

BIBLIOTECA Y EXCEPCIONES

CARLOS ISAAC CARABALLO CASAÑAS
IVET J CABRERA SANTANA

INDICE

- I. CUADERNO DE BATALLA
- II. CÓDIGO DE INTERÉS
- III. MVC
- IV. CAPTURAS EN EJECUCIÓN
- V. EXCEPCIONES Y ERRORES
- VI. RETROALIMENTACIÓN
- VII. CONCLUSIONES Y OPINIONES
- VIII. ENLACES

CUADERNO DE BATALLA

Martes 22 de octubre:

Comenzamos el trabajo aprendiendo el funcionamiento del Visual Studio creando 4 proyectos dentro de una misma solución (Biblioteca, Controllers, Models y Views), pero no nos estaba funcionando por varios errores que nos iban pasando por la inexperiencia con el Visual Studio y C#.

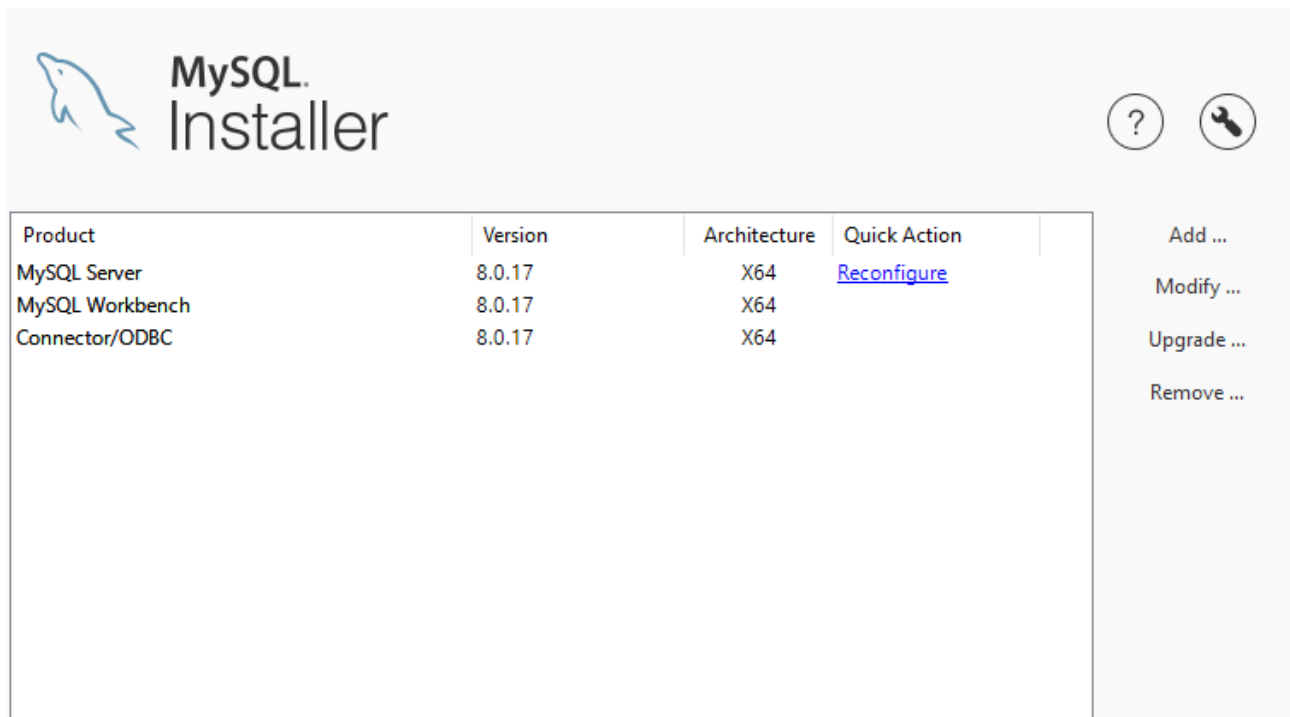
Este día no conseguimos avanzar con el proyecto, pero aprendimos sobre el funcionamiento del Visual Studio para el resto de los días.

Miércoles 23 octubre:

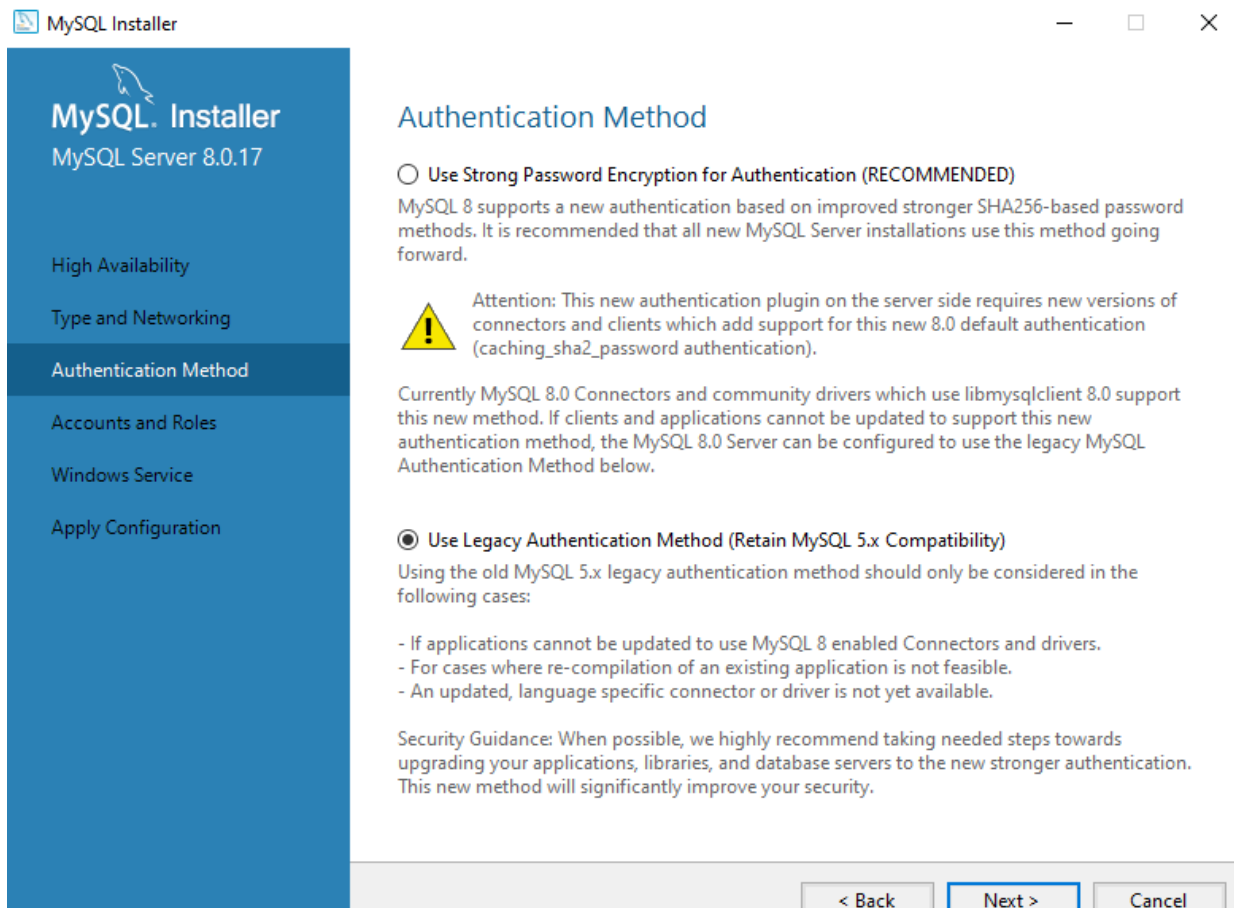
Seguíamos aprendiendo del programa, pero ya avanzamos en el proyecto. Los principales problemas que tuvimos el día anterior fueron a la hora de crear los proyectos, ya que elegíamos mal el tipo de plantilla para cada proyecto. Al principio todos nuestros proyectos eran bibliotecas de clase excepto la de Views que era Aplicación de Windows Forms. Luego cambiamos esa elección a que nuestro proyecto Controllers sea la Aplicación de Windows Forms, nuestro proyecto Models es de tipo Biblioteca de clases y nuestro proyecto Views es de tipo Biblioteca de controles de Windows Forms. Por lo tanto nuestra solución ahora tienen 3 proyectos habiendo borrado el de Biblioteca. Nuestro Controller ahora es un proyecto capaz de abrir ventanas, las Views es donde tenemos la biblioteca de ventanas y nuestro Models es donde tenemos una biblioteca de clases.

Jueves 24 octubre:

Teniendo ahora nuestros proyectos bien seleccionados comenzamos a programar nuestro modelo MVC. Este día tuvimos un problema con la conexión del MySQL que hacía que no pudiéramos conectarnos desde el Visual Studio. Después de buscar en internet encontramos la solución que era un problema de configuración del MySQL para el cual tuvimos que abrir el MySQL Installer y abrir Reconfigure en MySQL Server



Una vez abierto, llegado al paso de Authentication Method hay que elegir el segundo punto tal y como se ve en la foto



Viernes 25 octubre:

El proyecto ya funciona como MVC, así que hoy nos dedicamos a hacer las excepciones. Decidimos crear una clase nueva (CustomException) dentro del proyecto (Models). Este día no tuvimos grandes problemas.

CÓDIGO DE INTERÉS

```
public static void chargeTable()
{
    alumnoView.TableAlumno.Rows.Clear();
    ArrayList alumnosList = alumnoDao.GetAllAlumnos();
    for (int i = 0; i < alumnosList.Count; i++)
    {
        int n = alumnoView.TableAlumno.Rows.Add();
        alumnoView.TableAlumno.Rows[n].Cells[0].Value = ((Alumno)alumnosList[i]).registro;
        alumnoView.TableAlumno.Rows[n].Cells[1].Value = ((Alumno)alumnosList[i]).dni;
        alumnoView.TableAlumno.Rows[n].Cells[2].Value = ((Alumno)alumnosList[i]).nombre;
        alumnoView.TableAlumno.Rows[n].Cells[3].Value = ((Alumno)alumnosList[i]).apellido1;
        alumnoView.TableAlumno.Rows[n].Cells[4].Value = ((Alumno)alumnosList[i]).apellido2;
    }
}
```

El código que tenemos para mostrar los datos en la tabla nada más abrir el programa.

```
public ArrayList GetAllAlumnos()
{
    ArrayList alumnosList = new ArrayList();
    MySqlCommand command = new MySqlCommand("SELECT * FROM alumnos", this.conexion);
    MySqlDataReader data = command.ExecuteReader();
    while (data.Read())
    {
        Alumno alumno = new Alumno(data.GetInt32(0), data.GetString(1), data.GetString(2), data.GetString(3), data.GetString(4));
        alumnosList.Add(alumno);
    }
    data.Close();
    return alumnosList;
}
```

Dentro de AlumnoDao es donde tenemos el código para recoger todos los datos de la base de datos y mostrarlos según el código mostrado anteriormente.

```

public void DeleteAlumno(int registro)
{
    sql = new MySqlCommand("SELECT * FROM prestamos", this.conexion);
    MySqlDataReader data = sql.ExecuteReader();
    while (data.Read())
    {
        int registroNow = 0;
        if (Int32.TryParse(data.GetString(1), out registroNow))
        {
            if (registro == registroNow)
            {
                data.Close();
                CustomException.UserHaveLoanThrow(registro);
            }
        }
    }
    data.Close();

    sql = new MySqlCommand("delete from alumnos where registro = " + registro, this.conexion);
    int numChanges = sql.ExecuteNonQuery();

    if (numChanges <= 0)
    {
        CustomException.RegisterNotExistThrow(registro);
    }
}

```

Método para borrar alumnos de la base de datos. Primero busca dentro de la tabla préstamos de la base de datos si existe el registro seleccionado, si lo encuentra lanza un tipo de excepción USER_HAVE_LOAN dentro de la clase CustomException. Si no se ha encontrado ningún registro en la tabla de préstamos, entonces borra el registro de la tabla alumnos. Si no consigue borrar ningún registro entonces lanza una excepción de tipo REGISTER_NOTEXISTS dentro de la misma clase.

```

public void AddAlumno(Alumno alumno)
{
    sql = new MySqlCommand(string.Format("Insert into alumnos values('{0}', '{1}', '{2}', '{3}', '{4}'))",
        alumno.registro, alumno.dni, alumno.nombre, alumno.apellido1, alumno.apellido2), this.conexion);

    int numChanges = sql.ExecuteNonQuery();
}

public void UpdateAlumno(Alumno alumno)
{
    sql = new MySqlCommand(string.Format("Update alumnos set dni='{1}', nombre='{2}', " +
        " apellido1='{3}', apellido2='{4}' where registro = '{0}'", alumno.registro, alumno.dni, alumno.nombre, alumno.apellido1, alumno.apellido2),
        this.conexion);

    int numChanges = sql.ExecuteNonQuery();

    if (numChanges <= 0)
    {
        CustomException.RegisterNotExistThrow(alumno.registro);
    }
}

```

Estos son los códigos para añadir y actualizar un registro.

```
public class CustomException : Exception
{
    public int type { get; }
    public string message { get; }
    public const int REGISTER_NOTEXISTS = 2;
    public const int USER_HAVE_LOAN = 3;

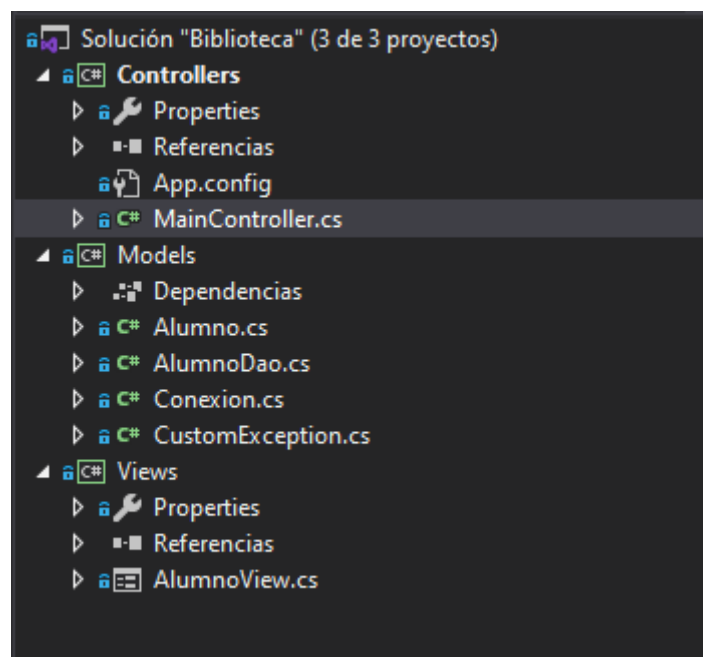
    public CustomException(string message,int type)
    {
        this.message = message;
        this.type = type;
    }

    public static void RegisterNotExistThrow(int registro)
    {
        throw new CustomException("Error : El registro " + registro + " no existe", CustomException.REGISTER_NOTEXISTS);
    }

    public static void UserHaveLoanThrow(int registro)
    {
        throw new CustomException("Error : El registro " + registro + " tiene un prestamo pendiente", CustomException.USER_HAVE_LOAN);
    }
}
```

Aquí creamos la nueva clase CustomException para poder crear nuestras propias excepciones y lanzarlas luego en el programa.

MVC



Este es el esquema de nuestro proyecto terminado.

En el proyecto Controllors tenemos referencias de Models, de Views y del paquete de MySQL. El de Models tiene sólo el de MySQL.

CAPTURAS EN EJECUCIÓN

Alumnos

	Registro	DNI	Nombre	1º Apellido	2º Apellido
▶	3	44F	James	Bond	Del Campo
	4	64a	JUAN MIGUEL	AFONSO	ALONSO
	5	73a	SERGIO	AFONSO	MARRERO
	6	74a	MAR...	AFONSO	MEDINA
	7	79a	HERIBERTO	AFONSO	PEREZ
	8	83a	DESIREE RAIZA	AFONSO	RAMIREZ
	9	85a	JUANA ISABEL	AFONSO	REYES
	10	87a	CARMEN G.	AFONSO	SANTANA
	11	96a	JOSE ANGEL	AGUIAR	MARTIN

Registro:

DNI:

Nombre:

1º Apellido:

2º Apellido:

Esta es nuestra vista al iniciarse.

Alumnos

	Registro	DNI	Nombre	1º Apellido	2º Apellido
▶	3	44F	James	Bond	Del Campo
	4	64a	JUAN MIGUEL	AFONSO	ALONSO
	5	73a	SERGIO	AFONSO	MARRERO
	6	74a	MAR...	AFONSO	MEDINA
	7	79a	HERIBERTO	AFONSO	PEREZ
	8	83a	DESIREE RAIZA	AFONSO	RAMIREZ
	9	85a	JUANA ISABEL	AFONSO	REYES
	10	87a	CARMEN G.	AFONSO	SANTANA
	11	96a	JOSE ANGEL	AGUIAR	MARTIN

Registro:


DNI:

Nombre:

1º Apellido:

2º Apellido:

Error

 Error : El registro 1 no existe

Asi es como se ven los errores.

EXCEPCIONES Y ERRORES

```
try
{
    alumnoDao.AddAlumno(alumno);
    chargeTable();
}
catch (MySqlException ex)
{
    string duplicateError = string.Format("Duplicate entry '{0}' for key 'PRIMARY'",alumno.registro.ToString());
    if (ex.Message == duplicateError)
    {
        MessageBox.Show(alumnoView, "Error : El registro " + alumno.registro + " ya existe",
            "Error Fatal", MessageBoxButtons.OK, MessageBoxIcon.Error);
    }
    else {
        MessageBox.Show(alumnoView, "Error Fatal: No se puede ejecutar la sentencia", "Error Fatal", MessageBoxButtons.OK, MessageBoxIcon.Error);
        Application.Exit();
    }
}
```

Aquí controlamos excepciones de MySQL. Conseguimos controlar que si la excepción se ha lanzado porque el registro ya estaba creado, mostramos un error al usuario. Si es un error que no teníamos previsto entonces cerramos el programa.

```
try
{
    alumnoDao.UpdateAlumno(alumno);
    chargeTable();
}
catch (CustomException ex)
{
    switch (ex.type)
    {
        case CustomException.REGISTER_NOTEXISTS:
            MessageBox.Show(alumnoView, ex.message, "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);
            chargeTable();
            break;
    }
}
catch (MySqlException)
{
    MessageBox.Show(alumnoView, "Error Fatal: No se puede ejecutar la sentencia", "Error Fatal", MessageBoxButtons.OK, MessageBoxIcon.Error);
    Application.Exit();
}
```

En la excepción CustomException comprobamos que si el usuario intenta actualizar un registro que no existe, entonces se lo informamos con un error. Si se da un error MySQL exception se cierra el programa.

```

try
{
    alumnoDao.DeleteAlumno(registro);
    chargeTable();
}
catch (CustomException ex)
{
    switch (ex.type)
    {
        case CustomException.REGISTER_NOTEXISTS:
            MessageBox.Show(alumnoView, ex.message, "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);
            chargeTable();
            break;
        case CustomException.USER HAVE LOAN:
            MessageBox.Show(alumnoView, ex.message, "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);
            chargeTable();
            break;
    }
}
catch (MySqlException)
{
    MessageBox.Show(alumnoView, "Error Fatal: No se puede ejecutar la sentencia", "Error Fatal", MessageBoxButtons.OK, MessageBoxIcon.Error);
    Application.Exit();
}
}

```

En el CustomException controlamos dos tipos de errores. Si el usuario intenta eliminar un registro que no existe o si intenta eliminar un usuario que ya tiene un préstamo pendiente. Cada tipo de error se lo informamos al usuario.

RETROALIMENTACIÓN

The screenshot shows a window titled "Alumnos" with a table of student records and a form below it for adding a new student.

	Registro	DNI	Nombre	1º Apellido	2º Apellido
▶	3	44F	James	Bond	Del Campo
	4	64a	JUAN MIGUEL	AFONSO	ALONSO
	5	73a	SERGIO	AFONSO	MARRERO
	6	74a	MAR...	AFONSO	MEDINA
	7	79a	HERIBERTO	AFONSO	PEREZ
	8	83a	DESIREE RAIZA	AFONSO	RAMIREZ
	9	85a	JUANA ISABEL	AFONSO	REYES
	10	87a	CARMEN G.	AFONSO	SANTANA
	11	96a	JOSE ANGEL	AGUIAR	MARTIN

Below the table, there is a form with the following fields and buttons:

- Registro:** A dropdown menu with the value "1" selected.
- DNI:** A text input field.
- Nombre:** A text input field.
- 1º Apellido:** A text input field.
- 2º Apellido:** A text input field.
- Buttons:** "Añadir", "Actualizar", and "Eliminar".

Para añadir un usuario, debes comprobar que el número de registro no exista en la tabla y se debe rellenar todos los campos con los datos correspondientes y pulsar el botón añadir.

Para actualizar un usuario, se debe comprobar que ese usuario exista en la tabla y se debe rellenar todos los campos con los datos correspondientes y pulsar el botón actualizar.

Para eliminar un registro, se debe comprobar que el usuario exista en la tabla. Solamente se debe seleccionar el campo registro que se desee eliminar.

CONCLUSIONES Y OPINIONES

Trabajar con Visual Studio fue un poco confuso al principio ya que teníamos que aprender nuevos conceptos como los espacios de nombre, aprender que tipo de proyectos elegir y sus normas. Además siempre que queríamos actualizar el proyecto en github teníamos que cerrar el programa o nos saltaba un error.

ENLACES

El enlace que más utilizamos para seguir el proyecto fue el siguiente:

<https://docs.microsoft.com/es-es/dotnet/api/system.windows.forms?view=netframework-4.8>