# Rapport technique

## Projet Gestion des étudiants

Encadré par :M.Ghailani Mohamed

Réalise par :

Naila Hssassa

Rihab Bahid

Chaymae Ghazi

Niema El ghazouani

# Bibliothèque de classes

## La classe connexion.cs:

```
using MySql.Data.MySqlClient;
using System;
using System.Collections.Generic;
using System.Data;
using System.Data.SqlClient;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Reflection;

namespace projet
{
    enum dbtype
    {
        mysql, sqlserver
    }
    class Connexion
    {
        static IDbConnection con = null;
        static IDbCommand cmd = null;
        public static IDbConnection Con { get => con; set => con = value; }
        public static IDbCommand Cmd { get => cmd; set => cmd = value; }
        public static void connect(dbtype Dbtype, string dbname, string host, string user,
string password = "", int port = 0)
        {

            if (con == null)
            {
                if (Dbtype == dbtype.mysql)
                {
                    con = new MySqlConnection("user id=" + user + ";password=" +
password + ";server=" + host + ";persistsecurityinfo=True;port=" + port +
";database=" + dbname);
                    cmd = new MySqlCommand();
                    if (con.State.ToString() == "Closed")
                    {
                        con.Open();
                        cmd.Connection = con;
                        Console.WriteLine(" connexion valide");
                    }
                }

                //   case "sqlserver":
                else if (Dbtype == dbtype.sqlserver)
                {
                    con = new SqlConnection(@"Data
Source=localhost\SQLEXPRESS;Initial Catalog=" + dbname + ";User ID=" +
user + ";Password=" + password + ";Integrated Security=True");
                    cmd = new SqlCommand();
                    if (con.State.ToString() == "Closed")
                    {
                        con.Open();
                        cmd.Connection = con;
                        Console.WriteLine("connexion valide");
                    }
                }

                else
                    throw new Exception("connexion invalide");

            }
        }
        public static int IUD(string req)
        {
            cmd.CommandText = req;
            return cmd.ExecuteNonQuery();
        }
        public static IDataReader Select(string req)
        {
            cmd.CommandText = req;
            return (IDataReader)cmd.ExecuteReader();
        }

        public static List<String> getchamps(String table)
        {

            List<String> LE = new List<String>();
            string sql = "SELECT COLUMN_NAME FROM
INFORMATION_SCHEMA.COLUMNS WHERE TABLE_NAME =" + "'" + table + "'";

            IDataReader Dr = Select(sql);
            while (Dr.Read())
                LE.Add(Dr.GetString(0));

            Dr.Close();

            return LE;


        }

    }
}
```

## La classe Model.cs:

```
using MySql.Data.MySqlClient;
using Newtonsoft.Json;
using System;
using System.Collections.Generic;
using System.Data;
using System.Data.SqlClient;
using System.Dynamic;
using System.Linq;
using System.Reflection;
using System.Text;
using System.Threading.Tasks;

namespace projet
{
    public class Model
    {
        public int id = 0;
        public static string sql = "";


        public Dictionary<string, T> ToDictionary<T>(object obj)
        {
            var json = JsonConvert.SerializeObject(obj);
            var dico = JsonConvert.DeserializeObject<Dictionary<string, T>>(json);
            return dico;
        }

        public int idfromcode(string cd)
        {
            Dictionary<string, object> dc = new Dictionary<string, object>();
            dc.Add("code", cd);
            List<dynamic> l= this.Select(dc);
            if (l.Count!=0) {
                Dictionary<string, string> ddc = ToDictionary<string>(l[0]);

                foreach (KeyValuePair<string, string> m in ddc)
                {
                    if (m.Key.Equals("id"))
                        return int.Parse(m.Value);

                } }return 0;
             }
```

```
  public int idnote(string ce,string cm)
     {
        Dictionary<string, object> dc = new Dictionary<string, object>();
        dc.Add("code_eleve", ce);
        dc.Add("code_mat", cm);
        List<dynamic> l = this.Select(dc);
        if (l.Count != 0)
        {
           Dictionary<string, string> ddc = ToDictionary<string>(l[0]);

           foreach (KeyValuePair<string, string> m in ddc)
           {
              if (m.Key.Equals("id"))
                 return int.Parse(m.Value);
           }
        }
        return 0;
     }
     private static dynamic DictionaryToObject(Dictionary<String, object> dico)
     {
        return dico.Aggregate(new ExpandoObject() as IDictionary<string,
Object>, (a, p) => { a.Add(p.Key, p.Value); return a; });
     }

     public Dictionary<string, string> GetChampsndvalue()
     {
        Type T = this.GetType();
        var pack = new Dictionary<string, string>();
        //
        PropertyInfo[] properties = T.GetProperties();
        foreach (PropertyInfo property in properties)
        {

           pack.Add(property.Name, property.GetValue(this).ToString());
        }
        return pack;

     }
```

```
public dynamic find()
{
    Dictionary<string, object> dico = new Dictionary<string, object>();
    Dictionary<string, string> dico2 = new Dictionary<string, string>();
    sql = "select * from " + this.GetType().Name + " where id=" + id;

    IDataReader dataReader = Connexion.Select(sql);
    if (dataReader.Read())
    {
        Console.WriteLine("existe");
        for (int i = 0; i < dataReader.FieldCount; i++)
        {
            dico.Add(dataReader.GetName(i), dataReader.GetValue(i));
        }
        dataReader.Close();
        return DictionaryToObject(dico);

    }
    else
    {
        Console.WriteLine("n'existe pas");
        dataReader.Close();
        return null;
    }
}
```

```
public static dynamic find<T>(int id)
{
Dictionary<string, object> dico = new Dictionary<string, object>();
sql = "select * from " + typeof(T).Name + " where id=" + id;
IDataReader dataReader = Connexion.Select(sql);
if (dataReader.Read()){
Console.WriteLine("existe");
for (int i = 0; i < dataReader.FieldCount; i++)
{dico.Add(dataReader.GetName(i), dataReader.GetValue(i));}
dataReader.Close();
return DictionaryToObject(dico);
} else{
Console.WriteLine("n'existe pas");
dataReader.Close();
return null;
}}
public int save(string storedProcedureName = "")
{ Dictionary<string, string> dico = this.GetChampsndvalue();
var instance = this.find();
try{
//Utilisation d'une procedure stockee
IDbCommand cmd;
if (storedProcedureName == "") throw new Exception();
if (Connexion.Con.GetType().Name == "MySqlConnection")
cmd = new MySqlCommand(storedProcedureName,
(MySqlConnection)Connexion.Con);
else
cmd = new SqlCommand(storedProcedureName, (SqlConnection)Connexion.Con);
cmd.CommandType = CommandType.StoredProcedure;
int i = 1;
foreach (var value in dico.Values)
{var p = cmd.CreateParameter(); p.ParameterName = "p" + i;
p.Value = value;
cmd.Parameters.Add(p);
i++;
}
cmd.ExecuteNonQuery();
Console.WriteLine("stocked procedure used");
return 1;
}
catch
{
  if (!(instance is null))
  { //cas de modification
    this.id = instance.id;
    sql = "update  " + this.GetType().Name + " set ";
   foreach (var temp in dico)
    {
      if (temp.Key != nameof(id)&&(!(temp.Key.Equals("code_fil"))))
      {
        //convertir un string sous forme d'un nombre en int
        //cas d'un varchar
        if (!int.TryParse(temp.Value, out _))
          sql += temp.Key + "='" + temp.Value + "' ,";
        else
          sql += temp.Key + "=" + temp.Value + " ,";
       }}
    sql = sql.Remove(sql.Length - 1);
    sql += " where id =" + this.id; }
else
  {
    //cas d'insertion
    sql = "insert into " + this.GetType().Name + " ( ";
    foreach (var key in dico.Keys)
    {
      sql += key + ",";
    }
    sql = sql.Remove(sql.Length - 1);
    sql += ") values( ";
    foreach (var value in dico.Values)
    {
      if (value is String)
        sql += "'" + value + "'" + ",";
      else
        sql += value + ",";
    }
    sql = sql.Remove(sql.Length - 1);
    sql += ")";
  }
  Console.WriteLine("sql command used");
  return Connexion.IUD(sql);
}  }
```

```
public int delete(string storedProcedureName = "")
      {
         try
         {
            Dictionary<string, string> dico = this.GetChampsndvalue();
            //int id =0;
            if (this.GetType().Name == "eleve") id = idfromcode(dico["Code"]);
            else if (this.GetType().Name == "note") id = idnote(dico["Code_eleve"],
dico["Code_mat"]);
            //Utilisation d'une procedure stockee

            IDbCommand cmd;
            if (storedProcedureName == "") throw new Exception();

            if (Connexion.Con.GetType().Name == "MySqlConnection")
               cmd = new MySqlCommand(storedProcedureName,
(MySqlConnection)Connexion.Con);
            else
               cmd = new SqlCommand(storedProcedureName,
(SqlConnection)Connexion.Con);
            cmd.CommandType = CommandType.StoredProcedure;
            var p = cmd.CreateParameter();
            p.ParameterName = "p1";
            p.Value = id;
            cmd.Parameters.Add(p);
            cmd.ExecuteNonQuery();
            Console.WriteLine("stocked procedure used");
            return 1;

         }
         catch
         {
            var instance = this.find();
            // this.id = instance.id;
            if (!(instance is null))
               sql = "delete from " + this.GetType().Name + " where id= " + id;
            return Connexion.IUD(sql);
         }
      }
```

```
//critere de selection
public static List<dynamic> Select<T>(Dictionary<string, object> dico)
{
   //construction de la req
   sql = "select * from " + typeof(T).Name + " where ";
   foreach (KeyValuePair<string, object> kvp in dico)
   {
      if (kvp.Value is string)
      {
         sql += kvp.Key + " = " + "'" + kvp.Value + "'" + " AND ";
      }
      else
         sql += kvp.Key + " = " + kvp.Value + " AND ";
   }
   sql = sql.Remove(sql.Length - 4);
   //
   IDataReader dataReader = Connexion.Select(sql);

   List<object> list = new List<object>();

   while (dataReader.Read())
   {
      Dictionary<string, object> buff = new Dictionary<string, object>();
      for (int i = 0; i < dataReader.FieldCount; i++)
      {
         buff.Add(dataReader.GetName(i), dataReader.GetValue(i));
      }
      list.Add(DictionaryToObject(buff));

   }
   dataReader.Close();
   return (List<dynamic>)list;
}
```

```
public List<dynamic> Select(Dictionary<string, object> dico)
    {
        //construction de la req

        sql = "select * from " + this.GetType().Name + " where ";
       // if (dico["code"] == null)
       // {
            foreach (KeyValuePair<string, object> kvp in dico)
            {
                if (kvp.Value is string)
                {
                    sql += kvp.Key + " = " + "'" + kvp.Value + "'" + " AND ";
                }
                else
                    sql += kvp.Key + " = " + kvp.Value + " AND ";
            }

            sql = sql.Remove(sql.Length - 4);
       // }
       // else { sql += " code = '" + dico["code"]+"'"; }
        //
        IDataReader dataReader = Connexion.Select(sql);

        List<object> list = new List<object>();

        while (dataReader.Read())
        {
            Dictionary<string, object> buff = new Dictionary<string, object>();
            for (int i = 0; i < dataReader.FieldCount; i++)
            {
                buff.Add(dataReader.GetName(i), dataReader.GetValue(i));
            }
            list.Add(DictionaryToObject(buff));

        }
        dataReader.Close();
        return (List<dynamic>)list;
    }
    public List<dynamic> All()
    {
      List<dynamic> Liste = new List<dynamic>();
      Dictionary<string, object> dico = new Dictionary<string, object>();
      string req = "select * from " + this.GetType().Name;
      IDataReader dataReader = Connexion.Select(req);
      while (dataReader.Read())
      {
        Dictionary<string, object> buff = new Dictionary<string, object>();
        for (int i = 0; i < dataReader.FieldCount; i++)
        {
          buff.Add(dataReader.GetName(i), dataReader.GetValue(i));
        }
        Liste.Add(DictionaryToObject(buff));
      }
      dataReader.Close();
      return Liste;
    }
    public static List<dynamic> All<T>()
    {
      //  List<dynamic> Liste = new List<dynamic>();
      Dictionary<string, object> dico = new Dictionary<string, object>();
      string req = "select * from " + typeof(T).Name;
      IDataReader dataReader = Connexion.Select(req);


      List<object> list = new List<object>();

      while (dataReader.Read())
      {
        Dictionary<string, object> dc = new Dictionary<string, object>();
        for (int i = 0; i < dataReader.FieldCount; i++)
        {
          dc.Add(dataReader.GetName(i), dataReader.GetValue(i));
        }
        list.Add(DictionaryToObject(dc));
      }
      dataReader.Close();
      return (List<dynamic>)list;
    } }
}
```

## La classe Filiere.cs:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace projet
{
    class filiere:Model
    {
        string code;
        string designation;

        public string Code { get => code; set => code = value; }
        public string Designation { get => designation; set => designation = value; }
    }
}
```

## La classe Eleve.cs:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using MySql.Data;
using System.Data;
namespace projet
{
    class eleve: Model
 {      string code;
        string nom;
        string prenom;
        string niveau;
        string code_Fil;
        public string Code { get => code; set => code = value; }
        public string Nom { get => nom; set => nom = value; }
        public string Prenom { get => prenom; set => prenom = value; }
        public string Niveau { get => niveau; set => niveau = value; }
        public string Code_Fil { get => code_Fil; set => code_Fil = value; }
        public eleve(string cd, string nm, string prnm, string nv, string cf)
        {   code = cd;
            nom = nm;
            prenom = prnm;
            niveau = nv;
            code_Fil = cf;
        }
        public eleve()  { }
            public static List<eleve> afficher(eleve o = null)
        {
            List<eleve> LE = new List<eleve>();
        string sql = "select * from eleve ";
            if (o != null)
                sql += "where id=" + o.id;
                IDataReader Dr = Connexion.Select(sql);
            while (Dr.Read())
                LE.Add(new eleve(Dr.GetString(1), Dr.GetString(2), Dr.GetString(3),
Dr.GetString(4), Dr.GetString(5)));
            Dr.Close();
            return LE;
        }}}
```

## La classe matiere.cs:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace projet
{
    class matiere:Model
    {
        string code, designation, VH, code_module;

        public string Code { get => code; set => code = value; }
        public string Designation { get => designation; set => designation = value; }
        public string VH1 { get => VH; set => VH = value; }
        public string Code_module { get => code_module; set => code_module = value; }
    }
}
```

## La classe notes.cs:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace projet
{
    class notes:Model
    {
        string code_eleve;
            string code_mat;
        float note;

        public string Code_eleve { get => code_eleve; set => code_eleve = value; }
        public string Code_mat { get => code_mat; set => code_mat = value; }
        public float Note { get => note; set => note = value; }
    }
}
```

trigger insert_note : Vérification de l'insertion des notes entre 0 et 20 et affichage d'erreur lors de l'insertion des note négatives ou supérieur à 20.

```
delimiter //
create trigger insert_note before insert on notes for each row
begin
if new.note<0 or new.note>20 then
SIGNAL SQLSTATE '45000'
SET MESSAGE_TEXT = "La note saisi n'est pas valide";
end if;
end //
delimiter ;
```

La fonctionne moyennemod:permet de calculer le moyenne d'un module pour un etudiant.

```
delimiter //
CREATE  FUNCTION `moyennemod`( codemod varchar(30), codeeleve varchar(30)) RETURNS float
READS SQL DATA
DETERMINISTIC
BEGIN

Declare moymod float;
select avg(no.note) from notes no where no.code_mat in ( select ma.code from matiere ma where
ma.code_module=codemod) and no.code_eleve=codeeleve into moymod;
return moymod;
end //
delimiter ;
```

La procédure CursorLoop:
cette procédure permet d'insérer les moyennes de tous les modules d'un eleve dans une table notesmode.

```
create procedure CursorLoop (IN niv varchar(20), IN filo varchar(20), IN code_elv varchar(30))
 begin
 DECLARE updateDone INT DEFAULT 0;
DECLARE ch_done INT DEFAULT 0;
 DECLARE x float;
 DECLARE CodeMod varchar(30);
 DECLARE CUR_Code_Mod CURSOR FOR SELECT distinct code from module where niveau= niv and
code_fil= filo;
DECLARE CONTINUE HANDLER FOR NOT FOUND SET updateDone = 1;
 OPEN CUR_Code_Mod;
 pl:loop
 FETCH CUR_Code_Mod INTO CodeMod;
 set x= moyennemod(CodeMod, code_elv);
  IF updateDone =1 THEN
        LEAVE pl;
    END IF;
 insert into notesmode values( code_elv  , CodeMod ,x);

end loop;
 close CUR_Code_Mod;
end //
 delimiter ;
```

Trigger calcule_moyenne: permet de calculer le moyenne annuelle d'un élève si il a tous les notes des matières.

```
delimiter //
CREATE TRIGGER calcule_moyenne AFTER INSERT ON notes FOR EACH ROW
BEGIN
Declare moy double;
Declare a integer;
 Declare b integer;
 Declare filo varchar(30);
Declare niv varchar(30);
Declare c varchar(30);
Declare x float;
Declare z float;
set moy := 0;
select mo.niveau from module mo where mo.code in (select ma.code_module from matiere ma where
ma.code=new.code_mat) into niv;
select count(*) from notes no where no.code_eleve=new.code_eleve into a;
select fil.code from filiere fil where fil.code in ( select mo.code_fil from module mo where mo.code in
(select ma.code_module from matiere ma where ma.code=new.code_mat)) into filo;
select count(*) from matiere ma where ma.code_module in (select mo.code from module mo where
mo.code_fil=filo and mo.niveau=niv) into b;
If(a=b) then
call CursorLoop(niv , filo, new.code_eleve);
select avg(note) from  notesmode where code_eleve=new.code_eleve into z;
 insert into Moyennes( code_eleve,code_fil, niveau, moyenne) values(new.code_eleve,filo,niv,z);
End if ;
END //
delimiter ;
```

## Trigger delete_Moyennes: supprimer une moyenne d'un élève lors de la suppression de la  note d'une matière.

```
delimiter //
create trigger delete_Moyennes
after delete on notes
for each row
begin
delete from moyennes where code_eleve=old.code_eleve;
END//
 delimiter ;
```

## Trigger update_Moyennes: mise à jour de la moyenne lors de la modification d'une note d'une matière.

```
delimiter //
create trigger update_Moyennes
after update on notes
for each row
begin
declare moyenne float;
select avg(note) into moyenne from notes where code_eleve=new.code_eleve;
update moyennes set moyenne=moyenne where code_eleve=new.code_eleve;
END//
delimiter ;
```

Trigger trigg_op before:permet de changer le niveau de l'etudiant

- si l'etudiant en AP1 et la moyenne>=10 et le nombre de modules <10 est 5 ,le niveau de l'etudiant sera 2.
- si la filiere de l'etudiant n'etait pas Ap et il a comme moyenne 12 ou plus et le nombre des modules<10 est 4 et le niveau est 3 le niveau de l'etudiant sera  diplome

```
delimiter //
CREATE TRIGGER trigg_op before INSERT ON moyennes FOR EACH ROW
BEGIN
declare a int;
select count(*) from notesmode where note<10 and code_eleve=new.code_eleve into a;
if(new.code_fil='Ap' and new.moyenne>=10 and a<5) then
 if(new.niveau='1') then
 update eleve set niveau='2'  where code=new.code_eleve;
    else update eleve set niveau='1', code_fil='ginf'  where code=new.code_eleve;
    end if;
      end if;
 if(new.code_fil!='Ap' and new.moyenne>=12 and a<4) then
 if(new.niveau='3') then
    update eleve el set el.niveau='Diplome' where el.code=new.code_eleve;

    else update eleve set niveau = cast((cast(new.niveau as float)+1) as char(30)) where
code=new.code_eleve;
  end if;

END if;
end //
delimiter ;
```

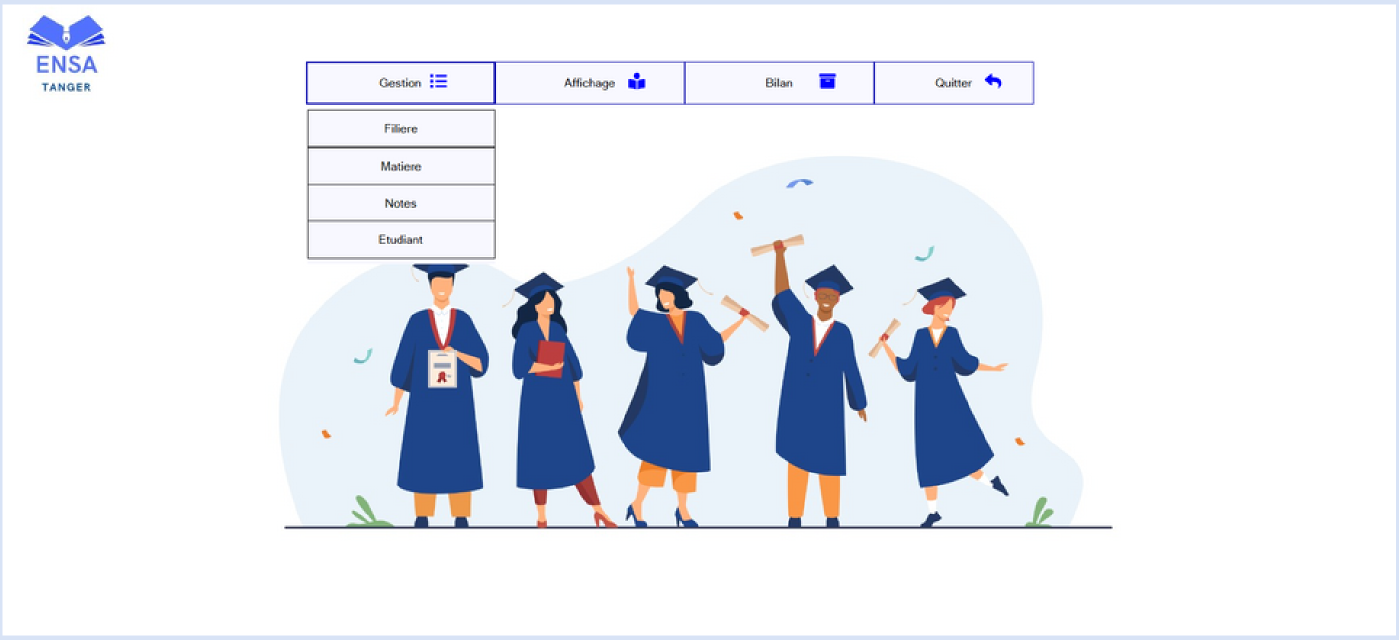
# Code de génération de bilan en format Excel

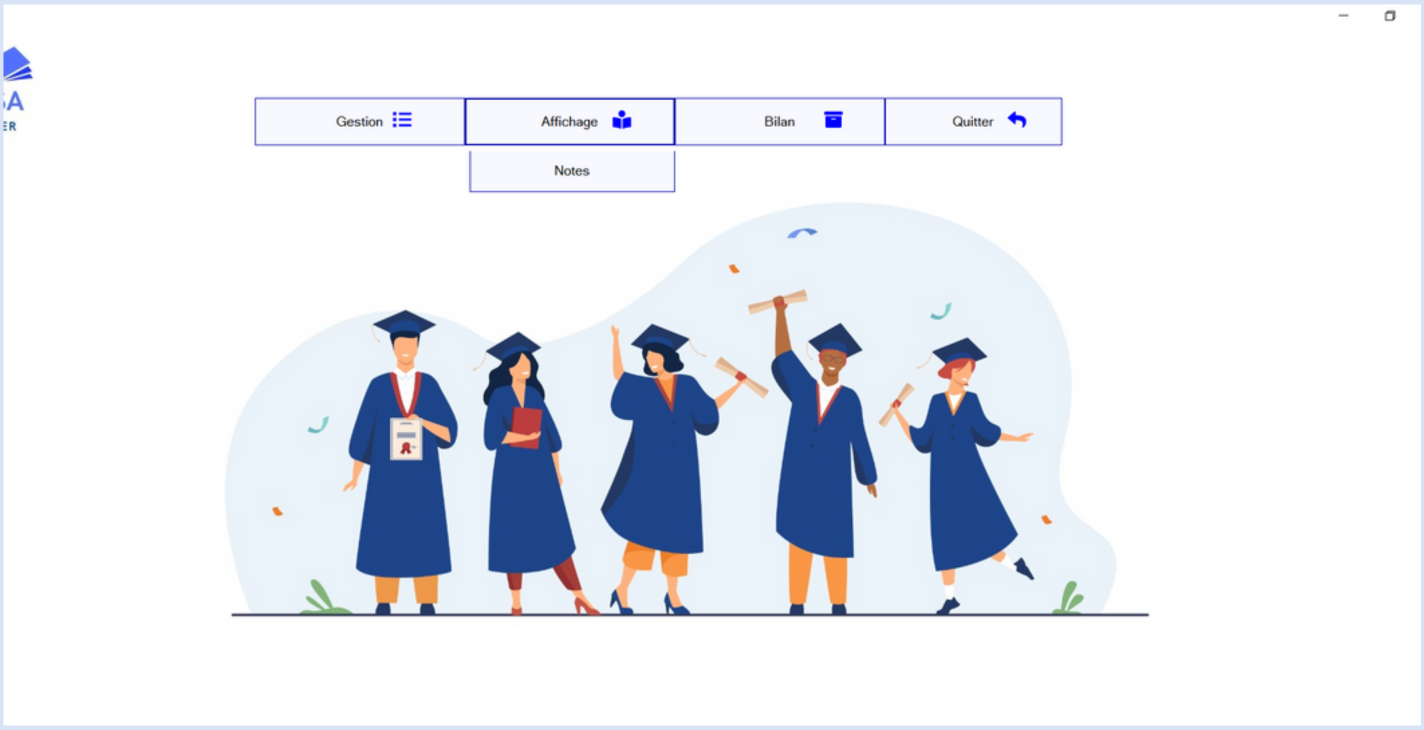
```
        private void button2_Click_1(object sender, EventArgs e)
      { if (textBoxpath.Text.Length != 0)
          {
              l_cmd.CommandText = "select * from moyennes where code_fil='" + comboBoxfl.Text +
"' and niveau='" + comboBoxnv.Text + "'";
              MySqlDataReader lrd = l_cmd.ExecuteReader();
              Dt.Load(lrd);
              ExcelPackage.LicenseContext = OfficeOpenXml.LicenseContext.NonCommercial;


          using (var package = new ExcelPackage(new FileInfo("MyWorkbook.xlsx")))
          {

          }
          ExcelPackage excelPackage = new ExcelPackage();
          ExcelWorksheet excelWorksheet = excelPackage.Workbook.Worksheets.Add("Sheet1");

              filepath = textBoxpath.Text;
              // Création du fichier Excel
              FileInfo filePath = new FileInfo(filepath);
              //excelPackage.SaveAs(new FileInfo(filepath));
              ExcelPackage l_xlp = new ExcelPackage();
              ExcelWorksheet l_xlw = l_xlp.Workbook.Worksheets.Add("Data");
              l_xlw.Cells["A1"].LoadFromDataTable(Dt, true);
              l_xlp.SaveAs(filePath);
              System.Diagnostics.Process.Start(textBoxpath.Text);
          }
      }
  }
```

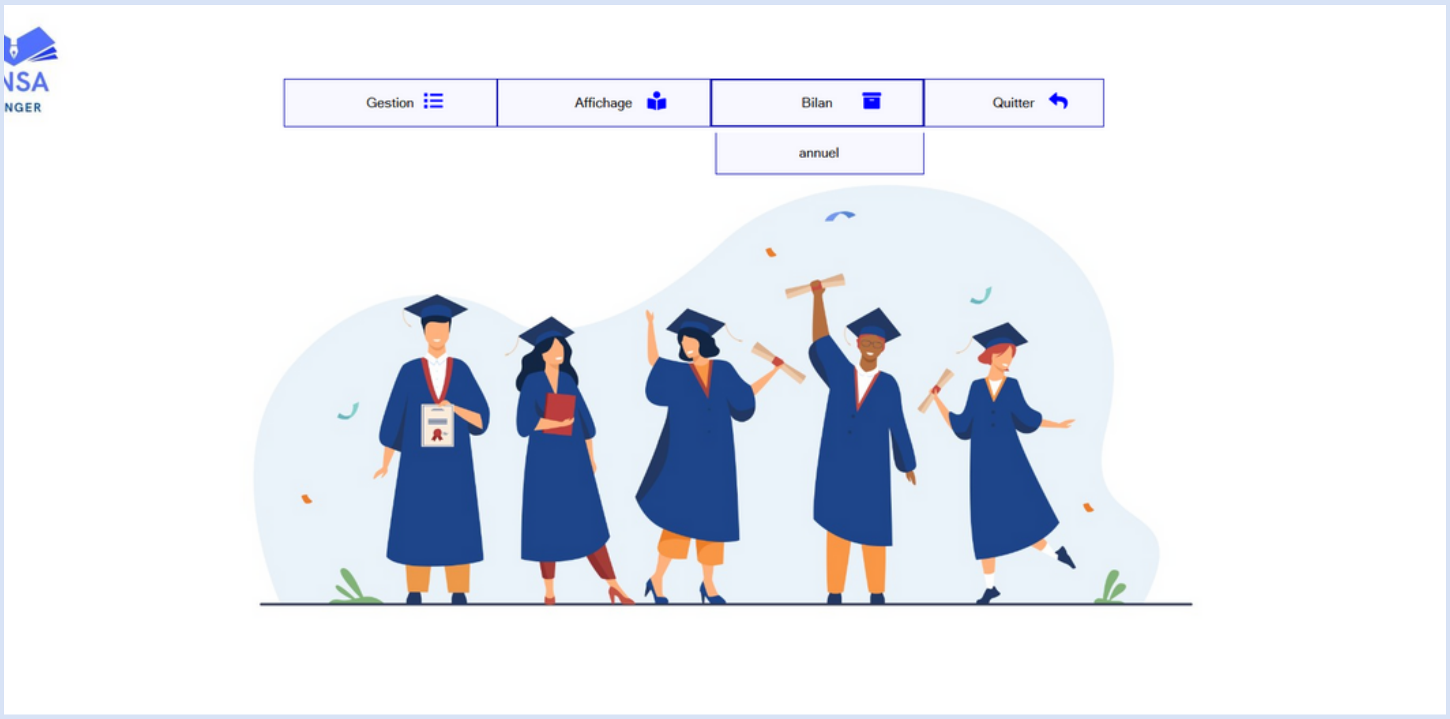
## procédures stockées

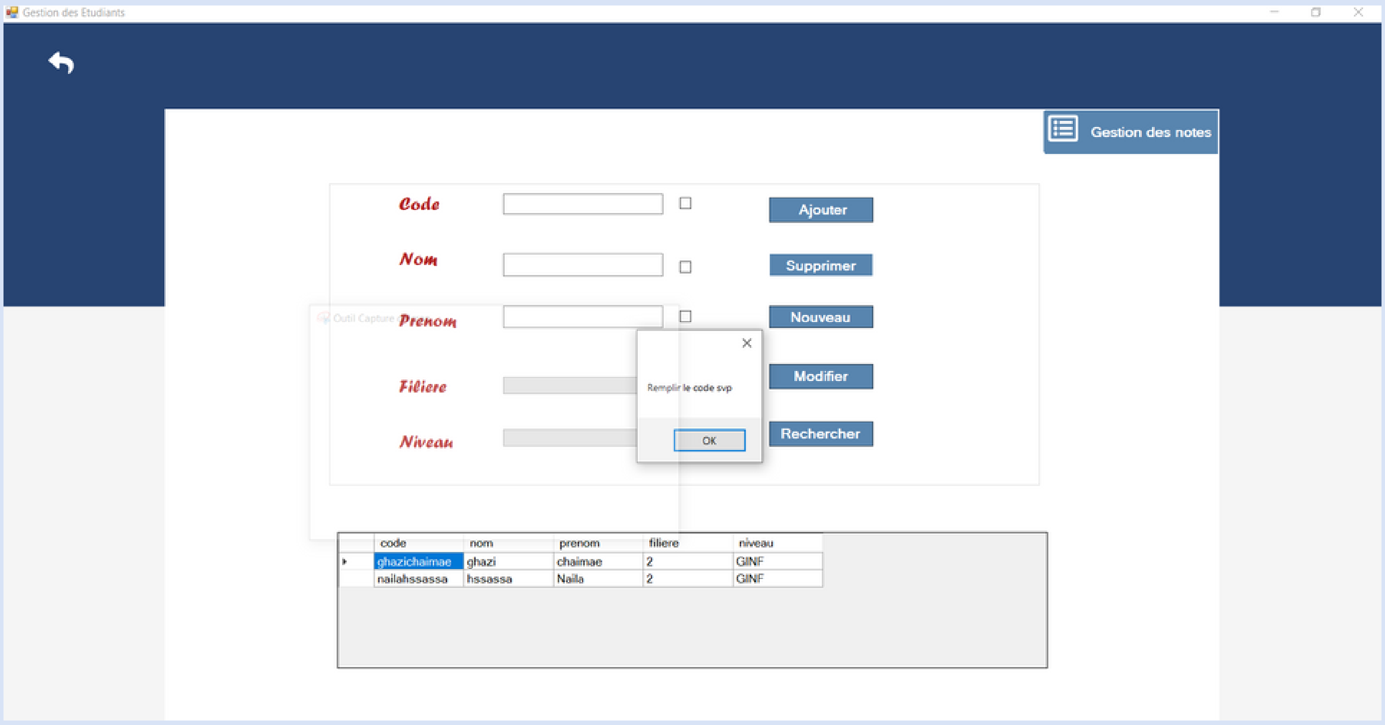
```
DELIMITER $$
CREATE DEFINER=`root`@`localhost` PROCEDURE `ajout_note`(in codee varchar(25) ,in mat varchar(25),in
note float)
begin
insert into notes(code_eleve, code_mat, note) values(codee,mat,note);
END$$
DELIMITER ;

DELIMITER $$
CREATE DEFINER=`root`@`localhost` PROCEDURE `Ajouter_elv`(IN p1 char(30), IN p2 char(30), IN p3
char(30), IN p4 char(30), IN p5 char(30))
BEGIN
insert into eleve(code, nom, prenom, niveau, code_fil) values(p1, p2, p3, p4, p5);
END$$
DELIMITER ;
##############
DELIMITER $$
CREATE DEFINER=`root`@`localhost` PROCEDURE `Modifier_elv`(IN p1 char(30), IN p2 char(30), IN p3
char(30), IN p4 char(30), IN p5 char(30))
BEGIN
UPDATE eleve
SET nom = p2, prenom = p3, niveau = p4, code_fil = p5
WHERE code = p1 ;
END$$
DELIMITER ;
###############
DELIMITER $$
CREATE DEFINER=`root`@`localhost` PROCEDURE `Modifier_note`(IN p1 char(30), IN p2 char(30))
BEGIN
UPDATE note
SET note = p3
WHERE id = p1 ;
END$$
DELIMITER ;
##########
DELIMITER $$
CREATE DEFINER=`root`@`localhost` PROCEDURE `Supprimer_elv`(IN p1 integer)
BEGIN
Delete from eleve WHERE id = p1 ;
END$$
DELIMITER ;
###########
DELIMITER $$
CREATE DEFINER=`root`@`localhost` PROCEDURE `Supprimer_note`(IN p1 integer)
BEGIN
Delete from note WHERE id = p1 ;
END$$
DELIMITER ;
```

# Interfaces


ENSA
TANGER
Gestion
Affichage
Bilan
Quitter
Filiere
Matiere
Notes
Etudiant


Gestion
Affichage
Bilan
Quitter
Notes



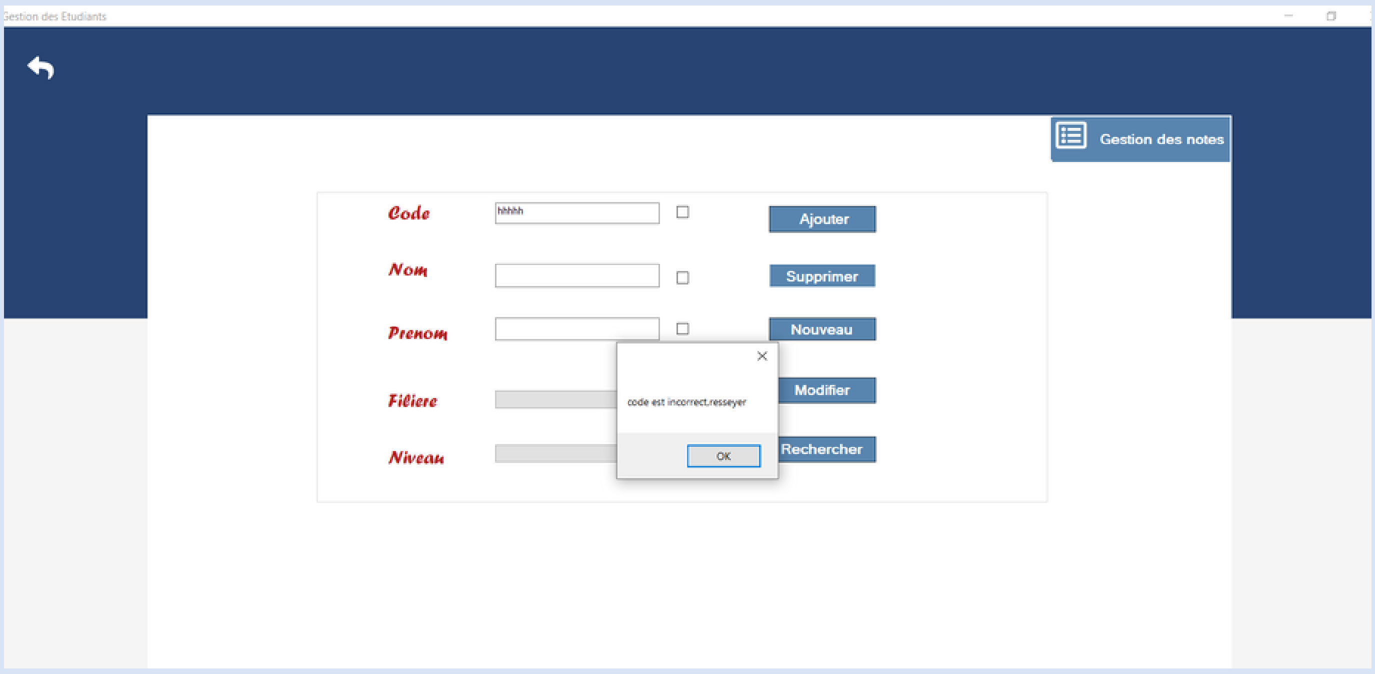


Remplir le code svp

| code | nom | prenom | filiere | niveau |
|---|---|---|---|---|
| ghazichaimae | ghazi | chaimae | 2 | GINF |
| nailahssassa | hssassa | Naila | 2 | GINF |

## L'interface de gestion des étudiants permet:

- L'Ajout d'un étudiants en appelant la méthode Save de la classe Model, (sans paramètres pour un ajout avec une requête ou avec le nom de la procédure stockée désirant l'utilisé dans l'ajout comme paramètre).
- La suppression après confirmation d'un étudiant par son id (utilisation de la fonction idfromCode())
- La modification des données d'un étudiant en appelant la méthode Save sans ou avec paramètre(cas d'usage de procédure stockée).
- La recherche multicritère des étudiants basée sur les champs cochées(si le champs du code est sélectionné la recherche par les autres champs sera négligé, et si aucun champs n'est rempli tous les étudiants seront affiché)
- Le bouton Nouveau permet de réinitialiser l'interface.



Code: hhhhh

Nom:

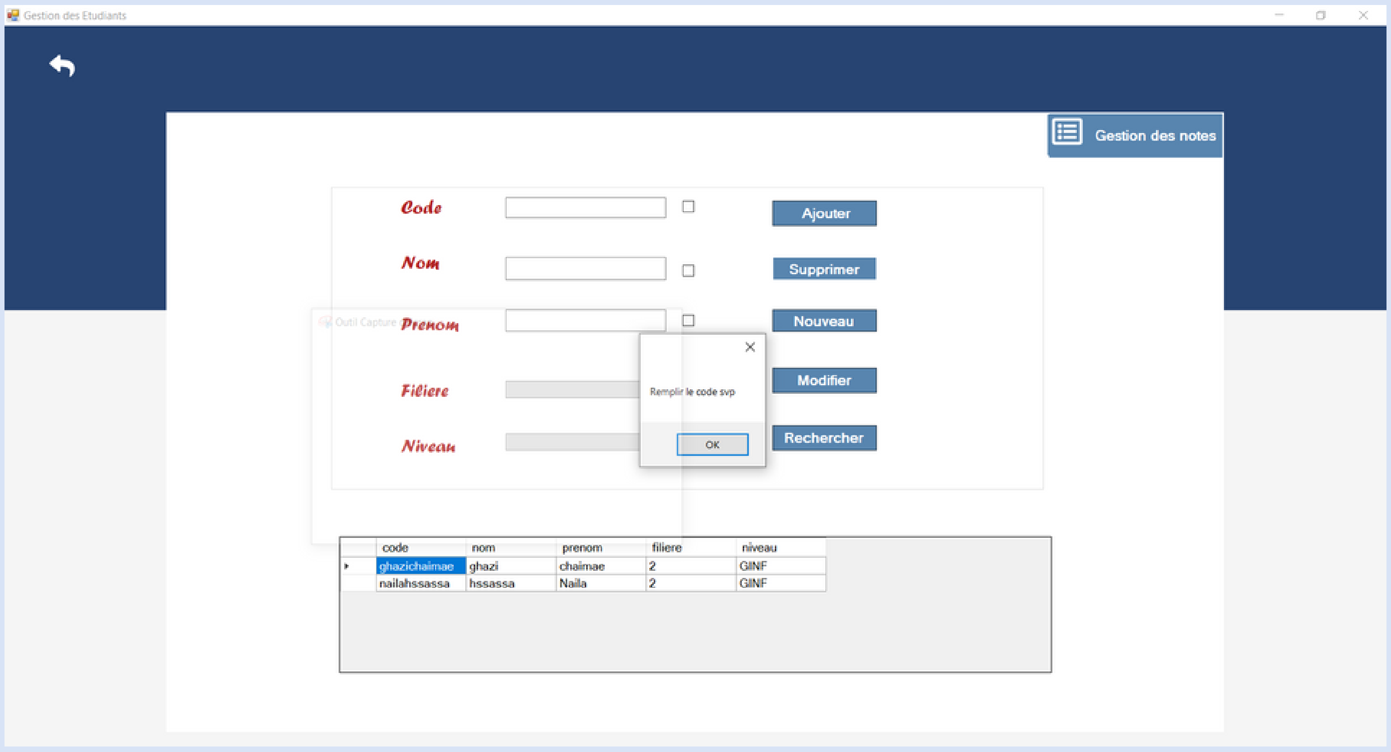
Prenom:

Filiere:

Niveau:

code est incorrect.resseyer

- Un click sur le bouton Gestion des notes permet lancer la fenêtre de gestion des notes specifique au etudiants dans le code doit etre saisi dans l'interface de gestion des etudiants. Si aucun code n'est saisi un message d'erreur sera affiché.



Code:

Nom:

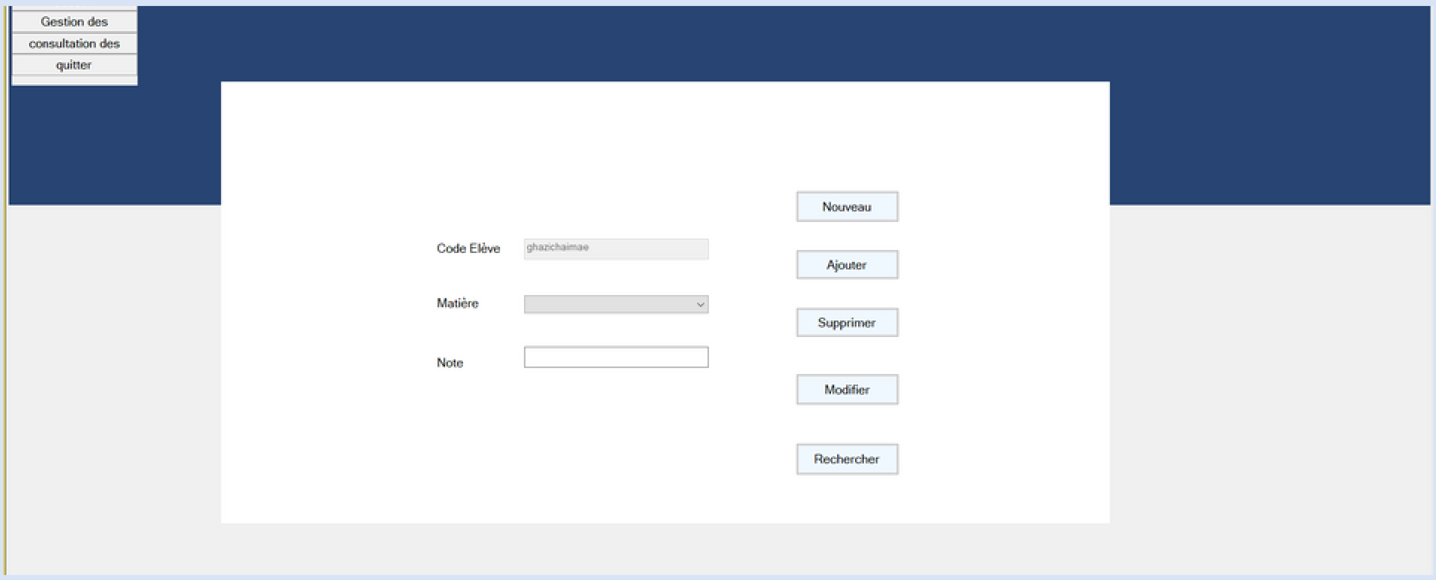
Prenom:

Filiere:

Niveau:

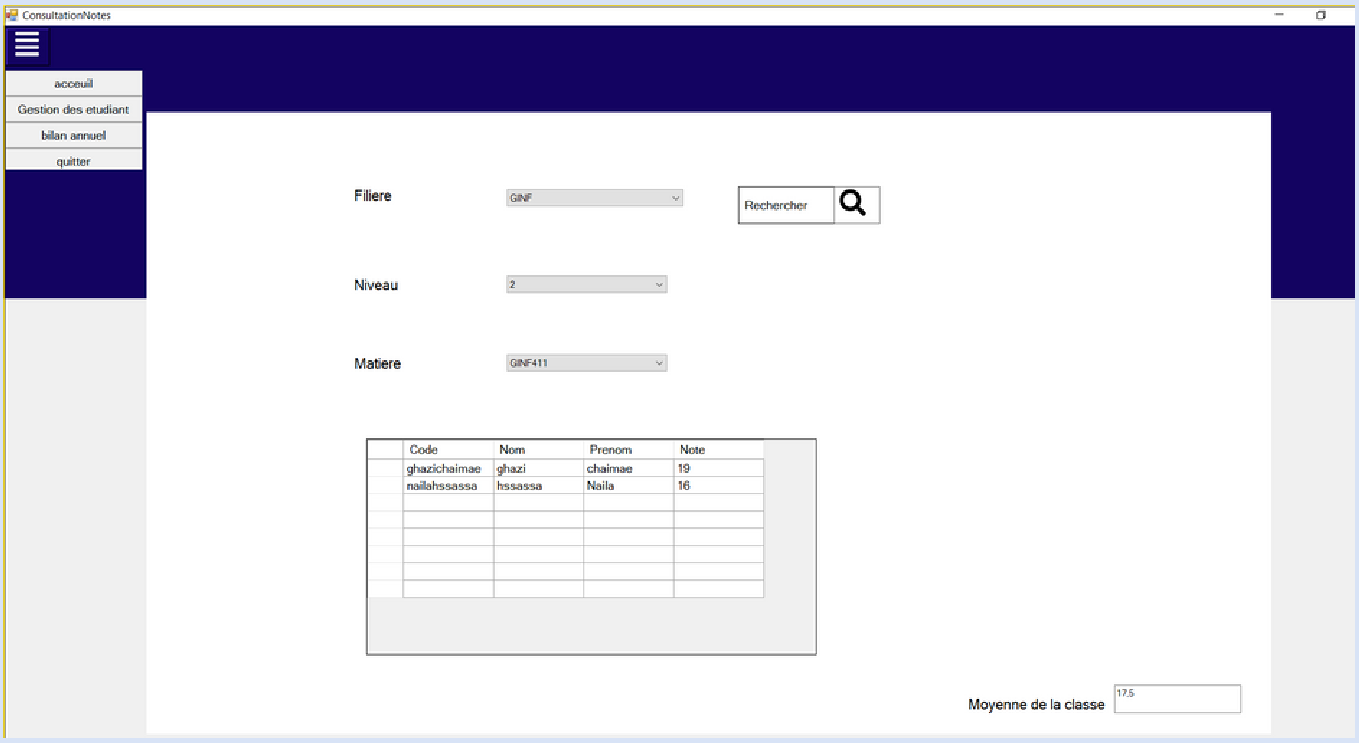
Remplir le code svp

| code | nom | prenom | filiere | niveau |
|---|---|---|---|---|
| ghazichaimae | ghazi | chaimae | 2 | GINF |
| nailahssassa | hssassa | Naila | 2 | GINF |



Code Elève: ghazichaimae
Matière:
Note:

- Une fois un bon code d'élève est saisi l'interface de gestion des notes est lancé.
- Et automatiquement la liste des matières étudiées.
- Pareille a la dernière fenêtre un utilisateur peut Ajouter, Supprimer, Modifier ou Recherche une note d'un étudiant.

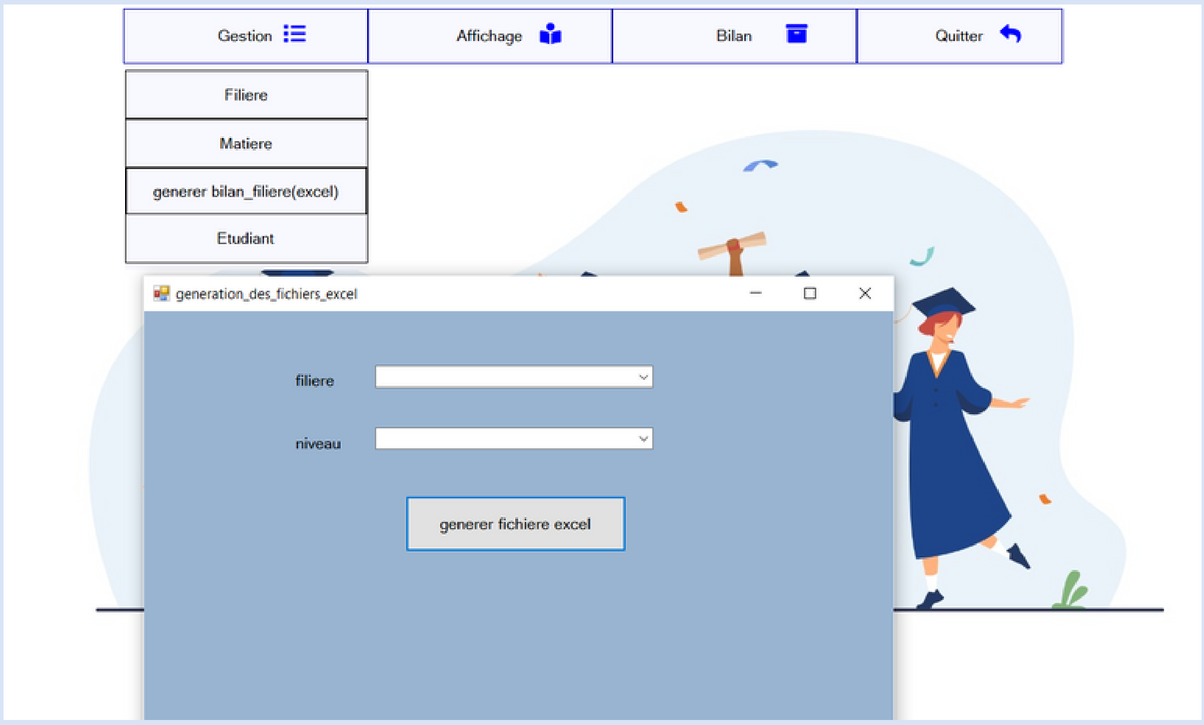


Filiere: GINF
Niveau: 2
Matiere: GINF411

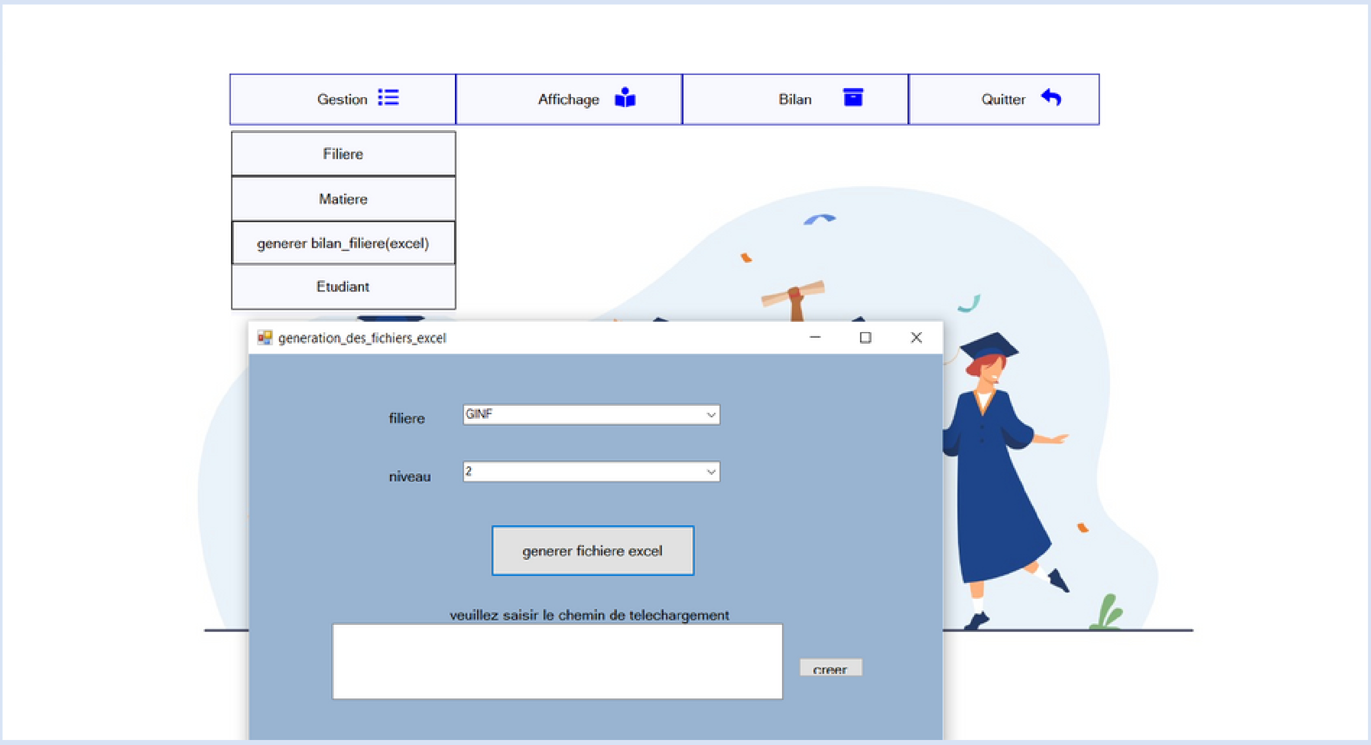
| Code | Nom | Prenom | Note |
|---|---|---|---|
| ghazichaimae | ghazi | chaimae | 19 |
| nailahssassa | hssassa | Naila | 16 |

Moyenne de la classe: 17,5

- L'interface de consultation des notes permet l'affichage des moyennes d'une classe dans un module choisi par l'utilisateur ainsi que la moyenne de la classe dans ce module.



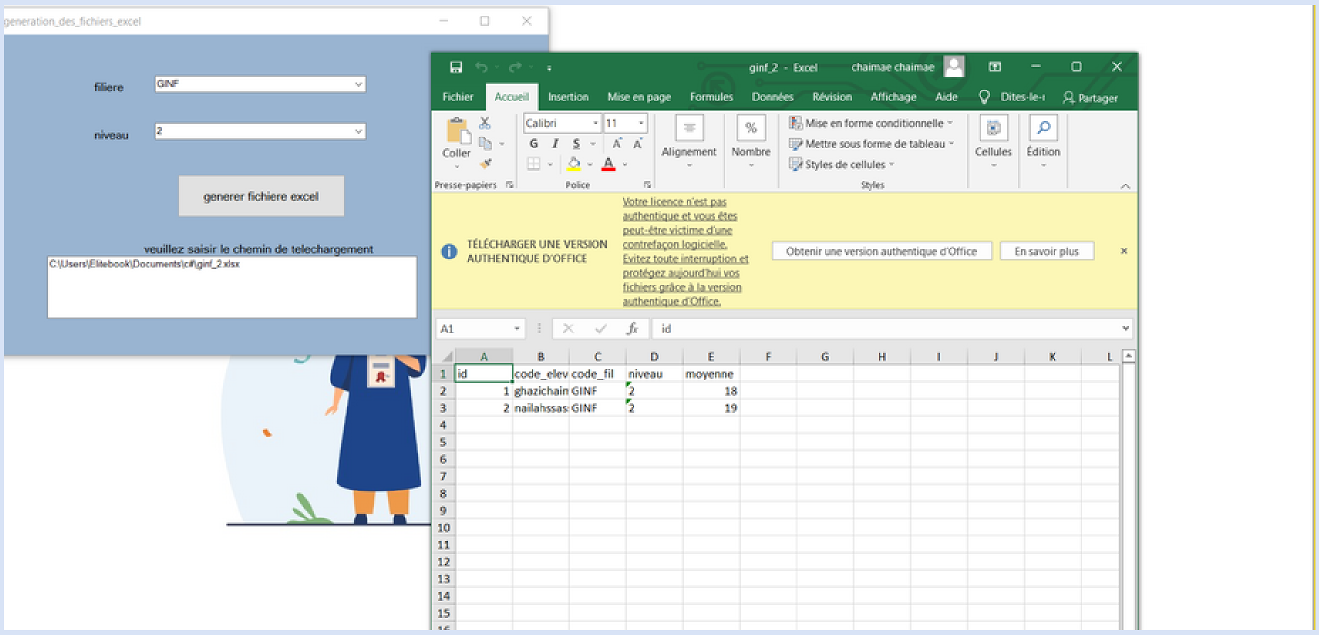
- La fenêtre de génération de bilan permet la création d'un fichier Excel contenant la liste des étudiants d'une classe avec leurs moyennes annuelles dans l'emplacement spécifié par l'utilisateur. Le fichier sera automatiquement ouvert après la génération.



filiere GINF

niveau 2

veuillez saisir le chemin de telechargement



filiere: GINF

niveau: 2

veuillez saisir le chemin de telechargement

C:\Users\Elitebook\Documents\cif\ginf_2.xlsx

TÉLÉCHARGER UNE VERSION AUTHENTIQUE D'OFFICE

Votre licence n'est pas authentique et vous êtes peut-être victime d'une contrefaçon logicielle. Evitez toute interruption et protégez aujourd'hui vos fichiers grâce à la version authentique d'Office.

A1 id

| | A | B | C | D | E |
|---|---|---|---|---|---|
| 1 | id | code_elev | code_fil | niveau | moyenne |
| 2 | 1 | ghazichain | GINF | 2 | 18 |
| 3 | 2 | nailahssas | GINF | 2 | 19 |