Gestión de la Información
Trabajo en Grupo

Miembros:

Inbal Hasson Tapiero
Francisco Javier Aguayo Villalba

Tecnologías Usadas:

C# (.NET)
Fluent NHibernate
SQLite

Contenido de la Presentación

01

Entorno de Programación

Donde hablaremos sobre las tecnologías utilizadas

02

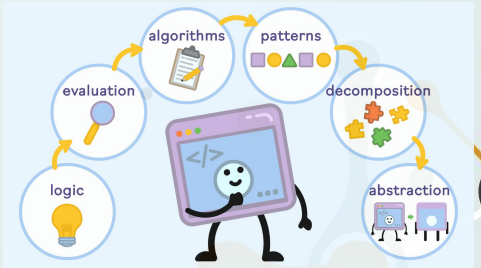
Desarrollo de la Aplicación

Donde mostraremos el código implementado

03

Demostración del Software

Donde enseñaremos el programa en ejecución



Características de C#

01

- Lenguaje Multiplataforma, Modular y Orientado a Objetos.
- Fácil de usar por su sintaxis sencilla y moderna, con elementos de Java, C++, Javascript...
- Adecuado para desarrollar todo tipo de proyectos basados en .NET.
- Gran cantidad de librerías disponibles y actualizaciones frecuentes.

Características de .NET Core

- Framework de desarrollo de Microsoft, lanzado en 2002.
- Plataforma gratuita, multiplataforma y de código abierto diseñada para compilar muchos tipos de aplicaciones diferentes.
- Proporciona gran cantidad de utilidades para un desarrollo de software eficiente.
- Facilita en gran medida el desarrollo de software escalable y de alto rendimiento.

Ventajas

- Accesibles y de fácil aprendizaje.
- Gran rendimiento al generar código específico dependiendo de la plataforma en que se usan.
- Código legible que simplifica el mantenimiento.

Desventajas

- El manejo de la memoria que ofrece su recolector de basura consume gran cantidad de recursos.
- Tiempo de compilación más largo que otros lenguajes de programación.

01 Algunas Aplicaciones Escritas en C#

Características y uso de

FLUENT NHIBERNATE

01

- Es la conversión de Hibernate de Java a C#, de código abierto y muy usada dentro del desarrollo en .NET.
- Facilita el mapeo de atributos entre la base de datos y el modelo de objetos.
- La versión Fluent se olvida del lenguaje de marcado XML y nos proporciona una versión de la tecnología más accesible y dinámica.

01

Características de SQLite

- Herramienta de software libre multiplataforma, útil en dispositivos empotrados o con pocas capacidades de hardware.
- Presenta un alto rendimiento, es sencilla, eficaz y potente.
- No necesita el uso de servidores, facilitando mucho el proceso de desarrollo.
- Ideal para bases de datos ligeras, ya que se encuentra contenida en un único archivo muy liviano. (<300 KB)

Usando 

Ventajas

- Presenta una lectura y escritura de datos muy rápida.
- Gran portabilidad de los datos entre un sistema y otro.
- Permite guardar muchísima información en un espacio reducido.

Desventajas

- SQLite sólo trabaja con cuatro tipos de datos(INTEGER, REAL, BLOB y TEXT) lo que limita su uso en muchos proyectos.
- No permite que más de un usuario interactúe con la base de datos.

Desarrollo de la Aplicación:

Clases de entidad y de mapeo

Clase Muestra

```
public class Muestra
{
    3 referencias
    public virtual int ID { get; protected set; }
    6 referencias
    public virtual string? NIF_Paciente { get; set; }
    6 referencias
    public virtual string? Cultivo { get; set; }
    6 referencias
    public virtual Solucion Solucion { get; set; }
```

Clase MuestraMap

```
public class MuestraMap : ClassMap<Muestra>
{
    0 referencias
    public MuestraMap()
    {
        Id(x => x.ID);
        Map(x => x.NIF_Paciente);
        Map(x => x.Cultivo);
        References(x => x.Solucion);
    }
}
```

Claves compuestas

Referencia a la entidad Rol

```
public PermisoMap()
{
    CompositeId()
        .KeyReference(x => x.rol, "id_rol")
        .KeyProperty(x => x.pantalla);
    Map(x => x.acceso);
    Map(x => x.insertar);
    Map(x => x.modificar);
    Map(x => x.borrar);
}
```

Conexión con la base de datos

```
3 referencias
public static ISessionFactory CreateSessionFactory()
{

    return Fluently.Configure()
        .Database(SQLiteConfiguration.Standard
            .UsingFile(DbFile))
        .Mappings(m =>
            m.FluentMappings.AddFromAssemblyOf<ConexionBD>())
        .BuildSessionFactory();
}
```

Consultas SQL

```
var sessionFactory = ConexionBD.CreateSessionFactory();

using (var session = sessionFactory.OpenSession())
{
    var usuarios = session.Query<Usuario>()
        .Where(x => x.nif == usuarioTextBox.Text && x.password == passwordTextBox.Text)
        .ToArray();
    if (usuarios.Length > 0)
    {
        usuario = usuarios[0];
    }

}
```

Insertado, borrado y actualizado de datos

Insertado de una muestra

```
using (var transaction = session.BeginTransaction())
{
    Solucion solucion = (Solucion)solucionListBox.SelectedItem;
    Muestra muestra = new Muestra(solucion)
    {
        NIF_Paciente = nifTextBox.Text,
        Cultivo = cultivoTextBox.Text,
    };
    session.SaveOrUpdate(solucion);

    transaction.Commit();
}
```

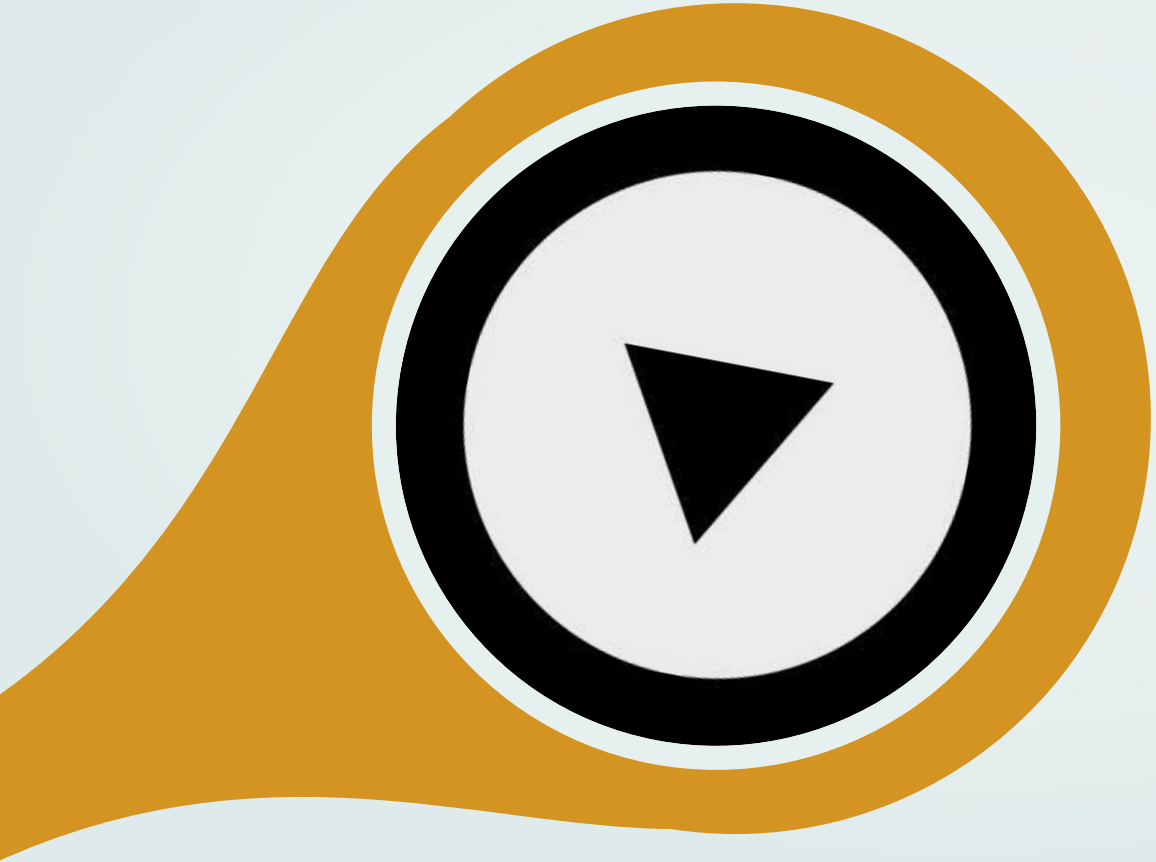
Actualizado de una muestra

```
using (var transaction = session.BeginTransaction())
{
    muestraSeleccionada.NIF_Paciente = nifTextBox.Text;
    muestraSeleccionada.Cultivo = cultivoTextBox.Text;
    muestraSeleccionada.cambiaSolucion((Solucion)solucionListBox.SelectedItem);

    transaction.Commit();
}
updateDataGridView();
```

Borrado de una muestra

```
using (var transaction = session.BeginTransaction())
{
    muestraSeleccionada.Delete();
    session.Delete(muestraSeleccionada);
    muestraSeleccionada = null;

    transaction.Commit();
}
```

Demostración de la aplicación



FIN