En este archivo voy a realizar la documentación técnica sobre mi proyecto Centro Deportivo

Documentación Técnica

2n DAM

Alex Martinez Juan

Índice

1. Introducción 2
1.1. Tecnologías utilizadas 2
2. Arquitectura MVVM 2
2.1. Model (Modelo) 2
2.2. View (Vista) 3
2.3. ViewModel 3
3. Diagrama de capas 4
4. Estructura del proyecto 4
4.1. CentroDeportivo.Model 4
4.2. CentroDeportivo.ViewModel 5
4.3. CentroDeportivo.View 6
4.4. CentroDeportivo.Tests 6
4.5. CentroDeportivo.Informes 7
5. Componentes Principales 7
5.1. Models 7
5.1.1. Clase Socios 7
5.1.2. Clase Actividades 7
5.1.3. Clase Reservas 7
5.2. ViewModels 8
5.2.1. BaseViewModel 8
5.2.2. MainWindowViewModel 8
5.2.3. SociosViewModel 8
5.2.4. ActividadesViewModel 9
5.2.5. ReservasViewModel 9
5.2.6. InformesViewModel 10
5.3. Commands 10
5.3.1. RelayCommand 10
5.4. Acceso a datos 10
5.4.1. Services y Interfaces 11
5.4.2. Entity Framework Context 11
6. Diagrama de base de datos 11
6.1. Esquema de tablas 11
6.2. Relaciones 12

1. Introducción

Centro Deportivo es una aplicación de escritorio desarrollada en **WPF** utilizando **.NET Framework 4.8** que implementa el patrón arquitectónico **MVVM** (Model-View-ViewModel). La aplicación permite gestionar socios, actividades y reservas de un centro deportivo, además de generar informes mediante **Crystal Reports** y realizar testing con **MSTest**.

1.1. Tecnologías utilizadas

- **WPF** (Windows Presentation Foundation)
- **.NET Framework 4.8**
- **Entity Framework 6** (para acceso a datos)
- **Crystal Reports** (para generación de informes)
- **MSTest** (para pruebas unitarias y de integración)

2. Arquitectura MVVM

El patrón **MVVM** divide la aplicación en tres capas claramente diferenciadas que garantizan la separación de responsabilidades y facilitan el mantenimiento y escalabilidad del código.

2.1. Model (Modelo)

Responsabilidad: Representa los datos y reglas de negocio de la aplicación.

- Clases POCO generadas por Entity Framework.
- No contienen lógica de presentación.

Clases principales:

- Socios: Representa un socio del centro deportivo.
- Actividades: Representa una actividad disponible.
- Reservas: Representa la relación entre socios y actividades. Es decir cuando un socio reserva una actividad.

2.2. View (Vista)

Responsabilidad: Define la interfaz gráfica de usuario en XAML.

- Sin lógica de negocio en el code-behind.
- Utiliza **Data Binding** para conectarse con el ViewModel.
- Los eventos se gestionan mediante **Commands**.

Vistas principales:

- MainWindow.xaml: Ventana principal con navegación.
- SociosView.xaml: Gestión de socios.
- ActividadesView.xaml: Gestión de actividades.
- ReservasView.xaml: Gestión de reservas.
- InformesView.xaml: Selección de informes.

2.3. ViewModel

Responsabilidad: Intermediario entre Model y View.

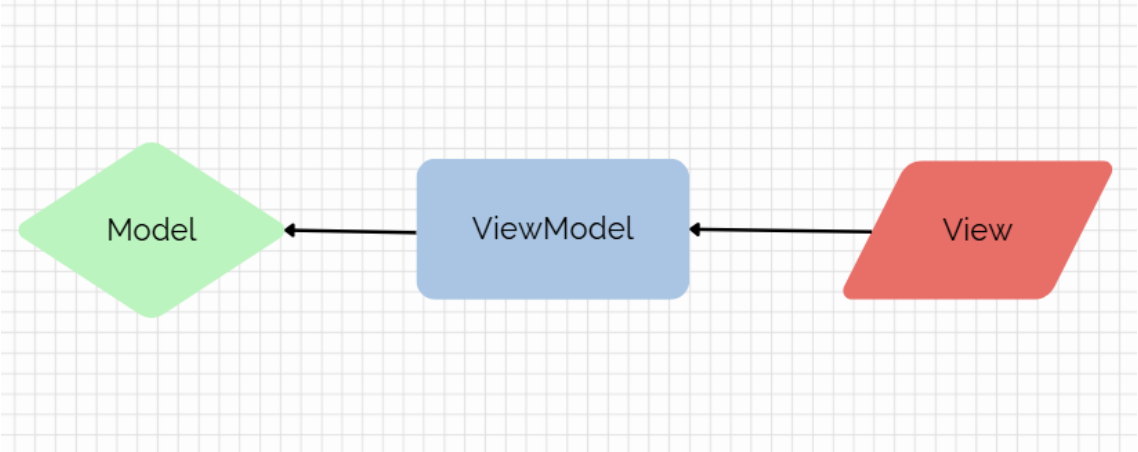
- Implementa **INotifyPropertyChanged** para notificar cambios.
- Contiene **Commands** para responder a acciones del usuario.
- Expone colecciones **ObservableCollection<T>** para listas dinámicas.
- Valida datos antes de interactuar con los servicios.

ViewModels principales:

- MainWindowViewModel: Control de navegación.
- SociosViewModel: Lógica de gestión de socios-
- ActividadesViewModel: Lógica de gestión de actividades.
- ReservasViewModel: Lógica de gestión de reservas.
- InformesViewModel: Lógica de generación de informes.

3. Diagrama de capas

El diagrama de capas es el siguiente:



4. Estructura del proyecto

El proyecto está organizado en **3 proyectos principales (Model, ViewModel y View)** y **2 proyectos secundarios (Informes y Tests)**:

4.1. CentroDeportivo.Model

Descripción: Capa de datos y entidades del dominio.

Contenido:

- Socios.cs: Entidad de socios.
- Actividades.cs: Entidad de actividades.
- Reservas.cs: Entidad de reservas.
- CentroDeportivoEntities: DbContext de Entity Framework.

Responsabilidades:

- Definir el modelo de datos.
- Mapear tablas de la base de datos.

4.2. CentroDeportivo.ViewModel

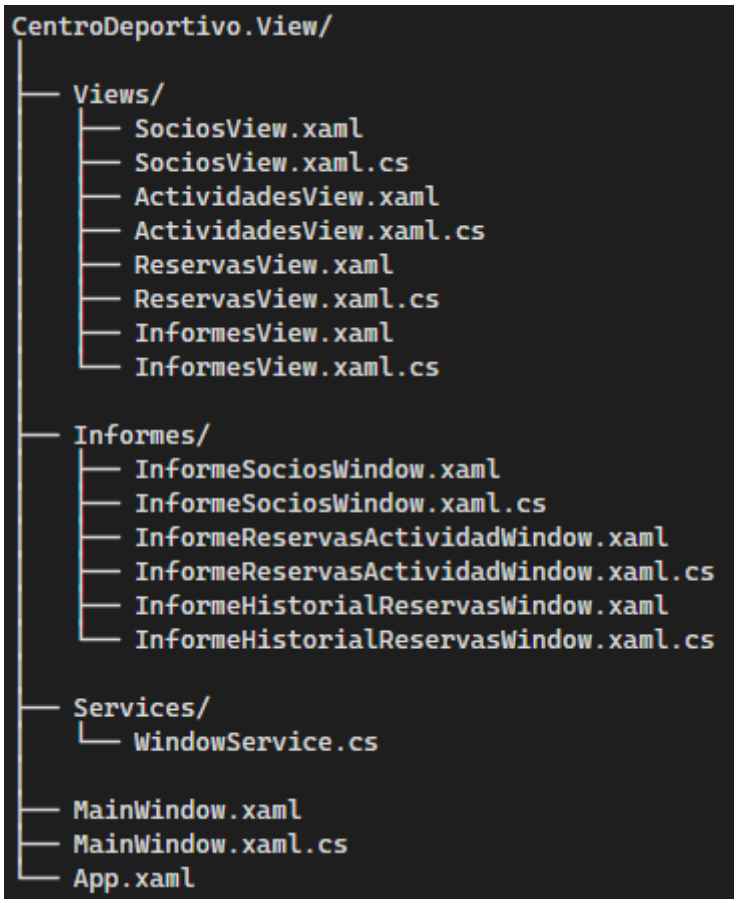
Descripción: Capa de lógica de presentación.

Estructura de carpetas:

4.3. CentroDeportivo.View

Descripción: Capa de presentación (interfaz gráfica).

Estructura de carpetas:



4.4. CentroDeportivo.Tests

Descripción: Proyecto de pruebas unitarias y de integración.

Contenido:

- EmailTests.cs: Valida formato de emails.
- FechaReservaTests.cs: Valida fechas de reservas.
- AforoTests.cs: Valida control de aforo

4.5. CentroDeportivo.Informes

Descripción: Proyecto de informes a través de Crystal Reports.

Contenido:

- crSociosMaestro.rpt: Informe con listado de socios.
- crReservasActividad.rpt: Informe sobre reservas por actividad con % ocupación.
- crHistorialReservas.rpt: Informe de reservas por socio (agrupados por socio).

5. Componentes Principales

5.1. Models

Los **Models** son las entidades que representan los datos del dominio. Estas clases son generadas automáticamente por **Entity Framework** a partir del esquema de la base de datos (Database First).

5.1.1. Clase Socios

Características:

- Propiedades de navegación hacia Reservas.
- Validación de unicidad del email en base de datos.
- Restricción: no se puede eliminar si tiene reservas asociadas.

5.1.2. Clase Actividades

Características:

- Control de aforo máximo.
- Propiedad de navegación hacia Reservas.
- Restricción: no se puede eliminar si tiene reservas asociadas.

5.1.3. Clase Reservas

Características:

- Relación muchos a muchos entre Socios y Actividades.
- Validación de aforo disponible antes de crear/editar.
- Validación de fechas (no se permiten fechas pasadas).

5.2. ViewModels

Los **ViewModels** son la capa intermedia entre la vista y el modelo. Gestionan la lógica de presentación y exponen datos a través de propiedades enlazadas.

5.2.1. BaseViewModel

Clase base que implementa INotifyPropertyChanged. Todos los ViewModels heredan de esta clase.

Propósito:

- Centralizar la implementación de **INotifyPropertyChanged.**
- Evitar código duplicado.
- Garantizar notificación de cambios a la vista.

5.2.2. MainWindowViewModel

Controla la navegación entre vistas.

Propiedades principales:

- VistaActual: Vista actualmente visible.
- Titulo: Título dinámico de la ventana

Commands:

- MostrarSociosCommand
- MostrarActividadesCommand
- MostrarReservasCommand
- MostrarInformesCommand

5.2.3. SociosViewModel

Gestiona la lógica de la vista de socios.

Propiedades principales:

- Socios: ObservableCollection<Socios> - Lista de socios.
- SocioSeleccionado: Socio seleccionado en el DataGrid.
- Nombre, Email, Activo: Propiedades del formulario.
- MensajeError: Mensajes de validación.

Commands:

- CrearCommand: Crea un nuevo socio
- EditarCommand: Edita el socio seleccionado

- EliminarCommand: Elimina el socio seleccionado

Validaciones:

- Email válido mediante expresión regular
- Email único en la base de datos
- Campos obligatorios

5.2.4. ActividadesViewModel

Gestiona la lógica de la vista de actividades.

Propiedades principales:

- Actividades: ObservableCollection<Actividades>
- ActividadSeleccionada: Actividad seleccionada
- Nombre, AforoMaximo: Propiedades del formulario

Validaciones:

- Nombre no vacío
- Aforo máximo mayor que 0
- No se puede eliminar si tiene reservas

5.2.5. ReservasViewModel

Gestiona la lógica de la vista de reservas.

Propiedades principales:

- Reservas: ObservableCollection<Reservas>
- Socios: ObservableCollection<Socios> (solo activos)
- Actividades: ObservableCollection<Actividades>
- ReservaSeleccionada: Reserva seleccionada
- SocioSeleccionado, ActividadSeleccionada, Fecha

Validaciones:

- Fecha no puede ser anterior a hoy
- Validación de aforo disponible antes de crear/editar

5.2.6. InformesViewModel

Gestiona la generación de informes.

Propiedades:

- Actividades: Lista para seleccionar actividad en el informe
- ActividadSeleccionada: Actividad para filtrar informe

Commands:

- InformeSociosCommand: Genera informe maestro de socios
- InformeReservasActividadCommand: Genera informe de reservas por actividad
- InformeHistorialCommand: Genera historial de reservas

5.3. Commands

Los **Commands** permiten desacoplar la lógica de negocio de los eventos de la interfaz gráfica. En lugar de usar eventos como **Click**, se utilizan comandos que pueden ser ejecutados y deshabilitados dinámicamente.

5.3.1. RelayCommand

Implementación genérica de ICommand.

Características:

- _execute: Acción a ejecutar
- _canExecute: Condición para habilitar/deshabilitar el comando
- CanExecuteChanged: Evento que reevalúa automáticamente el estado del comando

5.4. Acceso a datos

La capa de acceso a datos se implementa mediante el patrón Repository utilizando **servicios** que encapsulan las operaciones CRUD.

5.4.1. Services y Interfaces

SocioService, ReservaService, ReservaService y sus interfaces.

Características importantes:

- Uso de using para gestionar correctamente el contexto
- Manejo de excepciones con mensajes descriptivos
- Detección específica de errores de unicidad y claves foráneas
- En ellas se realizan las operaciones CRUD

5.4.2. Entity Framework Context

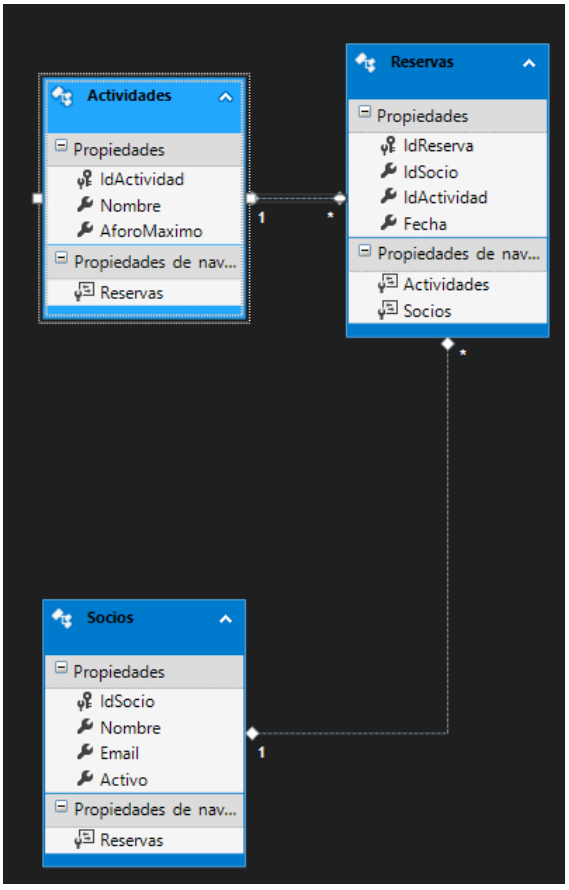
El contexto de Entity Framework (**CentroDeportivoEntities**) gestiona la conexión con la base de datos y el mapeo objeto-relacional.

Características:

- Generado automáticamente desde la base de datos (Database First)
- Connection string configurado en App.config

6. Diagrama de base de datos

6.1. Esquema de tablas



6.2. Relaciones

- **SOCIOS → RESERVAS (1:N)**
  - Un socio puede tener múltiples reservas
  - Una reserva pertenece a un solo socio

- **ACTIVIDADES → RESERVAS (1:N)**
  - Una actividad puede tener múltiples reservas
  - Una reserva pertenece a una sola actividad

- **SOCIOS ↔ ACTIVIDADES (N:M a través de RESERVAS)**
  - Relación muchos a muchos implementada mediante tabla intermedia