CIFP JUAN DE COLONIA

Memorias de Biblionet

Desarrollo de interfaces: Proyecto 2 Eva

Alexis López Briongos Dam2t

05/02/2024

Índice

Introducción al proyecto

1. **Título del Proyecto**: Biblionet
2. **Descripción del Proyecto**:
    - Esta aplicación tiene como objetivo principal facilitar a los usuarios la obtención de libros a través de préstamos. Los usuarios estándar pueden realizar solicitudes de préstamo, mientras que los usuarios administradores tienen la capacidad de gestionar aspectos clave como usuarios, libros, préstamos, y las incidencias reportadas por los usuarios.
3. **Objetivos: El proyecto tiene como objetivos:**
    - Facilitar a los usuarios estándar la adquisición de libros a través de un sistema eficiente de préstamos.
    - Permitir a los usuarios administradores la gestión efectiva de usuarios, libros, préstamos, y el manejo de incidencias.
4. **Público Objetivo**:
    - La aplicación está diseñada para atender a dos grupos de usuarios:
    - Usuarios Estándar: Personas que desean acceder a libros a través de préstamos.
    - Usuarios Administradores: Personal encargado de administrar y supervisar el sistema, gestionando usuarios, libros, préstamos, y abordando incidencias.
5. **Justificación**:
    - La creación de esta aplicación responde a la necesidad de proporcionar una plataforma eficiente y accesible para la obtención de libros mediante préstamos. La inclusión de funcionalidades administrativas facilita la gestión integral del sistema, permitiendo un control efectivo de usuarios, libros y préstamos. La implementación de esta solución contribuirá a mejorar la experiencia de los usuarios al mismo tiempo que optimiza la administración y operación del sistema.

Construcción del proyecto

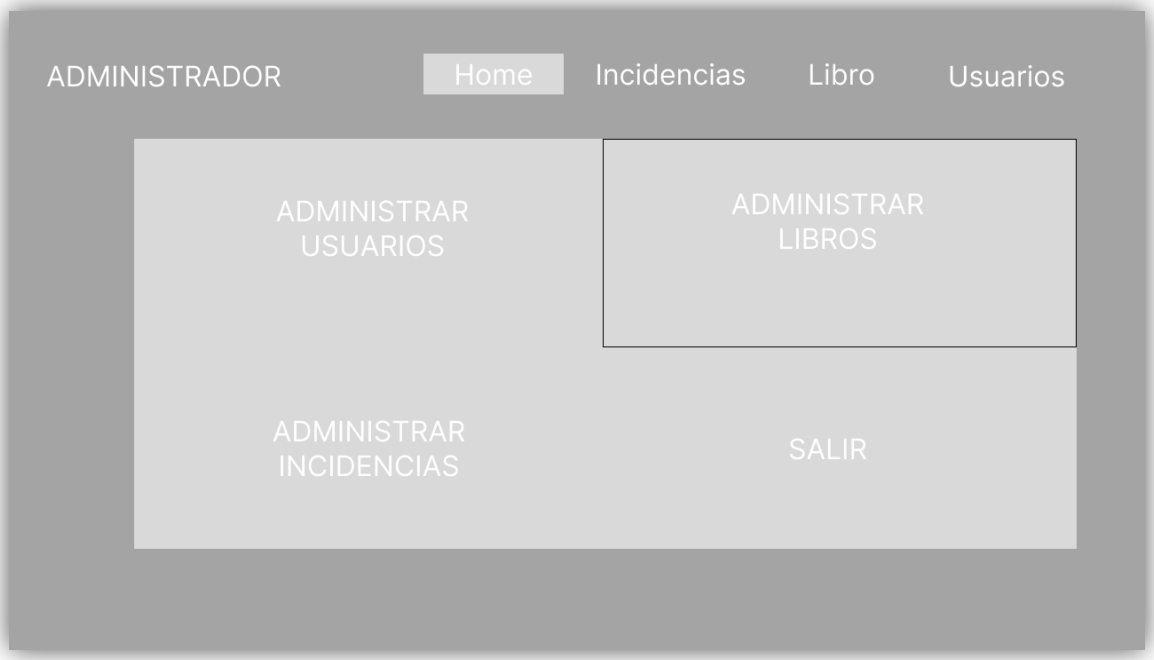
1. **Boceto** : este es la primera actualización del proyecto en el que he elaborado un boceto básico (está sujeto a cambios) en el que muestro las siguientes ventanas:

- Ventana de Inicio de sesión

- Ventana de creación de usuario

- Interfaz principal de Usuarios estándar.

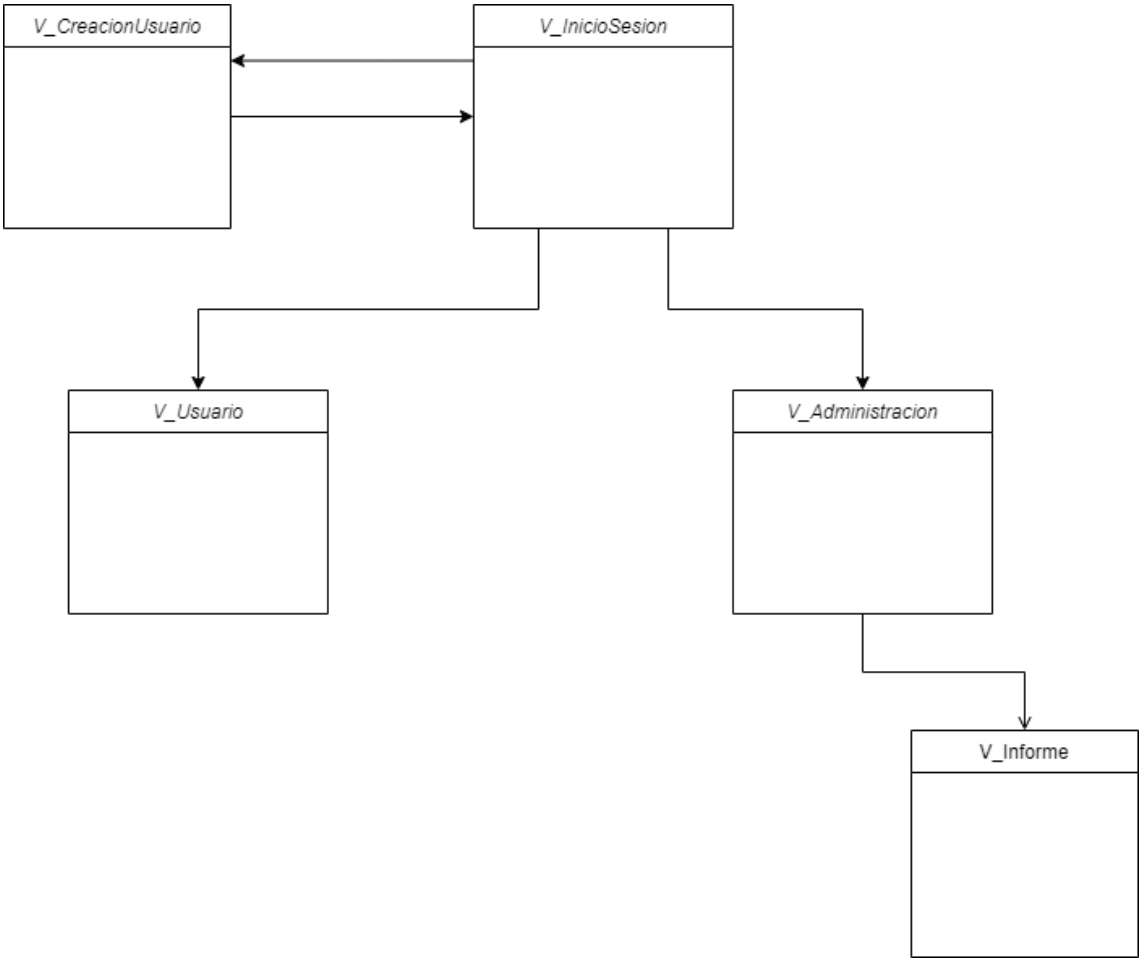
- Interfaz usuario estándar donde podemos visualizar los libros para poder pedir prestados.



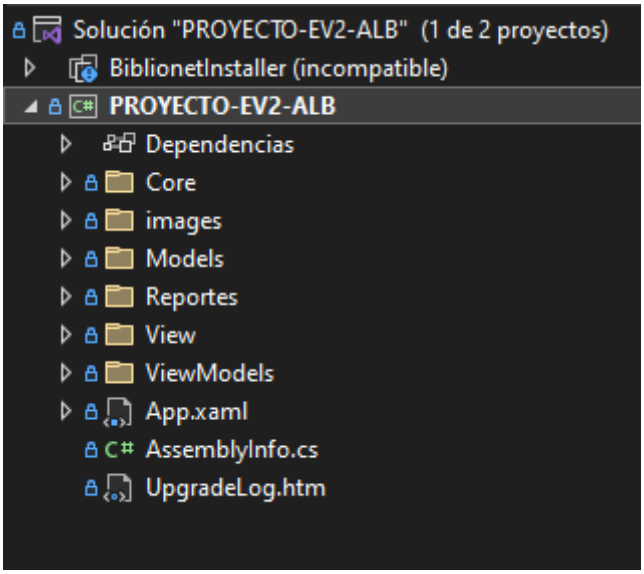
- Ventana de usuarios de tipo “Administrador” donde podemos visualizar las principales funcionalidades con este rol.

- Ventana de Administración de Libros en la cual podemos realizar operaciones tales como agregar, modificar y eliminar libros.

2. Estructuración del proyecto:

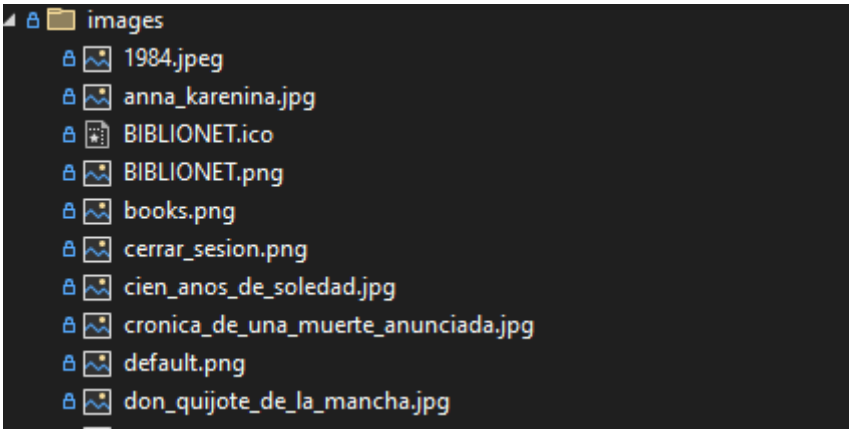


Estas son las ventanas en las cuales operaremos durante todo el flujo de ejecución de la aplicación.

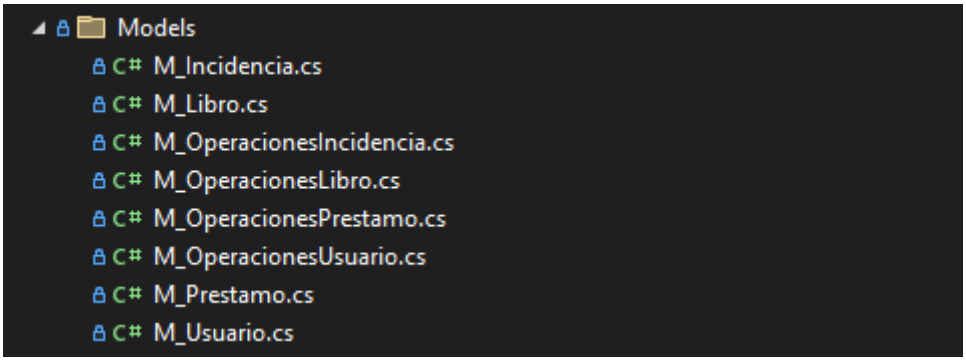


- Este es el cuerpo de la aplicación, en el cual contienen los siguientes componentes:

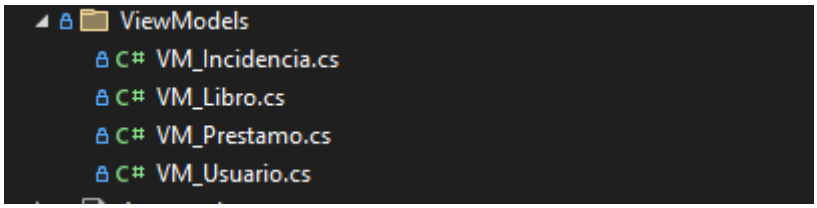
- **Core:** esta carpeta contiene todos los recursos de utilidad tales como:
  - **BBDD_Biblioteca.sql:** este script SQL sirve para generar la base de datos con su estructura y datos.
  - **Comandos.cs:** clase en la cual contiene todos los comandos que luego utilizaremos en las vistas de los botones y atajos de los mismos.
  - **Conexión.cs:** clase (SINGLETON) en la cual contiene todos los métodos y atributos para establecer con el conector MySQL a la base de datos.
  - **Utils.cs:** clase en la cual contiene métodos tales como conversión de imágenes BitMapImage a un array de Bytes[] y viceversa para el almacenado de imágenes en la base de datos.



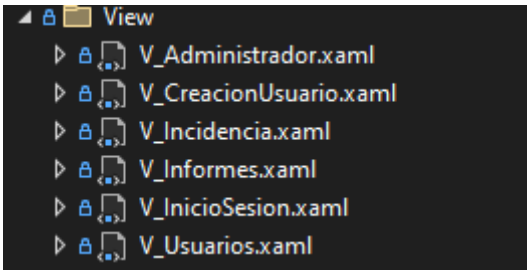
- **Images:** esta carpeta contiene todas las imágenes de las portadas de los libros, iconos, etc.



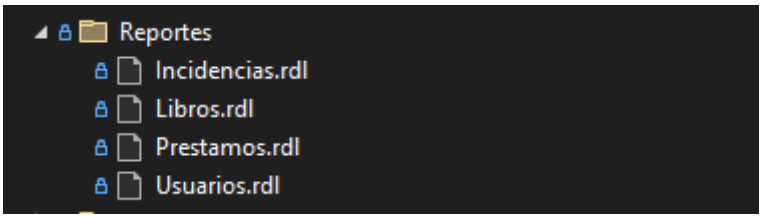
- **Models:** esta carpeta contiene todas las clases en las que guardaremos los datos de las tablas en objetos y todas las operaciones CRUD de dichos objetos.



- **ViewModels:** esta carpeta contiene todas las clases en las que se ven involucradas las lógicas de negocio y la comunicación entre las vistas y los modelos.



- **View:** esta carpeta engloba todas las clases relacionadas con la interfaz de usuario, diseñadas para la interacción visual de las ventanas.



- **Reportes:** en esta carpeta, se almacenan todos los documentos con extensión .rdl, los cuales despliegan informes detallados generados a partir de las diversas funcionalidades de la aplicación.


```
<Application.Resources>
    <Style x:Key="RoundedTextBoxStyle" TargetType="TextBox">
        <Setter Property="FontSize" Value="20"/>

        <Setter Property="Height" Value="25"/>
        <Setter Property="VerticalContentAlignment" Value="Center"/>
        <Setter Property="Template">
            <Setter.Value>
                <ControlTemplate TargetType="{x:Type TextBox}">
                    <Border CornerRadius="5" VerticalAlignment="Center" BorderThickness="1" BorderBrush="Black" Backg
                        <ScrollViewer x:Name="PART_ContentHost"/>
                    </Border>
                    <ControlTemplate.Triggers>
                        <Trigger Property="IsMouseOver" Value="True">
                            <Setter  Property="Effect" >
                                <Setter.Value>
                                    <DropShadowEffect BlurRadius="10" Color="Aquamarine" Opacity="0.5" ShadowDepth="0
```

- **App.xaml:** En esta clase, se han centralizado todos los estilos que impactan en los controles de la aplicación, abarcando elementos como botones, cuadros de texto, y más.

3. Patrón MVVM

- Este proyecto se ha estructurado empleando el patrón MVVM (Modelo-Vista-ViewModel) con el objetivo de lograr una abstracción eficiente tanto de la información como de la interacción del usuario con la aplicación.
- A continuación mostraré un ejemplo de interacción entre las diferentes capas de la interfaz de usuario (estándar).

V_Usuarios

```
4 referencias
public partial class V_Usuarios : Window
{

    private VM_Libro vm_libro;
    private VM_Usuario vm_usuario;
    private VM_Prestamo vm_prestamo;
    private VM_Incidencia vm_incidencia;
    private ObservableCollection<M_Prestamo> listaPrestamos;
    private M_Usuario usuarioSesion;
```

Atributos de la clase V_Usuarios

- Esta es la clase V_Usuarios la cual hace la función de vista para que el usuario pueda comunicarse con la aplicación.

```
1 referencia
public V_Usuarios(M_Usuario usuarioSesion)
{
    InitializeComponent();
    //Instanciamos los viewmodels y la lista de prestamos
    vm_libro = new VM_Libro();
    vm_incidencia = new VM_Incidencia();
    vm_prestamo = new VM_Prestamo();
    vm_usuario = new VM_Usuario();
    listaPrestamos = new ObservableCollection<M_Prestamo>();
    this.usuarioSesion = usuarioSesion;

    //Asignamos el nombre de usuario a la ventana de usuario
    tbUsuario.Text = usuarioSesion.Nombre;

    actualizarListas();
}
```

- En el constructor de la clase le pasaremos un objeto de tipo M_Usuario (modelo del usuario) e inicializamos todos los ViewModels relacionado con las funcionalidades que tiene el usuario estándar en la aplicación.

```
//Evento para limpiar los campos de la incidencia
1 referencia
private void limpiarCampos()...

//Evento para filtrar los libros, si el texto está vacío se muestran todos los libros
1 referencia
private void tbxBuscar_TextChanged(object sender, TextChangedEventArgs e)...

//Evento para cerrar la ventana creando un mensaje de confirmación
2 referencias
private void Window_Closing(object sender, System.ComponentModel.CancelEventArgs e)...


Eventos para actualizar los ListBox

Eventos botones

Navegacion entre pestañas

Comandos
}
```

Métodos de la clase V_Usuarios

- También esta clase tendrá todos los métodos necesarios para poder intercambiar y visualizar información con el ViewModel afectado.

VM_Usuario

```
namespace PROYECTO_EV2_ALB.ViewModels
{
    12 referencias
    public class VM_Usuario
    {


        private Models.M_OperacionesUsuario operacionesUsuario ;
        private ObservableCollection<Models.M_Usuario> listaUsuarios;

        6 referencias
        public VM_Usuario()
        {
            operacionesUsuario = new M_OperacionesUsuario();


            actualizarLista();
        }

        //Actualizamos la lista de usuarios
        1 referencia
        public void actualizarLista()...


        //Metodos Get y Set
        1 referencia
        public ObservableCollection<M_Usuario> ListaUsuarios...

        Metodos CRUD


        Validaciones
    }
}
```

- En la clase VM_Usuario será el intermediario, el medio de conexión que manejara la lógica de negocio entre el modelo(datos) y las entradas del usuario.
- Obtendrá métodos de validación para las entradas del usuario y los métodos CRUD para obtener información o actualizarla.

M_Usuario

```
39 referencias
public class M_Usuario

{



    private int id_usuario;
    private string nombre;
    private string contrasena;
    private string email;
    private string tipo_usuario;
    private Boolean bloqueado;
    private Boolean prestamo_activo;

    5 referencias
    public M_Usuario()
    {

    }

    0 referencias
    public M_Usuario(int id_usuario, string nombre, string contrasena, string tipo_usuario, string email, bool bloqueado,bool pr


    8 referencias
    public int Id_usuario { get => id_usuario; set => id_usuario = value; }
    14 referencias
    public string Nombre { get => nombre; set => nombre = value; }
    7 referencias
    public string Contrasena { get => contrasena; set => contrasena = value; }
    5 referencias
    public string Email { get => email; set => email = value; }
    8 referencias
    public string Tipo_usuario { get => tipo_usuario; set => tipo_usuario = value; }
    6 referencias
    public bool Bloqueado { get => bloqueado; set => bloqueado = value;}
    7 referencias
    public bool Prestamo_activo { get => prestamo_activo; set => prestamo_activo = value; }
```

- En esta clase, hemos implementado un objeto Usuario siguiendo el paradigma de programación orientada a objetos (POO). Este objeto POCO (Plain Old CLR Object) se utiliza para recuperar información de las tablas y almacenarla de manera estructurada.

M_OperacionesUsuario

```
3 referencias
public class M_OperacionesUsuario
{
    public ObservableCollection<M_Usuario> listaUsuarios;

    1 referencia
    public M_OperacionesUsuario()
    {
        listaUsuarios = new ObservableCollection<M_Usuario>();
    }

    1 referencia
    public void insertarUsuario(M_Usuario usuarioNuevo)...

    1 referencia
    public ObservableCollection<M_Usuario> obtenerUsuarios()...

    1 referencia
    public void desbloquearUsuario(string nombre)...

    1 referencia
    public void bloquearUsuario(string nombre)...

    1 referencia
    public void actualizarUsuario(M_Usuario usuario)...


}
```

- En esta clase, se han centralizado todas las operaciones CRUD (Crear, Leer, Actualizar, Eliminar) que establecen la comunicación con la base de datos real.

4. Conclusiones

¿Con qué dificultades nos hemos encontrado desde el proyecto inicial?

- La comunicación entre las capas y la actualización en tiempo real de los datos.
- Diseño de las interfaces.

¿Hemos tenido aplicar algún cambio de la idea inicial? ¿Por qué?

- Sí, añadido de funcionalidad en el apartado de administración y reestructuración de los objetos préstamos para que pudiera albergar en él otros objetos.
- Mejora visual de la interfaz para aprovechar más el espacio y sea más atractiva al ojo humano.

¿Hemos cumplido los objetivos especificados inicialmente?

- Sí, incluso he añadido más funcionalidades de las planeadas en un principio.

¿Hay alguna propuesta de mejora que se podría plantear para el futuro?

- Más mejoras en la interfaz visual.
- Realización de informes con alguna herramienta más sencilla y efectiva.