جامعة مولاي إسماعيل
UNIVERSITÉ MOULAY ISMAÏL
EST MEKNÈS
المدرسة العليا للتكنولوجيا
ÉCOLE SUPÉRIEURE DE TECHNOLOGIE
Département GI

Module : UML & DotNet

Elément : .NET

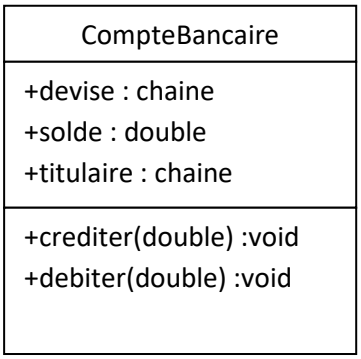
Chapitre 1 : Maîtriser la syntaxe et les concepts du langage C#

1 Syntaxe de base des classes et héritage

1.1 Écriture d'une classe

Une fois la modélisation d'une classe terminée, il est possible de la traduire dans n'importe quel langage de programmation orienté objet.

Considérant la classe suivante :



Voici la traduction en C# de la classe CompteBancaire.

```
public class CompteBancaire
{
    public string titulaire;
    public double solde;
    public string devise;

    public void Crediter(double montant)
    {
        solde = solde + montant;
    }
    public void Debiter(double montant)
    {
        solde = solde - montant;
    }
}
```

La définition d'une classe commence par le mot-clé class. On retrouve ensuite la définition des champs (attributs) et des méthodes de la classe. On remarque que les méthodes utilisent (et modifient) les valeurs des attributs.

Une méthode est une sorte de mini-programme. On peut y déclarer des variables locales et y utiliser tous les éléments de programmation déjà connus (alternatives, boucles, etc.).

1.2 Utilisation d'une classe

En utilisant le modèle fourni par la classe, il est possible de créer autant d'objets que nécessaire. Différents objets d'une même classe disposent des mêmes attributs et des mêmes méthodes, mais les valeurs des attributs sont différentes pour chaque objet. Par exemple, tous les comptes bancaires auront un solde, mais sauf exception, ce solde sera différent pour chaque compte.

Le programme C# ci-dessous utilise la classe CompteBancaire définie plus haut pour créer le compte de Amine et y effectuer deux opérations.

```
static void Main(string[] args)
{
   CompteBancaire compteAmine;         // déclaration d'un nouvel objet
   compteAmine = new CompteBancaire();  // instanciation de cet objet

   // affectations de valeurs aux attributs
   compteAmine.titulaire = "Amine";
   compteAmine.solde = 0;
   compteAmine.devise = "DH";

   // appels des méthodes
   compteAmine.Crediter(300);
   compteAmine.Debiter(500);

   string description = "Le solde du compte de " + compteAmine.titulaire +
      " est de " + compteAmine.solde + " " + compteAmine.devise;
   Console.WriteLine(description);
   Console.Read();
}
```

1.3 Déclaration et instanciation

On remarque que la création de l'objet `compteAmine` se fait en deux étapes :

1. Déclaration de l'objet ;
2. Instanciation de l'objet :

```
CompteBancaire compteAmine;         // déclaration d'un nouvel objet
compteAmine = new CompteBancaire();  // instanciation de cet objet
// ...
```

La déclaration permet de créer une nouvelle variable, appelée `compteAmine` dans l'exemple ci-dessus. À ce stade, aucune réservation de mémoire n'a eu lieu pour cet objet. Il est donc inutilisable.

Le type de la variable compteAmine est CompteBancaire. Le type d'un objet est sa classe.

L'instanciation utilise l'opérateur new, qui permet de réserver une zone mémoire spécifique pour l'objet. On dit que l'objet est instancié par l'opérateur new. Sans cette étape indispensable, l'objet déclaré ne peut pas être utilisé.

DÉFINITION : l'instanciation est l'opération qui consiste à créer un nouvel objet à partir d'une classe, au moyen de l'opérateur new.

Il est courant de rassembler sur une même ligne déclaration et instanciation d'un objet.

```
CompteBancaire compteAmine = new CompteBancaire();  // déclaration et instanciation d'un nouvel objet
// ...
```

ATTENTION : ne pas confondre instanciation et initialisation :

- instancier, c'est créer un nouvel objet (opérateur new) ;
- initialiser, c'est donner une valeur initiale à quelque chose (opérateur =).

1.4 Ajout d'une méthode

Comme il est lourd de construire la description d'un compte bancaire à chaque fois que l'on en a besoin, on voudrait ajouter une opération de description du compte. Cette action est réalisée par une nouvelle méthode nommée Decrire ajoutée à la classe CompteBancaire. La description est renvoyée sous la forme d'une chaîne de caractères (`string`).

```
public class CompteBancaire
{
  // ...

  // Renvoie la description d'un compte
  public string Decrire()
  {
    string description = "Le solde du compte de " +  titulaire + " est de " + solde + " " + devise;
    return description;
  }
}
```

1.5 Instanciation de plusieurs objets

Voici un autre programme qui gère les comptes de Amine et de Khalil. Il permet à l'utilisateur de saisir un montant qui sera débité à Amine et crédité à Khalil.

```
class Program
```

```
{
    static void Main(string[] args)
    {
        CompteBancaire compteAmine = new CompteBancaire();
        compteAmine.titulaire = "Amine";
        compteAmine.solde = 500;
        compteAmine.devise = "DH";

        CompteBancaire compteKhalil = new CompteBancaire();
        compteKhalil.titulaire = "Khalil";
        compteKhalil.solde = 150;
        compteKhalil.devise = "DH";

        Console.Write("Entrez le montant du transfert : ");
        double montantTransfert = Convert.ToDouble(Console.ReadLine());
        compteAmine.Debiter(montantTransfert);
        compteKhalil.Crediter(montantTransfert);

        Console.WriteLine(compteAmine.Decrire());
        Console.WriteLine(compteKhalil.Decrire());
        Console.ReadKey();


    }
}
```

1.6 La relation d'héritage

L'héritage est l'un des mécanismes fondamentaux de la POO. Il permet de créer des classes à partir de classes existantes.

Exemple d'utilisation

```
// Définit un compte bancaire
public class CompteBancaire
{
    private string titulaire; // Titulaire du compte
    private double solde;     // Solde du compte
    private string devise;    // Devise du compte

    public string Titulaire
    {
        get { return titulaire; }
    }

    public double Solde
    {
        get { return solde; }
    }
```

```
    public string Devise
    {
        get { return devise; }
    }

    // Constructeur
    public CompteBancaire(string leTitulaire, double soldeInitial, string laDevise)
    {
        titulaire = leTitulaire;
        solde = soldeInitial;
        devise = laDevise;
    }

    // Ajoute un montant au compte
    public void Crediter(double montant)
    {
        solde = solde + montant;
    }

    // Retire un montant au compte
    public void Debiter(double montant)
    {
        solde = solde - montant;
    }

    // Renvoie la description du compte
    public string Decrire()
    {
        string description = "Le solde du compte de " + titulaire + " est de " + solde + " " + devise;
        return description;
    }
}
```

Supposons maintenant que nous ayons à gérer un nouveau type de compte : le compte épargne. Comme un compte classique, un compte épargne possède un titulaire, un solde et une devise. Sa spécificité est qu'il permet d'appliquer des intérêts à l'argent déposé sur le compte.

Bien sûr, il serait possible de concevoir une classe CompteEpargne totalement distincte de la classe CompteBancaire. Cependant, on constate qu'un compte épargne possède toutes les caractéristiques d'un compte bancaire plus des caractéristiques spécifiques. Nous allons donc définir un compte épargne par **héritage** de la définition d'un compte bancaire.

```
public class CompteEpargne : CompteBancaire
{
    private double tauxInteret;

    public CompteEpargne(string leTitulaire, double soldeInitial, string laDevise, double leTauxInteret)
        : base(leTitulaire, soldeInitial, laDevise)
        // appel du constructeur de la classe CompteBancaire
```

```
    // le mot-clé "base" permet d'accéder à la classe parente
  {
    tauxInteret = leTauxInteret;
  }

  // Calcule et ajoute les intérêts au solde du compte
  public void AjouterInterets()
  {
    // ... (détaillé plus bas)
  }
}
```

Dans la déclaration class CompteEpargne : CompteBancaire, les deux-points spécifient que la classe CompteEpargne hérite de la classe CompteBancaire.

REMARQUE : d'autres langages comme Java ou PHP utilisent le mot-clé extends plutôt que le symbole : pour indiquer une relation d'héritage entre deux classes.

On observe que le constructeur de CompteEpargne fait appel au constructeur de CompteBancaire pour en initialiser les attributs. Le mot-clé base désigne la classe parente. Le constructeur de CompteEpargne initialise également l'attribut tauxInteret qui lui est propre.

1.6.1 Définition

DEFINITION : l'héritage est un mécanisme objet qui consiste à définir une classe à partir d'une classe existante. Une classe héritant d'une autre classe possède les caractéristiques de la classe initiale et peut définir ses propres éléments.

La nouvelle classe (ou classe dérivée) correspond à une spécialisation de la classe de base (appelée classe parente ou superclasse). On dit que l'héritage crée une relation de type est un entre les classes. Dans notre exemple, un compte épargne *est un* type particulier de compte bancaire.

NB : le constructeur d'une classe dérivée doit obligatoirement faire explicitement appel au constructeur de la classe parente lorsque celui-ci prend des paramètres. C'est le cas dans notre exemple.

1.6.2 Avantages

Grâce à la relation d'héritage, un objet de la classe `CompteEpargne` peut utiliser les fonctionnalités de la classe `CompteBancaire` sans avoir à les redéfinir. On peut donc débiter ou créditer un compte épargne exactement comme un compte bancaire.

```
double tauxInteret = 0.05;  // taux d'intérêt : 5%
CompteEpargne compteAhmed = new CompteEpargne("Ahmed", 100, "DH", tauxInteret);

// appel des méthodes de CompteBancaire sur le compte épargne
compteAhmed.Debiter(1000);
compteAhmed.Crediter(1500);
```

```
Console.WriteLine(compteAhmed.Decrire()); // Affiche 600 DH
```

Par contre, le calcul des intérêts (méthode AjouterInterets) ne peut se faire que sur un objet de la classe CompteEpargne. L'héritage est une relation *unidirectionnelle*.

```
// OK : compteAhmed est un compte épargne
compteAhmed.AjouterInterets();
Console.WriteLine(compteAhmed.Decrire());

CompteBancaire compteImane = new CompteBancaire("Imane", 100, "DH");
// Erreur : compteImane est un compte bancaire, pas un compte épargne
compteImane.AjouterInterets();
```

Grâce à l'héritage, il est possible de réutiliser les fonctionnalités d'une classe existante en la spécialisant. Il est également possible de spécialiser une classe dérivée.

On voit bien tous les avantages que l'héritage peut apporter : gain de temps de développement, amélioration de la qualité du code, création de hiérarchies de classes reflétant précisément le domaine d'étude, etc.

1.7 Héritage et encapsulation

Nous avons volontairement laissé de côté un point délicat. La méthode AjouterInterets de la classe CompteEpargne, qui doit ajouter les intérêts au solde, n'est pas encore définie. En voici une première version.

```
public class CompteEpargne : CompteBancaire
{
  // ...

  public void AjouterInterets()
  {
    // calcul des intérêts sur le solde
    double interets = solde * tauxInteret;
    // ajout des intérêts au solde
    solde += interets;
  }
}
```

Cependant, le compilateur nous signale qu'on ne peut pas accéder à solde.

Dans cet exemple, la classe dérivée CompteEpargne tente d'accéder à l'attribut solde qui appartient à la classe de base CompteBancaire. Cependant, cet attribut est défini avec le niveau de visibilité private ! Cela signifie qu'il n'est utilisable que dans la classe où il est défini, et non dans les classes dérivées.

Pour interdire l'accès à un membre d'une classe (attribut, propriété C# ou méthode) depuis l'extérieur tout en permettant son utilisation par une classe dérivée, il faut associer à ce membre un niveau de visibilité intermédiaire : protected.

```
public class CompteBancaire
{
   private string titulaire;
   protected double solde;   // attribut protégé
   private string devise;

   // ...
```

Une autre solution à ce problème consiste à laisser le champ solde privé et à définir un accesseur *protégé* pour modifier le solde depuis les méthodes de la classe CompteEpargne.

```
public class CompteBancaire
{
   private string titulaire;
   private double solde;
   private string devise;

   // ...

   public double Solde
   {
      get { return solde; }          // accesseur public pour la lecture
      protected set { solde = value; }  // mutateur protégé pour la modification
   }
   // public string Solde { get; protected set; } // Equivalent avec une propriété automatique

   // ...
```

Bien entendu, il faut alors utiliser le mutateur `Solde` et non plus l'attribut `solde` pour accéder au solde depuis la classe dérivée.

```
public class CompteEpargne : CompteBancaire
{
   // ...

   public void AjouterInterets()
   {
      // utilisation du mutateur Solde pour accéder au solde du compte
      double interets = Solde * tauxInteret;
      Solde += interets;
   }
}
```

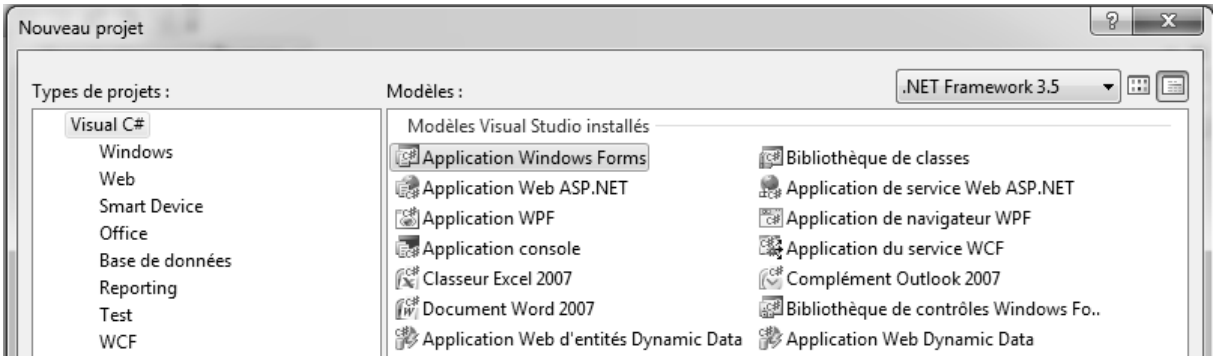
Le tableau ci-dessous rassemble les trois niveaux de visibilité utilisables.

Visibilité	Classe	Classes dérivées	Extérieur
`public`	X	X	X
`protected`	X	X	
`private`	X		

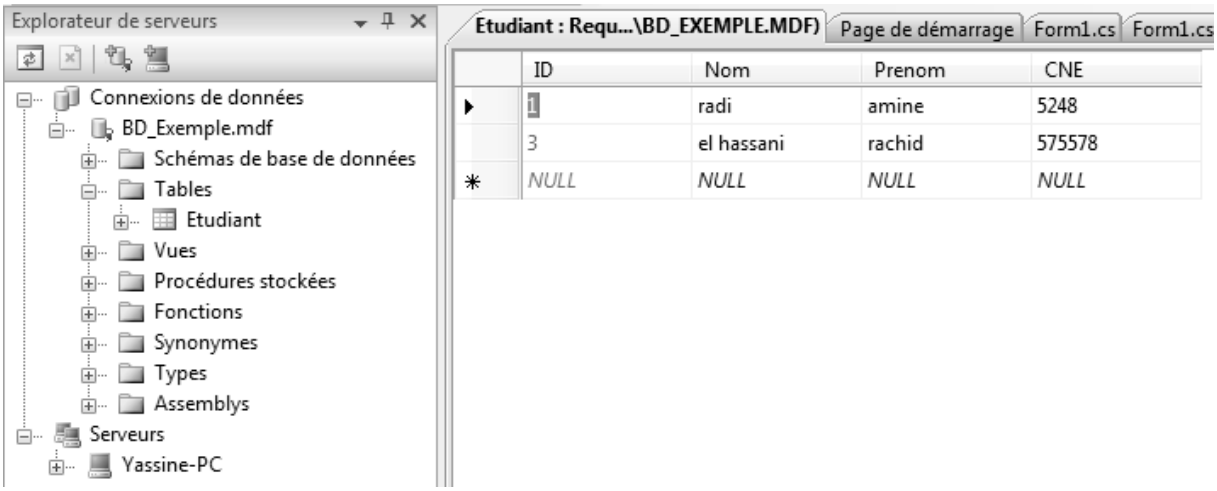
2 Les principes de C# à la réalisation d'applications riches (Windows Forms) et l'accès aux données d'une base

Ce titre montre comment insérer, mettre à jour et supprimer des enregistrements dans un DataGridView dans une application Windows Forms C #.

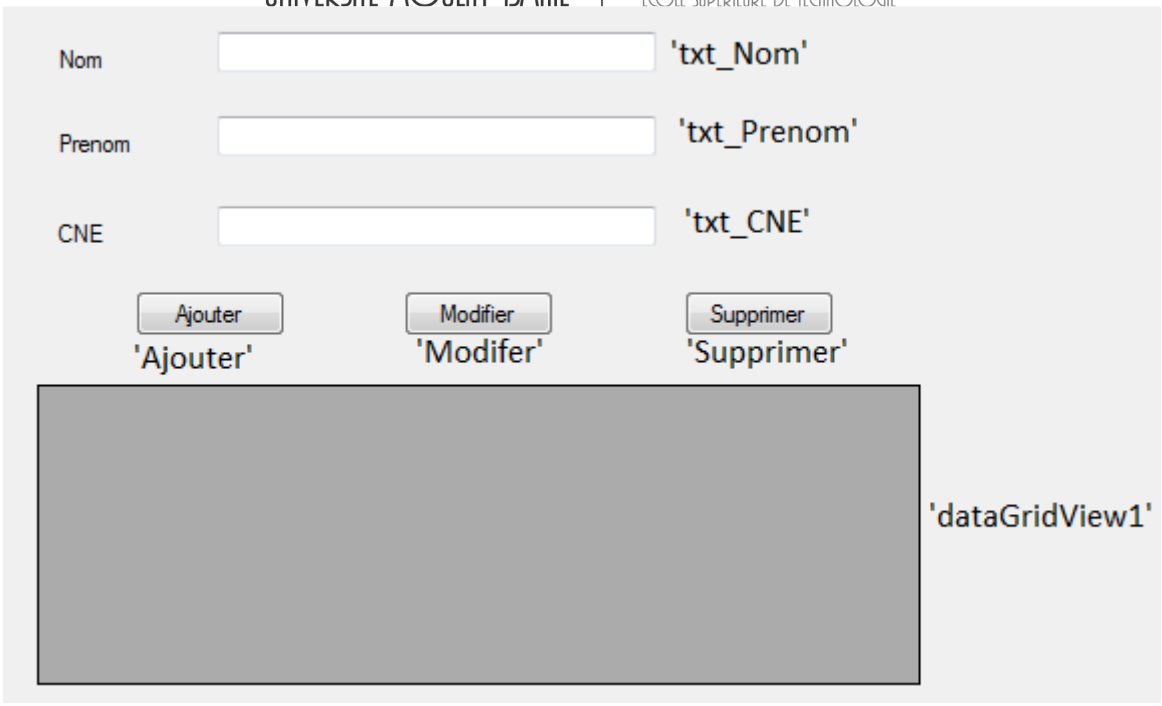
1. Créez une nouvelle application Windows Forms.



2. Créez une base de données (nommée BD_Exemple). Ajouter une table Etudiant. Voici le schéma de la table pour créer la table 'Etudiant'.



3. Créez un formulaire (nommé form1) et ajouter un contrôle Label, TextBox, Button et DataGridView à partir de la boîte à outils.



4. Maintenant, allez dans le code Form1.cs et ajoutez les espaces de noms System.Data et System.Data.SqlClient.

Form1.cs

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.Data.SqlClient;

namespace WindowsFormsApplication1
{
    public partial class Form1 : Form
    {

        SqlConnection con = new SqlConnection("Data
Source=.\\SQLEXPRESS;AttachDbFilename=C:\\Users\\Yassine\\Documents\\Visual
Studio
2008\\Projects\\WindowsFormsApplication1\\WindowsFormsApplication1\\BD_Exem
ple.mdf;Integrated Security=True;User Instance=True");
        SqlCommand cmd;
        SqlDataAdapter adapt;
        // Variable ID utilisée dans la mise à jour et la suppression
d'enregistrement
        int ID = 0;

        public Form1()
        {
            InitializeComponent();
```

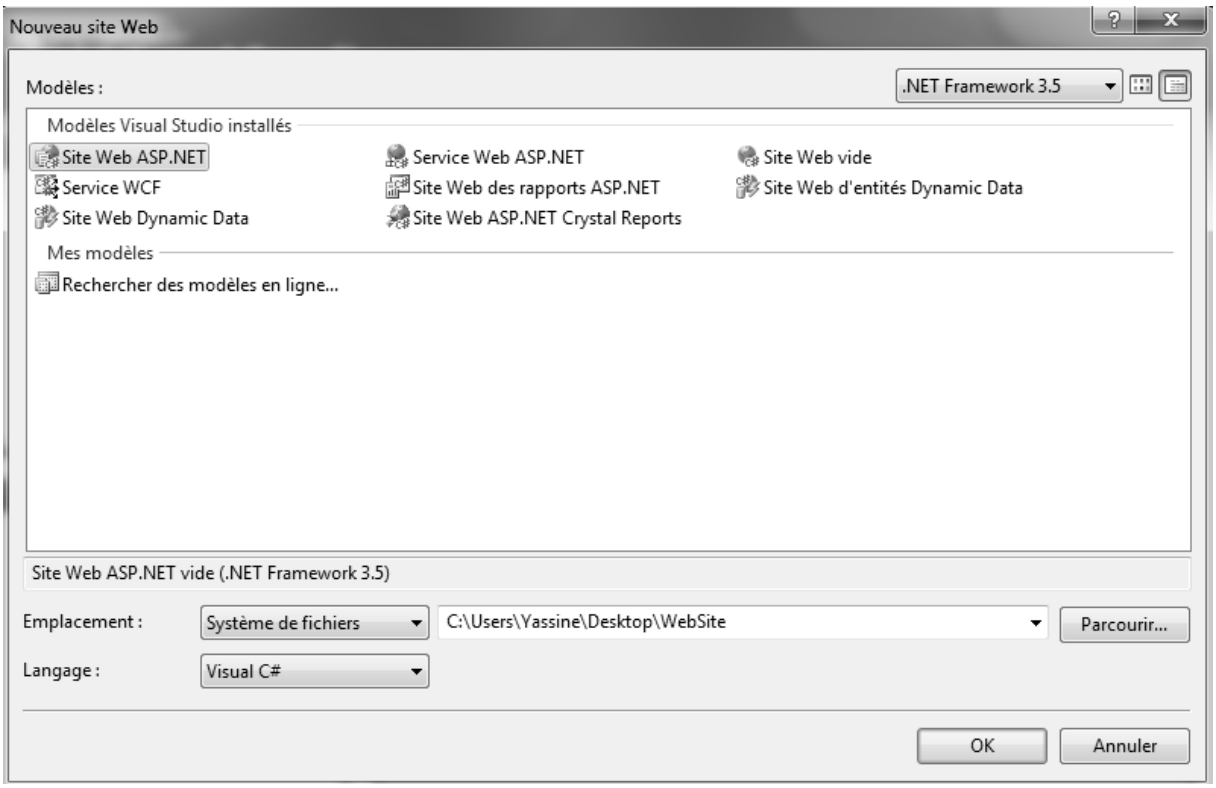
```
            DisplayData();
        }


        // Afficher les données dans DataGridView
        private void DisplayData()
        {
            con.Open();
            DataTable dt=new DataTable();
            adapt=new SqlDataAdapter("select * from Etudiant",con);
            adapt.Fill(dt);
            dataGridView1.DataSource = dt;
            con.Close();
        }
        //Effacer les données
        private void ClearData()
        {
            txt_Nom.Text = "";
            txt_Prenom.Text = "";
            txt_CNE.Text = "";
            ID = 0;
        }

        // événement clique du bouton ajouter
        private void Ajouter_Click(object sender, EventArgs e)
        {
            if (txt_Nom.Text != "" && txt_Prenom.Text != "")
            {
                cmd = new SqlCommand("insert into Etudiant(Nom,Prenom,CNE)
values(@nom,@prenom,@cne)", con);
                con.Open();
                cmd.Parameters.AddWithValue("@nom", txt_Nom.Text);
                cmd.Parameters.AddWithValue("@prenom", txt_Prenom.Text);
                cmd.Parameters.AddWithValue("@cne", txt_CNE.Text);
                cmd.ExecuteNonQuery();
                con.Close();
                MessageBox.Show("Enregistrement inséré avec succès");
                DisplayData();
                ClearData();
            }
            else
            {
                MessageBox.Show("SVP fournir plus des détails!");
            }
        }
        // événement clique du bouton modifier
        private void Modifier_Click(object sender, EventArgs e)
        {
            if (txt_Nom.Text != "" && txt_Prenom.Text != "")
            {
                cmd = new SqlCommand("update Etudiant set
Nom=@nom,Prenom=@prenom,CNE=@CNE where ID=@id", con);
                con.Open();
                cmd.Parameters.AddWithValue("@id", ID);
                cmd.Parameters.AddWithValue("@nom", txt_Nom.Text);
                cmd.Parameters.AddWithValue("@prenom", txt_Prenom.Text);
                cmd.Parameters.AddWithValue("@cne", txt_CNE.Text);
                cmd.ExecuteNonQuery();
                MessageBox.Show("Enregistrement mis à jour avec succès");
```

```
                con.Close();
                DisplayData();
                ClearData();
            }
            else
            {
                MessageBox.Show("Veuillez sélectionner Enregistrer pour
mettre à jour");
            }
        }
        // événement clique du bouton supprimer
        private void Supprimer_Click(object sender, EventArgs e)
        {
            if (ID != 0)
            {
                cmd = new SqlCommand("delete Etudiant where ID=@id", con);
                con.Open();
                cmd.Parameters.AddWithValue("@id", ID);
                cmd.ExecuteNonQuery();
                con.Close();
                MessageBox.Show("Enregistrement supprimé avec succès!");
                DisplayData();
                ClearData();
            }
            else
            {
                MessageBox.Show("Veuillez sélectionner Enregistrer pour
supprimer");
            }
        }
        // événement clique sur une ligne de dataGridView
        private void dataGridView1_CellContentClick(object sender,
DataGridViewCellEventArgs e)
        {
            ID =
Convert.ToInt32(dataGridView1.Rows[e.RowIndex].Cells[0].Value.ToString());
            txt_Nom.Text =
dataGridView1.Rows[e.RowIndex].Cells[1].Value.ToString();
            txt_Prenom.Text =
dataGridView1.Rows[e.RowIndex].Cells[2].Value.ToString();
            txt_CNE.Text =
dataGridView1.Rows[e.RowIndex].Cells[3].Value.ToString();

        }




    }
}
```

Chapitre 2 : Développement d'une application Web en ASP

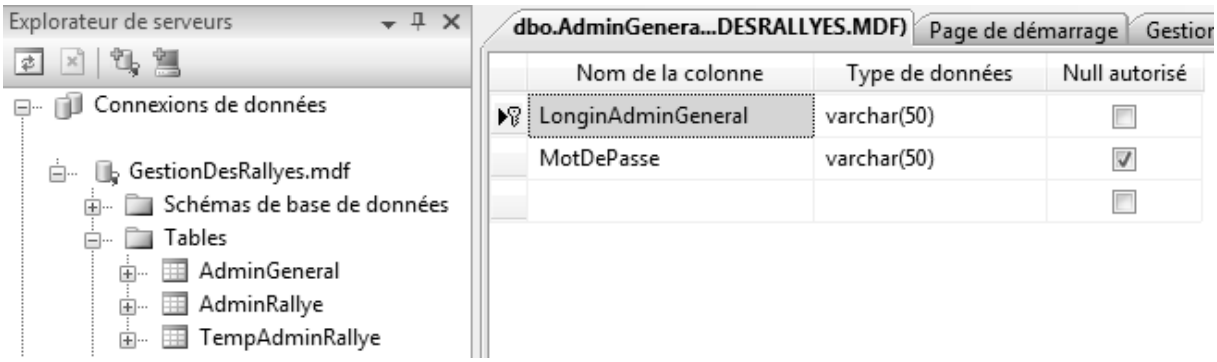
Ce chapitre montre comment développer une application web en ASP à l'aide du langage C #.

1. Créez un nouveau site web ASP.NET.

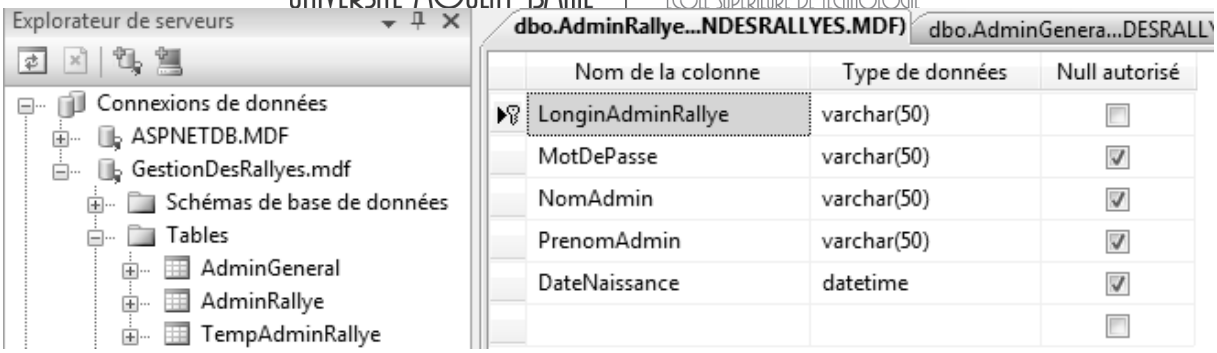


2. Créez une base de données (nommée : GestionDesRallyes). Ajouter trois tables.

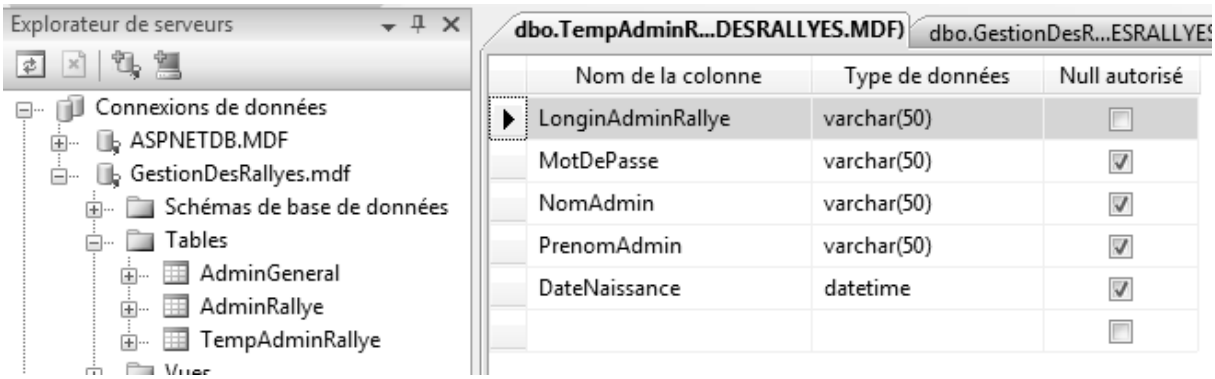
Voici le schéma de la table 'AdminGeneral'.



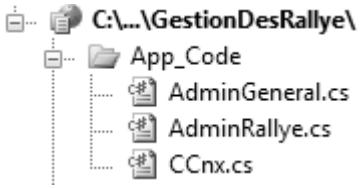
Voici le schéma de la table 'AdminRallye'.



Voici le schéma de la table 'TempAdminRallye'.



3. Maintenant, on va se déplacer dans l'explorateur de solution pour ajouter la classe de connexion et les classes correspondantes aux tables comme suit :



Ainsi le Contenu de la classe connexion se présente comme suit :

CCnx.cs

```
using System;
using System.Data;
using System.Configuration;
using System.Web;
using System.Web.Security;
using System.Web.UI;
using System.Web.UI.HtmlControls;
using System.Web.UI.WebControls;
using System.Web.UI.WebControls.WebParts;
using System.Data.SqlClient;
/// <summary>
/// Description résumée de CCnx
/// </summary>
public class CCnx
```

```
{
        public SqlDataAdapter dad;
        public DataTable Tab;
        public DataSet Dst;
        public SqlConnection Con;
        public SqlCommand Com;
        String S;

        public CCnx()
        { }

        public void Open()
        {
            S = "Data
Source=.\\SQLEXPRESS;AttachDbFilename=E:\\projet\\GestionDesRallye\\App_Dat
a\\GestionDesRallyes.mdf;Integrated Security=True;User Instance=True";
            Con = new SqlConnection(S);
            Con.Open();
        }

        public void Consultation(String Ch)
        {
            SqlConnection Con = new SqlConnection(S);
            SqlCommand Com = new SqlCommand();
            Com.Connection = Con;
            Com.CommandType = CommandType.Text;
            Com.CommandText = Ch;
            dad = new SqlDataAdapter(Com);
            Dst = new DataSet();
            dad.Fill(Dst, Com.CommandText);
            Tab = Dst.Tables[Com.CommandText];
        }
      }
```

La classe 'AdminGeneral' se présente comme suit :

AdminGeneral.cs

```
using System;
using System.Data;
using System.Configuration;
using System.Web;
using System.Web.Security;
using System.Web.UI;
using System.Web.UI.HtmlControls;
using System.Web.UI.WebControls;
using System.Web.UI.WebControls.WebParts;

/// <summary>
/// Description résumée de AdminGeneral
/// </summary>
public class AdminGeneral
{
    private string LoginAdminGeneral;
    private string MotDePasse;
    public CCnx OCnx;

    public AdminGeneral()
```

```
    { }

    public void SetLoginAdminGeneral(string LoginAdminGeneral)
    {
        this.LoginAdminGeneral = LoginAdminGeneral;
    }

    public string GetLoginAdminGeneral()
    {
        return this.LoginAdminGeneral;
    }

    public void SetMotDePasse(string MotDePasse)
    {
        this.MotDePasse = MotDePasse;
    }

    public string GetMotDePasse()
    {
        return this.MotDePasse;
    }


    string ss;
    DataRow row;

    public Boolean UnicId()
    {
        OCnx = new CCnx();
        OCnx.Open();
        OCnx.Consultation("select count(*) as p from AdminGeneral where
LonginAdminGeneral = '" + this.LoginAdminGeneral + "' and MotDePasse = '" +
this.MotDePasse + "'");
        row = OCnx.Tab.Rows[0];
        ss = row[0].ToString();
        if (ss == "0")
        {
            return false;

        }
        return true;
    }

}
```

La classe ‘AdminRallye’ se présente comme suit :

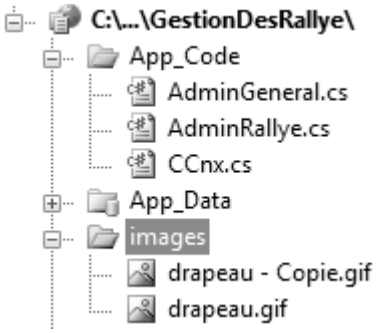
AdminGeneral.cs

```
using System;
using System.Data;
using System.Configuration;
using System.Web;
using System.Web.Security;
using System.Web.UI;
using System.Web.UI.HtmlControls;
using System.Web.UI.WebControls;
using System.Web.UI.WebControls.WebParts;
```

```
/// <summary>
/// Description résumée de AdminRallye
/// </summary>
public class AdminRallye
{
        private string LonginAdminRallye;
        private string MotDePasse;
        private string NomAdmin;
        private string PrenomAdmin;
        private DateTime DateNaissance;
        public CCnx OCnx;

        public AdminRallye()
        { }

        public void SetLonginAdminRallye(string LonginAdminRallye)
        {
            this.LonginAdminRallye = LonginAdminRallye;
        }

        public string GetLonginAdminRallye()
        {
            return this.LonginAdminRallye;
        }

        public void SetMotDePasse(string MotDePasse)
        {
            this.MotDePasse = MotDePasse;
        }

        public string GetMotDePasse()
        {
            return this.MotDePasse;
        }

        public void SetNomAdmin(string NomAdmin)
        {
            this.NomAdmin = NomAdmin;
        }

        public string GetNomAdmin()
        {
            return this.NomAdmin;
        }
        public void SetPrenomAdmin(string PrenomAdmin)
        {
            this.PrenomAdmin = PrenomAdmin;
        }

        public string GetPrenomAdmin()
        {
            return this.PrenomAdmin;
        }

        public void SetDateNaissance(DateTime DateNaissance)
        {
            this.DateNaissance = DateNaissance;
        }
```

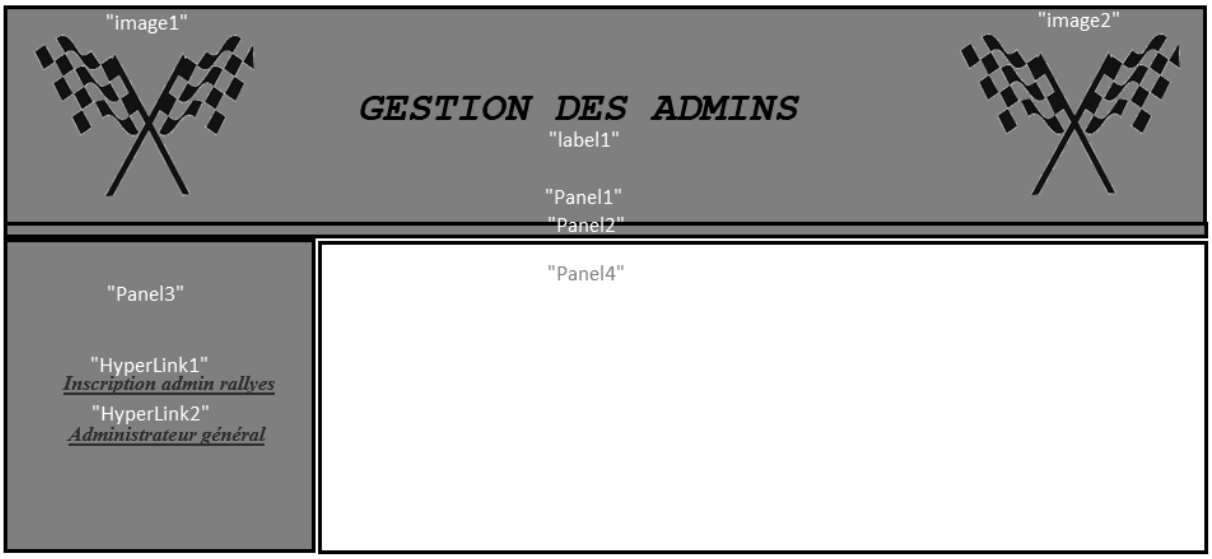
```
        public DateTime GetDateNaissance()
        {
            return this.DateNaissance;
        }

        public void ajouter()
        {
            CCnx OCnx = new CCnx();
            OCnx.Open();
            OCnx.Consultation("insert into AdminRallye values('" +
this.LonginAdminRallye + "','" + this.MotDePasse + "','" + this.NomAdmin +
"','" + this.PrenomAdmin + "','" + this.DateNaissance + "')");
        }

        public void ajouter1()
        {
            CCnx OCnX = new CCnx();
            OCnx.Open();
            OCnx.Consultation("insert into TempAdminRallye values('" +
this.LonginAdminRallye + "','" + this.MotDePasse + "','" + this.NomAdmin +
"','" + this.PrenomAdmin + "','" + this.DateNaissance + "')");
        }



        public void supprimer1()
        {
            CCnx OCnx = new CCnx();
            OCnx.Open();
            OCnx.Consultation("delete from TempAdminRallye where
LonginAdminRallye = '" + this.LonginAdminRallye + "'");
        }




        string ss;
        DataRow row;

        public Boolean UnicId()
        {
            OCnx = new CCnx();
            OCnx.Open();
            OCnx.Consultation("select count(*) as p from AdminRallye where
LonginAdminRallye = '" + this.LonginAdminRallye + "' and MotDePasse = '" +
this.MotDePasse + "'");
            row = OCnx.Tab.Rows[0];
            ss = row[0].ToString();
            if (ss == "0")
            {
                return false;

            }
            return true;
        }

}
```

4. Toujours dans l'explorateur de solution ajouter un dossier images dans lequel insérer deux photos qui seront utilisées dans la présentation de l'application web comme suit :

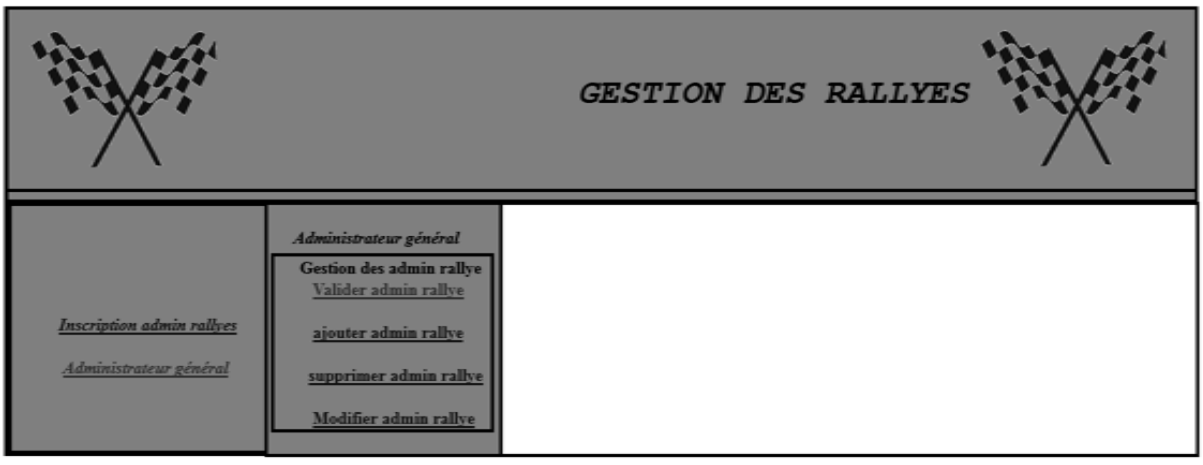


5. Maintenant Créer les pages master :

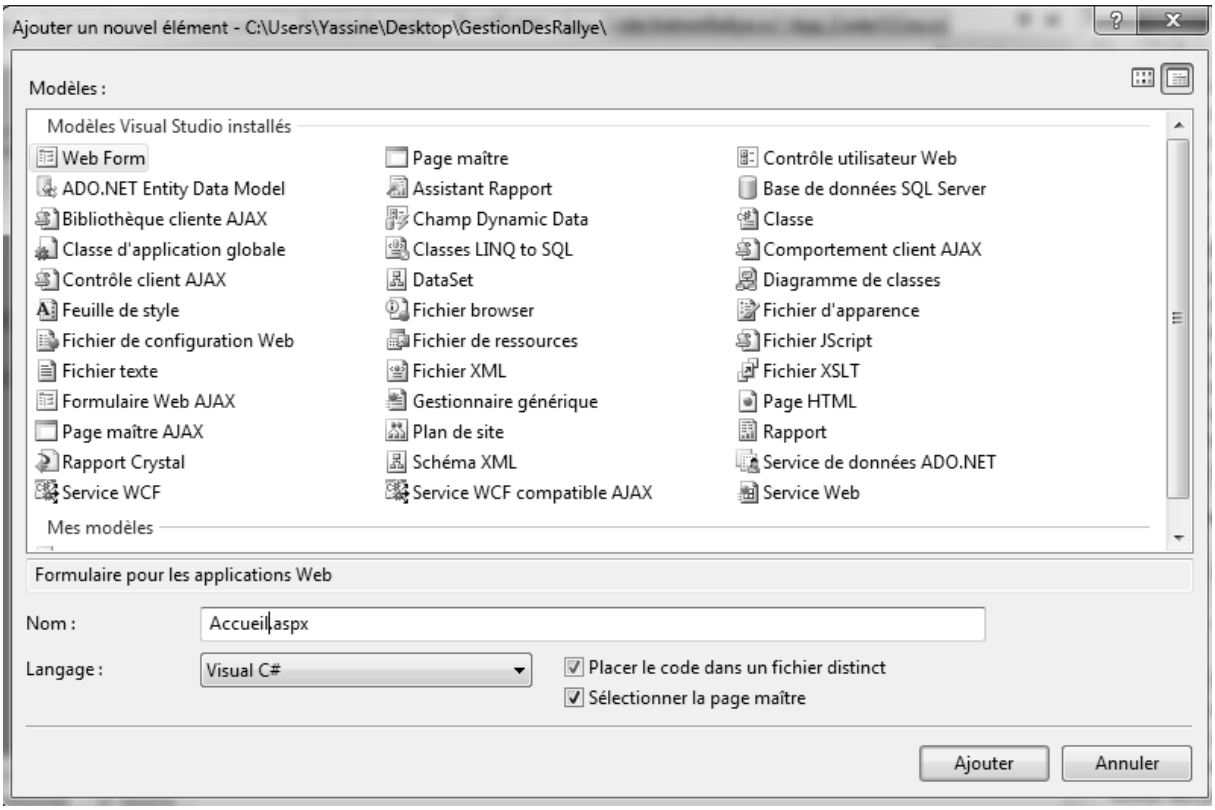
" Principale.master"



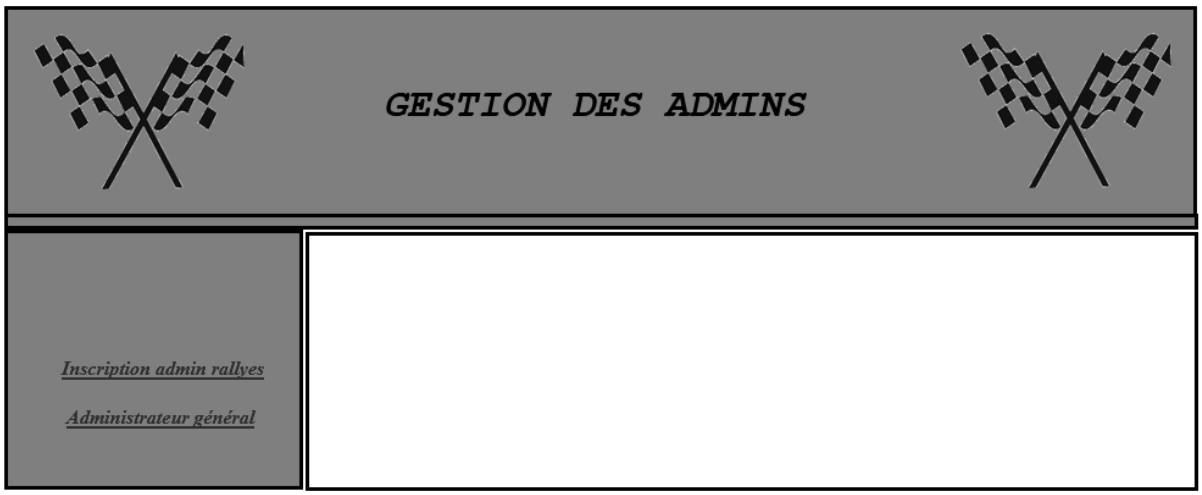
" GestionAdminGeneral.master"



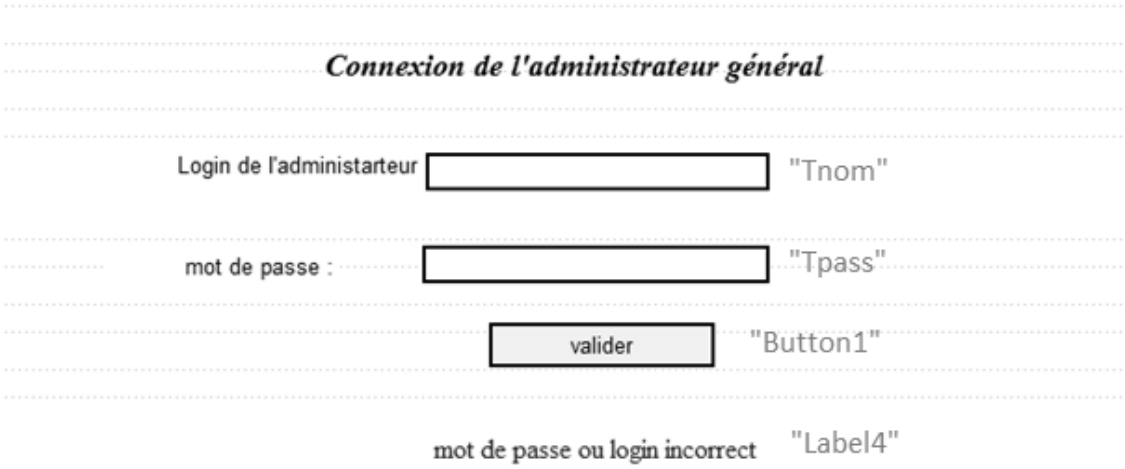
6. Ensuite dans l'explorateur de solution on va ajouter les" webform" en cochant "sélectionner page maitre":



Accueil.aspx



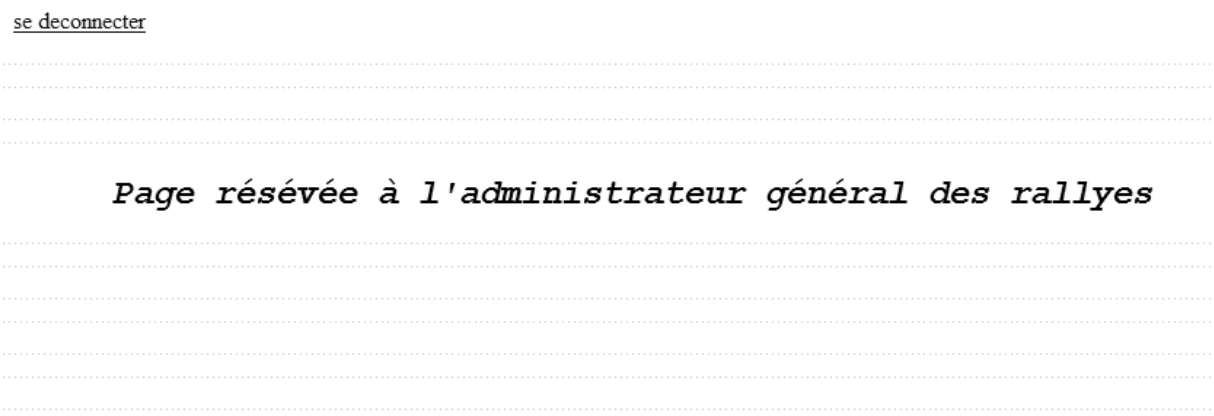
La page "AdminGeneral.aspx" qui hérite de "Principale.master"



La page "AdminGeneral.asp.cs"

```
using System;
using System.Collections;
using System.Configuration;
using System.Data;
using System.Web;
using System.Web.Security;
using System.Web.UI;
using System.Web.UI.HtmlControls;
using System.Web.UI.WebControls;
using System.Web.UI.WebControls.WebParts;

public partial class _Default : System.Web.UI.Page
{
    protected void Page_Load(object sender, EventArgs e)
    {
        Tnom.Focus();
    }
    protected void Button1_Click(object sender, EventArgs e)
    {
        AdminGeneral or = new AdminGeneral();
        or.SetLoginAdminGeneral(Tnom.Text); or.SetMotDePasse(Tpass.Text);
        if (or.UnicId())
        {
            Session["AdminGeneral"] = Tnom.Text;
            Response.Redirect("AdminGeneralPageAcueil.aspx");

        }
        else
        {
            Tnom.Text = "";
            Tpass.Text = "";
            Tnom.Focus();
            Label4.Visible = true;
        }

    }
```

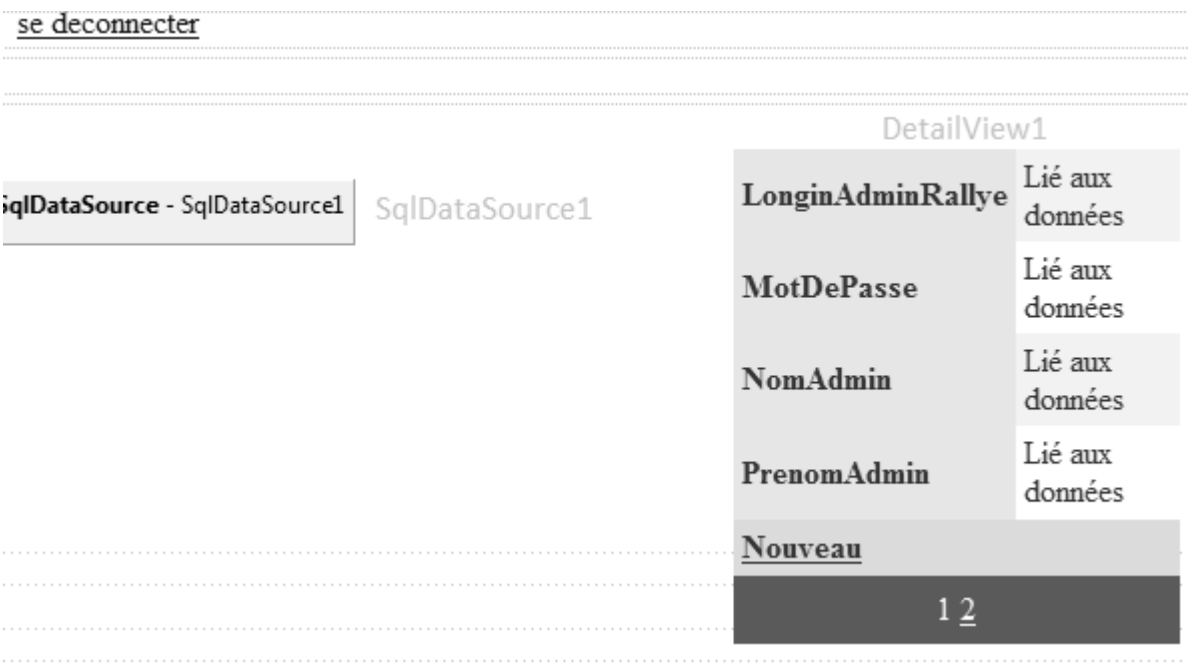
La page "AdminGeneralPageAccueil.aspx" qui hérite de " GestionAdminGeneral.master"



La page " AdminGeneralPageAccueil.asp.cs"

```
using System;

using System.Collections;

using System.Configuration;

using System.Data;

using System.Web;

using System.Web.Security;

using System.Web.UI;

using System.Web.UI.HtmlControls;

using System.Web.UI.WebControls;

using System.Web.UI.WebControls.WebParts;

public partial class _Default : System.Web.UI.Page
{
    protected void Page_Load(object sender, EventArgs e)
    {

        if ((String)Session["AdminGeneral"] == null)
        {
            Response.Redirect("AdminGeneral.aspx");
        }

    }

    protected void LinkButton3_Click(object sender, EventArgs e)
    {
        Session.Remove("AdminGeneral");
        Response.Redirect("AdminGeneral.aspx");
    }

}
```

La page " AjouterAdminRallye.aspx" qui hérite de " GestionAdminGeneral.master"



La page " AjouterAdminRallye.asp.cs"

```
using System;
using System.Collections;
using System.Configuration;
using System.Data;
using System.Web;
using System.Web.Security;
using System.Web.UI;
using System.Web.UI.HtmlControls;
using System.Web.UI.WebControls;
using System.Web.UI.WebControls.WebParts;

public partial class _Default : System.Web.UI.Page
{
    protected void Page_Load(object sender, EventArgs e)
    {
        if ((String)Session["AdminGeneral"] == null)
        {
            Response.Redirect("AdminGeneral.aspx");
        }
    }
    protected void LinkButton3_Click(object sender, EventArgs e)
    {
        Session.Remove("AdminGeneral");
        Response.Redirect("AdminGeneral.aspx");
    }
}
```

La page " ExclureAdminRallye.aspx" qui hérite de " GestionAdminGeneral.master"

La page " AjouterAdminRallye.asp.cs"

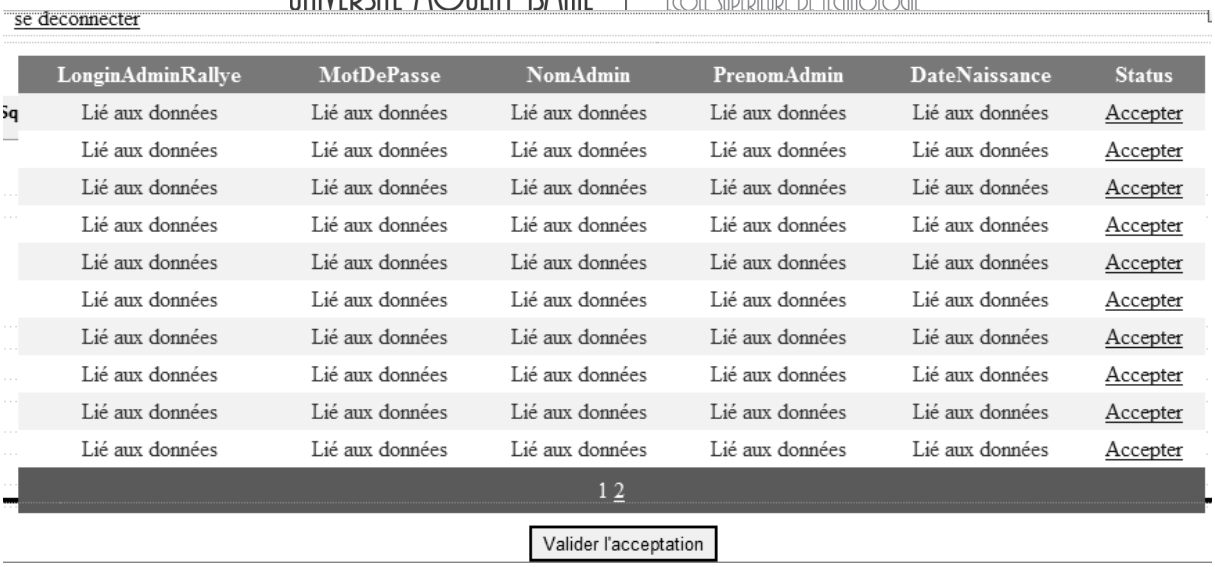
```
using System;
using System.Collections;
using System.Configuration;
using System.Data;
using System.Web;
using System.Web.Security;
using System.Web.UI;
using System.Web.UI.HtmlControls;
using System.Web.UI.WebControls;
using System.Web.UI.WebControls.WebParts;

public partial class _Default : System.Web.UI.Page
{
    protected void Page_Load(object sender, EventArgs e)
    {
        if ((String)Session["AdminGeneral"] == null)
        {
            Response.Redirect("AdminGeneral.aspx");
        }
    }
    protected void GridView1_SelectedIndexChanged1(object sender, EventArgs
e)
    {

    }

    protected void LinkButton3_Click(object sender, EventArgs e)
    {
        Session.Remove("AdminGeneral");
        Response.Redirect("AdminGeneral.aspx");
    }

}
```

La page "ModifierAdminRallye.aspx" qui hérite de " GestionAdminGeneral.master"

La page " ModifierAdminRallye.asp.cs"

```
using System;
using System.Collections;
using System.Configuration;
using System.Data;
using System.Web;
using System.Web.Security;
using System.Web.UI;
using System.Web.UI.HtmlControls;
using System.Web.UI.WebControls;
using System.Web.UI.WebControls.WebParts;

public partial class _Default : System.Web.UI.Page
{
    protected void Page_Load(object sender, EventArgs e)
    {
        if ((String)Session["AdminGeneral"] == null)
        {
            Response.Redirect("AdminGeneral.aspx");
        }
    }
    protected void GridView1_SelectedIndexChanged1(object sender, EventArgs
e)
    {

    }

    protected void LinkButton3_Click(object sender, EventArgs e)
    {
        Session.Remove("AdminGeneral");
        Response.Redirect("AdminGeneral.aspx");
    }

}
```

La page " ValiderAdminRallye.aspx" qui hérite de " GestionAdminGeneral.master"

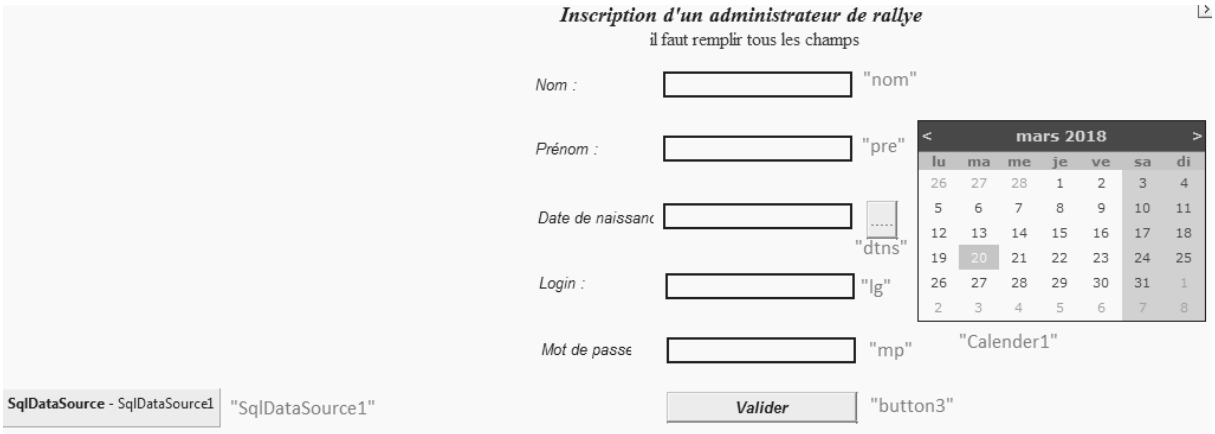


La page " ModifierAdminRallye.asp.cs"

```
using System;
using System.Collections;
using System.Configuration;
using System.Data;
using System.Web;
using System.Web.Security;
using System.Web.UI;
using System.Web.UI.HtmlControls;
using System.Web.UI.WebControls;
using System.Web.UI.WebControls.WebParts;

public partial class _Default : System.Web.UI.Page
{
    protected void Page_Load(object sender, EventArgs e)
    {
        if ((String)Session["AdminGeneral"] == null)
        {
            Response.Redirect("AdminGeneral.aspx");
        }
    }

    protected void GridView1_RowCommand(object sender,
GridViewCommandEventArgs e)
    {

        Int32 index = Convert.ToInt32(e.CommandArgument);
        AdminRallye Oadmr = new AdminRallye();
        Oadmr.SetLonginAdminRallye(GridView1.Rows[index].Cells[0].Text);
        Oadmr.SetMotDePasse(GridView1.Rows[index].Cells[1].Text);
        Oadmr.SetPrenomAdmin(GridView1.Rows[index].Cells[2].Text);
        Oadmr.SetNomAdmin(GridView1.Rows[index].Cells[3].Text);

Oadmr.SetDateNaissance(Convert.ToDateTime(GridView1.Rows[index].Cells[4].Te
xt));

        Oadmr.ajouter();
        Oadmr.supprimer1();
```

```
    }
    protected void Button1_Click(object sender, EventArgs e)
    {
        Response.Redirect("ValiderAdminRallye.aspx");
    }
    protected void GridView1_SelectedIndexChanged1(object sender, EventArgs
e)
    {

    }
    protected void LinkButton3_Click(object sender, EventArgs e)
    {
        Session.Remove("AdminGeneral");
        Response.Redirect("AdminGeneral.aspx");
    }
}
```

La page " InscriptionAdminRallye.aspx" qui hérite de " Principale.master"



La page " InscriptionAdminRallye.asp.cs"

```
using System;
using System.Collections;
using System.Configuration;
using System.Data;
using System.Web;
using System.Web.Security;
using System.Web.UI;
using System.Web.UI.HtmlControls;
using System.Web.UI.WebControls;
using System.Web.UI.WebControls.WebParts;

public partial class _Default : System.Web.UI.Page
{
    protected void Page_Load(object sender, EventArgs e)
    {
        nom.Focus();
    }



    protected void Calendar1_SelectionChanged(object sender, EventArgs e)
```

```
    {

        dtns.Text = Calendar1.SelectedDate.ToShortDateString();

        Calendar1.Visible = false;

    }

    protected void Button3_Click(object sender, EventArgs e)
    {
        if (lg.Text == "") { Label4.Visible = true; }
        else
        {
            SqlDataSource1.Insert();
            nom.Text = "";
            pre.Text = "";
            dtns.Text = "";
            mp.Text = "";
            lg.Text = "";
            nom.Focus();
        }
    }
    protected void pre_TextChanged(object sender, EventArgs e)
    {

    }
    protected void Button4_Click(object sender, EventArgs e)
    {
        Calendar1.Visible = !Calendar1.Visible;

        if (Calendar1.Visible & !String.IsNullOrEmpty(dtns.Text))
            Calendar1.VisibleDate = Calendar1.SelectedDate =
Convert.ToDateTime(dtns.Text);
    }
}
```