Лабораторна робота № 1

Інтерфейс віконного додатку

Мета: дослідження основних підходів та технологій реалізації інтерфейсу універсального додатку Windows. Обладнання: ПК, Visual Studio не нижче v.2010 (.Net Framework 4.5).

Хід роботи:

Репозиторій: https://github.com/DmitryKalinovskyi/WPF-TextEditor

Рис. 1. Вигляд віконного додатку.

Рис. 2. Вигляд віконного додатку
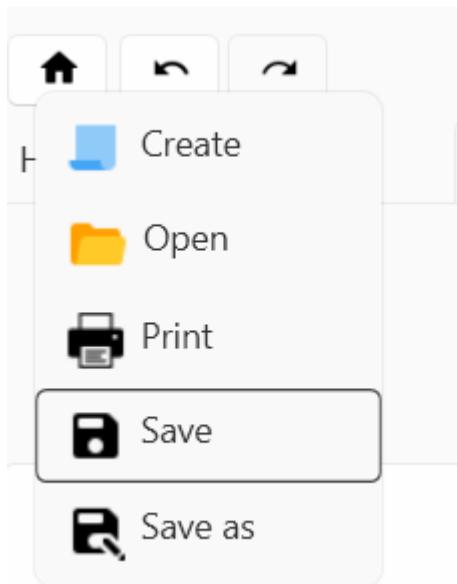
Рис. 3.
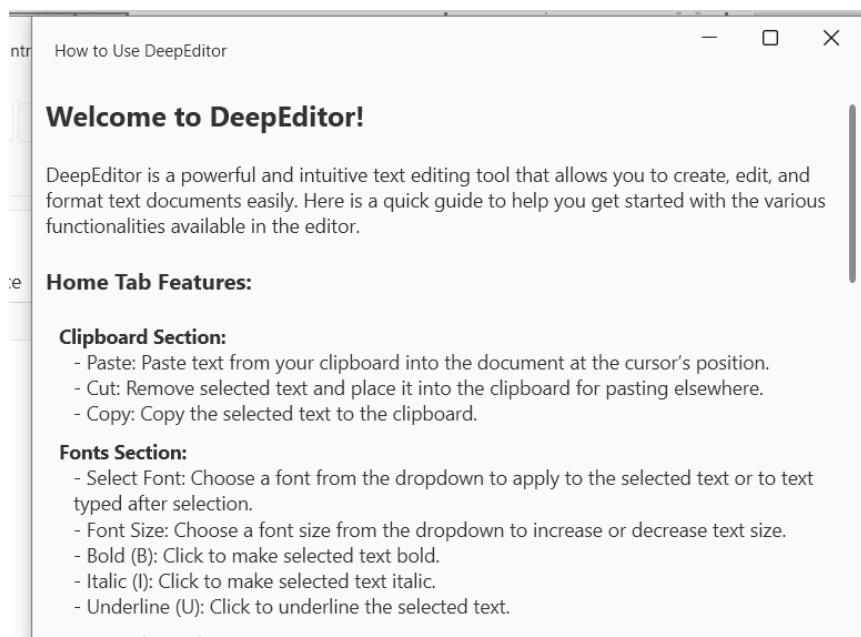
Рис. 4. Вікно справки.

Висновок: на цій практичній роботі я досліджував основні підходи та технології реалізації інтерфейсу універсального додатку Windows.

Лістинг програми:
MainWindow.xaml

```
<ui:FluentWindow x:Class="Word.MainWindow"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
```

```xml
    xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
    xmlns:local="clr-namespace:Word"
    xmlns:ui="http://schemas.lepo.co/wpfui/2022/xaml"
    xmlns:vm="clr-namespace:Word.ViewModels"
        xmlns:toolkit="http://schemas.xceed.com/wpf/xaml/toolkit"
    mc:Ignorable="d"
     Height="450" Width="800"

            WindowStartupLocation="CenterScreen">
    <ui:FluentWindow.Resources>
        <vm:MainWindowViewModel x:Key="MainWindowViewModel"/>
    </ui:FluentWindow.Resources>
    <ui:FluentWindow.DataContext>
        <StaticResource ResourceKey="MainWindowViewModel"/>
    </ui:FluentWindow.DataContext>
    <Grid>
        <Grid.RowDefinitions>
            <RowDefinition Height="60"/>
            <RowDefinition Height="auto"/>
            <RowDefinition Height="auto"/>
            <RowDefinition Height="*"/>
        </Grid.RowDefinitions>
        <ui:TitleBar Title="{Binding DocumentViewModel.Title}">
        </ui:TitleBar>

        <StackPanel Orientation="Horizontal" Grid.Row="1">
            <Button Click="Button_Click"
                    Margin="2, 2"
                    >
                <Button.ContextMenu>
                    <ContextMenu StaysOpen="true" x:Name="ContextMenu" DataContext="{StaticResource MainWindowViewModel}">
                        <!--<MenuItem Header="Open" Icon="{StaticResource }"/>-->
                        <MenuItem Header="Create" Command="{Binding DocumentViewModel.CreateNewCommand}">
                            <MenuItem.Icon>
                                <Image Source="{StaticResource PaperIcon24Source}" Width="24"/>
                            </MenuItem.Icon>
                        </MenuItem>
                        <MenuItem Header="Open"
                                Command="{Binding DocumentViewModel.OpenDocumentCommand}">
                            <MenuItem.Icon>
```

```xml
                    <Image Source="{StaticResource OpenedFolderIcon24Source}"
Width="25"/>
                </MenuItem.Icon>
            </MenuItem>
            <MenuItem Header="Print" Command="{Binding DocumentView-
Model.PrintCommand}">
                <MenuItem.Icon>
                    <Image Source="{StaticResource PrintIcon25Source}"
Width="24"/>
                </MenuItem.Icon>
            </MenuItem>
            <MenuItem Header="Save" Command="{Binding DocumentView-
Model.SaveCommand}">
                <MenuItem.Icon>
                    <Image Source="{StaticResource SaveIcon24Source}"
Width="24"/>
                </MenuItem.Icon>
            </MenuItem>
            <MenuItem Header="Save as"
                    Command="{Binding DocumentViewModel.Save-
AsCommand}">
                <MenuItem.Icon>
                    <Image Source="{StaticResource SaveAsIcon24Source}"
Width="24"/>
                </MenuItem.Icon>
            </MenuItem>
            <!--<MenuItem Header="Preferences" Click="MenuPreferences_Click"
>
                <MenuItem.Icon>
                    <Image Source="{StaticResource OptionsIcon24Source}"
Width="24"/>
                </MenuItem.Icon>
            </MenuItem>-->
            <!--<MenuItem Header="Recent Files">
                    <MenuItem Header="ReadMe.txt"/>
                    <MenuItem Header="Schedule.xls"/>
                </MenuItem>-->
        </ContextMenu>
      </Button.ContextMenu>
      <Button.Content>
        <StackPanel Orientation="Horizontal" >
          <Border Width="16">
            <StaticResource ResourceKey="HomeIcon24"/>
          </Border>
```

```xml
        <!--<TextBlock VerticalAlignment="A">Home</TextBlock>-->
      </StackPanel>
    </Button.Content>
  </Button>
  <Button Margin="2, 2" Command="{x:Static ApplicationCommands.Undo}"
                CommandTarget="{Binding ElementName=DocumentText-
Box}">
    <Border Width="16">
      <StaticResource ResourceKey="UndoIcon24"/>
    </Border>
  </Button>
  <Button Margin="2, 2" Command="{x:Static ApplicationCommands.Redo}"
                CommandTarget="{Binding ElementName=DocumentText-
Box}">
    <Border Width="16">
      <StaticResource ResourceKey="RedoIcon24"/>
    </Border>
  </Button>
</StackPanel>

<TabControl Grid.Row="2">
  <TabControl.Resources>
    <Style x:Key="ToolButton" TargetType="Button" BasedOn="{StaticRe-
source {x:Type Button}}">
      <Setter Property="Width" Value="36"/>
      <Setter Property="Height" Value="36"/>
      <!--<Setter Property="Padding" Value="10"/>-->
      <!--<Setter Property="Margin" Value="0, 0, 4, 0"/>-->
    </Style>
  </TabControl.Resources>
  <!--<TabItem>
    <TabItem.Style>
      <Style TargetType="TabItem">
        <Setter Property="Template">
          <Setter.Value>
            <ControlTemplate TargetType="TabItem">
              <Grid>
                --><!-- The content area that is not hit test visible --><!--
                <Border
            Background="Transparent"
            IsHitTestVisible="False">
                  --><!-- You can set a background here if needed --><!--
                </Border>
                --><!-- The interactive button --><!--
```

```xml
                                    <ContentPresenter ContentSource="Header" />
                                </Grid>
                            </ControlTemplate>
                        </Setter.Value>
                    </Setter>
                </Style>
            </TabItem.Style>
        </TabItem>-->

        <TabItem Header="Home">
            <Border Padding="10">
                <Border.Resources>
                    <Style x:Key="DockStyle" TargetType="DockPanel">
                        <Setter Property="Margin" Value="10, 0"/>
                        <Setter Property="VerticalAlignment" Value="Stretch"/>
                        <Setter Property="LastChildFill" Value="False"/>
                    </Style>
                    <Style x:Key="DockSub" TargetType="TextBlock">
                        <Setter Property="Foreground" Value="Gray"/>
                        <Setter Property="FontSize" Value="12"/>
                        <Setter Property="DockPanel.Dock" Value="Bottom"/>
                        <Setter Property="HorizontalAlignment" Value="Center"/>
                    </Style>
                </Border.Resources>
                <StackPanel Orientation="Horizontal">
                    <DockPanel Style="{StaticResource DockStyle}">
                        <Grid DockPanel.Dock="Top">
                            <Grid.ColumnDefinitions>
                                <ColumnDefinition/>
                                <ColumnDefinition/>
                            </Grid.ColumnDefinitions>

                            <Button Grid.Column="0"
                                Command="{x:Static ApplicationCommands.Paste}"
                                CommandTarget="{Binding ElementName=DocumentText-
Box}">

                                <StackPanel >
                                    <Border Width="31" HorizontalAlignment="Center">
                                        <StaticResource ResourceKey="PasteIcon48"/>
                                    </Border>
                                    <TextBlock HorizontalAlignment="Center">
                                    Paste
                                    </TextBlock>
                                </StackPanel>
```

```xml
            </Button>

            <StackPanel Grid.Column="1">
              <Button Command="{x:Static ApplicationCommands.Cut}"
                  CommandTarget="{Binding ElementName=DocumentText-
Box}">

                  <StackPanel Orientation="Horizontal">
                    <Border Width="12" VerticalAlignment="Center" Mar-
gin="0, 0, 2, 0">

                      <StaticResource ResourceKey="CutIcon24"/>
                    </Border>
                    <TextBlock VerticalAlignment="Center">
    Cut
                    </TextBlock>
                  </StackPanel>
              </Button>
              <Button Command="{x:Static ApplicationCommands.Copy}"
                  CommandTarget="{Binding ElementName=DocumentText-
Box}">
                  <StackPanel Orientation="Horizontal">
                    <Border Width="12" VerticalAlignment="Center" Mar-
gin="0, 0, 2, 0">

                      <StaticResource ResourceKey="CopyIcon24"/>
                    </Border>
                    <TextBlock VerticalAlignment="Center">Copy</Text-
Block>
                  </StackPanel>
              </Button>

            </StackPanel>
          </Grid>
          <!--<TextBlock DockPanel.Dock="Top">Hello</TextBlock>-->
          <TextBlock Style="{StaticResource DockSub}">Clipboard</Text-
Block>
        </DockPanel>
        <Separator/>
        <DockPanel Style="{StaticResource DockStyle}">
          <StackPanel Orientation="Horizontal" DockPanel.Dock="Top">
            <RibbonControlGroup>

              <ComboBox ItemsSource="{Binding Source={x:Static
Fonts.SystemFontFamilies}}"
                  SelectionChanged="ComboBox_SelectionChanged"
```

```
                              Text="Select Font..."
                              IsEditable="True"/>
                      <ComboBox x:Name="FontSizeComboBox"

                              Text="Size..."
                              IsEditable="True"
                              SelectionChanged="FontSize_SelectionChanged"
                              ItemsSource="{Binding FontSizes}"/>
                  </RibbonControlGroup>
              </StackPanel>
              <StackPanel Orientation="Horizontal" DockPanel.Dock="Top">
                  <RibbonButton Style="{StaticResource ToolButton}"
                          CommandTarget="{Binding ElementName=DocumentText-
Box}"

                          Command="{x:Static EditingCommands.ToggleBold}">
                      <StaticResource ResourceKey="BoldIcon24"/>
                  </RibbonButton>
                  <RibbonButton Style="{StaticResource ToolButton}"
                          CommandTarget="{Binding ElementName=DocumentText-
Box}"

                          Command="{x:Static EditingCommands.ToggleItalic}">
                      <StaticResource ResourceKey="ItalicIcon30"/>
                  </RibbonButton>
                  <RibbonButton Style="{StaticResource ToolButton}"
                          CommandTarget="{Binding ElementName=DocumentText-
Box}"

                          Command="{x:Static EditingCommands.ToggleUnderline}">
                      <StaticResource ResourceKey="UnderlineIcon25"/>
                  </RibbonButton>
              </StackPanel>
              <TextBlock Style="{StaticResource DockSub}">Fonts</TextBlock>
          </DockPanel>
          <Separator/>
          <DockPanel Style="{StaticResource DockStyle}">
              <StackPanel Orientation="Horizontal" DockPanel.Dock="Top">
                  <RibbonButton Style="{StaticResource ToolButton}"
      CommandTarget="{Binding ElementName=DocumentTextBox}"
      Command="{x:Static EditingCommands.AlignLeft}">
                          <StaticResource ResourceKey="AlignLeftIcon24"/>
                      </RibbonButton>
                  <RibbonButton Style="{StaticResource ToolButton}"
      CommandTarget="{Binding ElementName=DocumentTextBox}"
      Command="{x:Static EditingCommands.AlignCenter}">
                          <StaticResource ResourceKey="AlignCenterIcon24"/>
```

```
                </RibbonButton>
                <RibbonButton Style="{StaticResource ToolButton}"
    CommandTarget="{Binding ElementName=DocumentTextBox}"
    Command="{x:Static EditingCommands.AlignRight}">
                    <StaticResource ResourceKey="AlignRightIcon24"/>
                </RibbonButton>
                <RibbonButton Style="{StaticResource ToolButton}"
CommandTarget="{Binding ElementName=DocumentTextBox}"
Command="{x:Static EditingCommands.AlignJustify}">
                    <StaticResource ResourceKey="AlignJustifyIcon24"/>
                </RibbonButton>
            </StackPanel>
            <TextBlock Style="{StaticResource DockSub}">Paragraph</Text-
Block>
        </DockPanel>
      </StackPanel>
    </Border>
  </TabItem>
  <TabItem Header="Reference">
    <Border Padding="10">
      <Border.Resources>
        <Style x:Key="DockStyle" TargetType="DockPanel">
          <Setter Property="Margin" Value="10, 0"/>
          <Setter Property="VerticalAlignment" Value="Stretch"/>
          <Setter Property="LastChildFill" Value="False"/>
        </Style>
        <Style x:Key="DockSub" TargetType="TextBlock">
          <Setter Property="Foreground" Value="Gray"/>
          <Setter Property="FontSize" Value="12"/>
          <Setter Property="DockPanel.Dock" Value="Bottom"/>
          <Setter Property="HorizontalAlignment" Value="Center"/>
        </Style>
      </Border.Resources>
      <StackPanel Orientation="Horizontal">
        <DockPanel Style="{StaticResource DockStyle}">
          <Grid DockPanel.Dock="Top" >
            <StackPanel>
              <StackPanel Orientation="Horizontal">
                <Button Click="ReferenceButton_Click">
                  <StackPanel>
                    <Border Height="36">
                      <StaticResource ResourceKey="ReferenceIcon48"/>
                    </Border>
                    <TextBlock>Reference</TextBlock>
```

```
                        </StackPanel>
                    </Button>
                </StackPanel>
                <TextBlock Style="{StaticResource DockSub}">Help</Text-
Block>
            </StackPanel>
        </Grid>
    </DockPanel>
</StackPanel>
        </Border>
    </TabItem>
</TabControl>


<Grid Grid.Row="3">
    <!--Here we are using two way binding that updates content after each change,
that may reduce performance-->
    <toolkit:RichTextBox
        HorizontalAlignment="Center"
                IsEnabled="{Binding DocumentViewModel.IsEnabled}"
                x:Name="DocumentTextBox"
                Text="{Binding DocumentViewModel.Content, Mode=TwoWay,
UpdateSourceTrigger=PropertyChanged}"
                Style="{StaticResource {x:Type RichTextBox}}"
                >

    </toolkit:RichTextBox>
</Grid>
<!--<TextBox Grid.Row="2">

</TextBox>-->
    </Grid>
</ui:FluentWindow>
```

MainWindowViewModel.cs
```csharp
namespace Word.ViewModels
{
    public class MainWindowViewModel : ViewModelBase
    {
        public double[] FontSizes => _fontSizes;

        public DocumentViewModel DocumentViewModel { get; set; }

        public MainWindowViewModel()
        {
            DocumentViewModel = new DocumentViewModel();
        }

        private double[] _fontSizes = [3.0, 4.0, 5.0, 6.0, 6.5, 7.0, 7.5, 8.0, 8.5, 9.0,
9.5, 10.0, 10.5, 11.0, 11.5, 12.0, 12.5, 13.0, 13.5, 14.0, 15.0, 16.0, 17.0, 18.0, 19.0,
```

```
20.0, 22.0, 24.0, 26.0, 28.0, 30.0, 32.0, 34.0, 36.0, 38.0, 40.0, 44.0, 48.0, 52.0, 56.0,
60.0, 64.0, 68.0, 72.0, 76.0, 80.0, 88.0, 96.0, 104.0, 112.0, 120.0, 128.0, 136.0, 144.0
];
    }
}
```

DocumentViewModel.cs

```csharp
using Microsoft.Win32;
using System.Windows;
using System.Windows.Input;
using Word.Commands;
using Word.Models;
using Word.Services;

namespace Word.ViewModels
{
    public class DocumentViewModel : ViewModelBase
    {
        private DocumentRepository _documentRepository;
        private DocumentModel _documentModel;
        private PrintService? _printService;

        // Dependencies [Path, IsSaved]
        public string Title
        {
            get
            {
                string result = "DeepEditor";
                if (Path is not null)
                {
                    result += $": {System.IO.Path.GetFileName(Path)}";
                }
                if (!IsSaved) result += "*";
                return result;
            }
        }

        public string? Path
        {
            get => _documentModel.Path;
            set
            {
                _documentModel.Path = value;
                OnPropertyChanged(nameof(Path));
                OnPropertyChanged(nameof(Title));
            }
        }

        private bool _isSaved;
        public bool IsSaved
        {
            get => _isSaved;
            set
            {
                _isSaved = value;
                OnPropertyChanged(nameof(IsSaved));
                OnPropertyChanged(nameof(Title));
            }
        }

        public bool IsEnabled
        {
            get => _documentModel is not null;
        }
```

```csharp
        public string Content
        {
            get => _documentModel.Content ?? "";
            set
            {
                _documentModel.Content = value;
                IsSaved = false;
                OnPropertyChanged(nameof(Content));
            }
        }

        public DocumentViewModel()
        {
            _documentRepository = new DocumentRepository();
            _printService = new PrintService();
            _documentModel = new DocumentModel();

            // Initialize commands
            SaveAsCommand = new RelayCommand((param) => SaveAs());
            SaveCommand = new RelayCommand((param) => Save());
            OpenDocumentCommand = new RelayCommand((param) => OpenDocument());
            CreateNewCommand = new RelayCommand((param) => CreateNew());
            PrintCommand = new RelayCommand((param) =>
_printService.Print(_documentModel));
        }

        public void UpdateModel(DocumentModel model)
        {
            _documentModel = model;
            NotifyModelUpdate();
        }

        public void NotifyModelUpdate()
        {
            OnPropertyChanged(nameof(Title));
            OnPropertyChanged(nameof(Path));
            OnPropertyChanged(nameof(Content));
            OnPropertyChanged(nameof(IsEnabled));
            //OnPropertyChanged(nameof(SaveCommand));
            //OnPropertyChanged(nameof(SaveAsCommand));
        }

        #region Commands
        public ICommand SaveAsCommand { get; set; }
        public ICommand SaveCommand { get; set; }
        public ICommand OpenDocumentCommand { get; set; }
        public ICommand CreateNewCommand { get; set; }
        public ICommand PrintCommand { get; set; }

        public void CreateNew()
        {
            if (_documentModel is not null && !IsSaved)
            {
                // ask if we want to save our file.
                var result = MessageBox.Show("Want to save your changes?", "DeepEditor",
                    MessageBoxButton.YesNoCancel,
                    MessageBoxImage.Warning);

                if (result == MessageBoxResult.Yes)
                {
                    Save();
                }
                else if (result == MessageBoxResult.Cancel)
                {
                    return;
```

```csharp
                }
            }

            UpdateModel(new DocumentModel());
            IsSaved = true;
        }

        public void Save()
        {
            if (_documentModel is null) return;

            if (IsSaved) return;

            if (_documentModel.Path is null)
            {
                SaveAs();
                return;
            }

            _documentRepository.SaveDocument(_documentModel);
            IsSaved = true;
        }

        public void SaveAs()
        {
            if (_documentModel is null) return;

            var dialog = new SaveFileDialog();
            // set a default file name
            dialog.FileName = "document.rtf";
            // set filters – this can be done in properties as well

            dialog.Filter = "Document files (*.rtf)|*.rtf";
            dialog.ShowDialog();

            var documentModelResult = new DocumentModel()
            {
                Path = dialog.FileName,
                Content = _documentModel.Content
            };

            _documentRepository.SaveDocument(documentModelResult);
            UpdateModel(documentModelResult);
            IsSaved = true;
        }

        public void OpenDocument()
        {
            var dialog = new OpenFileDialog();
            dialog.Filter = "Document files (*.rtf)|*.rtf";
            var result = dialog.ShowDialog();

            if (result is null || result.Value == false)
            {
                // user decided to close dialog
                return;
            }

            var documentModel = _documentRepository.OpenDocument(dialog.FileName);
            if (documentModel is not null)
                UpdateModel(documentModel);
            IsSaved = true;
        }
        #endregion
```

```csharp
        }
}

DocumentRepository.cs
using System.IO;
using Word.Models;

namespace Word.Services
{
    public class DocumentRepository
    {
        public void SaveDocument(DocumentModel documentModel)
        {
            if (documentModel.Path is null)
            {
                throw new ArgumentNullException(nameof(documentModel.Path), "Document
path is required to save.");
            }

            using var file = new FileStream(documentModel.Path, FileMode.Create);
            using var streamWriter = new StreamWriter(file);
            streamWriter.Write(documentModel.Content);
        }

        public DocumentModel? OpenDocument(string path)
        {
            using var file = new FileStream(path, FileMode.Open);
            using var streamReader = new StreamReader(file);

            return new DocumentModel()
            {
                Path = path,
                Content = streamReader.ReadToEnd()
            };
        }
    }
}
```

PrintService.cs
```csharp
using System.IO;
using System.Text;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Documents;
using Word.Models;

namespace Word.Services
{
    public class PrintService
    {
        public void Print(DocumentModel document)
        {
            var dialog = new PrintDialog();

            // Check if the user confirmed the print dialog
            if (dialog.ShowDialog() == true)
            {
                // Create a FlowDocument with the content of the document
                FlowDocument flowDocument = ConvertRtfToFlowDocument(document.Content);

                // Create a DocumentPaginator from the FlowDocument
                IDocumentPaginatorSource documentPaginator = flowDocument;

                // Print the document using the PrintDialog
```

```csharp
                dialog.PrintDocument(documentPaginator.DocumentPaginator, "Print
Document");
            }
        }

        public FlowDocument ConvertRtfToFlowDocument(string rtfContent)
        {
            // Create a new FlowDocument
            FlowDocument flowDocument = new FlowDocument();

            // Use a RichTextBox to load the RTF content
            RichTextBox richTextBox = new RichTextBox();

            // Load the RTF content into the RichTextBox
            MemoryStream memoryStream = new
MemoryStream(Encoding.UTF8.GetBytes(rtfContent));
            TextRange textRange = new TextRange(richTextBox.Document.ContentStart,
richTextBox.Document.ContentEnd);

            // Try to load the RTF content into the TextRange
            textRange.Load(memoryStream, DataFormats.Rtf);

            // Return the document from the RichTextBox
            flowDocument = richTextBox.Document;

            return flowDocument;
        }
    }
}
```