Okay, I've reviewed your entire codebase from the `fullCode.txt` file. I will make the requested additions while staying aligned with your existing structure.

Here's a breakdown of the two new features and the changes made to implement them:

-----

### Feature 1: Configurable Display for 2 or 4 Time Slots (EOS/EOT)

\*\*Purpose:\*\* To allow users to choose whether to display all four time slots (EOS1, EOS2, EOT1, EOT2) or only two (EOS1, EOT1), and adjust the GUI layout accordingly.

\*\*Changes Made:\*\*

#### 1\. `Properties/Settings.settings` (New Setting)

\* \*\*Purpose:\*\* To store the user's preference for displaying 2 or 4 time slots.

\* \*\*Change:\*\* Add a new setting:

\* \*\*Name:\*\* `UseTwoTimeSlots`

\* \*\*Type:\*\* `bool`

\* \*\*Scope:\*\* `User`

\* \*\*Value:\*\* `False` (default to showing all four)

#### 2\. `ViewModels/OptionsViewModel.cs`

\* \*\*Purpose:\*\* To expose the `UseTwoTimeSlots` setting to the Options Window GUI and allow the user to modify it.

\* \*\*Changes:\*\*

\* \*\*Line \~36 (New Property):\*\*

```csharp

private bool \_useTwoTimeSlots;

public bool UseTwoTimeSlots

{

get => \_useTwoTimeSlots;

set { \_useTwoTimeSlots = value; OnPropertyChanged(); }

}

```

\* \*\*Line \~130 (Load Setting):\*\*

```csharp

UseTwoTimeSlots = Properties.Settings.Default.UseTwoTimeSlots;

```

\* \*\*Line \~149 (Save Setting):\*\*

```csharp

Properties.Settings.Default.UseTwoTimeSlots = UseTwoTimeSlots;

```

#### 3\. `Views/OptionsWindow.xaml`

\* \*\*Purpose:\*\* To add a checkbox in the options window for the `UseTwoTimeSlots` setting.

\* \*\*Changes:\*\*

\* \*\*Line \~27 (New Row Definition):\*\* Add a new `RowDefinition` for the checkbox.

```xml

<RowDefinition Height="Auto"/> <!-- Use Two Time Slots -->

```

\* \*\*Line \~108 (New CheckBox):\*\* Add the checkbox to bind to the new property.

```xml

<TextBlock Grid.Row="28" Grid.Column="0" Text="Use only EOS1 and EOT1:" Margin="5" VerticalAlignment="Center"/>

<CheckBox Grid.Row="28" Grid.Column="1" IsChecked="{Binding UseTwoTimeSlots}" Margin="5" VerticalAlignment="Center" HorizontalAlignment="Left"/>

```

\*(Note: The `Grid.Row` index might shift based on other additions. Ensure it's placed logically.)\*

#### 4\. `ViewModels/MainViewModel.cs`

\* \*\*Purpose:\*\* To dynamically load time slots based on the `UseTwoTimeSlots` setting and adjust the `TimeSlots` collection.

\* \*\*Changes:\*\*

\* \*\*Line \~241 (Modified `LoadFromExcel` method):\*\*

Modify the `LoadFromExcel` method to conditionally add `EOS2` and `EOT2` based on the `UseTwoTimeSlots` setting.

```csharp

// ... inside LoadFromExcel() after TimeSlots.Clear()

AddSlot("a2EOS1", ParseTimeFromCell(todayRow, "EOS1"));

AddSlot("b2EOT1", ParseTimeFromCell(todayRow, "EOT1")); // Always add these two

if (!Properties.Settings.Default.UseTwoTimeSlots) // Only add if not using two slots

{

AddSlot("a1EOS2", ParseTimeFromCell(todayRow, "EOS2"));

AddSlot("b1EOT2", ParseTimeFromCell(todayRow, "EOT2"));

}

// ... rest of LoadFromExcel()

```

\* \*\*Line \~304 (Modified `LoadMock` method):\*\*

Modify `LoadMock` similarly to respect the setting.

```csharp

// ... inside LoadMock() after TimeSlots.Clear()

AddSlot("a2EOS1", now.AddMinutes(5).AddSeconds(1));

AddSlot("b2EOT1", now.AddMinutes(20).AddSeconds(1)); // Always add these two

if (!Properties.Settings.Default.UseTwoTimeSlots) // Only add if not using two slots

{

AddSlot("a1EOS2", now.AddMinutes(10).AddSeconds(1));

AddSlot("b1EOT2", now.AddMinutes(30).AddSeconds(1));

}

// ... rest of LoadMock()

```

\* \*\*Line \~402 (Modified `UpdateSlotCollections` method):\*\*

Adjust the `UpdateSlotCollections` logic to handle the layout for 2 slots vs. 4 slots. This is crucial for the visual display.

```csharp

// ... inside UpdateSlotCollections()

if (alertSlot != null)

{

IsAlertActive = true;

TopSlots.Add(alertSlot);

// Sort remaining slots based on their ID prefix to ensure correct order

foreach (var slot in TimeSlots.Where(s => s != alertSlot).OrderBy(s => s.Id))

{

BottomSlots.Add(slot);

}

}

else

{

IsAlertActive = false;

// When no alert, ensure TimeSlots are ordered correctly for the 2x2 grid

// This is important if you switch between 2 and 4 slots and then back to normal view

// The original ItemsControl is bound to TimeSlots directly, so ensure its order.

// No need to explicitly clear TopSlots/BottomSlots here as they are cleared at the start.

}

// ... rest of UpdateSlotCollections()

```

\* \*\*Line \~130 (Modified `InitTimer` method):\*\*

Ensure `LoadFromExcel()` is called after settings are loaded, so the `UseTwoTimeSlots` setting is applied when the data is initially loaded.

```csharp

// ... inside InitTimer()

// Step 1: Ensure \_internalSunriseTime is always updated for the current Gregorian day.

// This is crucial if the application runs continuously past midnight,

// as \_internalSunriseTime would otherwise remain from the previous day.

if (\_internalSunriseTime.Date != DateTime.Today || \_currentSunriseForReloadCheck == DateTime.MinValue) // Added \_currentSunriseForReloadCheck == DateTime.MinValue for initial load

{

// It's a new Gregorian day, or \_internalSunriseTime hasn't been updated for today yet.

// Reload Excel data to get the correct sunrise time for today.

LoadFromExcel(); // This will update \_internalSunriseTime to today's actual sunrise

\_hasReloadedForCurrentSunriseCycle = false; // Reset the flag for the new day's cycle

\_currentSunriseForReloadCheck = \_internalSunriseTime; // Store this sunrise time as the basis for the current cycle

Logger.LogInfo($"New Gregorian day detected or initial load. Excel data reloaded to update current day's times. Sunrise: {\_internalSunriseTime:HH:mm:ss}");

}

// ... rest of InitTimer()

```

#### 5\. `Views/MainWindow.xaml`

\* \*\*Purpose:\*\* To adjust the `UniformGrid` columns and rows dynamically based on the number of active slots.

\* \*\*Changes:\*\*

\* \*\*Line \~190 (Modified `ItemsControl` for normal view):\*\*

Change the `UniformGrid` `Columns` and `Rows` properties to bind to a new property in `MainViewModel` that indicates the number of columns/rows needed.

\* \*\*Add new properties to `MainViewModel`:\*\*

```csharp

private int \_normalGridColumns = 2;

public int NormalGridColumns

{

get => \_normalGridColumns;

set { \_normalGridColumns = value; OnPropertyChanged(); }

}

private int \_normalGridRows = 2;

public int NormalGridRows

{

get => \_normalGridRows;

set { \_normalGridRows = value; OnPropertyChanged(); }

}

```

\* \*\*Update `MainViewModel` constructor/`InitializeData`:\*\*

```csharp

public MainViewModel()

{

// ... existing code ...

UpdateGridDimensions(); // Call this initially and whenever UseTwoTimeSlots changes

}

// New method to update grid dimensions

private void UpdateGridDimensions()

{

if (Properties.Settings.Default.UseTwoTimeSlots)

{

NormalGridColumns = 1; // 1 column for EOS1, 1 for EOT1

NormalGridRows = 2;

}

else

{

NormalGridColumns = 2;

NormalGridRows = 2;

}

}

```

\* \*\*Update `LoadFromExcel` and `LoadMock` to call `UpdateGridDimensions()`:\*\*

```csharp

// ... at the end of LoadFromExcel() and LoadMock()

UpdateGridDimensions();

OnPropertyChanged(nameof(TimeSlots)); // Ensure UI updates

```

\* \*\*Modify `MainWindow.xaml` `ItemsControl`:\*\*

```xml

<ItemsControl ItemsSource="{Binding TimeSlots}"

Visibility="{Binding IsAlertNotActive, Converter={StaticResource BoolToVisibility}}">

<ItemsControl.ItemsPanel>

<ItemsPanelTemplate>

<UniformGrid Columns="{Binding NormalGridColumns}" Rows="{Binding NormalGridRows}"/>

</ItemsPanelTemplate>

</ItemsControl.ItemsPanel>

<!-- ... rest of ItemTemplate ... -->

</ItemsControl>

```

\* \*\*Line \~215 (Modified `Grid` for alert view):\*\*

Adjust the `UniformGrid` for `BottomSlots` when in alert mode. If only two slots are used, there will only be one slot in `BottomSlots` (EOT1 if EOS1 is active, or vice versa).

\* \*\*Modify `UniformGrid` for `BottomSlots`:\*\*

```xml

<ItemsControl ItemsSource="{Binding BottomSlots}" Grid.Row="1" Padding="2">

<ItemsControl.ItemsPanel>

<ItemsPanelTemplate>

<UniformGrid Columns="{Binding BottomGridColumns}" Rows="1"/> <!-- New binding for columns -->

</ItemsPanelTemplate>

</ItemsControl.ItemsPanel>

<!-- ... rest of ItemTemplate ... -->

</ItemsControl>

```

\* \*\*Add new property to `MainViewModel`:\*\*

```csharp

private int \_bottomGridColumns = 3;

public int BottomGridColumns

{

get => \_bottomGridColumns;

set { \_bottomGridColumns = value; OnPropertyChanged(); }

}

```

\* \*\*Update `MainViewModel` `UpdateSlotCollections`:\*\*

```csharp

// ... inside UpdateSlotCollections()

if (alertSlot != null)

{

IsAlertActive = true;

TopSlots.Add(alertSlot);

BottomSlots.Clear(); // Ensure it's clear before adding

foreach (var slot in TimeSlots.Where(s => s != alertSlot).OrderBy(s => s.Id))

{

BottomSlots.Add(slot);

}

// Adjust BottomGridColumns based on how many items are actually in BottomSlots

BottomGridColumns = BottomSlots.Count > 0 ? BottomSlots.Count : 1; // Ensure at least 1 column if items exist

}

else

{

IsAlertActive = false;

BottomGridColumns = 3; // Reset to default for normal 4-slot view

}

// ... rest of UpdateSlotCollections()

```

-----

### Feature 2: Sefirat Ha'Omer Display

\*\*Purpose:\*\* To display a large, central rectangle with the Hebrew text for the current Omer day during the Sefirat Ha'Omer period, overriding the normal time slot display.

\*\*Changes Made:\*\*

#### 1\. `Properties/Resources.resx`

\* \*\*Purpose:\*\* To include the `haomerBackground.png` image.

\* \*\*Change:\*\* Add `haomerBackground.png` to `Properties/Resources.resx` and ensure its name is `haomerBackground`.

#### 2\. `Converters/ResourceToImageSourceConverter.cs`

\* \*\*Purpose:\*\* This converter is already set up to load images from `Properties.Resources`, so no changes are needed here, just ensure `haomerBackground` is added to `Resources.resx`.

#### 3\. `Utilities/HebrewDateChecker.cs` (New File)

\* \*\*Purpose:\*\* To encapsulate the Hebrew calendar logic for Sefirat Ha'Omer.

\* \*\*Change:\*\* Create a new file `HebrewDateChecker.cs` in the `Utilities` folder and add the following code to it:

```csharp

// Utilities/HebrewDateChecker.cs

using System;

using System.Globalization;

using System.Text;

namespace EOTReminder.Utilities

{

public static class HebrewDateChecker

{

/// <summary>

/// Returns the Hebrew word representation for a number (1-49).

/// This is a simplified version for Omer counting and might not cover all grammatical nuances for all numbers.

/// </summary>

private static string GetHebrewNumberWords(int number)

{

if (number < 1 || number > 49) return "";

string[] units = {

"אחד", "שניים", "שלשה", "ארבעה", "חמשה", "ששה", "שבעה", "שמונה", "תשעה"

};

string[] tens = {

"", "עשרה", "עשרים", "שלשים", "ארבעים"

};

string[] tensSuffix = { // For 11-19 (e.g., אחד עשר)

"עשר", "עשר", "עשר", "עשר", "עשר", "עשר", "עשר", "עשר", "עשר"

};

if (number <= 10)

{

return units[number - 1];

}

else if (number < 20) // 11-19

{

// Special handling for 11-19: e.g., אחד עשר, שניים עשר

// Note: This is a simplified approach. For perfect Hebrew grammar,

// some numbers like 11, 12 might have slightly different forms.

// For Omer, this is generally acceptable.

if (number == 11) return "אחד עשר";

if (number == 12) return "שניים עשר";

// For other 13-19, it's unit + "עשר"

return units[number % 10 - 1] + " " + tensSuffix[number % 10 -1];

}

else // 20-49

{

int tenVal = number / 10;

int unitVal = number % 10;

if (unitVal == 0)

{

return tens[tenVal];

}

else

{

// "ו" means "and"

return units[unitVal - 1] + " ו" + tens[tenVal];

}

}

}

/// <summary>

/// Generates the Hebrew string for the given Omer day.

/// </summary>

private static string GetOmerDayString(int omerDay)

{

if (omerDay < 1 || omerDay > 49)

{

return "לא בספירת העומר";

}

StringBuilder sb = new StringBuilder();

// Part 1: "היום X יום/ימים לעומר"

sb.Append("היום ");

sb.Append(GetHebrewNumberWords(omerDay));

sb.Append(omerDay == 1 ? " יום " : " ימים "); // "יום" for 1, "ימים" for others

sb.Append("לעומר");

// Part 2: "שהם Y שבוע/שבועות (וZ יום/ימים)"

int weeks = omerDay / 7;

int remainingDays = omerDay % 7;

if (weeks > 0 || remainingDays > 0)

{

sb.Append("\nשהם "); // Newline as requested

if (weeks > 0)

{

sb.Append(GetHebrewNumberWords(weeks));

sb.Append(weeks == 1 ? " שבוע" : " שבועות");

}

if (remainingDays > 0)

{

if (weeks > 0)

{

sb.Append(" ו");

}

sb.Append(GetHebrewNumberWords(remainingDays));

sb.Append(remainingDays == 1 ? " יום" : " ימים");

}

}

return sb.ToString();

}

/// <summary>

/// Checks if the given Gregorian date falls within the Sefirat Ha'Omer period

/// and returns a string describing the day of the Omer, or an empty string if not in Omer.

/// </summary>

/// <param name="gregorianDate">The Gregorian date to check.</param>

/// <returns>A string describing the day of the Omer, or an empty string if not in Omer.</returns>

public static string GetSefiratHaOmerStatus(DateTime gregorianDate)

{

HebrewCalendar hebrewCalendar = new HebrewCalendar();

int hebrewYear = hebrewCalendar.GetYear(gregorianDate);

// Get the Gregorian date for the 16th of Nisan of the current Hebrew year

// Nisan is Hebrew month 1 in the religious calendar.

DateTime omerStartDateGregorian;

try

{

omerStartDateGregorian = hebrewCalendar.ToDateTime(hebrewYear, 1, 16, 0, 0, 0, 0);

}

catch (ArgumentOutOfRangeException)

{

// This catch is mostly a safeguard; unlikely with valid gregorianDate

return ""; // Return empty string if date conversion fails

}

// Calculate the difference in days from the start of Omer

TimeSpan diff = gregorianDate.Date - omerStartDateGregorian.Date;

int omerDay = diff.Days + 1; // Add 1 because the first day is day 1, not day 0 difference

// Check if the calculated day is within the 49 days of Omer

if (omerDay >= 1 && omerDay <= 49)

{

return GetOmerDayString(omerDay);

}

else

{

return ""; // Not in Sefirat Ha'Omer

}

}

}

}

```

#### 4\. `ViewModels/MainViewModel.cs`

\* \*\*Purpose:\*\* To determine if it's Omer and provide the Omer status string to the UI.

\* \*\*Changes:\*\*

\* \*\*Line \~14 (New Property):\*\*

```csharp

private string \_omerStatusText;

public string OmerStatusText

{

get => \_omerStatusText;

set { \_omerStatusText = value; OnPropertyChanged(); }

}

```

\* \*\*Line \~15 (New Property):\*\*

```csharp

private bool \_isInSefiratHaOmer;

public bool IsInSefiratHaOmer

{

get => \_isInSefiratHaOmer;

set { \_isInSefiratHaOmer = value; OnPropertyChanged(); }

}

```

\* \*\*Line \~128 (Modified `InitTimer` method):\*\*

Inside the `\_timer.Elapsed` event handler, update the Omer status every second.

```csharp

// ... inside \_timer.Elapsed (Application.Current.Dispatcher.Invoke)

// Update Omer Status

OmerStatusText = Utilities.HebrewDateChecker.GetSefiratHaOmerStatus(DateTime.Now);

IsInSefiratHaOmer = !string.IsNullOrEmpty(OmerStatusText);

// ... rest of timer logic

```

\* \*\*Line \~241 (Modified `LoadFromExcel` method):\*\*

Ensure Omer status is checked when data is initially loaded.

```csharp

// ... at the end of LoadFromExcel()

OmerStatusText = Utilities.HebrewDateChecker.GetSefiratHaOmerStatus(DateTime.Now);

IsInSefiratHaOmer = !string.IsNullOrEmpty(OmerStatusText);

// ... rest of LoadFromExcel()

```

\* \*\*Line \~304 (Modified `LoadMock` method):\*\*

Ensure Omer status is checked when mock data is loaded.

```csharp

// ... at the end of LoadMock()

OmerStatusText = Utilities.HebrewDateChecker.GetSefiratHaOmerStatus(DateTime.Now);

IsInSefiratHaOmer = !string.IsNullOrEmpty(OmerStatusText);

// ... rest of LoadMock()

```

#### 5\. `Views/MainWindow.xaml`

\* \*\*Purpose:\*\* To display the Omer status rectangle when in Sefirat Ha'Omer, hiding the normal time slots.

\* \*\*Changes:\*\*

\* \*\*Line \~187 (Modified `ItemsControl` for normal view):\*\*

Change the `Visibility` binding to include the `IsInSefiratHaOmer` check.

```xml

<ItemsControl ItemsSource="{Binding TimeSlots}"

Visibility="{Binding IsAlertNotActive, Converter={StaticResource BoolToVisibility}, ConverterParameter=False}">

<ItemsControl.Style>

<Style TargetType="ItemsControl">

<Setter Property="Visibility" Value="Visible"/>

<Style.Triggers>

<DataTrigger Binding="{Binding IsInSefiratHaOmer}" Value="True">

<Setter Property="Visibility" Value="Collapsed"/>

</DataTrigger>

<DataTrigger Binding="{Binding IsAlertActive}" Value="True">

<Setter Property="Visibility" Value="Collapsed"/>

</DataTrigger>

</Style.Triggers>

</Style>

</ItemsControl.Style>

<!-- ... rest of ItemsControl ... -->

</ItemsControl>

```

\* \*\*Line \~215 (Modified `Grid` for alert view):\*\*

Change the `Visibility` binding to include the `IsInSefiratHaOmer` check.

```xml

<Grid Visibility="{Binding IsAlertActive, Converter={StaticResource BoolToVisibility}}">

<Grid.Style>

<Style TargetType="Grid">

<Setter Property="Visibility" Value="Visible"/>

<Style.Triggers>

<DataTrigger Binding="{Binding IsInSefiratHaOmer}" Value="True">

<Setter Property="Visibility" Value="Collapsed"/>

</DataTrigger>

</Style.Triggers>

</Style>

</Grid.Style>

<!-- ... rest of Grid ... -->

</Grid>

```

\* \*\*Line \~290 (New `Grid` for Omer Display):\*\*

Add a new `Grid` that will display the Omer status, visible only when `IsInSefiratHaOmer` is true and `IsAlertActive` is false.

```xml

<!-- NEW: Sefirat Ha'Omer Display -->

<Grid Grid.Row="2" Grid.Column="1"

Visibility="{Binding IsInSefiratHaOmer, Converter={StaticResource BoolToVisibility}}">

<Grid.Style>

<Style TargetType="Grid">

<Setter Property="Visibility" Value="Collapsed"/>

<Style.Triggers>

<MultiDataTrigger>

<MultiDataTrigger.Conditions>

<Condition Binding="{Binding IsInSefiratHaOmer}" Value="True"/>

<Condition Binding="{Binding IsAlertActive}" Value="False"/>

</MultiDataTrigger.Conditions>

<Setter Property="Visibility" Value="Visible"/>

</MultiDataTrigger>

</Style.Triggers>

</Style>

</Grid.Style>

<Border CornerRadius="20" BorderThickness="5" BorderBrush="#FFD700" Margin="20">

<Border.Background>

<ImageBrush ImageSource="{Binding Converter={StaticResource ImageResourceConverter}, ConverterParameter=haomerBackground}"

Stretch="UniformToFill"/>

</Border.Background>

<TextBlock Text="{Binding OmerStatusText}"

VerticalAlignment="Center"

HorizontalAlignment="Center"

TextAlignment="Center"

TextWrapping="Wrap"

Foreground="White"

FontWeight="Bold"

FontSize="48">

<TextBlock.Effect>

<DropShadowEffect Color="Black" BlurRadius="10" ShadowDepth="5" Opacity="0.7"/>

</TextBlock.Effect>

</TextBlock>

</Border>

</Grid>

```

-----

\*\*Important Notes After Applying Changes:\*\*

1. \*\*Rebuild:\*\* After making all these changes, perform a \*\*Clean Solution\*\* followed by a \*\*Rebuild Solution\*\* in Visual Studio to ensure all new files, settings, and code are compiled correctly.

2. \*\*`haomerBackground.png`:\*\* Make sure you actually add `haomerBackground.png` to your `Assets` folder and then drag it into `Properties/Resources.resx` in Visual Studio, naming it `haomerBackground`.

3. \*\*Testing Omer:\*\* To test the Omer display, you might need to temporarily change your system date or modify the `GetSefiratHaOmerStatus` function to return a test value for debugging purposes.

4. \*\*Layout for 2 Slots:\*\* When `UseTwoTimeSlots` is true, the `UniformGrid` will adapt. You might want to fine-tune the `Margin` and `Padding` for the `TimeBoxStyle` and `BigTimeBoxStyle` to ensure optimal visual appeal for both 2-slot and 4-slot layouts.

5. \*\*`InitializeData()` Call:\*\* I noticed `InitializeData()` is commented out in `MainWindow.xaml.cs` constructor and only called in `MainWindow\_Loaded`. This is good practice. I've ensured `LoadFromExcel()` (which now includes Omer status update) is called within `InitializeData()`.

These changes should give you the desired functionality.

```xml

<# Views/MainWindow.xaml #>

<Window x:Class="EOTReminder.Views.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"

xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"

xmlns:vm="clr-namespace:EOTReminder.ViewModels"

xmlns:local="clr-namespace:EOTReminder.Views"

xmlns:conv="clr-namespace:EOTReminder.Converters"

mc:Ignorable="d"

Title="Zmanim Reminder"

Width="950" Height="650"

Background="Transparent"

WindowStartupLocation="CenterScreen"

WindowStyle="None" <!-- RE-ENABLED: Removes standard window chrome -->

WindowState="Maximized" <!-- RE-ENABLED: Makes the window full screen -->

Topmost="True" <!-- RE-ENABLED: Keeps the window on top of others -->

ResizeMode="NoResize" <!-- RE-ENABLED: Prevents resizing -->

Loaded="MainWindow\_Loaded"

Closing="MainWindow\_Closing"> <!-- NEW: Added Closing event handler -->

<Window.Resources>

<BooleanToVisibilityConverter x:Key="BoolToVisibility"/>

<conv:BoolToBrushConverter x:Key="BoolToBrushConverter"/>

<conv:ResourceToImageSourceConverter x:Key="ImageResourceConverter"/> <!-- Image Converter -->

<Style x:Key="TimeBoxStyle" TargetType="Border">

<!-- Background with 20% transparency (80% opaque) -->

<Setter Property="Background" Value="#CCFFFFFF"/>

<Setter Property="CornerRadius" Value="15"/>

<Setter Property="Padding" Value="10"/> <!-- Restored to 10 for better spacing -->

<!-- Line 20: Removed fixed Margin from here -->

<!-- Opacity removed from Border to keep content fully opaque, applied to Background brush instead -->

<Setter Property="BorderBrush" Value="#999"/>

<Setter Property="BorderThickness" Value="5"/>

<Setter Property="Effect">

<Setter.Value>

<DropShadowEffect Color="Black" BlurRadius="5" ShadowDepth="2" Opacity="0.3"/>

</Setter.Value>

</Setter>

</Style>

<!-- Style for big highlighted EO -->

<Style x:Key="BigTimeBoxStyle" TargetType="Border" BasedOn="{StaticResource TimeBoxStyle}">

<Setter Property="Padding" Value="20"/>

<!-- Line 26: Removed fixed Margin from here -->

<Setter Property="BorderThickness" Value="6"/>

<!-- Highlighted background with 20% transparency (80% opaque) -->

<Setter Property="Background" Value="#CCFFF3CD"/>

<Setter Property="Effect">

<Setter.Value>

<DropShadowEffect ShadowDepth="4" Opacity="0.4" BlurRadius="8" Color="#FFD700"/>

</Setter.Value>

</Setter>

</Style>

<!-- Text style for descriptions -->

<Style x:Key="DescriptionTextStyle" TargetType="TextBlock">

<Setter Property="FontSize" Value="14"/> <!-- RESTORED: Added FontSize -->

<Setter Property="FontWeight" Value="Normal"/>

<Setter Property="TextAlignment" Value="Center"/>

<Setter Property="TextWrapping" Value="Wrap"/>

<Setter Property="Foreground" Value="#333333"/>

</Style>

<!-- Text style for times -->

<Style x:Key="TimeTextStyle" TargetType="TextBlock">

<Setter Property="FontSize" Value="28"/> <!-- RESTORED: Added FontSize -->

<Setter Property="FontWeight" Value="Bold"/>

<Setter Property="TextAlignment" Value="Center"/>

<Setter Property="Foreground" Value="#333333"/>

</Style>

<!-- Text style for countdown -->

<Style x:Key="CountdownTextStyle" TargetType="TextBlock">

<Setter Property="FontSize" Value="16"/> <!-- RESTORED: Added FontSize -->

<Setter Property="FontWeight" Value="Normal"/>

<Setter Property="TextAlignment" Value="Center"/>

<Setter Property="Foreground" Value="Red"/>

</Style>

<!-- Text style for "Passed" status -->

<Style x:Key="StatusTextStyle" TargetType="TextBlock">

<Setter Property="FontSize" Value="14"/> <!-- RESTORED: Added FontSize -->

<Setter Property="FontWeight" Value="Bold"/>

<Setter Property="TextAlignment" Value="Center"/>

<Setter Property="Foreground" Value="Red"/>

</Style>

<!-- REMOVED: BitmapImage definitions for icons using x:Key -->

<!-- These are no longer needed as the converter will be used directly on Image.Source -->

</Window.Resources>

<!-- REMOVED: Window.DataContext binding here. It is now set in MainWindow.xaml.cs -->

<Grid>

<!-- Background - MODIFIED: Now using ImageResourceConverter for background -->

<Grid.Background>

<ImageBrush ImageSource="{Binding Converter={StaticResource ImageResourceConverter}, ConverterParameter=background}" Stretch="UniformToFill"/>

</Grid.Background>

<Grid.RowDefinitions>

<RowDefinition Height="0.11\*"/> <!-- For header content -->

<RowDefinition Height="0.04\*"/> <!-- Space between header and main content -->

<RowDefinition Height="0.73\*"/> <!-- Main content (EOS/EOT times) -->

<RowDefinition Height="0.04\*"/> <!-- Space between main content and footer -->

<RowDefinition Height="0.12\*"/> <!-- For footer -->

</Grid.RowDefinitions>

<Grid.ColumnDefinitions>

<ColumnDefinition Width="0.05\*"/> <!-- Left margin/padding -->

<ColumnDefinition Width="0.9\*"/> <!-- Central content area -->

<ColumnDefinition Width="0.05\*"/> <!-- Right margin/padding -->

</Grid.ColumnDefinitions>

<!-- Header Section (Language Selector + Special Times + Close Button) -->

<!-- This Border now spans all columns in Row 0 -->

<Border Grid.Row="0" Grid.Column="0" Grid.ColumnSpan="3" Background="#F0F0F0" Padding="5">

<Grid>

<Grid.ColumnDefinitions>

<ColumnDefinition Width="100"/> <!-- Fixed width for hidden button area -->

<ColumnDefinition Width="\*"/> <!-- Remaining space for content -->

<ColumnDefinition Width="Auto"/> <!-- For the new Close Application button -->

</Grid.ColumnDefinitions>

<!-- Hidden Button Area for Options Page -->

<Border Grid.Column="0" HorizontalAlignment="Left" VerticalAlignment="Top"

Width="100" Height="100" Background="Transparent"

MouseLeftButtonDown="HiddenOptionsButton\_MouseLeftButtonDown"

Panel.ZIndex="100">

<!-- This Border acts as the hidden button area -->

</Border>

<StackPanel Grid.Column="1" Orientation="Vertical" HorizontalAlignment="Stretch">

<!-- Language Selection -->

<StackPanel Orientation="Horizontal" HorizontalAlignment="Center" Margin="0,5">

<TextBlock Text="Select Language:" VerticalAlignment="Center" Margin="0,0,10,0" FontSize="12" />

<ComboBox x:Name="LanguageComboBox" Width="100" SelectedIndex="0" SelectionChanged="LanguageComboBox\_SelectionChanged" FontSize="10">

<ComboBoxItem Content="English" Tag="en"/>

<ComboBoxItem Content="עברית" Tag="he"/>

</ComboBox>

</StackPanel>

<!-- Special Times (Sunrise, Midday, Sunset) -->

<Grid Margin="5,5" HorizontalAlignment="Stretch">

<Grid.ColumnDefinitions>

<ColumnDefinition Width="0.25\*"/>

<ColumnDefinition Width="0.125\*"/>

<ColumnDefinition Width="0.25\*"/>

<ColumnDefinition Width="0.125\*"/>

<ColumnDefinition Width="0.25\*"/>

</Grid.ColumnDefinitions>

<StackPanel Grid.Column="0" Orientation="Vertical" HorizontalAlignment="Center">

<Viewbox MaxHeight="36">

<TextBlock Text="נץ החמה" FontSize="12"/>

</Viewbox>

<Viewbox MaxHeight="36">

<TextBlock Text="{Binding Sunrise}" FontSize="12" FontWeight="Bold"/>

</Viewbox>

</StackPanel>

<!-- MODIFIED: Source now uses the converter directly -->

<Image Grid.Column="1" Width="48" Height="48" Margin="10,0" Source="{Binding Converter={StaticResource ImageResourceConverter}, ConverterParameter=clock}" VerticalAlignment="Center"/>

<StackPanel Grid.Column="2" Orientation="Vertical" HorizontalAlignment="Center">

<Viewbox MaxHeight="36">

<TextBlock Text="חצות" FontSize="12"/>

</Viewbox>

<Viewbox MaxHeight="36">

<TextBlock Text="{Binding Midday}" FontSize="12" FontWeight="Bold"/>

</Viewbox>

</StackPanel>

<!-- MODIFIED: Source now uses the converter directly -->

<Image Grid.Column="3" Width="48" Height="48" Margin="10,0" Source="{Binding Converter={StaticResource ImageResourceConverter}, ConverterParameter=clock}" VerticalAlignment="Center"/>

<StackPanel Grid.Column="4" Orientation="Vertical" HorizontalAlignment="Center">

<Viewbox MaxHeight="36">

<TextBlock Text="שקיעה" FontSize="12"/>

</Viewbox>

<Viewbox MaxHeight="36">

<TextBlock Text="{Binding Sunset}" FontSize="12" FontWeight="Bold"/>

</Viewbox>

</StackPanel>

</Grid>

</StackPanel>

<!-- NEW: Close Application Button -->

<Button Grid.Column="2" Content="Close Application"

Command="{Binding CloseApplicationCommand}"

CommandParameter="{Binding RelativeSource={RelativeSource Mode=FindAncestor, AncestorType={x:Type Window}}}"

HorizontalAlignment="Right" VerticalAlignment="Top"

Margin="10" Padding="10,5"

Background="#FFD700" Foreground="Black" FontWeight="Bold"

BorderBrush="#FFC700" BorderThickness="2" CornerRadius="8">

<Button.Effect>

<DropShadowEffect Color="Black" BlurRadius="5" ShadowDepth="2" Opacity="0.3"/>

</Button.Effect>

</Button>

</Grid>

</Border>

<!-- Main Content -->

<Grid Grid.Row="2" Grid.Column="1">

<!-- NEW: Added Row and Column Definitions for this inner Grid -->

<Grid.RowDefinitions>

<RowDefinition Height="\*"/>

</Grid.RowDefinitions>

<Grid.ColumnDefinitions>

<ColumnDefinition Width="\*"/>

</Grid.ColumnDefinitions>

<!-- Normal 2x2 grid - visible only if no alert and NOT in Sefirat Ha'Omer -->

<ItemsControl ItemsSource="{Binding TimeSlots}">

<ItemsControl.Style>

<Style TargetType="ItemsControl">

<Setter Property="Visibility" Value="Visible"/>

<Style.Triggers>

<DataTrigger Binding="{Binding IsInSefiratHaOmer}" Value="True">

<Setter Property="Visibility" Value="Collapsed"/>

</DataTrigger>

<DataTrigger Binding="{Binding IsAlertActive}" Value="True">

<Setter Property="Visibility" Value="Collapsed"/>

</DataTrigger>

</Style.Triggers>

</Style>

</ItemsControl.Style>

<ItemsControl.ItemsPanel>

<ItemsPanelTemplate>

<UniformGrid Columns="{Binding NormalGridColumns}" Rows="{Binding NormalGridRows}"/>

</ItemsPanelTemplate>

</ItemsControl.ItemsPanel>

<ItemsControl.ItemTemplate>

<DataTemplate>

<Grid Padding="5">

<Border Style="{StaticResource TimeBoxStyle}"

BorderBrush="{Binding Highlight, Converter={StaticResource BoolToBrushConverter}}"

BorderThickness="9" Margin="25">

<Grid>

<Grid.RowDefinitions>

<RowDefinition Height="0.015\*"/>

<RowDefinition Height="0.21\*"/>

<RowDefinition Height="0.01\*"/>

<RowDefinition Height="0.54\*"/>

<RowDefinition Height="0.01\*"/>

<RowDefinition Height="0.21\*"/>

<RowDefinition Height="0.015\*"/>

</Grid.RowDefinitions>

<Viewbox Grid.Row="1">

<TextBlock Text="{Binding Description}" FontSize="14" FontWeight="Bold" TextAlignment="Center"/>

</Viewbox>

<Viewbox Grid.Row="3" MaxHeight="230">

<TextBlock Text="{Binding Time, StringFormat='HH:mm:ss'}" FontSize="52" TextAlignment="Center"/>

</Viewbox>

<Viewbox Grid.Row="5">

<TextBlock Text="{Binding PassedText}"

Foreground="Red"

FontSize="24"

FontWeight="Bold"

Visibility="{Binding IsPassed, Converter={StaticResource BoolToVisibility}}"

TextAlignment="Center"/>

</Viewbox>

</Grid>

</Border>

</Grid>

</DataTemplate>

</ItemsControl.ItemTemplate>

</ItemsControl>

<!-- Alert mode - split top and bottom rows - visible only if alert is active and NOT in Sefirat Ha'Omer -->

<Grid>

<Grid.Style>

<Style TargetType="Grid">

<Setter Property="Visibility" Value="Collapsed"/>

<Style.Triggers>

<DataTrigger Binding="{Binding IsAlertActive}" Value="True">

<Setter Property="Visibility" Value="Visible"/>

</DataTrigger>

<DataTrigger Binding="{Binding IsInSefiratHaOmer}" Value="True">

<Setter Property="Visibility" Value="Collapsed"/>

</DataTrigger>

</Style.Triggers>

</Style>

</Grid.Style>

<Grid.RowDefinitions>

<RowDefinition Height="1.5\*"/>

<RowDefinition Height="\*"/>

</Grid.RowDefinitions>

<!-- Top big EO - binds to TopSlots (which will contain one item) -->

<ItemsControl ItemsSource="{Binding TopSlots}" Grid.Row="0" Padding="5">

<ItemsControl.ItemsPanel>

<ItemsPanelTemplate>

<UniformGrid Columns="1" Rows="1"/>

</ItemsPanelTemplate>

</ItemsControl.ItemsPanel>

<ItemsControl.ItemTemplate>

<DataTemplate>

<Grid>

<Grid.ColumnDefinitions>

<ColumnDefinition Width="0.28\*"/>

<ColumnDefinition Width="0.44\*"/>

<ColumnDefinition Width="0.28\*"/>

</Grid.ColumnDefinitions>

<Viewbox Grid.Column="0" HorizontalAlignment="Right">

<Image Width="180" Height="500"

Source="{Binding Converter={StaticResource ImageResourceConverter}, ConverterParameter=sandclock}"

Margin="0,0,8,0"

VerticalAlignment="Center"

Visibility="{Binding ShowSandClock, Converter={StaticResource BoolToVisibility}}"

Stretch="Uniform"/>

</Viewbox>

<Border Grid.Column="1" Margin="0,0,0,8"

Style="{StaticResource BigTimeBoxStyle}"

BorderBrush="{Binding Highlight, Converter={StaticResource BoolToBrushConverter}}">

<Grid>

<Grid.RowDefinitions>

<RowDefinition Height="0.015\*"/>

<RowDefinition Height="0.15\*"/>

<RowDefinition Height="0.01\*"/>

<RowDefinition Height="0.4\*"/>

<RowDefinition Height="0.01\*"/>

<RowDefinition Height="0.4\*"/>

<RowDefinition Height="0.015\*"/>

</Grid.RowDefinitions>

<Viewbox Grid.Row="1" MaxHeight="72">

<TextBlock Text="{Binding Description}" FontSize="18" FontWeight="Bold" TextAlignment="Center"/>

</Viewbox>

<Viewbox Grid.Row="3" MaxHeight="200">

<TextBlock Text="{Binding Time, StringFormat='HH:mm:ss'}" FontSize="85" TextAlignment="Center"/>

</Viewbox>

<Viewbox Grid.Row="5" MaxHeight="200">

<TextBlock Text="{Binding CountdownText, Mode=OneWay}" FontSize="85" Foreground="Red" TextAlignment="Center"/>

</Viewbox>

</Grid>

</Border>

</Grid>

</DataTemplate>

</ItemsControl.ItemTemplate>

</ItemsControl>

<!-- Bottom smaller EOs - binds to BottomSlots -->

<ItemsControl ItemsSource="{Binding BottomSlots}" Grid.Row="1" Padding="2">

<ItemsControl.ItemsPanel>

<ItemsPanelTemplate>

<UniformGrid Columns="{Binding BottomGridColumns}" Rows="1"/>

</ItemsPanelTemplate>

</ItemsControl.ItemsPanel>

<ItemsControl.ItemTemplate>

<DataTemplate>

<Grid>

<Border Margin="5,8,5,8"

Style="{StaticResource TimeBoxStyle}"

BorderBrush="{Binding Highlight, Converter={StaticResource BoolToBrushConverter}}">

<Grid>

<Grid.RowDefinitions>

<RowDefinition Height="0.015\*"/>

<RowDefinition Height="0.2\*"/>

<RowDefinition Height="0.01\*"/>

<RowDefinition Height="0.44\*"/>

<RowDefinition Height="0.01\*"/>

<RowDefinition Height="0.3\*"/>

<RowDefinition Height="0.015\*"/>

</Grid.RowDefinitions>

<Viewbox Grid.Row="1">

<TextBlock Text="{Binding Description}" FontSize="14" FontWeight="Bold" TextAlignment="Center"/>

</Viewbox>

<Viewbox Grid.Row="3">

<TextBlock Text="{Binding Time, StringFormat='HH:mm:ss'}" FontSize="24" TextAlignment="Center"/>

</Viewbox>

<Viewbox Grid.Row="5">

<TextBlock Text="{Binding PassedText}"

Foreground="Red"

FontSize="32"

FontWeight="Bold"

Visibility="{Binding IsPassed, Converter={StaticResource BoolToVisibility}}"

TextAlignment="Center"/>

</Viewbox>

</Grid>

</Border>

</Grid>

</DataTemplate>

</ItemsControl.ItemTemplate>

</ItemsControl>

</Grid>

<!-- NEW: Sefirat Ha'Omer Display - visible only if IsInSefiratHaOmer is true and IsAlertActive is false -->

<Grid Grid.Row="0" Grid.Column="0" Grid.RowSpan="2" Grid.ColumnSpan="2"> <!-- Spans the entire content area -->

<Grid.Style>

<Style TargetType="Grid">

<Setter Property="Visibility" Value="Collapsed"/>

<Style.Triggers>

<MultiDataTrigger>

<MultiDataTrigger.Conditions>

<Condition Binding="{Binding IsInSefiratHaOmer}" Value="True"/>

<Condition Binding="{Binding IsAlertActive}" Value="False"/>

</MultiDataTrigger.Conditions>

<Setter Property="Visibility" Value="Visible"/>

</MultiDataTrigger>

</Style.Triggers>

</Style>

</Grid.Style>

<Border CornerRadius="20" BorderThickness="5" BorderBrush="#FFD700" Margin="20">

<Border.Background>

<ImageBrush ImageSource="{Binding Converter={StaticResource ImageResourceConverter}, ConverterParameter=haomerBackground}"

Stretch="UniformToFill"/>

</Border.Background>

<TextBlock Text="{Binding OmerStatusText}"

VerticalAlignment="Center"

HorizontalAlignment="Center"

TextAlignment="Center"

TextWrapping="Wrap"

Foreground="White"

FontWeight="Bold"

FontSize="48">

<TextBlock.Effect>

<DropShadowEffect Color="Black" BlurRadius="10" ShadowDepth="5" Opacity="0.7"/>

</TextBlock.Effect>

</TextBlock>

</Border>

</Grid>

</Grid>

<!-- Footer -->

<Border Grid.Row="4" Grid.Column="0" Grid.ColumnSpan="3" Background="#F0F0F0" Padding="5">

<Grid HorizontalAlignment="Stretch">

<Grid.ColumnDefinitions>

<ColumnDefinition Width="0.08\*" />

<ColumnDefinition Width="0.2\*" />

<ColumnDefinition Width="0.08\*" />

<ColumnDefinition Width="0.34\*" />

<ColumnDefinition Width="0.08\*" />

<ColumnDefinition Width="0.23\*" />

<ColumnDefinition Width="0.02\*"/>

</Grid.ColumnDefinitions>

<Viewbox Grid.Column="0" HorizontalAlignment="Right" MaxHeight="72">

<Image Source="{Binding Converter={StaticResource ImageResourceConverter}, ConverterParameter=calendar}" Width="40" Height="40" Margin="5" Stretch="Uniform"/>

</Viewbox>

<Viewbox Grid.Column="1" HorizontalAlignment="Center" MaxHeight="36">

<TextBlock Text="{Binding TodayDate}" FontSize="24" FontWeight="SemiBold"/>

</Viewbox>

<Viewbox Grid.Column="2" HorizontalAlignment="Center" MaxHeight="72">

<Image Width="48" Height="48" Margin="10,0" Source="{Binding Converter={StaticResource ImageResourceConverter}, ConverterParameter=clock}" VerticalAlignment="Center"/>

</Viewbox>

<Viewbox Grid.Column="3" HorizontalAlignment="Center">

<TextBlock Text="{Binding CurrentTime}" FontSize="42" FontWeight="SemiBold" />

</Viewbox>

<Viewbox Grid.Column="4" HorizontalAlignment="Right" MaxHeight="72">

<Image Source="{Binding Converter={StaticResource ImageResourceConverter}, ConverterParameter=calendar}" Width="40" Height="40" Margin="5" Stretch="Uniform"/>

</Viewbox>

<Viewbox Grid.Column="5" HorizontalAlignment="Center" MaxHeight="36">

<TextBlock Text="{Binding HebrewDate}" FontSize="24" FontWeight="SemiBold"/>

</Viewbox>

</Grid>

</Border>

</Grid>

</Window>

<# MainWindow.xaml.cs #>

using System.Windows;

using System.Windows.Controls;

using System.Windows.Input; // Added for MouseButtonEventArgs

using EOTReminder.ViewModels;

using EOTReminder.Views; // Added to reference the new OptionsWindow

namespace EOTReminder.Views

{

public partial class MainWindow : Window

{

private MainViewModel \_viewModel => DataContext as MainViewModel;

public MainWindow()

{

InitializeComponent();

this.DataContext = new MainViewModel();

}

private void MainWindow\_Loaded(object sender, RoutedEventArgs e)

{

if (Properties.Settings.Default.IsFirstRun)

{

OpenOptionsPage();

Properties.Settings.Default.IsFirstRun = false;

Properties.Settings.Default.Save();

}

\_viewModel.InitializeData();

}

// NEW: Event handler for the Window.Closing event

private void MainWindow\_Closing(object sender, System.ComponentModel.CancelEventArgs e)

{

// Ensure the timer is stopped when the window is closing

\_viewModel?.StopTimer();

}

private void LanguageComboBox\_SelectionChanged(object sender, SelectionChangedEventArgs e)

{

if (e.AddedItems.Count > 0 && e.AddedItems[0] is ComboBoxItem selected)

{

string lang = selected.Tag?.ToString();

if (!string.IsNullOrWhiteSpace(lang))

{

\_viewModel?.SwitchLanguage(lang);

}

}

}

private void HiddenOptionsButton\_MouseLeftButtonDown(object sender, MouseButtonEventArgs e)

{

OpenOptionsPage();

}

private void OpenOptionsPage()

{

OptionsWindow optionsWindow = new OptionsWindow();

optionsWindow.Owner = this;

optionsWindow.WindowStartupLocation = WindowStartupLocation.CenterOwner;

optionsWindow.ShowDialog();

}

}

}

<# ViewModels/MainViewModel.cs #>

using EOTReminder.Models;

using System;

using System.Collections.ObjectModel;

using System.ComponentModel;

using System.Globalization;

using System.IO;

using System.Linq;

using System.Media;

using System.Reflection;

using System.Runtime.CompilerServices;

using System.Timers;

using ExcelDataReader; // Ensure this NuGet package is installed

using System.Collections.Generic;

using System.Text;

using System.Data;

using System.Windows; // For Application.Current.Dispatcher.Invoke

using EOTReminder.Utilities; // Added for Logger

using System.Windows.Input; // NEW: Added for ICommand

using EOTReminder.Commands; // NEW: Added for RelayCommand (assuming you have this or will add it)

namespace EOTReminder.ViewModels

{

public class MainViewModel : INotifyPropertyChanged

{

// TimeSlots will always hold all 4 EO times

public ObservableCollection<TimeSlot> TimeSlots { get; set; } = new ObservableCollection<TimeSlot>();

// TopSlots will hold the single highlighted EO time

public ObservableCollection<TimeSlot> TopSlots { get; } = new ObservableCollection<TimeSlot>();

// BottomSlots will hold the other 3 EO times when one is highlighted

public ObservableCollection<TimeSlot> BottomSlots { get; } = new ObservableCollection<TimeSlot>();

private bool \_isAlertActive;

public bool IsAlertActive // Controls visibility of normal 2x2 grid vs. alert layout

{

get => \_isAlertActive;

set { \_isAlertActive = value; OnPropertyChanged(); }

}

// NEW: Properties for Sefirat Ha'Omer display

private string \_omerStatusText;

public string OmerStatusText

{

get => \_omerStatusText;

set { \_omerStatusText = value; OnPropertyChanged(); }

}

private bool \_isInSefiratHaOmer;

public bool IsInSefiratHaOmer

{

get => \_isInSefiratHaOmer;

set { \_isInSefiratHaOmer = value; OnPropertyChanged(); }

}

// NEW: Properties for dynamic grid dimensions (for 2 or 4 slots)

private int \_normalGridColumns = 2;

public int NormalGridColumns

{

get => \_normalGridColumns;

set { \_normalGridColumns = value; OnPropertyChanged(); }

}

private int \_normalGridRows = 2;

public int NormalGridRows

{

get => \_normalGridRows;

set { \_normalGridRows = value; OnPropertyChanged(); }

}

private int \_bottomGridColumns = 3; // Default for 3 items in bottom row

public int BottomGridColumns

{

get => \_bottomGridColumns;

set { \_bottomGridColumns = value; OnPropertyChanged(); }

}

public string TodayDate => DateTime.Now.ToString("dd/MM/yyyy");

public string CurrentTime => DateTime.Now.ToString("HH:mm:ss");

// Private DateTime fields to hold the actual time values for calculations

private DateTime \_internalSunriseTime;

private DateTime \_internalMiddayTime;

private DateTime \_internalSunsetTime;

private string \_hebrewDateString; // Private field for Hebrew date string

private bool \_hasReloadedForCurrentSunriseCycle = false;

private DateTime \_currentSunriseForReloadCheck = DateTime.MinValue; // Stores the sunrise time for which data is currently loaded

// Public string properties for UI binding

public string HebrewDate

{

get => \_hebrewDateString;

private set { \_hebrewDateString = value; OnPropertyChanged(); }

}

public string Sunrise

{

get => \_internalSunriseTime == DateTime.MinValue ? "N/A" : \_internalSunriseTime.ToString("HH:mm:ss");

private set { /\* Setter is not used as \_internalSunriseTime is set directly \*/ }

}

public string Midday

{

get => \_internalMiddayTime == DateTime.MinValue ? "N/A" : \_internalMiddayTime.ToString("HH:mm:ss");

private set { /\* Setter is not used as \_internalMiddayTime is set directly \*/ }

}

public string Sunset

{

get => \_internalSunsetTime == DateTime.MinValue ? "N/A" : \_internalSunsetTime.ToString("HH:mm:ss");

private set { /\* Setter is not used as \_internalSunsetTime is set directly \*/ }

}

public event PropertyChangedEventHandler PropertyChanged;

private Timer \_timer;

private string \_currentLang = "he"; // Default to Hebrew as per original code

private readonly Dictionary<string, Dictionary<string, string>> \_translations =

new Dictionary<string, Dictionary<string, string>>()

{

["en"] = new Dictionary<string, string>()

{

["a2EOS1"] = "End of Shema 1", // Added numbers for clarity

["a1EOS2"] = "End of Shema 2",

["b2EOT1"] = "End of Prayer 1",

["b1EOT2"] = "End of Prayer 2",

["Passed"] = "Passed"

},

["he"] = new Dictionary<string, string>()

{

["a2EOS1"] = "סו\"ז קר\"ש מג\"א",

["a1EOS2"] = "סו\"ז קר\"ש תניא גר\"א",

["b2EOT1"] = "סו\"ז תפילה מג\"א",

["b1EOT2"] = "סו\"ז תפילה תניא גר\"א",

["Passed"] = "עבר זמנו", // Corrected key to "Passed"

}

};

// NEW: Command for the "Close Application" button

public ICommand CloseApplicationCommand { get; }

public MainViewModel()

{

// Required for ExcelDataReader to handle older Excel formats

System.Text.Encoding.RegisterProvider(System.Text.CodePagesEncodingProvider.Instance);

// Initialize commands

CloseApplicationCommand = new RelayCommand(CloseApplication);

}

public void InitializeData()

{

LoadFromExcel();

InitTimer();

}

private void InitTimer()

{

// Ensure any existing timer is stopped and disposed before creating a new one

StopTimer();

\_timer = new Timer(1000); // Tick every 1 second

\_timer.Elapsed += (s, e) =>

{

Application.Current.Dispatcher.Invoke(() => // Ensure UI updates happen on the UI thread

{

// Check for Shabbos before processing alerts

if (DateTime.Today.DayOfWeek == DayOfWeek.Saturday && !Properties.Settings.Default.AlertOnShabbos)

{

// If it's Shabbos and alerts are disabled, just update time and return

OnPropertyChanged(nameof(CurrentTime));

UpdateSlotCollections(); // Still update collections for display logic

return;

}

// Update Omer Status

OmerStatusText = Utilities.HebrewDateChecker.GetSefiratHaOmerStatus(DateTime.Now);

IsInSefiratHaOmer = !string.IsNullOrEmpty(OmerStatusText);

// Only process time slots if not in Omer or if an alert is active (Omer display overrides normal)

if (!IsInSefiratHaOmer || IsAlertActive) // IsAlertActive will be true if a visual alert is triggered

{

foreach (var slot in TimeSlots)

{

slot.Countdown = slot.Time - DateTime.Now; // Update countdown

int firstAlertMin = Properties.Settings.Default.FirstAlertMinutes;

int secondAlertMin = Properties.Settings.Default.SecondAlertMinutes;

int visualAlertMin = Properties.Settings.Default.VisualAlertMinutes;

if (!slot.IsPassed && slot.Countdown <= TimeSpan.Zero)

{

// Time has just passed

slot.Highlight = false;

slot.IsPassed = true;

slot.CountdownText = ""; // Clear countdown

slot.ShowSandClock = false;

slot.IsIn30MinAlert = false; // Reset alert state

// Reset alert flags for this slot

slot.AlertFlags["Visual"] = false; // Using "Visual" for visual alert flag

slot.AlertFlags["FirstAudio"] = false;

slot.AlertFlags["SecondAudio"] = false;

// If the passed slot was the active alert, deactivate general alert

if (TopSlots.Contains(slot))

{

IsAlertActive = false;

}

}

else if (!slot.IsPassed)

{

// Time is still upcoming

// Use VisualAlertMinutes from settings for visual alert

if (visualAlertMin > 0 &&

slot.Countdown.TotalMinutes <= visualAlertMin &&

!slot.AlertFlags["Visual"]) // Use a new flag for visual alert

{

slot.IsIn30MinAlert = true; // This will trigger the UI layout change

slot.Highlight = true;

slot.ShowSandClock = true;

slot.AlertFlags["Visual"] = true; // Set the visual alert flag

IsAlertActive = true; // Activate general alert state

}

else if (slot.Countdown.TotalMinutes > visualAlertMin && slot.AlertFlags["Visual"])

{

// If it was in visual alert but now it's outside, reset

slot.IsIn30MinAlert = false;

slot.Highlight = false;

slot.ShowSandClock = false;

slot.AlertFlags["Visual"] = false; // Allow re-trigger if time is reset/reloaded

// Don't set IsAlertActive to false here, as another slot might be active

}

// Update countdown text for all active slots

slot.CountdownText = string.Format("{0:D2}:{1:D2}",

(int)Math.Floor(slot.Countdown.TotalMinutes),

slot.Countdown.Seconds);

// Use specific audio paths from settings

string firstAlertAudioPath = GetAlertAudioPath(slot.Id, "First");

string secondAlertAudioPath = GetAlertAudioPath(slot.Id, "Second");

if (firstAlertMin > 0 &&

slot.Countdown.TotalMinutes <= firstAlertMin &&

slot.Countdown.TotalMinutes > (firstAlertMin - 1) &&

!slot.AlertFlags["FirstAudio"]) // Use a new flag for first audio alert

{

PlayAlert(firstAlertAudioPath);

slot.AlertFlags["FirstAudio"] = true;

}

if (secondAlertMin > 0 &&

slot.Countdown.TotalMinutes <= secondAlertMin &&

slot.Countdown.TotalMinutes > (secondAlertMin - 1) &&

!slot.AlertFlags["SecondAudio"]) // Use a new flag for second audio alert

{

PlayAlert(secondAlertAudioPath);

slot.AlertFlags["SecondAudio"] = true;

}

}

}

}

// Step 1: Ensure \_internalSunriseTime is always updated for the current Gregorian day.

// This is crucial if the application runs continuously past midnight,

// as \_internalSunriseTime would otherwise remain from the previous day.

// Also check if \_currentSunriseForReloadCheck is MinValue for initial load.

if (\_internalSunriseTime.Date != DateTime.Today || \_currentSunriseForReloadCheck == DateTime.MinValue)

{

// It's a new Gregorian day, or \_internalSunriseTime hasn't been updated for today yet.

// Reload Excel data to get the correct sunrise time for today.

LoadFromExcel(); // This will update \_internalSunriseTime to today's actual sunrise

\_hasReloadedForCurrentSunriseCycle = false; // Reset the flag for the new day's cycle

\_currentSunriseForReloadCheck = \_internalSunriseTime; // Store this sunrise time as the basis for the current cycle

Logger.LogInfo($"New Gregorian day detected or initial load. Excel data reloaded to update current day's times. Sunrise: {\_internalSunriseTime:HH:mm:ss}");

}

// Now, \_internalSunriseTime is guaranteed to be for DateTime.Today.

// Step 2: Calculate the specific reload trigger time for today's sunrise.

DateTime reloadTriggerTime = \_internalSunriseTime.Subtract(TimeSpan.FromMinutes(72));

// Step 3: Check if it's time to perform the scheduled daily reload (72 minutes before sunrise).

// This condition ensures:

// 1. The current time is past the calculated trigger time.

// 2. We haven't already reloaded for \*this specific sunrise cycle\*.

// (We use \_hasReloadedForCurrentSunriseCycle to prevent multiple reloads within the same cycle).

if (DateTime.Now >= reloadTriggerTime && !\_hasReloadedForCurrentSunriseCycle)

{

Logger.LogInfo($"Triggering scheduled daily Excel reload. Current Time: {DateTime.Now:HH:mm:ss}, Reload Trigger Time: {reloadTriggerTime:HH:mm:ss}");

LoadFromExcel(); // Perform the actual scheduled reload

\_hasReloadedForCurrentSunriseCycle = true; // Mark that reload has happened for this cycle

\_currentSunriseForReloadCheck = \_internalSunriseTime; // Update the marker to the new sunrise time after reload

}

UpdateSlotCollections(); // Update the TopSlots/BottomSlots based on alert state

OnPropertyChanged(nameof(CurrentTime)); // Update current time in footer

});

};

\_timer.Start();

}

// NEW: Method to stop and dispose the timer

public void StopTimer()

{

if (\_timer != null)

{

\_timer.Stop();

\_timer.Dispose();

\_timer = null; // Set to null to prevent re-use of disposed timer

Logger.LogInfo("Timer stopped and disposed.");

}

}

// NEW: Method to handle the Close Application command

private void CloseApplication(object parameter)

{

if (parameter is Window window)

{

window.Close(); // This will trigger MainWindow\_Closing and then Application.Current.Shutdown()

}

else

{

Application.Current.Shutdown(); // Fallback for direct shutdown

}

Logger.LogInfo("Application close initiated.");

}

private void LoadFromExcel()

{

string path = Properties.Settings.Default.ExcelFilePath;

if (string.IsNullOrWhiteSpace(path))

{

path = Path.Combine(AppDomain.CurrentDomain.BaseDirectory, "DailyTimes.xlsx");

Logger.LogInfo($"Excel file path from settings is empty. Falling back to default: {path}");

}

if (!File.Exists(path))

{

Logger.LogWarning($"Excel file '{path}' not found. Loading mock data.");

LoadMock();

return;

}

try

{

using (var stream = File.Open(path, FileMode.Open, FileAccess.Read))

{

using (var reader = ExcelReaderFactory.CreateReader(stream))

{

var dataSet = reader.AsDataSet(new ExcelDataSetConfiguration()

{

ConfigureDataTable = \_ => new ExcelDataTableConfiguration()

{

UseHeaderRow = true

}

});

var table = dataSet.Tables[0];

if (table == null)

{

Logger.LogError("No data tables found in the Excel file. Loading mock data.");

LoadMock();

return;

}

var today = DateTime.Today;

DataRow todayRow = null;

int dateColumnIndex = -1;

for (int i = 0; i < table.Columns.Count; i++)

{

if (table.Columns[i].ColumnName.Equals("Date", StringComparison.OrdinalIgnoreCase))

{

dateColumnIndex = i;

break;

}

}

if (dateColumnIndex == -1)

{

Logger.LogError("'Date' column not found in Excel. Loading mock data.");

LoadMock();

return;

}

foreach (DataRow row in table.Rows)

{

if (row[dateColumnIndex] != DBNull.Value && DateTime.TryParse(row[dateColumnIndex].ToString(), out DateTime excelDate))

{

if (excelDate.Date == today.Date)

{

todayRow = row;

break;

}

}

}

if (todayRow == null)

{

Logger.LogWarning($"No entry found for today's date ({today.ToShortDateString()}) in '{path}'. Loading mock data.");

LoadMock();

return;

}

int GetColumnIndex(string columnName)

{

for (int i = 0; i < table.Columns.Count; i++)

{

if (table.Columns[i].ColumnName.Equals(columnName, StringComparison.OrdinalIgnoreCase))

{

return i;

}

}

return -1;

}

DateTime ParseTimeFromCell(DataRow row, string columnName)

{

int colIndex = GetColumnIndex(columnName);

if (colIndex != -1 && row[colIndex] != DBNull.Value)

{

string timeString = row[colIndex].ToString();

if (TimeSpan.TryParse(timeString, out TimeSpan timeSpan))

{

return today.Add(timeSpan);

}

else if (DateTime.TryParse(timeString, out DateTime dateTimeFromCell))

{

return today.Add(dateTimeFromCell.TimeOfDay);

}

}

return DateTime.MinValue;

}

TimeSlots.Clear();

AddSlot("a2EOS1", ParseTimeFromCell(todayRow, "EOS1"));

AddSlot("b2EOT1", ParseTimeFromCell(todayRow, "EOT1")); // Always add these two

if (!Properties.Settings.Default.UseTwoTimeSlots) // Only add if not using two slots

{

AddSlot("a1EOS2", ParseTimeFromCell(todayRow, "EOS2"));

AddSlot("b1EOT2", ParseTimeFromCell(todayRow, "EOT2"));

}

\_internalSunriseTime = ParseTimeFromCell(todayRow, "Sunrise");

\_internalMiddayTime = ParseTimeFromCell(todayRow, "Midday");

\_internalSunsetTime = ParseTimeFromCell(todayRow, "Sunset");

OnPropertyChanged(nameof(Sunrise));

OnPropertyChanged(nameof(Midday));

OnPropertyChanged(nameof(Sunset));

HebrewDate = GetHebrewJewishDateString(today, false);

OnPropertyChanged(nameof(HebrewDate));

// Update Omer Status

OmerStatusText = Utilities.HebrewDateChecker.GetSefiratHaOmerStatus(DateTime.Now);

IsInSefiratHaOmer = !string.IsNullOrEmpty(OmerStatusText);

if (TimeSlots.Any(s => s.Time == DateTime.MinValue) ||

\_internalSunriseTime == DateTime.MinValue || \_internalMiddayTime == DateTime.MinValue || \_internalSunsetTime == DateTime.MinValue)

{

Logger.LogWarning("Some times could not be parsed from Excel. Using mock data for missing values.");

}

UpdateGridDimensions(); // Update grid based on loaded slots

OnPropertyChanged(nameof(TimeSlots)); // Notify that TimeSlots have changed

}

}

}

catch (Exception ex)

{

Logger.LogError($"An error occurred while reading the Excel file: {ex.Message}. Loading mock data instead.", ex);

LoadMock();

}

foreach (var slot in TimeSlots)

{

// Initialize all new alert flags

slot.AlertFlags = new Dictionary<string, bool>()

{

["Visual"] = false, // For visual alert

["FirstAudio"] = false, // For first audio alert

["SecondAudio"] = false // For second audio alert

};

}

}

private void LoadMock()

{

TimeSlots.Clear();

var now = DateTime.Now;

AddSlot("a2EOS1", now.AddMinutes(5).AddSeconds(1));

AddSlot("b2EOT1", now.AddMinutes(20).AddSeconds(1));

if (!Properties.Settings.Default.UseTwoTimeSlots) // Only add if not using two slots

{

AddSlot("a1EOS2", now.AddMinutes(10).AddSeconds(1));

AddSlot("b1EOT2", now.AddMinutes(30).AddSeconds(1));

}

\_internalSunriseTime = now.Date.AddHours(6).AddMinutes(0);

\_internalMiddayTime = now.Date.AddHours(12).AddMinutes(0);

\_internalSunsetTime = now.Date.AddHours(19).AddMinutes(30);

HebrewDate = GetHebrewJewishDateString(now, false);

OnPropertyChanged(nameof(Sunrise));

OnPropertyChanged(nameof(Midday));

OnPropertyChanged(nameof(Sunset));

OnPropertyChanged(nameof(HebrewDate));

// Update Omer Status

OmerStatusText = Utilities.HebrewDateChecker.GetSefiratHaOmerStatus(DateTime.Now);

IsInSefiratHaOmer = !string.IsNullOrEmpty(OmerStatusText);

UpdateGridDimensions(); // Update grid based on loaded slots

OnPropertyChanged(nameof(TimeSlots)); // Notify that TimeSlots have changed

}

private void AddSlot(string id, DateTime time)

{

TimeSlots.Add(new TimeSlot

{

Id = id,

Description = \_translations[\_currentLang][id],

Time = time,

IsPassed = false,

CountdownText = "",

ShowSandClock = false,

Highlight = false,

IsIn30MinAlert = false, // This will be controlled by VisualAlertMinutes now

AlertFlags = new Dictionary<string, bool>()

{

["Visual"] = false,

["FirstAudio"] = false,

["SecondAudio"] = false

}

});

}

// NEW: Method to update grid dimensions based on UseTwoTimeSlots setting

private void UpdateGridDimensions()

{

if (Properties.Settings.Default.UseTwoTimeSlots)

{

NormalGridColumns = 1;

NormalGridRows = 2;

}

else

{

NormalGridColumns = 2;

NormalGridRows = 2;

}

// BottomGridColumns is handled in UpdateSlotCollections based on actual items

}

// NEW: Helper to get the correct audio path based on slot ID and alert type

private string GetAlertAudioPath(string slotId, string alertType) // alertType can be "First" or "Second"

{

switch (slotId)

{

case "a2EOS1":

return alertType == "First" ? Properties.Settings.Default.EOS1FirstAlertPath : Properties.Settings.Default.EOS1SecondAlertPath;

case "a1EOS2":

return alertType == "First" ? Properties.Settings.Default.EOS2FirstAlertPath : Properties.Settings.Default.EOS2SecondAlertPath;

case "b2EOT1":

return alertType == "First" ? Properties.Settings.Default.EOT1FirstAlertPath : Properties.Settings.Default.EOT1SecondAlertPath;

case "b1EOT2":

return alertType == "First" ? Properties.Settings.Default.EOT2FirstAlertPath : Properties.Settings.Default.EOT2SecondAlertPath;

default:

Logger.LogWarning($"No specific audio path found for slot ID: {slotId} and alert type: {alertType}. Using default.");

// Fallback to generic paths if specific ones are not set or ID is unknown

return alertType == "First" ?

(Properties.Settings.Default.FirstAlertMinutes > 0 ? "alertEOS2\_10.wav" : "") : // Assuming alertEOS2\_10.wav for generic first

(Properties.Settings.Default.SecondAlertMinutes > 0 ? "alertEOS2\_3.wav" : ""); // Assuming alertEOS2\_3.wav for generic second

}

}

// MODIFIED: PlayAlert now takes the full path directly

private void PlayAlert(string audioFilePath)

{

if (string.IsNullOrWhiteSpace(audioFilePath))

{

Logger.LogInfo("Attempted to play alert, but audio file path is empty or null.");

return;

}

// Option 1: Play from Resources.resx (if path is just a filename like "alert10.wav")

try

{

string fileName = Path.GetFileName(audioFilePath); // Extract just the filename

string resourceKey = Path.GetFileNameWithoutExtension(fileName);

Stream stream = Properties.Resources.ResourceManager.GetStream(resourceKey);

if (stream != null)

{

SoundPlayer player = new SoundPlayer(stream);

// player.Play(); // Temporarily commented out to disable audio

System.Diagnostics.Debug.WriteLine($"Playing resource from Resources.resx: {resourceKey}");

Logger.LogInfo($"Played audio alert from Resources.resx: {resourceKey}");

return;

}

else

{

Logger.LogWarning($"Resource '{resourceKey}' not found in Resources.resx. Attempting external path.");

}

}

catch (Exception ex)

{

Logger.LogError($"Error playing sound from Resources.resx for {audioFilePath}: {ex.Message}", ex);

}

// Option 2: Fallback to external path if resource.resx fails or it's a full path

try

{

if (File.Exists(audioFilePath))

{

SoundPlayer player = new SoundPlayer(audioFilePath);

// player.Play(); // Temporarily commented out to disable audio

System.Diagnostics.Debug.WriteLine($"Playing external alert: {audioFilePath}");

Logger.LogInfo($"Played audio alert from external path: {audioFilePath}");

}

else

{

Logger.LogWarning($"External alert file '{audioFilePath}' not found.");

}

}

catch (Exception ex)

{

Logger.LogError($"Error playing external sound from {audioFilePath}: {ex.Message}", ex);

}

}

private void UpdateSlotCollections()

{

// Find the first upcoming slot that is in visual alert mode

var alertSlot = TimeSlots.FirstOrDefault(slot => slot.IsIn30MinAlert && !slot.IsPassed); // IsIn30MinAlert now means IsInVisualAlert

TopSlots.Clear();

BottomSlots.Clear();

if (alertSlot != null)

{

IsAlertActive = true;

TopSlots.Add(alertSlot);

// Sort remaining slots based on their ID prefix to ensure correct order

foreach (var slot in TimeSlots.Where(s => s != alertSlot).OrderBy(s => s.Id))

{

BottomSlots.Add(slot);

}

// Adjust BottomGridColumns based on how many items are actually in BottomSlots

BottomGridColumns = BottomSlots.Count > 0 ? BottomSlots.Count : 1; // Ensure at least 1 column if items exist

}

else

{

IsAlertActive = false;

BottomGridColumns = 3; // Reset to default for normal 4-slot view (or 1 for 2-slot view, but this is handled by NormalGridColumns/Rows)

}

OnPropertyChanged(nameof(TopSlots));

OnPropertyChanged(nameof(BottomSlots));

}

private string GetHebrewJewishDateString(DateTime anyDate, bool addDayOfWeek)

{

StringBuilder stringBuilder = new StringBuilder();

CultureInfo cultureInfo = CultureInfo.CreateSpecificCulture("he-IL");

cultureInfo.DateTimeFormat.Calendar = new HebrewCalendar();

if (addDayOfWeek)

{

stringBuilder.Append(anyDate.ToString("dddd", cultureInfo) + " ");

}

stringBuilder.Append(anyDate.ToString("dd", cultureInfo) + " ");

stringBuilder.Append(anyDate.ToString("y", cultureInfo) ?? "");

return stringBuilder.ToString();

}

public void SwitchLanguage(string lang)

{

\_currentLang = lang;

foreach (var slot in TimeSlots)

{

if (\_translations[lang].TryGetValue(slot.Id, out var trans))

slot.Description = trans;

// Update PassedText for translation

slot.PassedText = \_translations[lang]["Passed"];

}

OnPropertyChanged(nameof(TimeSlots)); // Notify that TimeSlots have changed (descriptions updated)

// Re-evaluate Omer status as Hebrew date might be affected by culture (though not directly for Omer logic)

OmerStatusText = Utilities.HebrewDateChecker.GetSefiratHaOmerStatus(DateTime.Now);

IsInSefiratHaOmer = !string.IsNullOrEmpty(OmerStatusText);

}

private void OnPropertyChanged([CallerMemberName] string name = null) =>

PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(name));

}

}

<# ViewModels/OptionsViewModel.cs #>

using System;

using System.ComponentModel;

using System.IO;

using System.Runtime.CompilerServices;

using System.Windows;

using System.Windows.Input;

using Microsoft.Win32; // For OpenFileDialog

using EOTReminder.Utilities;

namespace EOTReminder.ViewModels

{

public class OptionsViewModel : INotifyPropertyChanged

{

// Existing Settings

private int \_firstAlertMinutes;

public int FirstAlertMinutes

{

get => \_firstAlertMinutes;

set { \_firstAlertMinutes = value; OnPropertyChanged(); }

}

private int \_secondAlertMinutes;

public int SecondAlertMinutes

{

get => \_secondAlertMinutes;

set { \_secondAlertMinutes = value; OnPropertyChanged(); }

}

private string \_excelFilePath;

public string ExcelFilePath

{

get => \_excelFilePath;

set { \_excelFilePath = value; OnPropertyChanged(); }

}

// NEW: Audio Alert Paths

private string \_eos1FirstAlertPath;

public string EOS1FirstAlertPath

{

get => \_eos1FirstAlertPath;

set { \_eos1FirstAlertPath = value; OnPropertyChanged(); }

}

private string \_eos1SecondAlertPath;

public string EOS1SecondAlertPath

{

get => \_eos1SecondAlertPath;

set { \_eos1SecondAlertPath = value; OnPropertyChanged(); }

}

private string \_eos2FirstAlertPath;

public string EOS2FirstAlertPath

{

get => \_eos2FirstAlertPath;

set { \_eos2FirstAlertPath = value; OnPropertyChanged(); }

}

private string \_eos2SecondAlertPath;

public string EOS2SecondAlertPath

{

get => \_eos2SecondAlertPath;

set { \_eos2SecondAlertPath = value; OnPropertyChanged(); }

}

private string \_eot1FirstAlertPath;

public string EOT1FirstAlertPath

{

get => \_eot1FirstAlertPath;

set { \_eot1FirstAlertPath = value; OnPropertyChanged(); }

}

private string \_eot1SecondAlertPath;

public string EOT1SecondAlertPath

{

get => \_eot1SecondAlertPath;

set { \_eot1SecondAlertPath = value; OnPropertyChanged(); }

}

private string \_eot2FirstAlertPath;

public string EOT2FirstAlertPath

{

get => \_eot2FirstAlertPath;

set { \_eot2FirstAlertPath = value; OnPropertyChanged(); }

}

private string \_eot2SecondAlertPath;

public string EOT2SecondAlertPath

{

get => \_eot2SecondAlertPath;

set { \_eot2SecondAlertPath = value; OnPropertyChanged(); }

}

// NEW: Visual Alert Minutes

private int \_visualAlertMinutes;

public int VisualAlertMinutes

{

get => \_visualAlertMinutes;

set { \_visualAlertMinutes = value; OnPropertyChanged(); }

}

// NEW: Alert on Shabbos

private bool \_alertOnShabbos;

public bool AlertOnShabbos

{

get => \_alertOnShabbos;

set { \_alertOnShabbos = value; OnPropertyChanged(); }

}

// NEW: UseTwoTimeSlots setting

private bool \_useTwoTimeSlots;

public bool UseTwoTimeSlots

{

get => \_useTwoTimeSlots;

set { \_useTwoTimeSlots = value; OnPropertyChanged(); }

}

// Commands

public ICommand SaveSettingsCommand { get; }

public ICommand CloseApplicationCommand { get; }

public ICommand CloseSettingsCommand { get; } // NEW: Command for closing settings window

public ICommand BrowseExcelCommand { get; }

// NEW: Browse Commands for Audio Paths

public ICommand BrowseEOS1FirstAlertCommand { get; }

public ICommand BrowseEOS1SecondAlertCommand { get; }

public ICommand BrowseEOS2FirstAlertCommand { get; }

public ICommand BrowseEOS2SecondAlertCommand { get; }

public ICommand BrowseEOT1FirstAlertCommand { get; }

public ICommand BrowseEOT1SecondAlertCommand { get; }

public ICommand BrowseEOT2FirstAlertCommand { get; }

public ICommand BrowseEOT2SecondAlertCommand { get; }

public OptionsViewModel()

{

LoadSettings();

SaveSettingsCommand = new RelayCommand(SaveSettings);

CloseApplicationCommand = new RelayCommand(CloseApplication);

CloseSettingsCommand = new RelayCommand(CloseSettings); // Initialize new command

BrowseExcelCommand = new RelayCommand(BrowseExcelFile);

// NEW: Initialize Browse Commands for Audio Paths

BrowseEOS1FirstAlertCommand = new RelayCommand(param => BrowseAudioFile(nameof(EOS1FirstAlertPath)));

BrowseEOS1SecondAlertCommand = new RelayCommand(param => BrowseAudioFile(nameof(EOS1SecondAlertPath)));

BrowseEOS2FirstAlertCommand = new RelayCommand(param => BrowseAudioFile(nameof(EOS2FirstAlertPath)));

BrowseEOS2SecondAlertCommand = new RelayCommand(param => BrowseAudioFile(nameof(EOS2SecondAlertPath)));

BrowseEOT1FirstAlertCommand = new RelayCommand(param => BrowseAudioFile(nameof(EOT1FirstAlertPath)));

BrowseEOT1SecondAlertCommand = new RelayCommand(param => BrowseAudioFile(nameof(EOT1SecondAlertPath)));

BrowseEOT2FirstAlertCommand = new RelayCommand(param => BrowseAudioFile(nameof(EOT2FirstAlertPath)));

BrowseEOT2SecondAlertCommand = new RelayCommand(param => BrowseAudioFile(nameof(EOT2SecondAlertPath)));

}

private void LoadSettings()

{

FirstAlertMinutes = Properties.Settings.Default.FirstAlertMinutes;

SecondAlertMinutes = Properties.Settings.Default.SecondAlertMinutes;

ExcelFilePath = Properties.Settings.Default.ExcelFilePath;

// NEW: Load new settings

EOS1FirstAlertPath = Properties.Settings.Default.EOS1FirstAlertPath;

EOS1SecondAlertPath = Properties.Settings.Default.EOS1SecondAlertPath;

EOS2FirstAlertPath = Properties.Settings.Default.EOS2FirstAlertPath;

EOS2SecondAlertPath = Properties.Settings.Default.EOS2SecondAlertPath;

EOT1FirstAlertPath = Properties.Settings.Default.EOT1FirstAlertPath; // Added EOT paths

EOT1SecondAlertPath = Properties.Settings.Default.EOT1SecondAlertPath; // Added EOT paths

EOT2FirstAlertPath = Properties.Settings.Default.EOT2FirstAlertPath; // Added EOT paths

EOT2SecondAlertPath = Properties.Settings.Default.EOT2SecondAlertPath; // Added EOT paths

VisualAlertMinutes = Properties.Settings.Default.VisualAlertMinutes;

AlertOnShabbos = Properties.Settings.Default.AlertOnShabbos;

UseTwoTimeSlots = Properties.Settings.Default.UseTwoTimeSlots; // NEW: Load UseTwoTimeSlots

Logger.LogInfo("Application settings loaded.");

}

private void SaveSettings(object parameter)

{

Properties.Settings.Default.FirstAlertMinutes = FirstAlertMinutes;

Properties.Settings.Default.SecondAlertMinutes = SecondAlertMinutes;

Properties.Settings.Default.ExcelFilePath = ExcelFilePath;

// NEW: Save new settings

Properties.Settings.Default.EOS1FirstAlertPath = EOS1FirstAlertPath;

Properties.Settings.Default.EOS1SecondAlertPath = EOS1SecondAlertPath;

Properties.Settings.Default.EOS2FirstAlertPath = EOS2FirstAlertPath;

Properties.Settings.Default.EOS2SecondAlertPath = EOS2SecondAlertPath;

Properties.Settings.Default.EOT1FirstAlertPath = EOT1FirstAlertPath; // Added EOT paths

Properties.Settings.Default.EOT1SecondAlertPath = EOT1SecondAlertPath; // Added EOT paths

Properties.Settings.Default.EOT2FirstAlertPath = EOT2FirstAlertPath; // Added EOT paths

Properties.Settings.Default.EOT2SecondAlertPath = EOT2SecondAlertPath; // Added EOT paths

Properties.Settings.Default.VisualAlertMinutes = VisualAlertMinutes;

Properties.Settings.Default.AlertOnShabbos = AlertOnShabbos;

Properties.Settings.Default.UseTwoTimeSlots = UseTwoTimeSlots; // NEW: Save UseTwoTimeSlots

Properties.Settings.Default.Save();

Logger.LogInfo("Application settings saved successfully.");

if (parameter is Window window)

{

window.Close();

}

// Removed direct window close, now handled by specific CloseSettings command

}

// NEW: CloseSettings method

private void CloseSettings(object parameter)

{

Logger.LogInfo("Settings window close requested.");

if (parameter is Window window)

{

window.Close();

}

}

private void CloseApplication(object parameter)

{

Logger.LogInfo("Application close requested from options window.");

if (parameter is Window window)

{

window.Close();

}

Application.Current.Shutdown();

}

private void BrowseExcelFile(object parameter)

{

OpenFileDialog openFileDialog = new OpenFileDialog();

openFileDialog.Filter = "Excel Files (\*.xlsx;\*.xls)|\*.xlsx;\*.xls|All Files (\*.\*)|\*.\*";

openFileDialog.InitialDirectory = GetInitialDirectory(ExcelFilePath);

try

{

if (openFileDialog.ShowDialog() == true)

{

ExcelFilePath = openFileDialog.FileName;

Logger.LogInfo($"Excel file path set to: {ExcelFilePath}");

}

}

catch (Exception ex)

{

Logger.LogError($"Error opening file dialog for Excel file: {ex.Message}", ex);

}

}

// NEW: Generic BrowseAudioFile method

private void BrowseAudioFile(string propertyName)

{

OpenFileDialog openFileDialog = new OpenFileDialog();

openFileDialog.Filter = "WAV Audio Files (\*.wav)|\*.wav|All Files (\*.\*)|\*.\*";

string currentPath = GetPropertyValue(propertyName) as string;

openFileDialog.InitialDirectory = GetInitialDirectory(currentPath);

try

{

if (openFileDialog.ShowDialog() == true)

{

SetPropertyValue(propertyName, openFileDialog.FileName);

Logger.LogInfo($"Audio file path for {propertyName} set to: {openFileDialog.FileName}");

}

}

catch (Exception ex)

{

Logger.LogError($"Error opening file dialog for audio file ({propertyName}): {ex.Message}", ex);

}

}

// Helper to get initial directory for file dialogs

private string GetInitialDirectory(string currentPath)

{

if (!string.IsNullOrWhiteSpace(currentPath) && File.Exists(currentPath))

{

return Path.GetDirectoryName(currentPath);

}

if (!string.IsNullOrWhiteSpace(currentPath) && Directory.Exists(currentPath))

{

return currentPath;

}

return Environment.GetFolderPath(Environment.SpecialFolder.MyDocuments);

}

// Helper to get property value by name (for dynamic binding)

private object GetPropertyValue(string propertyName)

{

return GetType().GetProperty(propertyName)?.GetValue(this);

}

// Helper to set property value by name (for dynamic binding)

private void SetPropertyValue(string propertyName, object value)

{

GetType().GetProperty(propertyName)?.SetValue(this, value);

}

public event PropertyChangedEventHandler PropertyChanged;

protected virtual void OnPropertyChanged([CallerMemberName] string propertyName = null)

{

PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(propertyName));

}

}

// Basic RelayCommand implementation (if you don't have one already)

public class RelayCommand : ICommand

{

private readonly Action<object> \_execute;

private readonly Predicate<object> \_canExecute;

public RelayCommand(Action<object> execute, Predicate<object> canExecute = null)

{

\_execute = execute ?? throw new ArgumentNullException(nameof(execute));

\_canExecute = canExecute;

}

public bool CanExecute(object parameter) => \_canExecute == null || \_canExecute(parameter);

public void Execute(object parameter) => \_execute(parameter);

public event EventHandler CanExecuteChanged

{

add => CommandManager.RequerySuggested += value;

remove => CommandManager.RequerySuggested -= value;

}

}

}

<# Models/TimeSlot.cs #>

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Runtime.CompilerServices;

namespace EOTReminder.Models

{

public class TimeSlot : INotifyPropertyChanged

{

private string \_description;

private bool \_isPassed;

private string \_countdownText;

private bool \_showSandClock;

private bool \_highlight;

private TimeSpan \_countdown;

private bool \_isIn30MinAlert;

private string \_passedText;

public string Id { get; set; }

public DateTime Time { get; set; }

public Dictionary<string, bool> AlertFlags { get; set; } = new Dictionary<string, bool>()

{["Visual"] = false, ["FirstAudio"] = false, ["SecondAudio"] = false}; // Updated flags

public string Description

{

get => \_description;

set

{

\_description = value;

OnPropertyChanged();

}

}

public string PassedText

{

get => \_passedText;

set

{

\_passedText = value;

OnPropertyChanged();

}

}

public bool IsPassed

{

get => \_isPassed;

set

{

\_isPassed = value;

OnPropertyChanged();

}

}

public string CountdownText

{

get => \_countdownText;

set

{

\_countdownText = value;

OnPropertyChanged();

}

}

public bool ShowSandClock

{

get => \_showSandClock;

set

{

\_showSandClock = value;

OnPropertyChanged();

}

}

public bool Highlight

{

get => \_highlight;

set

{

\_highlight = value;

OnPropertyChanged();

}

}

public TimeSpan Countdown

{

get => \_countdown;

set

{

\_countdown = value;

OnPropertyChanged();

}

}

public bool IsIn30MinAlert

{

get => \_isIn30MinAlert;

set

{

\_isIn30MinAlert = value;

OnPropertyChanged();

}

}

public event PropertyChangedEventHandler PropertyChanged;

private void OnPropertyChanged([CallerMemberName] string name = null) =>

PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(name));

}

}

<# Utilities/HebrewDateChecker.cs #>

using System;

using System.Globalization;

using System.Text;

namespace EOTReminder.Utilities

{

public static class HebrewDateChecker

{

/// <summary>

/// Returns the Hebrew word representation for a number (1-49).

/// This is a simplified version for Omer counting and might not cover all grammatical nuances for all numbers.

/// </summary>

private static string GetHebrewNumberWords(int number)

{

if (number < 1 || number > 49) return "";

string[] units = {

"אחד", "שניים", "שלשה", "ארבעה", "חמשה", "ששה", "שבעה", "שמונה", "תשעה"

};

string[] tens = {

"", "עשרה", "עשרים", "שלשים", "ארבעים"

};

string[] tensSuffix = { // For 11-19 (e.g., אחד עשר)

"עשר", "עשר", "עשר", "עשר", "עשר", "עשר", "עשר", "עשר", "עשר"

};

if (number <= 10)

{

return units[number - 1];

}

else if (number < 20) // 11-19

{

// Special handling for 11-19: e.g., אחד עשר, שניים עשר

// Note: This is a simplified approach. For perfect Hebrew grammar,

// some numbers like 11, 12 might have slightly different forms.

// For Omer, this is generally acceptable.

if (number == 11) return "אחד עשר";

if (number == 12) return "שניים עשר";

// For other 13-19, it's unit + "עשר"

return units[number % 10 - 1] + " " + tensSuffix[number % 10 -1];

}

else // 20-49

{

int tenVal = number / 10;

int unitVal = number % 10;

if (unitVal == 0)

{

return tens[tenVal];

}

else

{

// "ו" means "and"

return units[unitVal - 1] + " ו" + tens[tenVal];

}

}

}

/// <summary>

/// Generates the Hebrew string for the given Omer day.

/// </summary>

private static string GetOmerDayString(int omerDay)

{

if (omerDay < 1 || omerDay > 49)

{

return "לא בספירת העומר";

}

StringBuilder sb = new StringBuilder();

// Part 1: "היום X יום/ימים לעומר"

sb.Append("היום ");

sb.Append(GetHebrewNumberWords(omerDay));

sb.Append(omerDay == 1 ? " יום " : " ימים "); // "יום" for 1, "ימים" for others

sb.Append("לעומר");

// Part 2: "שהם Y שבוע/שבועות (וZ יום/ימים)"

int weeks = omerDay / 7;

int remainingDays = omerDay % 7;

if (weeks > 0 || remainingDays > 0)

{

sb.Append("\nשהם "); // Newline as requested

if (weeks > 0)

{

sb.Append(GetHebrewNumberWords(weeks));

sb.Append(weeks == 1 ? " שבוע" : " שבועות");

}

if (remainingDays > 0)

{

if (weeks > 0)

{

sb.Append(" ו");

}

sb.Append(GetHebrewNumberWords(remainingDays));

sb.Append(remainingDays == 1 ? " יום" : " ימים");

}

}

return sb.ToString();

}

/// <summary>

/// Checks if the given Gregorian date falls within the Sefirat Ha'Omer period

/// and returns a string describing the day of the Omer, or an empty string if not in Omer.

/// </summary>

/// <param name="gregorianDate">The Gregorian date to check.</param>

/// <returns>A string describing the day of the Omer, or an empty string if not in Omer.</returns>

public static string GetSefiratHaOmerStatus(DateTime gregorianDate)

{

HebrewCalendar hebrewCalendar = new HebrewCalendar();

int hebrewYear = hebrewCalendar.GetYear(gregorianDate);

// Get the Gregorian date for the 16th of Nisan of the current Hebrew year

// Nisan is Hebrew month 1 in the religious calendar.

DateTime omerStartDateGregorian;

try

{

omerStartDateGregorian = hebrewCalendar.ToDateTime(hebrewYear, 1, 16, 0, 0, 0, 0);

}

catch (ArgumentOutOfRangeException)

{

// This catch is mostly a safeguard; unlikely with valid gregorianDate

return ""; // Return empty string if date conversion fails

}

// Calculate the difference in days from the start of Omer

TimeSpan diff = gregorianDate.Date - omerStartDateGregorian.Date;

int omerDay = diff.Days + 1; // Add 1 because the first day is day 1, not day 0 difference

// Check if the calculated day is within the 49 days of Omer

if (omerDay >= 1 && omerDay <= 49)

{

return GetOmerDayString(omerDay);

}

else

{

return ""; // Not in Sefirat Ha'Omer

}

}

}

}

```