant une séparation claire des responsabilités.
- **Encapsulation** : La classe *Eleve* encapsule correctement les attributs des étudiants, améliorant ainsi l'intégrité des données.
- **Security (Requêtes paramétrées)** : Empêche l'injection SQL en utilisant des requêtes paramétrées basées sur un dictionnaire.
- **Fonctionnalités CRUD de base** : Fournit les opérations fondamentales sur la base de données (Insertion, Mise à jour, Suppression, Recherche, Recherche globale).
- **Interface WinForms** : Offre une interface utilisateur simple pour interagir avec la base de données.

Chapitre 3

Inconvénients

- **Absence de gestion des exceptions** : Aucune gestion des erreurs dans les opérations de base de données, ce qui peut entraîner des plantages si une connexion échoue ou si une requête SQL est invalide.
- **Fermeture de connexion et *Singleton Pattern* manquante** : La connexion à la base de données est ouverte mais jamais fermée explicitement, ce qui peut entraîner des fuites de mémoire. De plus, le *pattern* Singleton n'est pas implémenté pour éviter l'ouverture multiple de la même base de données.
- **Identifiants de base de données en dur** : La méthode de connexion dans *Connexion* utilise des valeurs par défaut codées en dur (par exemple, nom d'utilisateur root, sans mot de passe), ce qui constitue un risque de sécurité.
- **Manque de retour d'erreur pour les utilisateurs** : En cas d'échec d'une opération, aucun mécanisme ne permet de notifier l'utilisateur via l'interface.

Chapitre 4

Améliorations suggérées

- **Implémenter la gestion des exceptions :**
 - Ajouter des blocs *try-catch* dans toutes les opérations de base de données pour gérer les problèmes de connexion et les erreurs SQL.
 - Afficher des messages d'erreur significatifs aux utilisateurs au lieu de provoquer des échecs silencieux.
- **Gérer correctement la connexion à la base de données :**
 - Implémenter la fermeture explicite des connexions (`CloseConnection()`) après la fermeture de l'application pour éviter les fuites de mémoire.

```
public void CloseConnection()
{
    try
    {
        if (cnx != null && cnx.State == ConnectionState.Open)
        {
            cnx.Close();
            cnx.Dispose();
            cnx = null;
            Console.WriteLine("Connexion closed");
        }
    }
    catch (Exception ex)
    {
        Console.WriteLine("Erreur lors de la fermeture de la connexion : " + ex.Message);
    }
}
```

FIGURE 4.1 – Exemple de méthode `CloseConnection()` pour fermer la connexion à la BDD.

- Fermer la base de données après la fermeture de l'application.

```

namespace Gestion_Ecole
{
    internal static class Program
    {
        private static Connexion connexion = Connexion.GetInstance();
        /// <summary>
        /// The main entry point for the application.
        /// </summary>
        [STAThread]
        static void Main()
        {
            // To customize application configuration such as set high DPI settings or default font,
            // see https://aka.ms/applicationconfiguration.
            ApplicationConfiguration.Initialize();
            Application.Run(new Form1());
            connexion.CloseConnection();
        }
    }
}

```

FIGURE 4.2 – Exemple d’appel à `CloseConnection()` après la fermeture du formulaire principal.

— **Singleton Pattern :**

- Mettre en place le *pattern* Singleton pour éviter l’ouverture multiple de la même base de données.

```

internal class Connexion : IConnexion
{
    // Instance unique de la connexion (Singleton)
    ❖ IL code
    private static Connexion instance;

    // Objet pour la connexion MySQL
    ❖ IL code
    private MySqlConnection cnx;
    ❖ IL code
    private MySqlCommand cmd;

    /// <summary>
    /// Constructeur privé pour empêcher l'instanciation directe de la classe.
    /// Utilise le **modèle Singleton** pour garantir une seule instance.
    /// </summary>
    ❖ IL code
    private Connexion()
    {
    }
}

```

FIGURE 4.3 – *Connexion* implémentant le *Singleton Pattern* (partie 1).

```
public static Connexion GetInstance()
{
    if (instance == null)
    {
        instance = new Connexion();
    }
    return instance;
}
```

FIGURE 4.4 – *GetInstance()* permettant de n’instancier qu’une seule fois la connexion (partie 2).

- **Renforcer la sécurité :**
 - Éviter l’utilisation d’identifiants de base de données en dur dans le code et externaliser ces paramètres dans un fichier de configuration.
 - Utiliser des mécanismes de gestion de mots de passe sécurisés.
- **Améliorer l’interface utilisateur :**
 - Fournir des retours d’erreurs clairs à l’utilisateur via des *MessageBox* ou d’autres éléments visuels.
 - Ajouter des contrôles de validité pour les champs saisis dans les formulaires (par exemple, empêcher les champs obligatoires d’être vides).

Chapitre 5

Conclusion

L'application *Gestion_Ecole* est un système de gestion des étudiants bien structuré, respectant de bonnes pratiques de conception. Cependant, plusieurs améliorations sont nécessaires, notamment en matière de gestion des exceptions, de gestion des connexions, d'amélioration de l'interface utilisateur et d'implémentation de la fonction de mise à jour. En corrigeant ces problèmes, l'application deviendra plus robuste, sécurisée et conviviale.