En este archivo voy a realizar la documentación técnica sobre mi proyecto Sistema de Gestión de Expedientes para Despacho de Abogados

Documentación Técnica

2n DAM

Alex Martinez Juan

Índice

1. Introducción

Sistema de Gestión de Despacho de Abogados Lexenda 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 clientes, expedientes jurídicos, actuaciones, citas y generar informes mediante **Crystal Reports**, además de 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, reglas de negocio y acceso a datos de la aplicación.

- Clases POCO generadas por Entity Framework.
- Capa de **Repositories** para acceso a datos.
- Capa de **Services** para lógica de negocio y validaciones.
- No contienen lógica de presentación.

Clases principales:

- **Clientes:** Representa un cliente del despacho.
- **Expedientes:** Representa un expediente jurídico.
- **Actuaciones:** Representa las actuaciones realizadas en un expediente.
- **Citas:** Representa citas asociadas a clientes o expedientes.
- **EstadosExpediente:** Catálogo de estados posibles de un expediente.

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 mediante menú superior.
- **ClientesView.xaml**: Gestión de clientes.
- **ExpedientesView.xaml:** Gestión de expedientes jurídicos.
- **ActuacionesView.xaml:** Gestión de actuaciones por expediente.
- **CitasView.xaml:** Gestión de agenda de citas.
- **InformesView.xaml:** Visualización de informes Crystal Reports.

2.3. ViewModel

Responsabilidad: Intermediario entre Model y View.

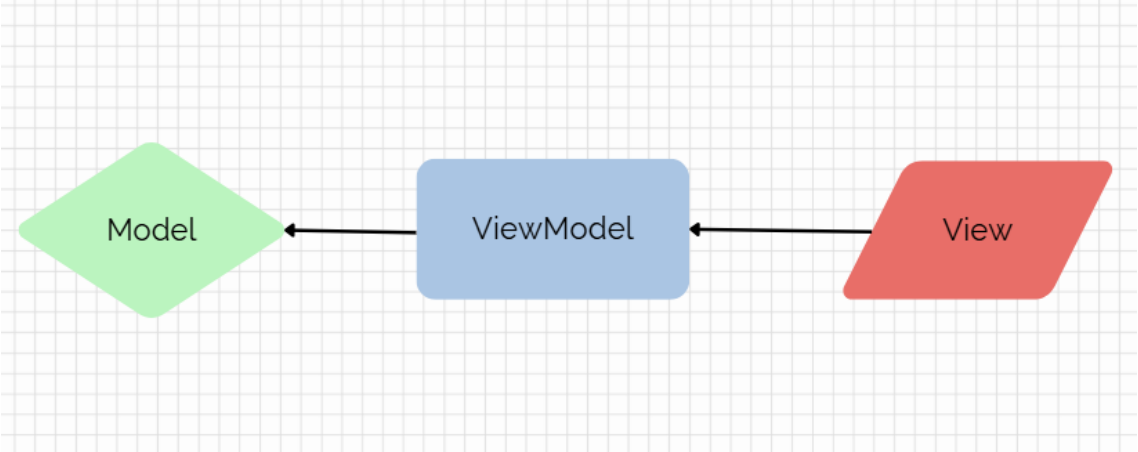
- Implementa **INotifyPropertyChanged** para notificar cambios.
- Contiene **Commands** para responder a acciones del usuario.
- Expone colecciones **ObservableCollection<T>** para listas dinámicas.
- Valida datos básicos antes de delegar en los servicios.

ViewModels principales:

- **MainWindowViewModel:** Control de navegación entre vistas.
- **ClientesViewModel:** Lógica de gestión de clientes.
- **ExpedientesViewModel:** Lógica de gestión de expedientes.
- **ActuacionesViewModel:** Lógica de gestión de actuaciones.
- **CitasViewModel:** Lógica de gestión de citas.
- **InformesViewModel:** Lógica de generación y visualización 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. SistemaGestionDespacho.Model

Descripción: Capa de datos, entidades del dominio, repositorios y servicios de negocio.

Contenido:

Entidades (POCO generadas por EF):

- **Clientes.cs:** Entidad de clientes.
- **Expedientes.cs:** Entidad de expedientes.
- **Actuaciones.cs:** Entidad de actuaciones.
- **Citas.cs:** Entidad de citas.
- **EstadosExpediente.cs:** Catálogo de estados**.**

Repositories:

- **ClienteRepository.cs:** Operaciones CRUD sobre clientes.
- **ExpedienteRepository.cs:** Operaciones CRUD sobre expedientes.
- **ActuacionRepository.cs:** Operaciones CRUD sobre actuaciones.
- **CitaRepository.cs:** Operaciones CRUD sobre citas.
- **EstadoExpedienteRepository.cs:** Consulta de estados.

Services:

- **ClienteService.cs:** Validaciones y lógica de negocio de clientes.
- **ExpedienteService.cs:** Validaciones y generación de códigos de expedientes.
- **ActuacionService.cs:** Validaciones de actuaciones.
- **CitaService.cs:** Validaciones de citas y detección de conflictos.
- **EstadoExpedienteService.cs:** Consulta de estados.

DataSets para Informes:

- **dsInformes.xsd**: DataSet tipado para Crystal Reports.
- **InformeClientesHelper.cs:** Helper para cargar datos de clientes.
- **InformeExpedientesPorEstadoHelper.cs:** Helper para expedientes agrupados por estado.
- **InformeActuacionesHelper.cs:** Helper para actuaciones por expediente.

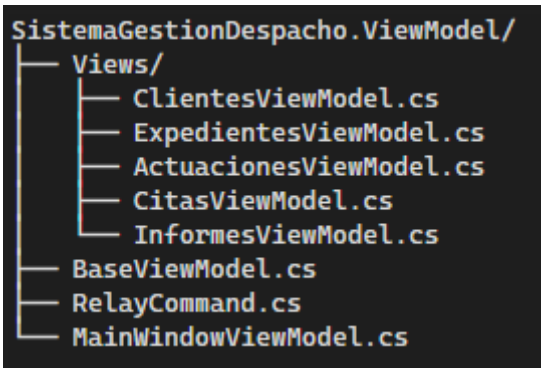
Responsabilidades:

- Definir el modelo de datos mediante Entity Framework.
- Implementar operaciones CRUD mediante repositorios.
- Centralizar validaciones de negocio en los servicios.
- Preparar datos para informes.

4.2. SistemaGestionDespacho.ViewModel

Descripción: Capa de lógica de presentación.

Estructura de carpetas:



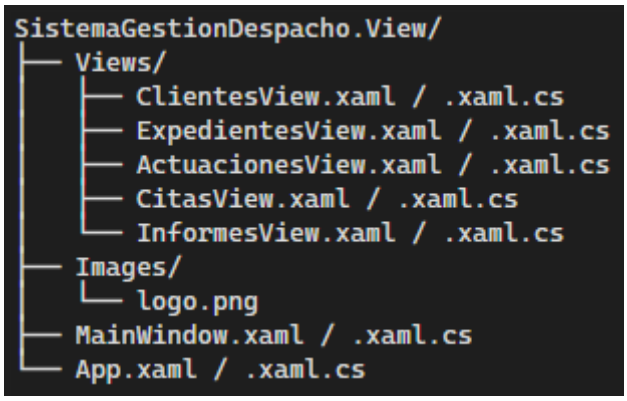
Responsabilidades:

- Gestionar la navegación entre vistas.
- Exponer propiedades enlazables para las vistas.
- Implementar comandos para acciones del usuario.
- Invocar servicios de la capa Model.
- Notificar cambios mediante INotifyPropertyChanged.

4.3. SistemaGestionDespacho.View

Descripción: Capa de presentación (interfaz gráfica).

Estructura de carpetas:



Responsabilidades:

- Definir la interfaz de usuario en XAML.
- Aplicar estilos y plantillas visuales.
- Conectar mediante Data Binding con ViewModels.
- Mínima lógica en code-behind (solo inicialización).

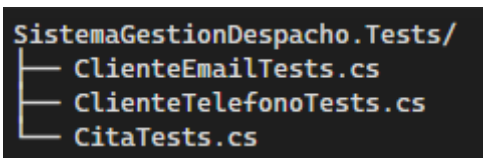
4.4. SistemaGestionDespacho.Tests

Descripción: Proyecto de pruebas unitarias y de integración.

Contenido:

- **ClienteEmailTests.cs:** Valida formato de emails.
- **ClienteTelefonoTests.cs:** Valida formato de teléfonos españoles.
- **CitaTests.cs:** Valida detección de conflictos de citas.

Estructura de carpetas:



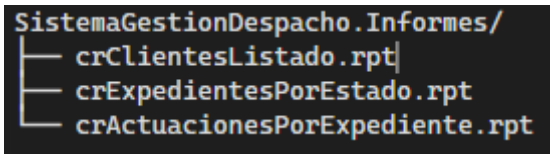
4.5. SistemaGestionDespacho.Informes

Descripción: Proyecto de informes a través de Crystal Reports.

Contenido:

- **crClientesListado.rpt:** Listado maestro de clientes.
- **crExpedientesPorEstado.rpt:** Expedientes agrupados por estado.
- **crActuacionesPorExpediente.rpt:** Actuaciones por expediente.

Estructura de carpetas:



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 Clientes

Características:

- Propiedades de navegación hacia Expedientes y Citas.
- Validación de unicidad del DNI/CIF en base de datos.
- Validación de formato de email y teléfono español.
- Restricción: no se puede desactivar si tiene expedientes abiertos o en curso.

5.1.2. Clase Expedientes

Características:

- Generación automática de código único (EXP-{timestamp}-{guid}).
- Relación con Clientes y EstadosExpediente.
- Propiedades de navegación hacia Actuaciones y Citas.
- Validación: solo se puede cerrar si tiene actuaciones registradas.
- Restricción: no se puede crear/editar para clientes desactivados.

5.1.3. Clase Actuaciones

Características:

- Relación muchos-a-uno con Expedientes.
- Validación de tipos permitidos (Llamada, Reunión, Escrito presentado, etc.).
- Restricción: no se pueden crear en expedientes cerrados.

5.1.4. Clase Citas

Características:

- Relación opcional con Clientes y Expedientes.
- Validación de conflictos de fecha/hora (no se permiten citas simultáneas).
- Estados: Pendiente, Realizada, Cancelada.
- Restricción: no se pueden crear para clientes desactivados.

5.1.5. Clase EstadosExpediente

Características:

- Catálogo de estados: Abierto, En curso, Archivado, Cerrado.
- Relación uno-a-muchos con Expedientes.

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 mediante **OnPropertyChanged(propertyName).**

5.2.2. MainWindowViewModel

Controla la navegación entre vistas.

Propiedades principales:

- VistaActual: Vista actualmente visible (BaseViewModel).
- Titulo: Título dinámico de la ventana.
- Propiedades booleanas para marcar el menú activo (EsClientes, EsExpedientes…)

Commands:

- MostrarClientesCommand
- MostrarExpedientesCommand
- MostrarActuacionesCommand
- MostrarCitasCommand
- MostrarInformesCommand

Funcionalidades adicionales:

- Métodos **AbrirActuaciones(expedienteId)** y **AbrirCitas(expedienteId)** para navegación contextual desde expedientes

5.2.3. ClientesViewModel

Gestiona la lógica de la vista de clientes.

Propiedades principales:

- Clientes: ObservableCollection<Clientes> - Lista de clientes.
- ClienteSeleccionado: Cliente seleccionado en el DataGrid.
- Nombre, Apellidos, DNI_CIF, Telefono, Email, Direccion, Activo: Propiedades del formulario.
- MensajeError: Mensajes de validación.
- FiltroNombre, FiltroDNI, FiltroActivos: Propiedades de filtrado.

Commands:

- CrearCommand: Crea un nuevo cliente
- EditarCommand: Edita el cliente seleccionado
- EliminarCommand: Desactiva el cliente seleccionado (soft delete)
- LimpiarCommand: Limpia el formulario
- BuscarCommand: Aplica filtros de búsqueda

Validaciones delegadas al Service:

- Email válido mediante expresión regular
- Teléfono español válido (9 dígitos, comienza por 6-9)
- DNI español válido (8 dígitos + letra)
- DNI único en la base de datos
- No desactivar clientes con expedientes abiertos/en curso

5.2.4. ExpedientesViewModel

Gestiona la lógica de la vista de expedientes.

Propiedades principales:

- Expedientes: ObservableCollection<Expedientes>.
- ExpedienteSeleccionado: Expediente seleccionado.
- ClienteFormulario, TipoFormulario, EstadoFormulario, Titulo, Descripcion, FechaApertura.
- Propiedades de filtro: FiltroCliente, FiltroTipo, FiltroEstado, FiltroDesde, FiltroHasta.

Commands:

- CrearCommand: Crea un nuevo expediente
- EditarCommand: Edita el expediente seleccionado
- CerrarCommand: Cierra el expediente (requiere actuaciones registradas)
- LimpiarCommand: Limpia el formulario
- BuscarCommand: Aplica filtros de búsqueda
- LimpiarFiltrosCommand: Limpia los filtros aplicados
- VerActuacionesCommand: Navega a actuaciones del expediente
- VerCitasCommand: Navega a citas del expediente

Validaciones delegadas al Service:

- Campos obligatorios (título, descripción, tipo, cliente, estado).
- Cliente debe estar activo.
- Solo se puede cerrar si tiene actuaciones.
- Generación automática de código único.

5.2.5. ActuacionesViewModel

Gestiona la lógica de la vista de actuaciones.

Propiedades principales:

- Actuaciones: ObservableCollection<Actuaciones>.
- ActuacionSeleccionada: Actuación seleccionada.
- ExpedienteSeleccionado, Tipo, Fecha, Descripcion.
- Propiedades de filtro: FiltroExpediente, FiltroTipo, FiltroFecha.

Commands:

- CrearCommand: Crea una nueva actuación
- EditarCommand: Edita la actuación seleccionada
- EliminarCommand: Elimina la actuación (hard delete)
- LimpiarCommand: Limpia el formulario
- BuscarCommand: Aplica filtros de búsqueda
- LimpiarFiltrosCommand: Limpia los filtros aplicados

Validaciones delegadas al Service:

- Expediente obligatorio y existente.
- No se pueden crear actuaciones en expedientes cerrados.
- Tipo debe ser válido (lista predefinida).
- Fecha obligatoria.
- Descripción obligatoria.

Constructor especial:

- Permite instanciar con expedienteId para filtrar actuaciones de un expediente específico (navegación desde ExpedientesView).

5.2.6. CitasViewModel

Gestiona la lógica de la vista de citas.

Propiedades principales:

- Citas: ObservableCollection<Citas>.
- CitaSeleccionada: Cita seleccionada.
- ClienteFormulario, ExpedienteFormulario, FechaHora, Hora, Lugar, Motivo, Estado.
- ClienteEnabled, ExpedienteEnabled: Controlan habilitación de combos según origen.
- Propiedades de filtro: FiltroFecha, FiltroEstado, FiltroCliente, FiltroExpediente.

Commands:

- CrearCommand: Crea una nueva cita
- EditarCommand: Edita la cita seleccionada
- EliminarCommand: Cancela la cita
- LimpiarCommand: Limpia el formulario
- BuscarCommand: Aplica filtros de búsqueda
- LimpiarFiltrosCommand: Limpia los filtros aplicados

Validaciones delegadas al Service:

- Debe seleccionarse cliente o expediente.
- Fecha y hora obligatorias.
- Lugar y motivo obligatorios.
- Estado válido (Pendiente, Realizada, Cancelada).
- Cliente debe estar activo.
- No se permiten citas en la misma fecha/hora (excepto canceladas).

Constructor especial:

- Permite instanciar con expedienteId para crear citas asociadas a un expediente específico.

5.2.7. InformesViewModel

Gestiona la generación y visualización de informes Crystal Reports.

Propiedades:

- InformeActual: ReportDocument - Informe actualmente cargado en el visor.

Commands:

- MostrarInformeClientesCommand: Genera informe de listado de clientes
- MostrarInformeExpedientesCommand: Genera informe de expedientes por estado
- MostrarInformeActuacionesCommand: Genera informe de actuaciones por expediente

Funcionamiento:

- Utiliza clases Helper para cargar datos en dsInformes.
- Crea instancias de Crystal Reports (.rpt).
- Asigna el DataSet como origen de datos.
- Actualiza la propiedad InformeActual para refrescar el visor.

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

Ventajas:

- Reutilizable en todos los ViewModels.
- Permite habilitar/deshabilitar botones dinámicamente según el estado de la aplicación.
- Simplifica el código XAML mediante Command="{Binding NombreCommand}".

5.4. Acceso a datos

La capa de acceso a datos se implementa mediante el patrón **Repository** y **Service Layer**, separando las operaciones de base de datos de la lógica de negocio.

5.4.1. Repositorios

Los repositorios encapsulan las operaciones CRUD directas sobre la base de datos utilizando Entity Framework.

Características:

- Uso de **using** para gestionar correctamente el contexto de Entity Framework.
- Métodos típicos: **GetAll(), GetById(id), Add(entity), Update(entity), Delete(entity)**.
- Uso de **AsNoTracking()** para consultas de solo lectura y mejorar rendimiento.
- Actualización de entidades mediante **Find()** para evitar problemas de tracking en **Update().**

5.4.2. Services

Los servicios contienen la lógica de negocio y validaciones antes de delegar operaciones a los repositorios.

Características:

- Inyección manual de repositorios en el constructor.
- Métodos de validación privados (ValidarCliente, ValidarActuacion, etc.).
- Manejo de excepciones con mensajes descriptivos.
- Validaciones específicas del dominio (unicidad, referencias, estados, formatos).

5.4.3. Entity Framework Context

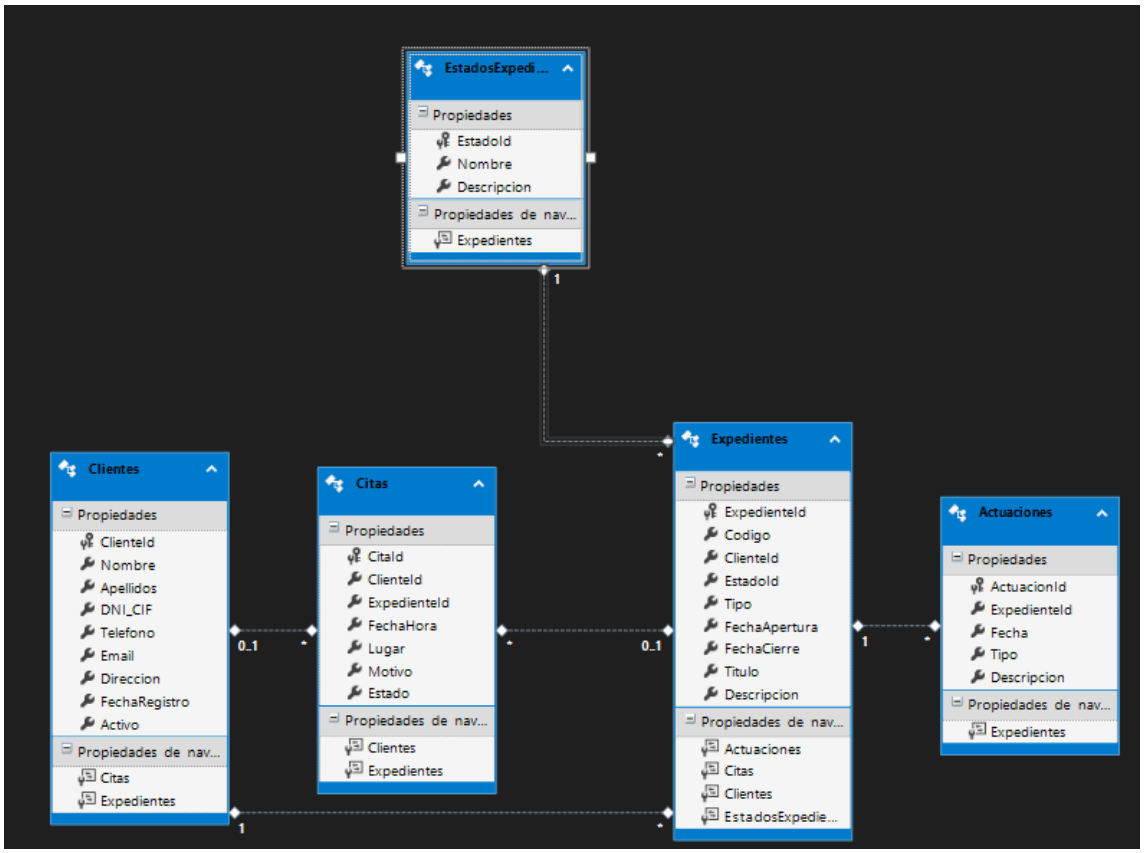
El contexto de Entity Framework (**SistemaGestionDespachoEntities**) gestiona la conexión con la base de datos y el mapeo objeto-relacional.

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

- **CLIENTES → EXPEDIENTES (1:N)**
  - Un cliente puede tener múltiples expedientes
  - Un expediente pertenece a un solo cliente
  - Restricción: No se puede desactivar un cliente con expedientes abiertos/en curso

- **CLIENTES → CITAS (1:N)**
  - Un cliente puede tener múltiples citas
  - Una cita puede pertenecer opcionalmente a un cliente
  - Restricción: No se pueden crear citas para clientes desactivados

- **ESTADOSEXPEDIENTE → EXPEDIENTES (1:N)**
  - Un estado puede aplicarse a múltiples expedientes
  - Un expediente tiene un solo estado
  - Estados predefinidos: Abierto, En curso, Archivado, Cerrado

- **EXPEDIENTES → ACTUACIONES (1:N)**
  - Un expediente puede tener múltiples actuaciones
  - Una actuación pertenece a un solo expediente
  - Restricción: No se pueden crear actuaciones en expedientes cerrados
  - Restricción: Un expediente solo se puede cerrar si tiene actuaciones

- **EXPEDIENTES → CITAS (1:N)**
  - Un expediente puede tener múltiples citas
  - Una cita puede pertenecer opcionalmente a un expediente
  - Relación opcional para permitir citas independientes de expedientes