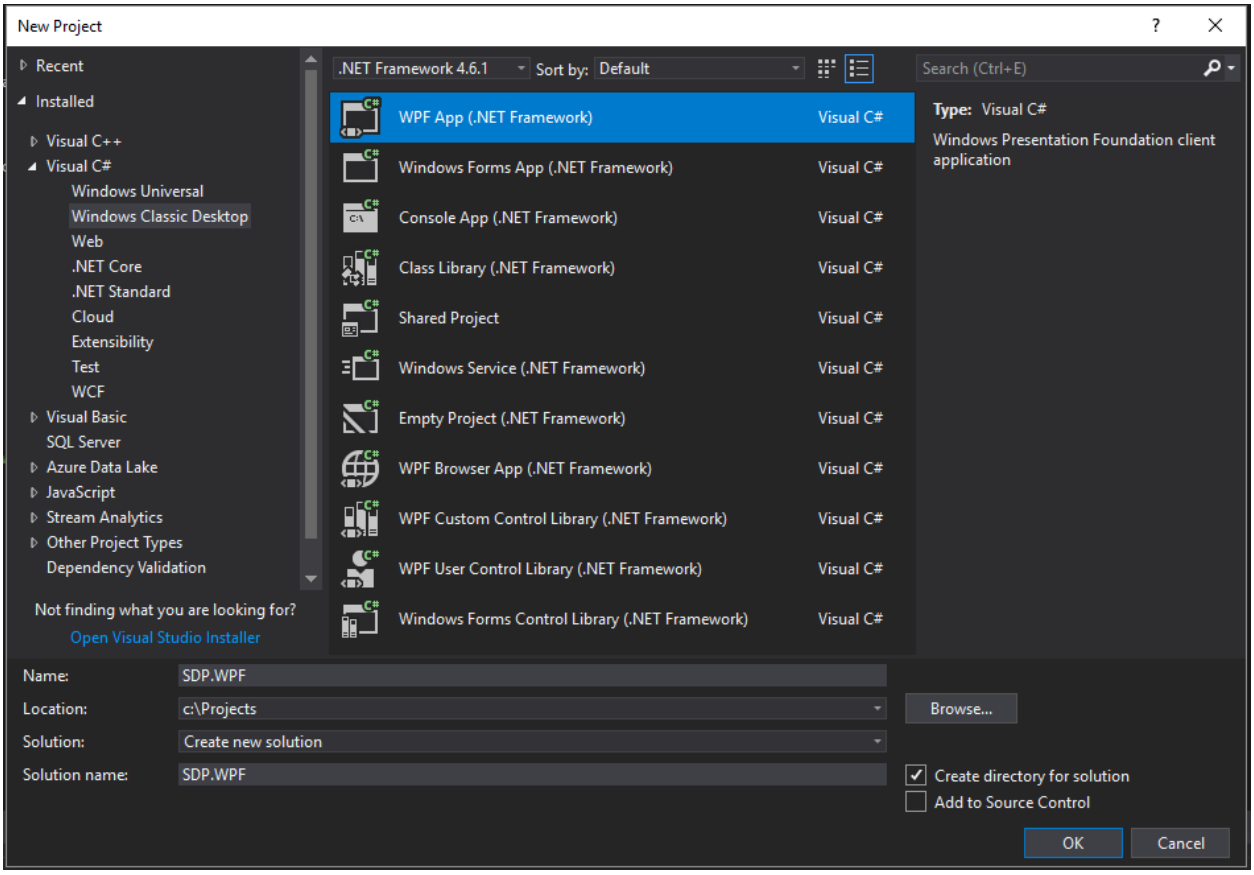
GlobalLogic

Student Development Program
C# WPF

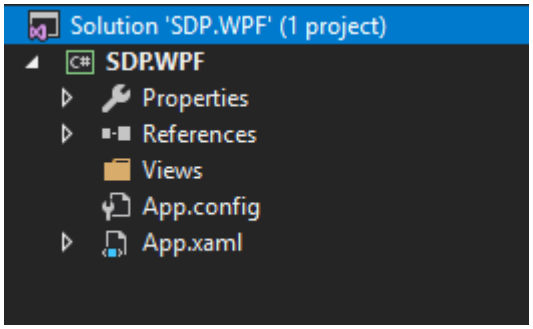
Piotr Wilczak
Krzysztof Krywiak
Paweł Trumiński

1.New WPF Application

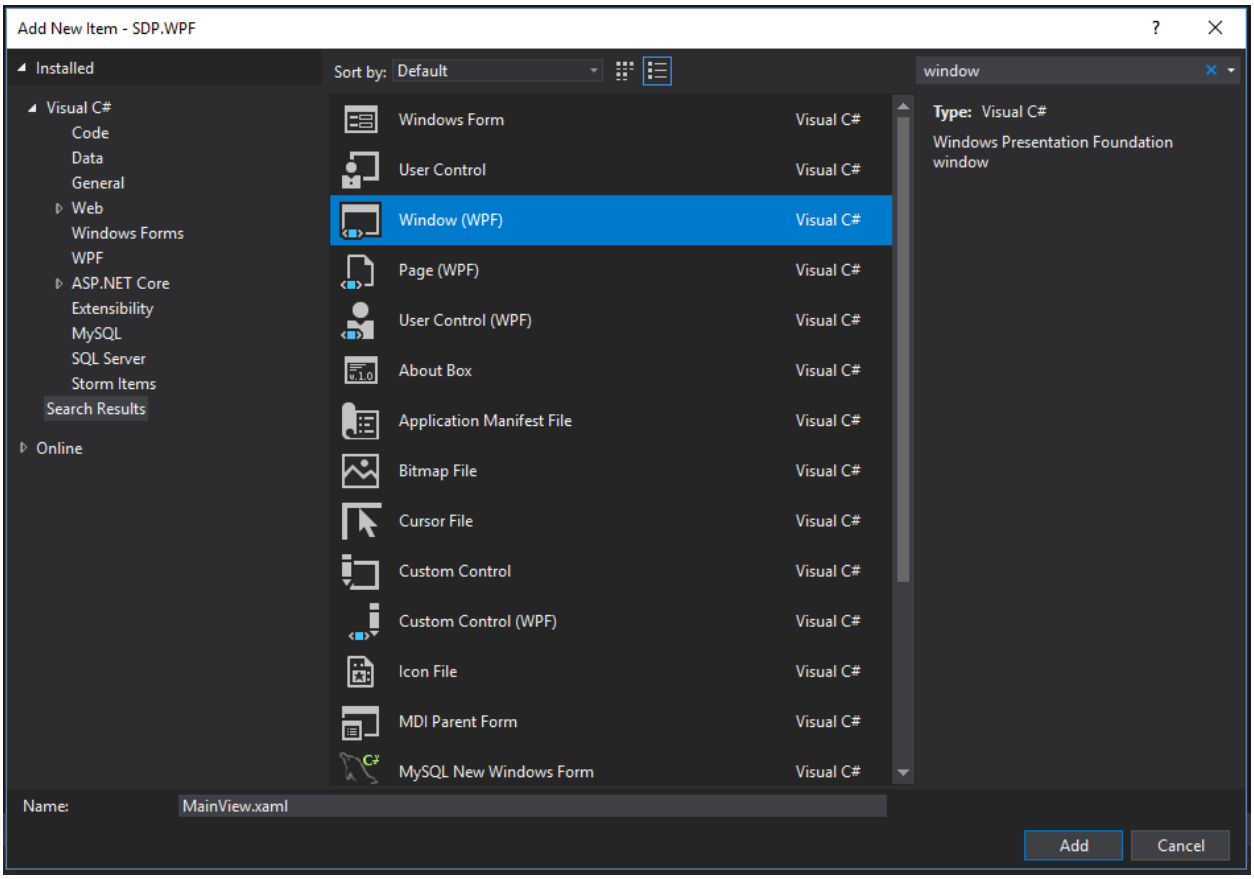


1.1 Delete MainWindow.xaml

1.2 Add folder Views



1.3 Add new file MainView.xaml in folder Views



Now we have to set *MainView.xaml* as default page for our application. *App.xaml.cs* will usually look like this for a new project.

```
namespace SDP.WPF
{
    /// <summary>
    /// Interaction logic for App.xaml
    /// </summary>
    3 references
    public partial class App : Application
    {

    }
}
```

1.3 Add OnStartup method to App.xaml.cs

```
private void OnStartup(object sender, StartupEventArgs e)
{
        Views.MainView view = new Views.MainView();
        view.Show();
}
```

In App.xaml there is property StartupUri, it indicates the file MainWindow.xaml, which we removed.

```
<Application x:Class="SDP.WPF.App"
                 xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
                 xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
                 xmlns:local="clr-namespace:SDP.WPF"
                 StartupUri="MainWindow.xaml">
<Application.Resources>

    </Application.Resources>
</Application>
```

1.4 Replace StartupUri

```
 Startup="OnStartup">
```

1.5 Add *TextBlock* in *Views/MainView.xaml.*

```
<TextBlock HorizontalAlignment="Center" VerticalAlignment="Center" FontSize="72">
            Hello, WPF!
</TextBlock>
```

2.Displaying data

2.1 Add new folder Models

2.2 Add new file Models/BindableBase.cs

```
public class BindableBase : INotifyPropertyChanged
{
    public event PropertyChangedEventHandler PropertyChanged;
    protected void OnPropertyChanged([CallerMemberName] string property = null)
    {
        PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(property));
    }
}
```

2.3 Add new file Models/Book.cs

```
public class Book : BindableBase
{
    private int _inventoryNumber;
    private string _author;
    private string _title;
    private string _yearPublisher;
    private decimal _price;

    public int InventoryNumber
    {
        get => _inventoryNumber;
        set
        {
            _inventoryNumber = value;
            OnPropertyChanged();
        }
    }

    public string Author
    {
        get => _author;
        set
        {
            _author = value;
            OnPropertyChanged();
        }
    }

    public string Title
    {
        get => _title;
        set
```

```
        {
            _title = value;
            OnPropertyChanged();
        }
    }

    public string YearPublisher
    {
        get => _yearPublisher;
        set
        {
            _yearPublisher = value;
            OnPropertyChanged();
        }
    }

    public decimal Price
    {
        get => _price;
        set
        {
            _price = value;
            OnPropertyChanged();
        }
    }

    public void Clear()
    {
        InventoryNumber = 0;
        Author = string.Empty;
        Title = string.Empty;
        YearPublisher = string.Empty;
        Price = 0;
    }

public Book(int inventoryNumber, string author, string title, string yearPublisher,
decimal price)
    {
        this.InventoryNumber = inventoryNumber;
        this.Author = author;
        this.Title = title;
        this.YearPublisher = yearPublisher;
        this.Price = price;
    }
}
```

2.4 In Views/MainView.xaml in main <Grid> specify the default behavior of rows.

```
<Grid.RowDefinitions>
       <RowDefinition Height="Auto" />
       <RowDefinition Height="*" />
</Grid.RowDefinitions>
```

2.5 Create a <ListView> in Views/MainView.xaml

```
<ListView  Grid.Row="1" BorderBrush="White" ItemsSource="{Binding Path=Books}"
HorizontalAlignment="Stretch">
    <ListView.View>
        <GridView>
            <GridViewColumn Header="Inventory Number"
              DisplayMemberBinding="{Binding Path=InventoryNumber}" />
            <GridViewColumn Header="Author"
              DisplayMemberBinding="{Binding Path=Author}" />
            <GridViewColumn Header="Title"
              DisplayMemberBinding="{Binding Path=Title}" />
            <GridViewColumn Header="Year/Publisher"
              DisplayMemberBinding="{Binding Path=YearPublisher}" />
            <GridViewColumn Header="Price"
              DisplayMemberBinding="{Binding Path=Price}" />
        </GridView>
    </ListView.View>
</ListView >
```

2.6 Run application

2.7 Add <ListView.Resources> to <ListView>

```
<ListView.Resources>
        <Style TargetType="{x:Type GridViewColumnHeader}">
            <Setter Property="FontWeight" Value="Bold" />
            <Setter Property="Foreground" Value="Maroon" />
            <Setter Property="Background" Value="LightGreen" />
        </Style>
</ListView.Resources>
```

2.8 Add folder ViewModels

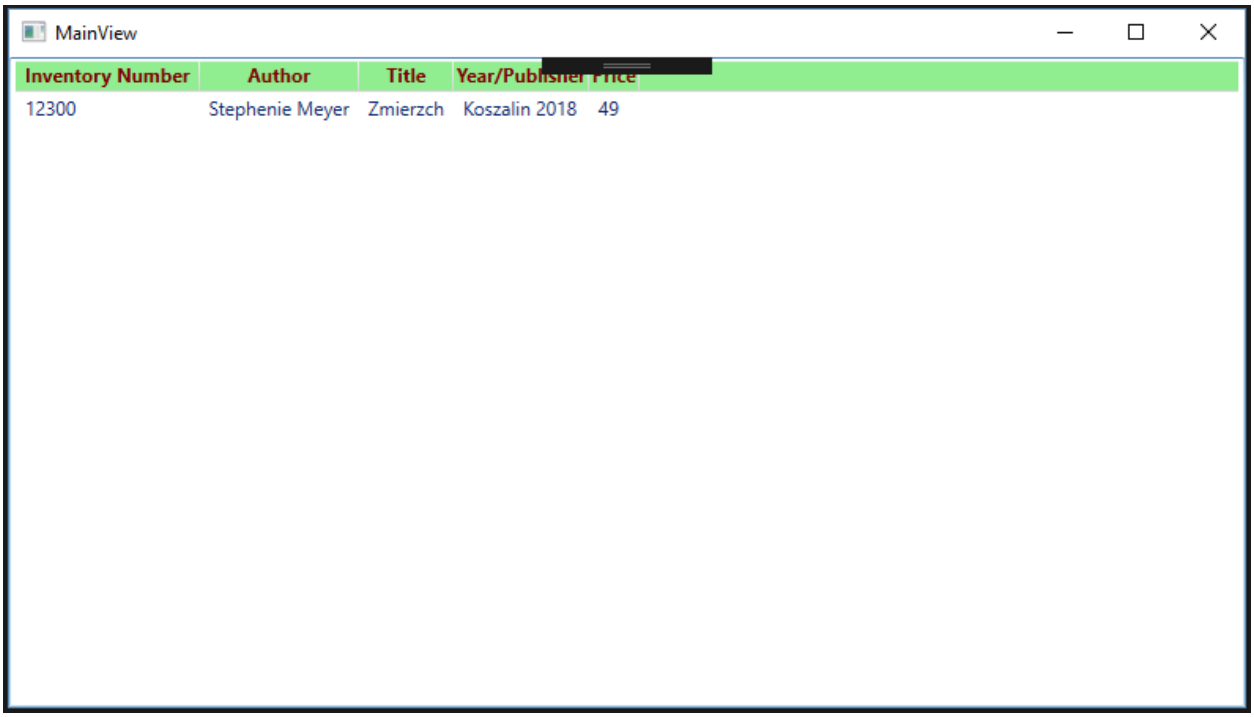
2.9 Add file ViewModels/MainViewModel.cs

```
class MainViewModel
{
    public ObservableCollection<Book> Books
    {
        get;
        set;
    }
    Book exampleBook = new Book(12300,"Stephenie Meyer","Zmierzch", "Koszalin 2018", 49);

    public MainViewModel()
    {
        Books = new ObservableCollection<Book>();
        Books.Add(exampleBook);
    }
}
```

2.10 Now in Views/MainView.xaml.cs specify DataContext.

```
public MainView()
{
    InitializeComponent();
    DataContext = new MainViewModel();
}
```

2.11 Run application.



3. Adding new book

3.1 In file Views/MainView.xaml above *<ListView…>* add section

```
<Grid Grid.Row="0" HorizontalAlignment="Left" Margin="5,25" VerticalAlignment="Center">

</Grid>
```

3.2 *<Grid…>* will contain 11 rows.

```
<Grid.RowDefinitions>
    <RowDefinition Height="Auto" />
    <RowDefinition Height="Auto" />
    <RowDefinition Height="Auto" />
    <RowDefinition Height="Auto" />
    <RowDefinition Height="Auto" />
    <RowDefinition Height="Auto" />
    <RowDefinition Height="Auto" />
    <RowDefinition Height="Auto" />
    <RowDefinition Height="Auto" />
    <RowDefinition Height="Auto" />
    <RowDefinition Height="Auto" />
</Grid.RowDefinitions>
```

3.3 Add textboxes and button

```
<TextBlock Grid.Row="0" Text="Inv. No." />
<TextBox Grid.Row="1" Width="200" Text="{Binding Path=NewBook.InventoryNumber,
Mode=TwoWay, UpdateSourceTrigger=PropertyChanged}" />

<TextBlock Grid.Row="2" Text="Author" />
<TextBox Grid.Row="3" Width="200" Text="{Binding Path=NewBook.Author, Mode=TwoWay,
UpdateSourceTrigger=PropertyChanged}" />

<TextBlock Grid.Row="4" Text="Title" />
<TextBox Grid.Row="5" Width="200" Text="{Binding Path=NewBook.Title, Mode=TwoWay,
UpdateSourceTrigger=PropertyChanged}" />

<TextBlock Grid.Row="6" Text="Year/Publisher" />
<TextBox Grid.Row="7" Width="200" Text="{Binding Path=NewBook.YearPublisher, Mode=TwoWay,
UpdateSourceTrigger=PropertyChanged}" />

<TextBlock Grid.Row="8" Text="Price" />
<TextBox Grid.Row="9" Width="200" Text="{Binding Path=NewBook.Price, Mode=TwoWay,
UpdateSourceTrigger=PropertyChanged}" />

<Button Grid.Row="10" x:Name="btnAddStudent" Content="Add new book" Command="{Binding
AddBook}" />
```

3.4 Run application

3.5 Add some styles

```
<Grid.Resources>
    <Style TargetType="{x:Type TextBlock}">
        <Setter Property="FontWeight" Value="Bold" />
        <Setter Property="HorizontalAlignment" Value="Left" />
    </Style>
</Grid.Resources>
```

3.6 Replace constructor in Models/Book.cs and add a default one

```
public Book(Book book) : this()
{
    InventoryNumber = book.InventoryNumber;
    Author = book.Author;
    Title = book.Title;
    YearPublisher = book.YearPublisher;
    Price = book.Price;
}

public Book()
{

}
```

3.7 Delete *exampleBook* from ViewModels/MainViewModel.cs

3.8 Add in ViewModels/MainViewModel.cs

```
public Book NewBook
{
      get;
      set;
}

public ICommand AddBook
{
      get;
      private set;
}

private void AddNewBook()
{
       Books.Add(new Book(NewBook));
 }
```

3.9 Add Commands folder to project

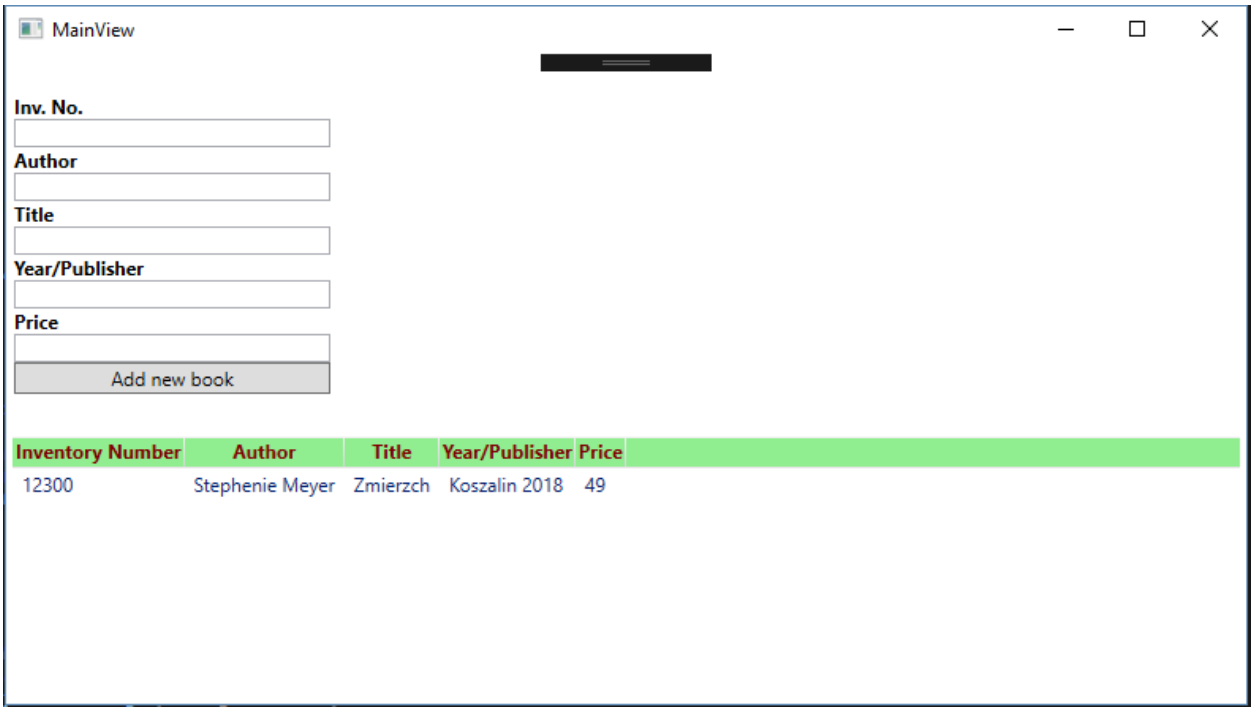
3.10 Add new file Commands/RelayCommand.cs

```
class RelayCommand : ICommand
    {
        public RelayCommand(Action<object> execute) : this(execute, null)
        {
        }
        public RelayCommand(Action<object> execute, Predicate<object> canExecute)
        {
            _execute = execute ?? throw new ArgumentNullException("execute");
            _canExecute = canExecute;
        }
        public bool CanExecute(object parameter)
        {
            return _canExecute == null ? true : _canExecute(parameter);
        }
        public event EventHandler CanExecuteChanged
        {
            add { CommandManager.RequerySuggested += value; }
            remove { CommandManager.RequerySuggested -= value; }
        }
        public void Execute(object parameter)
        {
            _execute(parameter);
        }
        private readonly Action<object> _execute;
        private readonly Predicate<object> _canExecute;
    }
```

3.11 Now in ViewModels/MainViewModel.cs bind it together

```
public MainViewModel()
{
    Books = new ObservableCollection<Book>();
    NewBook = new Book();
    AddBook = new RelayCommand(param => AddNewBook());
 }
```

3.12 Now it is possible to add new book

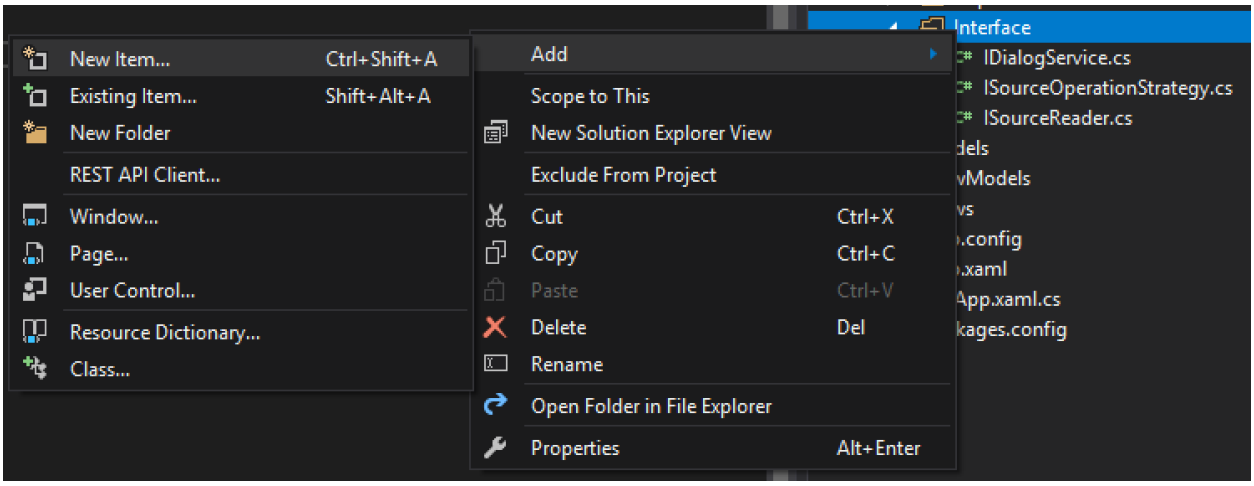


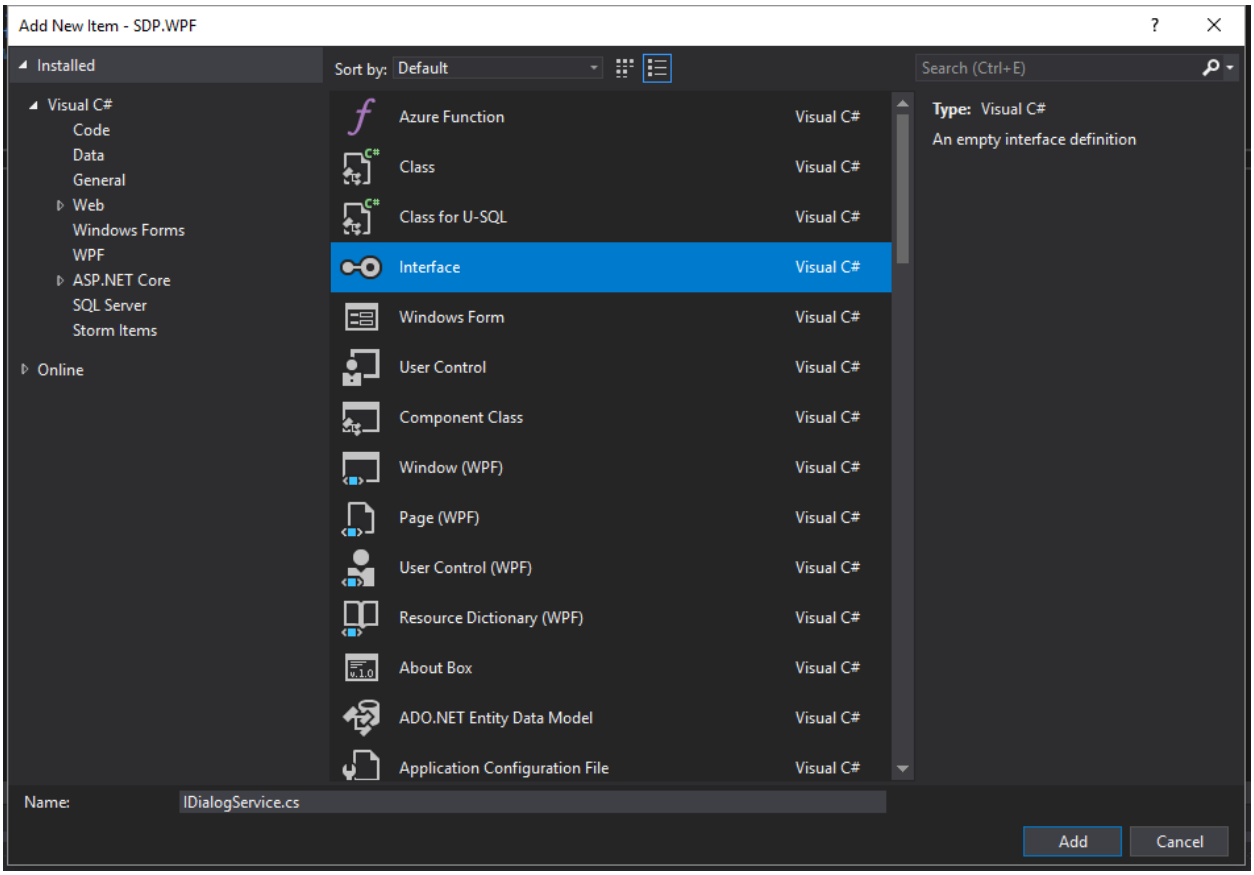
4. Implementing reading from file

4.1 Add new folder DataAccess

4.2 Add new folder DataAccess/Interface

4.3 Add following interfaces





- Interface/IDialogService.cs

```
public interface IDialogService
{
        string OpenFileDialog();
}
```

- Interface/ISourceReader.cs

```
public interface ISourceReader
{
        List<Book> ReadBooks(string filePath);
}
```

Now implement classes which will implement these interfaces

4.4 Add new folder DataAccess/Implementation

4.5 Add following implementations

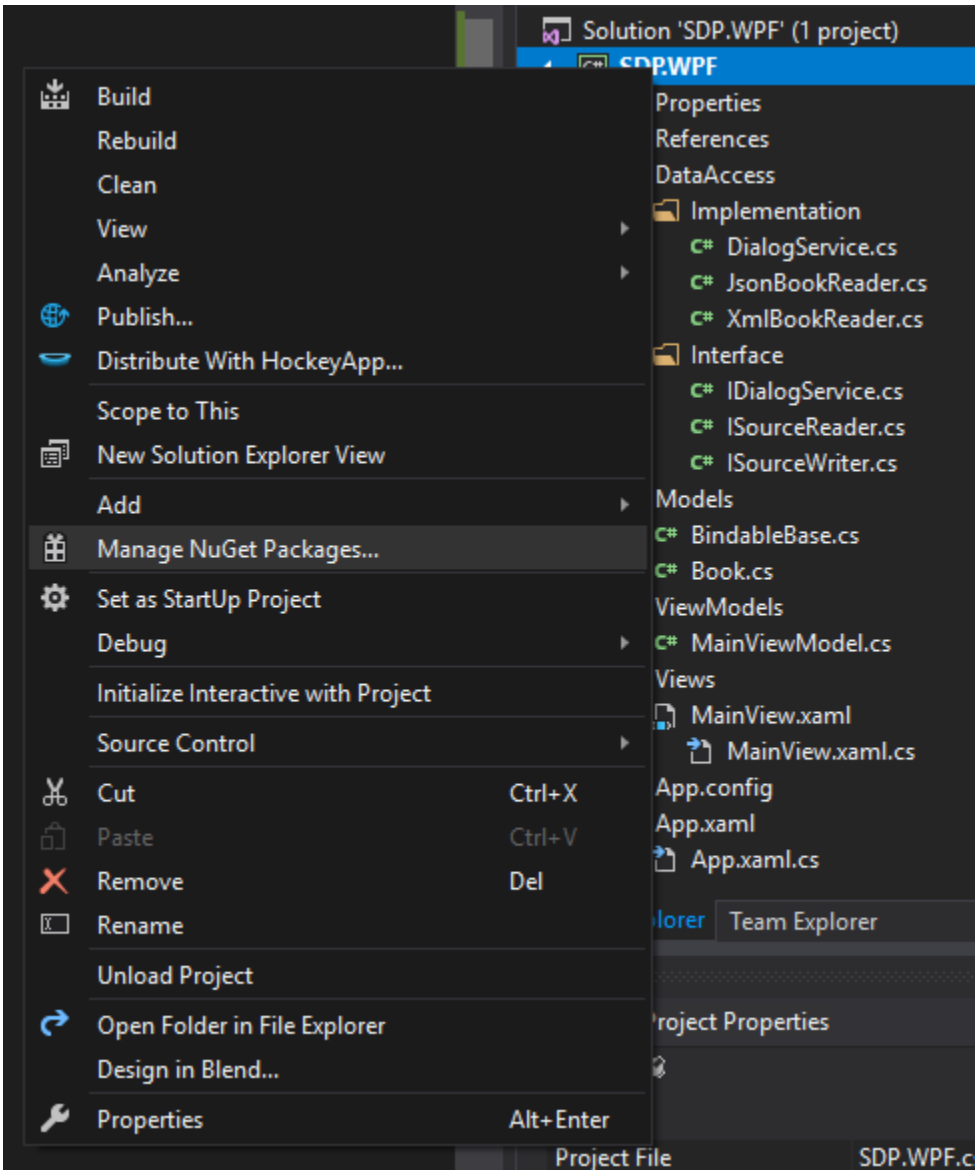
- DialogService.cs

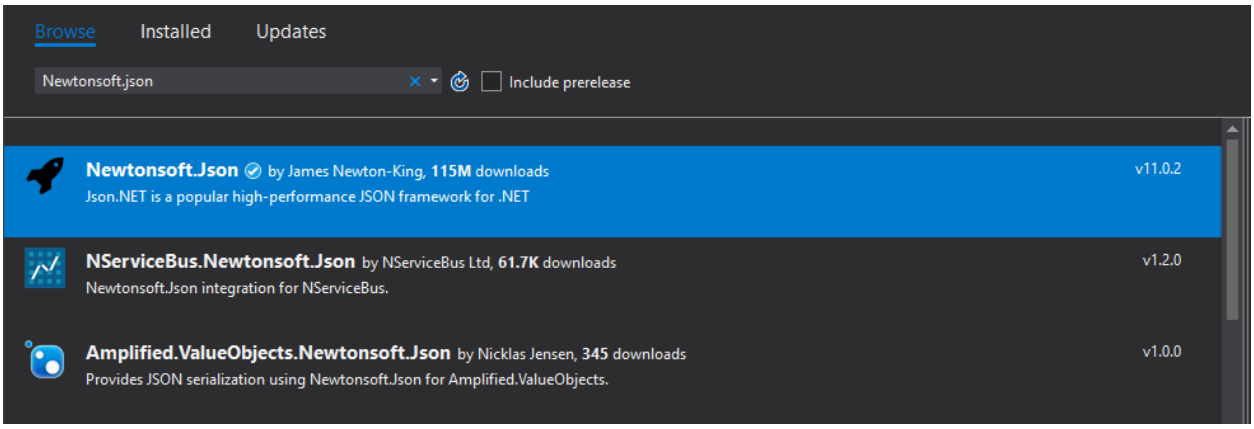
```
public class DialogService : IDialogService
    {
        public string OpenFileDialog()
        {
            var dialog = new OpenFileDialog
            {
                DefaultExt = ".json",
                Filter = "JSON Files (*.json)|*.json|XML Files (*.xml)|*.xml"
            };

            bool? result = dialog.ShowDialog();

            return result == true ? dialog.FileName : null;
        }
    }
```

- JsonBookReader.cs

```
class JsonBookReader : ISourceReader
    {
        public List<Book> ReadBooks(string path)
        {
            string books = File.ReadAllText(path);
   List<Book> BookList = JsonConvert.DeserializeObject<List<Book>>(books);

            return BookList;
        }
    }
```

Now we can see that there is a problem with JsonConvert namespace, we have to add a reference to Newtonsoft.Json.


Solution 'SDP.WPF' (1 project)
SDP.WPF
Build
Rebuild
Clean
View
Analyze
Publish...
Distribute With HockeyApp...
Scope to This
New Solution Explorer View
Add
Manage NuGet Packages...
Set as StartUp Project
Debug
Initialize Interactive with Project
Source Control
Cut Ctrl+X
Paste Ctrl+V
Remove Del
Rename
Unload Project
Open Folder in File Explorer
Design in Blend...
Properties Alt+Enter
Properties
References
DataAccess
Implementation
DialogService.cs
JsonBookReader.cs
XmlBookReader.cs
Interface
IDialogService.cs
ISourceReader.cs
ISourceWriter.cs
Models
BindableBase.cs
Book.cs
ViewModels
MainViewModel.cs
Views
MainView.xaml
MainView.xaml.cs
App.config
App.xaml
App.xaml.cs
Team Explorer
Project Properties
Project File
SDP.WPF.c



4.6 Add following elements in ViewModel/MainViewModel.cs

```
private readonly IDialogService _dialogService;
private readonly JsonBookReader _jsonBookReader;

public ICommand OpenFile
{
        get;
        private set;
}

private void LoadDataFromFile()
{
        var fileName = _dialogService.OpenFileDialog();

        if (fileName != null)
        {
        List<Book> bookList = _jsonBookReader.ReadBooks(fileName);
        Books.Clear();

                foreach (Book book in bookList)
                {
                        Books.Add(book);
                }
         }
}
```

4.7 Update constructor in ViewModels/MainViewModel.cs

```
public MainViewModel(IDialogService dialogService, JsonBookReader jsonBookReader)
{
        _dialogService = dialogService;
        _jsonBookReader = jsonBookReader;

        Books = new ObservableCollection<Book>();
        NewBook = new Book();

        AddBook = new RelayCommand(param => AddNewBook(), null);
        OpenFile = new RelayCommand(param => LoadDataFromFile(), null);
}
```

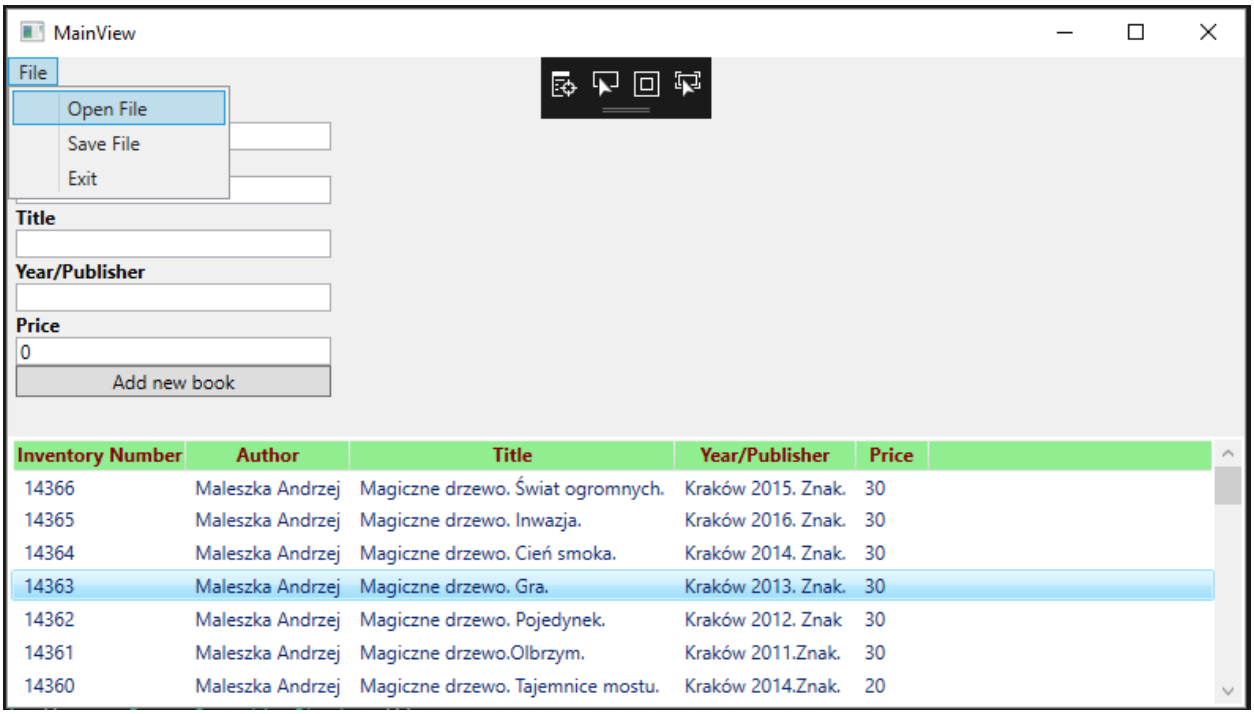
4.8 In MainView.xaml in main <Grid> add menu

```
<Menu DockPanel.Dock="Top">
        <MenuItem Header="_File">
                <MenuItem Header="_Open File" Command="{Binding OpenFile}"/>
                  <MenuItem Header="_Save File" Command="{Binding SaveFile}"/>
                  <MenuItem Header="_Exit"/>
              </MenuItem>
</Menu>
```

4.9 Now update DataContext in MainView.xaml.cs

```
public MainView()
{
    InitializeComponent();
    DataContext = new MainViewModel(new DialogService(), new JsonBookReader());
}
```

4.10 Now we can open JSON files, try to open file example_data.json



5. Reading XML files

5.1 Add new File DataAccess/Implementation/XmlBookReader.cs

```
class XmlBookReader : ISourceReader
{

static XmlSerializer serializer =
new XmlSerializer(typeof(List<Book>), new XmlRootAttribute("BookList"));

public List<Book> ReadBooks(string filePath)
        {
            using (var fileStream = File.OpenRead(filePath))
            {
                return (List<Book>)serializer.Deserialize(fileStream);
            }
        }
}
```

5.2 Add new file DataAccess/Interface/ISourceOperationStrategy.cs

```
public interface ISourceOperationStrategy
{
        ISourceReader GetReader(string fileName);
}
```

5.3 Add new file DataAccess/Implementation/SourceOperationStrategy.cs

```
public class SourceOperationStrategy : ISourceOperationStrategy
{
       private static readonly Dictionary<string, ISourceReader> _readers;

       static SourceOperationStrategy()
       {
              _readers = new Dictionary<string, ISourceReader>();

              _readers.Add(".xml", new XmlBookReader());
              _readers.Add(".json", new JsonBookReader());
       }

       public ISourceReader GetReader(string fileName)
       {
              var extension = Path.GetExtension(fileName);
              if (_readers.TryGetValue(extension, out ISourceReader reader))
                     return reader;

       return null;
       }
}
```

5.4 In ViewModels/MainViewModel.cs add following element

```
private readonly ISourceOperationStrategy _sourceOperations;
```

5.5 In ViewModels/MainViewModel.cs update following elements

```
public MainViewModel(IDialogService dialogService, ISourceOperationStrategy
sourceOperations)
{
       _dialogService = dialogService;
       _sourceOperations = sourceOperations;

       Books = new ObservableCollection<Book>();
       NewBook = new Book();

       AddBook = new RelayCommand(param => AddNewBook());
       OpenFile = new RelayCommand(param => LoadDataFromFile());
}

private void LoadDataFromFile()
{
       var fileName = _dialogService.OpenFileDialog();

       if (fileName != null)
       {
         List<Book> bookList = _sourceOperations.GetReader(fileName).ReadBooks(fileName);

         Books.Clear();

         foreach (Book book in bookList)
         {
              Books.Add(book);
         }
        }
}
```

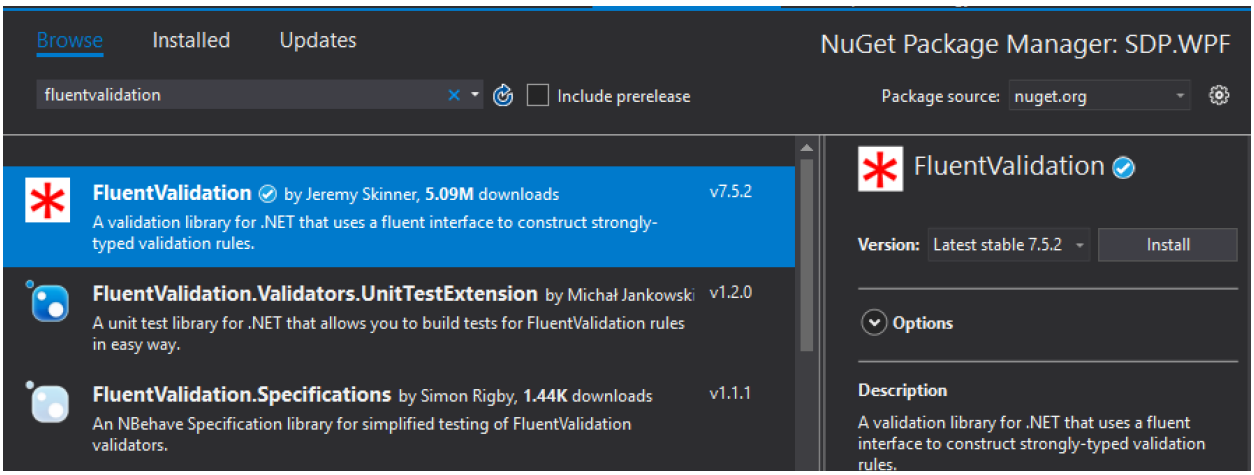
5.6 Update DataContext In Views/MainView.xaml.cs

```
DataContext = new MainViewModel(new DialogService(), new SourceOperationStrategy());
```

5.7 Try to open file example_data.xml

6. Implementing Validation

6.1 Add reference to FluentValidation, just like in step 4.5



6.2 Add new file Models/BookValidator.cs

```
public class BookValidator : AbstractValidator<Book>
{
        public BookValidator()
        {
            CascadeMode = CascadeMode.StopOnFirstFailure;

            RuleFor(b => b.InventoryNumber)
                 .NotEmpty().WithMessage("Inventory number can't be empty.")
                 .GreaterThan(9999).WithMessage("Inventory number must have 5 digits.")
                 .LessThan(100000).WithMessage("Inventory number must have 5 digits.");

            RuleFor(b => b.Title)
                 .NotEmpty().WithMessage("Title can't be empty.");

            RuleFor(b => b.Author)
                 .NotEmpty().WithMessage("Author can't be empty.");

            RuleFor(b => b.YearPublisher)
                 .NotEmpty().WithMessage("Year / Publisher can't be empty.");

            RuleFor(b => b.Price)
                 .NotEmpty().WithMessage("Price cannot be empty")
                 .GreaterThan(0).WithMessage("A book must have a price");
         }
}
```

6.3 Add *IDataErrorInfo* interface to class declaration in Models/Book.cs

```
public class Book : BindableBase, IDataErrorInfo
{
…
…
```

6.4 In file Models/Book.cs add following elements

```
private readonly BookValidator _bookValidator;

public bool IsValid
{
	get => _bookValidator.Validate(this).IsValid;
}
//////////////////////////////////////
public string Error
{
	get
	{
		if (_bookValidator != null)
		{
			var results = _bookValidator.Validate(this);

			if (results != null && results.Errors.Any())
			{
var errors =
String.Join(Environment.NewLine, results.Errors.Select(x => x.ErrorMessage).ToArray());
				return errors;
			}
		 }

	return String.Empty;
	 }
}
//////////////////////////////////////
public string this[string columnName]
{
	get
	{
var firstOrDefault = _bookValidator.Validate(this)
.Errors.FirstOrDefault(lol => lol.PropertyName == columnName);

		if (firstOrDefault != null)
		return _bookValidator != null ? firstOrDefault.ErrorMessage : String.Empty;

	 return String.Empty;
	 }
}
```

6.5 In file Models/Book.cs update default constructor

```
public Book()
{
	_bookValidator = new BookValidator();
}
```

6.6 In file ViewModels/MainViewModel.cs update method *AddNewBook()*

```
private void AddNewBook()
{
	if (NewBook.IsValid)
	{
		Books.Add(new Book(NewBook));
	}
}
```

6.7 In file Views/MainView.xaml in section <Grid.Resources> add following elements:

```
<ControlTemplate x:Key="ValidatedTextBoxTemplate">
	<StackPanel Orientation="Horizontal">
		<AdornedElementPlaceholder x:Name="textBox"/>
<TextBlock
VerticalAlignment="Center" FontWeight="Bold" FontSize="20" Text="!" Foreground="Red"/>
	</StackPanel>
</ControlTemplate>

<Style TargetType="{x:Type TextBox}">
	<Style.Triggers>
		<Trigger Property="Validation.HasError" Value="True">
<Setter Property="ToolTip" Value="{Binding RelativeSource={x:Static RelativeSource.Self},
Path=(Validation.Errors)[0].ErrorContent}" />
		</Trigger>
	</Style.Triggers>
<Setter Property="HorizontalAlignment" Value="Left" />
<Setter Property="Margin" Value="0, 0, 0, 10"/>
<Setter
Property="Validation.ErrorTemplate" Value="{StaticResource ValidatedTextBoxTemplate}" />
</Style>
```

6.8 Add validation to *<TextBox…/> elements*

```
<TextBlock Grid.Row="0" Text="Inv. No." />
<TextBox Grid.Row="1" Width="100" Text="{Binding Path=NewBook.InventoryNumber,
Mode=TwoWay, UpdateSourceTrigger=PropertyChanged, ValidatesOnDataErrors=True}" />

<TextBlock Grid.Row="2" Text="Author" />
<TextBox Grid.Row="3" Width="200" Text="{Binding Path=NewBook.Author, Mode=TwoWay,
UpdateSourceTrigger=PropertyChanged, ValidatesOnDataErrors=True}" />

<TextBlock Grid.Row="4" Text="Title" />
<TextBox Grid.Row="5" Width="200" Text="{Binding Path=NewBook.Title, Mode=TwoWay,
UpdateSourceTrigger=PropertyChanged, ValidatesOnDataErrors=True}" />

<TextBlock Grid.Row="6" Text="Year/Publisher" />
<TextBox Grid.Row="7" Width="200" Text="{Binding Path=NewBook.YearPublisher, Mode=TwoWay,
UpdateSourceTrigger=PropertyChanged, ValidatesOnDataErrors=True}" />

<TextBlock Grid.Row="8" Text="Price" />
<TextBox Grid.Row="9" Width="80" Text="{Binding Path=NewBook.Price, Mode=TwoWay,
UpdateSourceTrigger=PropertyChanged, ValidatesOnDataErrors=True}" />
```

7. Saving new books to file (Homework :-))

7.1 Add new file DataAccess/Interface/ISourceWriter.cs

```
public interface ISourceWriter
{
        void WriteBooks(string filePath, List<Book> books);
}
```

7.2 Add GetWriter method to DataAccess/Interface/IsourceOperationStrategy.cs

```
public interface ISourceOperationStrategy
{
         ISourceReader GetReader(string fileName);
         ISourceWriter GetWriter(string fileName);
}
```

7.3 In DataAccess/Implementation/SourceOperationStrategy.cs:

- Add declaration of *_writers*

  ```
  private static readonly Dictionary<string, ISourceWriter> _writers;
  ```

- Add method *GetWriter*

  ```
  public ISourceWriter GetWriter(string fileName)
   {
       var extension = Path.GetExtension(fileName);

       if (_writers.TryGetValue(extension, out ISourceWriter writer))
           return writer;

       return null;
   }
  ```

- In method SourceOperationStrategy

  ```
  _writers = new Dictionary<string, ISourceWriter>();
  ```

  Add JsonBookWriter and XmlBookWriter to _writers

7.4 In ViewModels/MainViewModel.cs

- Add SaveFile method
```
public ICommand SaveFile
 {
     get;
     private set;
 }
```

- Add to constructor following declaration
```
SaveFile = new RelayCommand(param => SaveDataToFile());
```

- Add new method SaveDataToFile()
  Implement saving to file, user will choose which format should be used

8. Create converter *

8.1 Add folder Converters

8.2 Add new file ToUpperConverter in folder Converters

8.3 Let ToUpperConverter class implements IValueConverter

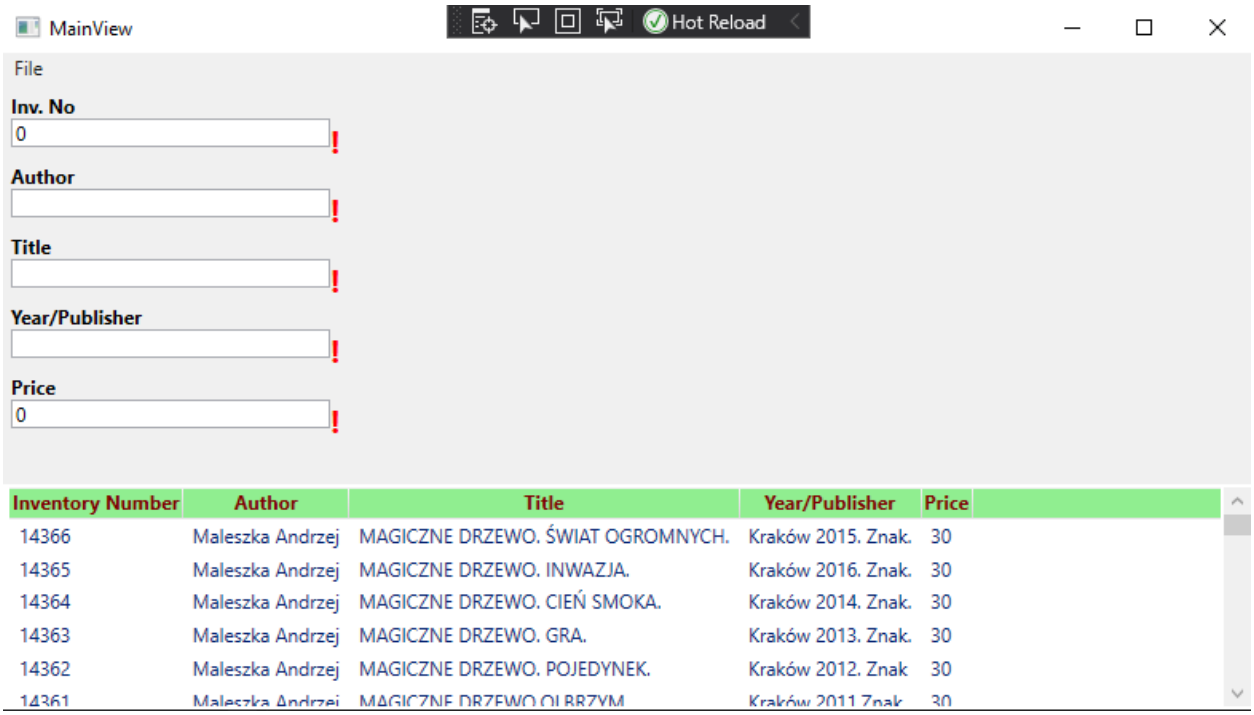
8.4 Implement Convert method to return string in upper-case – leave ConvertBack method as is

8.5 In MainView.xaml add namespace to Converters folder

8.6 Add ToUpperConverter to Grid.Resource

8.7 Apply ToUpperConverter to GridViewColumn for book Title

8.8 If everything wokrs correctly you should see that all titles in ListView are displayed in upper-case manner


MainView
Hot Reload
File
Inv. No
0
Author
Title
Year/Publisher
Price
0
Inventory Number
Author
Title
Year/Publisher
Price
14366 Maleszka Andrzej MAGICZNE DRZEWO. ŚWIAT OGROMNYCH. Kraków 2015. Znak. 30
14365 Maleszka Andrzej MAGICZNE DRZEWO. INWAZJA. Kraków 2016. Znak. 30
14364 Maleszka Andrzej MAGICZNE DRZEWO. CIEŃ SMOKA. Kraków 2014. Znak. 30
14363 Maleszka Andrzej MAGICZNE DRZEWO. GRA. Kraków 2013. Znak. 30
14362 Maleszka Andrzej MAGICZNE DRZEWO. POJEDYNEK. Kraków 2012. Znak 30