En primer lugar mostrar que vamos a conseguir.

Desarrollaremos una app que acceda a una colección de películas y las muestre mediante maestro detalle. Crearemos un app de escritorio con Microsoft Framework .Net 4.8 , C# como lenguaje y WPF como tecnología de vistas.

1. Abrir Visual Studio. Configurar vs. Plugins
2. Navegar por Visual Studio.
   * explorador de soluciones
   * explorador de servidores
   * que son los proyectos
   * menú compilar
3. ¿Qué vamos a desarrollar? Una APP de escritorio de películas usando inyección de dependencias y arquitectura limpia
   * Que es inyección de dependencias
   * Que es arquitectura limpia. Separación de responsabilidades
   * Simularemos un acceso a una api de películas
   * Usaremos MVVM e interfaz gráfica WPF. ¿que es WPF?
   * clonar la app desde github con una rama final
     1. Navegar por las carpetas del proyecto.
     2. Como está distribuido un proyecto de wpf
        1. Propiedades del proyecto
        2. Properties
        3. Referencias
        4. App.xaml
        5. App.CS
        6. Mainwindow.xaml
4. Crear proyecto vacio .net Framework WPF.
   * Añadir referencia a NEWTONJSON. Explicar que hará esta biblioteca
   * Crear carpetas Domain, Data, Core, Usecases, View. Esta será la arquitectura limpia. ######## INICIAREMOS LA ARQUITECTURA DE DENTRO HACIA FUERA ###########
5. Fase 1. Crear clases de dominio. Modelbase y Movie y lógica si la hubiese
   * Crear clase Modelbase. Implementa INotifyPropertyChanged. Crear método OnPropertyChanged
   * Crear clase Movie que herede de Modelbase
   * Crear propiedades para: (escribir prop TAB TAB para propiedades simples y propfull TAB TAB para propiedades completas). Añadir a las propiedades completas en el set llamada al método OnPropertyChanged(nameof(propiedad))
     1. string Title (completa)
     2. int Year (completa)
     3. string Rated;
     4. DateTime Released;
     5. double Runtimeminutes (completa)
     6. string Genre;
     7. string Director;
     8. string Writer;
     9. string Actors (completa)
     10. string Plot;
     11. string Language;
     12. string Country;
     13. string Awards;
     14. string Poster;
     15. string Metascore;
     16. string ImdbRating1;
     17. string ImdbVotes1;
     18. string ImdbID1;
     19. string Type;
     20. string Response;
   * Crear propiedades de solo lectura.
     1. public string Duration => TimeTools.GetTimeHM(RuntimeMinutes \* 60);
     2. public ImageSource PosterImage => Tools.GetImageFromPath($"{App.MyPath}\\{Poster}");
6. Fase 2. Crear clases e interfaces de la capa USESCASES O APLICACION.
   * Interfaces en carpeta ABSTRACTIONS
     1. Interface IMovieAPI. Añadir los métodos:
        1. IEnumerable<Movie> GetAllMovies();
        2. IEnumerable<Movie> MapperMovies();
     2. Interface IMovieRepository. Añadir métodos
        1. IEnumerable<Movie> GetMovies();
        2. void AddMovie(Movie movie);
        3. void DeleteMovie(Movie movie);
        4. void UpdateMovie(Movie movie);
        5. void SaveChanges();
   * Crear clase de caso de uso GetMoviesUseCase
     1. Crear constructor con argumento interface IMovieRepository (escribiendo ctor TAB TAB). Al ser el argumento una interface que ya hemos creado, podemos implementar sus métodos. Crear campo, ALT + Entrar sobre el argumento, generar campo
     2. Crear métodos. Cada uno de los métodos devuelve una colección de objetos ordenados o filtrados.
        1. public ObservableCollection<Movie> GetMoviesByYear()
        2. public ObservableCollection<Movie> GetMoviesByTitle()
        3. public ObservableCollection<Movie> GetMoviesByDuration()
        4. public ObservableCollection<Movie> GetMoviesPrevious2008()
7. Fase 3. Crear clases de la capa DATA o INFRASTRUCTURE
   * Clase MovieAPi. Implementa interfaz IMovieAPI de la capa aplicación
     1. Añadir constructor con la ruta de la base de datos simulando una url. Crear campo, ALT + Entrar sobre el argumento, generar campo.
     2. Añadir la deserializacion de json al método
   * Clase MovieRepository. Implementa la interfaz IMovieRepository de la capa APLICACIÓN
     1. Añadir constructor con argumento la interfaz de la api IMovieAPI Crear campo, ALT + Entrar sobre el argumento, generar campo
     2. Añadir el código a los métodos implementados
8. Fase 4. Creación de las clase Viewmodel
   * Clase BaseViewmodel.
     1. Implementa la interfaz INotifyPropertyChanged.
     2. Contiene propiedades y métodos de mensajes de usuario
   * Clase MainViewmodel
     1. Añadir constructor con argumento la clase GetMovieUseCases y crear campo
     2. Añadir propiedad ICommand MovieAction para ejecución de comandos
     3. Añadir propiedad completa string ActionSelected
     4. Añadir propiedad completa Movie MovieSelected
     5. Añadir propiedad completa OrderBy
     6. Añadir propiedad colección ObservableCollection<string> OrderBy. Añadir elementos al campo
     7. Añadir propiedad de solo lectura bool EditIsEnabled
     8. Añadir propiedad de solo lectura ObservableCollection<Movie> Movies
     9. Añadir método privado ExecuteAction(object parameter)
9. Fase 5. Crear inyección de dependencias en la clase de inicio App.cs
   * Crear campo con la ruta de la API (supuesta URL)
   * Crear una propiedad estática MainViewmodel
   * Sobrescribir el método OnStartUp. ALT+ENTRAR Generar invalidaciones
   * Añadir dentro del método:
     1. Instancia de un objeto IMovieAPI pasando como argumento la URL
     2. Instancia de un objeto IMovieRepository pasando como argumento el objeto del tipo IMovieAPI
     3. Instancia de un caso de uso pasando como argumento el objeto instanciado IMovieRepository
     4. Instancia de la propiedad MainViewmodel pasando como argumento el objeto caso de uso. Esta clase será la encargada de alimentar el DataContext de la vista.
10. Fase 6. Crear la vista
    * Crear Grid. Contenedores, StackPanel, DockPanel, etc
    * Crear Datagrid de películas. DataBinding
    * Crear detalle de la película seleccionada
    * Crear ComboBox. Colecciones
    * Crear barra de estado
    * Crear estilos
    * Crear botones y comandos
    * Cambiar un TextBlock por un TextBox y añadir actualización de propiedad para visualizar los cambios directos sobre la colección del datagrid.