ToolboxContainer.cs

using System;

using System.Windows.Controls;

using System.Windows.Input;

namespace WYSIWYGConfigurator

{

public class ToolboxContainer

{

public event EventHandler<ToolboxItemEventArgs> ItemDragged;

private readonly ListBox \_toolbox;

public ToolboxContainer(ListBox toolbox)

{

\_toolbox = toolbox;

\_toolbox.PreviewMouseDown += Toolbox\_PreviewMouseDown;

}

private void Toolbox\_PreviewMouseDown(object sender, MouseButtonEventArgs e)

{

if (\_toolbox.SelectedItem is ListBoxItem item && item.Content is string itemType)

{

ItemDragged?.Invoke(this, new ToolboxItemEventArgs { ItemType = itemType });

}

}

}

public class ToolboxItemEventArgs : EventArgs

{

public string ItemType { get; set; }

}

}

**PropertyContainer.cs:**

using System;

using System.Windows;

using System.Windows.Controls;

using System.Windows.Shapes;

namespace WYSIWYGConfigurator

{

public class PropertyContainer

{

private readonly StackPanel \_propertyBox;

private readonly Canvas \_designArea; // Reference to DesignArea for re-adding elements

public PropertyContainer(StackPanel propertyBox, Canvas designArea)

{

\_propertyBox = propertyBox;

\_designArea = designArea;

}

public void DisplayProperties(UIElement element)

{

\_propertyBox.Children.Clear();

if (element is Rectangle rectangle)

{

AddPropertyField("Width", rectangle.Width, value => rectangle.Width = Convert.ToDouble(value));

AddPropertyField("Height", rectangle.Height, value => rectangle.Height = Convert.ToDouble(value));

}

else if (element is TextBlock textBlock)

{

AddPropertyField("Text", textBlock.Text, value =>

{

// Remove and re-add the TextBlock to refresh it

RefreshElement(textBlock, new TextBlock

{

Text = value?.Trim() ?? string.Empty,

FontSize = textBlock.FontSize

});

});

AddPropertyField("FontSize", textBlock.FontSize, value =>

{

if (double.TryParse(value, out double fontSize) && fontSize > 0)

{

// Remove and re-add the TextBlock with updated font size

RefreshElement(textBlock, new TextBlock

{

Text = textBlock.Text,

FontSize = fontSize

});

}

});

}

}

private void AddPropertyField(string name, object value, Action<string> onValueChanged)

{

var textBox = new TextBox { Text = value.ToString() };

textBox.LostFocus += (sender, args) =>

{

onValueChanged(textBox.Text?.Trim());

};

\_propertyBox.Children.Add(new TextBlock { Text = name, Margin = new Thickness(0, 5, 0, 0) });

\_propertyBox.Children.Add(textBox);

}

private void RefreshElement(UIElement oldElement, UIElement newElement)

{

// Remove old element

\_designArea.Children.Remove(oldElement);

// Copy position of the old element

Canvas.SetLeft(newElement, Canvas.GetLeft(oldElement));

Canvas.SetTop(newElement, Canvas.GetTop(oldElement));

// Add new element to the DesignArea

\_designArea.Children.Add(newElement);

}

}

}

**MainWindow.xaml.cs**

using System;

using System.Windows;

using System.Windows.Controls;

using System.Windows.Input;

using System.Windows.Media;

using System.Windows.Shapes;

namespace WYSIWYGConfigurator

{

public partial class MainWindow : Window

{

private UIElement \_selectedElement; // Tracks the currently selected element

private readonly Rectangle \_selectionOverlay = new Rectangle

{

Stroke = Brushes.Black,

StrokeThickness = 2,

Fill = Brushes.Transparent // Transparent fill to show only the border

};

public MainWindow()

{

InitializeComponent();

// Set up Toolbox drag event

Toolbox.PreviewMouseDown += Toolbox\_PreviewMouseDown;

}

private void Toolbox\_PreviewMouseDown(object sender, MouseButtonEventArgs e)

{

if (sender is ListBox toolbox && toolbox.SelectedItem is ListBoxItem item)

{

DragDrop.DoDragDrop(toolbox, item.Content, DragDropEffects.Copy);

}

}

private void DesignArea\_DragEnter(object sender, DragEventArgs e)

{

if (!e.Data.GetDataPresent(DataFormats.Text))

{

e.Effects = DragDropEffects.None;

}

}

private void DesignArea\_Drop(object sender, DragEventArgs e)

{

if (e.Data.GetDataPresent(DataFormats.Text))

{

string tool = e.Data.GetData(DataFormats.Text) as string;

UIElement element = null;

if (tool == "Rectangle")

{

element = new Rectangle

{

Width = 100,

Height = 50,

Fill = Brushes.Blue

};

}

else if (tool == "Ellipse")

{

element = new Ellipse

{

Width = 100,

Height = 100,

Fill = Brushes.Red

};

}

else if (tool == "TextBlock")

{

element = new TextBlock

{

Text = "Sample Text",

FontSize = 14,

Foreground = Brushes.Black

};

}

if (element != null)

{

Canvas.SetLeft(element, e.GetPosition(DesignArea).X);

Canvas.SetTop(element, e.GetPosition(DesignArea).Y);

element.MouseLeftButtonDown += (s, ev) => SelectElement(element);

DesignArea.Children.Add(element);

}

}

}

private void DesignArea\_MouseDown(object sender, MouseButtonEventArgs e)

{

var clickedElement = e.Source as UIElement;

if (clickedElement != null)

{

SelectElement(clickedElement);

}

}

private void SelectElement(UIElement element)

{

if (\_selectedElement != null)

{

// Remove the previous selection overlay

DesignArea.Children.Remove(\_selectionOverlay);

}

\_selectedElement = element;

if (\_selectedElement != null)

{

// Add a selection overlay

if (\_selectedElement is FrameworkElement fe)

{

\_selectionOverlay.Width = fe.Width;

\_selectionOverlay.Height = fe.Height;

Canvas.SetLeft(\_selectionOverlay, Canvas.GetLeft(\_selectedElement));

Canvas.SetTop(\_selectionOverlay, Canvas.GetTop(\_selectedElement));

DesignArea.Children.Add(\_selectionOverlay);

}

// Update the property box

UpdatePropertyBox();

}

}

private void UpdatePropertyBox()

{

PropertyBox.Children.Clear(); // Clear previous properties

if (\_selectedElement is Rectangle rect)

{

PropertyBox.Children.Add(new TextBlock { Text = "Width:" });

var widthTextBox = new TextBox

{

Text = rect.Width.ToString(),

Margin = new Thickness(0, 5, 0, 5)

};

widthTextBox.LostFocus += (sender, args) =>

{

if (double.TryParse(((TextBox)sender).Text, out double width))

{

rect.Width = width;

\_selectionOverlay.Width = width; // Update overlay size

}

};

PropertyBox.Children.Add(widthTextBox);

PropertyBox.Children.Add(new TextBlock { Text = "Height:" });

var heightTextBox = new TextBox

{

Text = rect.Height.ToString(),

Margin = new Thickness(0, 5, 0, 5)

};

heightTextBox.LostFocus += (sender, args) =>

{

if (double.TryParse(((TextBox)sender).Text, out double height))

{

rect.Height = height;

\_selectionOverlay.Height = height; // Update overlay size

}

};

PropertyBox.Children.Add(heightTextBox);

}

else if (\_selectedElement is Ellipse ellipse)

{

PropertyBox.Children.Add(new TextBlock { Text = "Width:" });

var widthTextBox = new TextBox

{

Text = ellipse.Width.ToString(),

Margin = new Thickness(0, 5, 0, 5)

};

widthTextBox.LostFocus += (sender, args) =>

{

if (double.TryParse(((TextBox)sender).Text, out double width))

{

ellipse.Width = width;

\_selectionOverlay.Width = width; // Update overlay size

}

};

PropertyBox.Children.Add(widthTextBox);

PropertyBox.Children.Add(new TextBlock { Text = "Height:" });

var heightTextBox = new TextBox

{

Text = ellipse.Height.ToString(),

Margin = new Thickness(0, 5, 0, 5)

};

heightTextBox.LostFocus += (sender, args) =>

{

if (double.TryParse(((TextBox)sender).Text, out double height))

{

ellipse.Height = height;

\_selectionOverlay.Height = height; // Update overlay size

}

};

PropertyBox.Children.Add(heightTextBox);

}

else if (\_selectedElement is TextBlock textBlock)

{

PropertyBox.Children.Add(new TextBlock { Text = "Text:" });

var textTextBox = new TextBox

{

Text = textBlock.Text,

Margin = new Thickness(0, 5, 0, 5)

};

textTextBox.LostFocus += (sender, args) =>

{

textBlock.Text = ((TextBox)sender).Text;

};

PropertyBox.Children.Add(textTextBox);

PropertyBox.Children.Add(new TextBlock { Text = "Font Size:" });

var fontSizeTextBox = new TextBox

{

Text = textBlock.FontSize.ToString(),

Margin = new Thickness(0, 5, 0, 5)

};

fontSizeTextBox.LostFocus += (sender, args) =>

{

if (double.TryParse(((TextBox)sender).Text, out double fontSize))

{

textBlock.FontSize = fontSize;

}

};

PropertyBox.Children.Add(fontSizeTextBox);

}

else

{

PropertyBox.Children.Add(new TextBlock { Text = "No properties available." });

}

}

}

}

**MainWindow.xaml:**<Window x:Class="WYSIWYGConfigurator.MainWindow"

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

Title="WYSIWYG Configurator" Height="600" Width="800">

<Grid>

<!-- Define Grid Columns -->

<Grid.ColumnDefinitions>

<ColumnDefinition Width="200" />

<!-- Toolbox -->

<ColumnDefinition Width="\*" />

<!-- Design Area -->

<ColumnDefinition Width="200" />

<!-- Property Box -->

</Grid.ColumnDefinitions>

<!-- Toolbox -->

<ListBox Grid.Column="0" Background="LightGray" Margin="5" Name="Toolbox">

<ListBoxItem Content="Rectangle" />

<ListBoxItem Content="Ellipse" />

<ListBoxItem Content="TextBlock" />

</ListBox>

<!-- Design Area -->

<Canvas Grid.Column="1" Background="White" Margin="5" Name="DesignArea"

AllowDrop="True" Drop="DesignArea\_Drop" MouseDown="DesignArea\_MouseDown">

<!-- Dynamically added elements will appear here -->

</Canvas>

<!-- Property Box -->

<StackPanel Grid.Column="2" Background="LightGray" Margin="5" Name="PropertyBox">

<TextBlock Text="Properties" FontWeight="Bold" Margin="5" />

<!-- Dynamic property fields will appear here -->

</StackPanel>

</Grid>

</Window>

**DesignAreaContainer.cs**

using System;

using System.Collections.Generic;

using System.Windows;

using System.Windows.Controls;

using System.Windows.Shapes;

namespace WYSIWYGConfigurator

{

public class DesignAreaContainer

{

private readonly Canvas \_designArea;

private readonly List<UIElement> \_elements;

public DesignAreaContainer(Canvas designArea)

{

\_designArea = designArea;

\_elements = new List<UIElement>();

\_designArea.AllowDrop = true;

\_designArea.Drop += DesignArea\_Drop;

}

private void DesignArea\_Drop(object sender, DragEventArgs e)

{

if (e.Data.GetData(DataFormats.StringFormat) is string itemType)

{

Point dropPosition = e.GetPosition(\_designArea);

AddElement(itemType, dropPosition);

}

}

public void AddElement(string itemType, Point position)

{

UIElement element = itemType switch

{

"Rectangle" => new Rectangle { Width = 100, Height = 50, Fill = System.Windows.Media.Brushes.Blue },

"Ellipse" => new Ellipse { Width = 100, Height = 100, Fill = System.Windows.Media.Brushes.Red },

"TextBlock" => CreateTextBlock(),

\_ => null

};

if (element != null)

{

Canvas.SetLeft(element, position.X);

Canvas.SetTop(element, position.Y);

\_designArea.Children.Add(element);

\_elements.Add(element);

// Ensure the canvas redraws

\_designArea.InvalidateVisual();

}

}

private TextBlock CreateTextBlock()

{

return new TextBlock

{

Text = "Sample Text",

FontSize = 14,

TextWrapping = TextWrapping.Wrap

};

}

public bool Contains(UIElement element) => \_elements.Contains(element);

public void UpdateElement(UIElement element)

{

if (\_elements.Contains(element))

{

// Force the element to redraw

element.InvalidateVisual();

\_designArea.InvalidateVisual();

}

}

}

}