

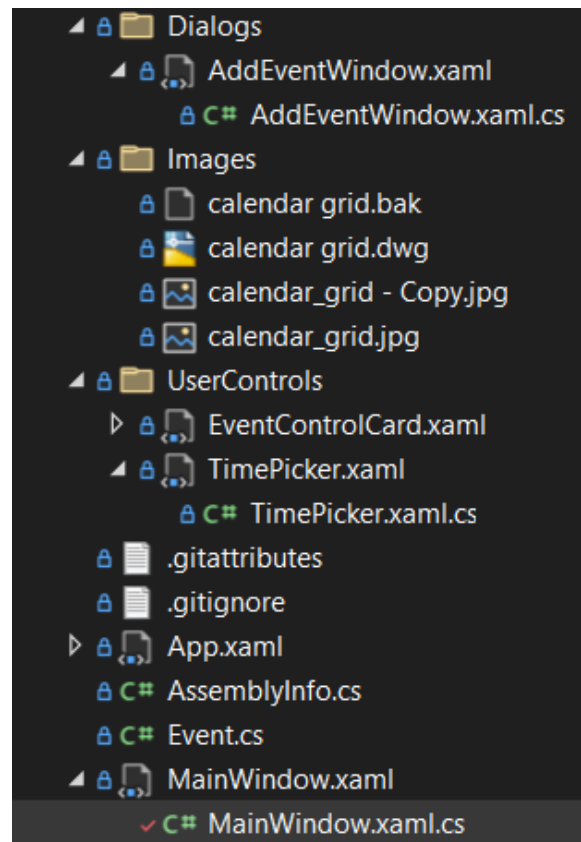
## 1. Część 1 z 2 – głównie backend

### 1.1. Opis aplikacji

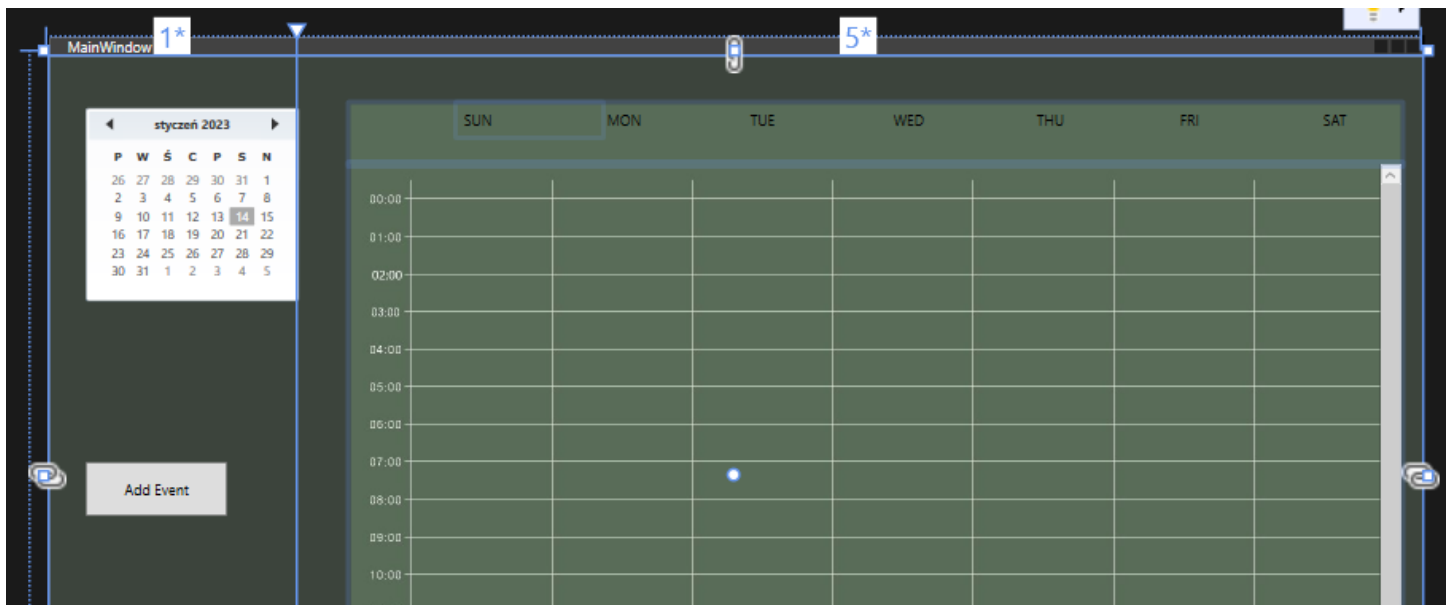
Jest to aplikacja do zarządzania wydarzeniami. Pozwala ona dodawać wydarzenia oraz przeglądać zaplanowane wydarzenia w wybranym tygodniu, podobnie do Google Calendar. Aplikacja napisana w języku angielskim.

### 1.2. Opis kodu

Struktura projektu:



MainWindow:



3 Głównie elementy: Kalendarz, przycisk dodania wydarzenia, płaszczyzna dla umieszczenia wydarzeń.

Płaszczyzna narysowana ręcznie w AutoCad.

Główne pole: Grid, 2 kolumny. Etykiety dni tygodnia są statyczne, etykiety numeru dnia są przywiązane poprzez Context do klasy MainViewModel, która jest zbiorczą klasą zawierającą wszystkie zmienne do których

```
<Grid Name="plnMainGrid" Background="#FF3C443C">
    <Grid.ColumnDefinitions>
        <ColumnDefinition Width="1*" />
        <ColumnDefinition Width="5*" />
    </Grid.ColumnDefinitions>
    <Calendar x:Name="Calendar" Grid.Column="0" Margin="30,40,0,0" />
    <Button x:Name="AddEventButton" Grid.Column="0" Margin="30,340,60,320" Content="Add Eve
        Click="AddEventButton_Click" />
    <DockPanel Width="880" Margin="30,40,5,5" Grid.Column="1" Panel.ZIndex="5" >
        <Canvas DockPanel.Dock="Top" Background="#FF586C58" Height="50">
            <StackPanel Orientation="Horizontal" Canvas.Left="90" Canvas.Top="0">
                <Label Content="SUN" Width="120" />
                <Label Content="MON" Width="120" />
                <Label Content="TUE" Width="120" />
                <Label Content="WED" Width="120" />
                <Label Content="THU" Width="120" />
                <Label Content="FRI" Width="120" />
                <Label Content="SAT" />
            </StackPanel>
            <StackPanel Orientation="Horizontal" Canvas.Top="20" Canvas.Left="95">
                <TextBlock Text="{Binding Sunday, StringFormat={}{0: dd}}" Width="122"/>
                <TextBlock Text="{Binding Monday, StringFormat={}{0: dd}}" Width="117"/>
                <TextBlock Text="{Binding Tuesday, StringFormat={}{0: dd}}" Width="120"/>
                <TextBlock Text="{Binding Wednesday, StringFormat={}{0: dd}}" Width="120"/>
                <TextBlock Text="{Binding Thursday, StringFormat={}{0: dd}}" Width="118"/>
                <TextBlock Text="{Binding Friday, StringFormat={}{0: dd}}" Width="120"/>
                <TextBlock Text="{Binding Saturday, StringFormat={}{0: dd}}" />
            </StackPanel>
        </Canvas>
    </DockPanel>
    <ScrollViewer>
        <Canvas x:Name="EventsCanvasPanel" Height="794" Width="880" >
```

będą przywiązane elementy umieszczone na MainWindow aplikacji.

```
public class MainViewModel : INotifyPropertyChanged
{
    public event PropertyChangedEventHandler PropertyChanged;
    private DateTime sunday, monday, tuesday, wednesday, thursday, friday, saturday, selectedDate;
    10 references
    public DateTime Sunday...
    1 reference
    public DateTime Monday...
    1 reference
    public DateTime Tuesday...
    1 reference
    public DateTime Wednesday...
    1 reference
    public DateTime Thursday...
    1 reference
    public DateTime Friday...
    1 reference
    public DateTime Saturday
    {
        get { return saturday; }
        set
        {
            saturday = value;
            RaisePropertyChanged("Saturday");
        }
    }
    2 references
    public DateTime SelectedDate
    {
        get { return selectedDate; }
```

Aby działać jako Context klasa implementuje interfejs INotifyPropertyChanged, a zmiana pól powoduje wywołanie metody RaisePropertyChanged().

```
8 references
protected void RaisePropertyChanged(string propertyName)
{
    PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(propertyName));
}
0 references
```

Oprócz MainViewModel, MainWindow zawiera ObservableCollection Events ze wszystkimi wydarzeniami:

```
134 public partial class MainWindow : Window
135 {
136     3 references
137     public ObservableCollection<Event> Events { get; set; }
138     public MainViewModel ViewModel;
139
140     0 references
141     public MainWindow()
142     {
143         InitializeComponent();
144         Events = new ObservableCollection<Event>();
145
146         Calendar.SelectedDate = DateTime.Today;
147         ViewModel = new MainViewModel(Calendar.SelectedDates[0]);
148         this.DataContext = ViewModel;
149
150         Calendar.SelectedDatesChanged += Calendar_SelectedDatesChanged;
151     }
```

Typ kolekcji ObservableCollection w WPF potrzebny dla zmiany wyświetlanych elementów przywiązanych do zmieniających się obiektów.

Linia 150 – zaznaczenie daty w kalendarzu wywołuje metodę Calendar\_SelectedDatesChanged().

```
158 private void Calendar_SelectedDatesChanged(object sender, SelectionChangedEventArgs e)
159 {
160     // Clear the events list
161     EventsCanvasPanel.Children.Clear();
162
163     // Get the selected week
164     ViewModel.SelectedDate = Calendar.SelectedDates[0];
165
166     // Add the events for the selected week to the list
167     foreach (Event ev in Events)
168     {
169         if (ev.Day >= ViewModel.Sunday && ev.Day < ViewModel.Sunday.AddDays(7))
170         {
171             var eventCard = new EventControlCard(ev);
172
173             switch (ev.Day.DayOfWeek)
174             {
175                 case DayOfWeek.Sunday:
176                     Canvas.SetTop(eventCard, 29 + (int)Math.Round((ev.StartTime.TotalSeconds / 114.5));
177                     EventsCanvasPanel.Children.Add(eventCard);
178             }
179         }
180     }
181 }
```

Przy zmianie zaznaczonej daty w kalendarzu:

1. Usuwa się wszystkie wydarzenia z oła wydarzeń.

2. Zmieniają się wartości etykiet numeru dnia tygodnia (linia 164)
3. Na pole wydarzeń dodawane są wszystkie wydarzenia z wybranego tygodnia. Każde wydarzenie pozycjonowane zgodnie z dniem i czasem, a rozmiar zależy od trwałości wydarzenia.

Dodawanie wydarzenia odbywa się metodą `AddEventButton_Click()`.

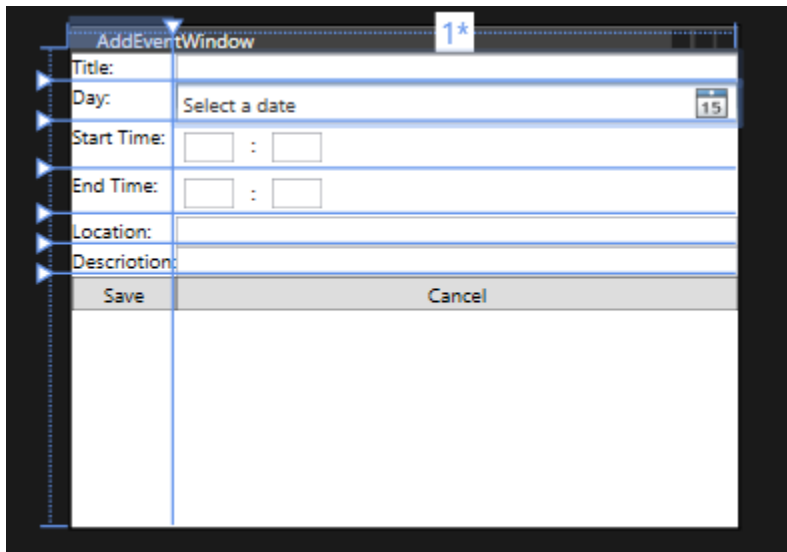
```
private void AddEventButton_Click(object sender, RoutedEventArgs e)
{
    // Display a dialog to add a new event
    AddEventWindow addEventWindow = new AddEventWindow();
    addEventWindow.ShowDialog();

    // If the user clicks "OK" in the dialog, add the event to the list
    if (addEventWindow.DialogResult.HasValue && addEventWindow.DialogResult.Value == true)
    {
        Event ev = addEventWindow.Event;
        Events.Add(ev);
    }

    //Refreshing the schedule plane
    Calendar_SelectedDatesChanged(Calendar, null);
}
```

Wywołuje ono dialog `AddEventWindow`, ostatnia linia dla ponownego usuwania i dodawania wydarzeń razem z nowym wydarzeniem.

`AddEventWindow` jest dialogiem dla dodawania wydarzenia. Oprócz prostych etykiet, przycisków i pól tekstowych zawiera ono również ręcznie stworzony kontroler `TimePicker` dla wyboru godziny, a data wybiera się istniejącym kontrolerem `DayPicker`. Tworzy i zwraca obiekt wydarzenia.

The image shows a screenshot of a Windows application window titled "AddEventWindow". The window contains several input fields: "Title:" followed by a text box; "Day:" followed by a text box containing "Select a date" and a small calendar icon showing the number "15"; "Start Time:" followed by two text boxes separated by a colon; "End Time:" followed by two text boxes separated by a colon; "Location:" followed by a text box; and "Description:" followed by a text box. At the bottom of the window are two buttons: "Save" and "Cancel". The window has a standard Windows title bar with a maximize button and a close button.

Klasa wydarzeń `Event`

```

public class Event
{
    4 references
    public DateTime Day { get; set; }
    3 references
    public TimeSpan StartTime { get; set; }
    2 references
    public TimeSpan EndTime { get; set; }
    1 reference
    public string Title { get; set; }
    1 reference
    public string Description { get; set; }
    1 reference
    public string Location { get; set; }
    0 references
    public int CartHeight
    {
        // (int) and Round because '(int)' rounds always to smallest number (31.8 -> 31)
        get { return (int)Math.Round((EndTime - StartTime).TotalSeconds/112.5) ; }
    }
}

```

Oprócz pól związanych z wydarzeniem zawiera pole CartHeight które zwraca wysokość karty wydarzenia we właściwym typie int. Karta wydarzenia jest właśnie stworzoną kontrollerem EventControlCard.



```

9  <WrapPanel Orientation="Vertical" Width="113" Height="{Binding CartHeight}" Background="LightBlue">
10  <TextBlock Text="{Binding Title}" FontWeight="Bold" Height="auto" Width="115" TextWrapping="Wrap"/>
11  <TextBlock Text="{Binding StartTime}" Height="auto" TextWrapping="Wrap"/>
12  <TextBlock Text="{Binding EndTime}" Height="auto" TextWrapping="Wrap"/>
13  <TextBlock Text="{Binding Location}" Height="auto" TextWrapping="Wrap"/>
14  <TextBlock Text="{Binding Description}" Height="auto" TextWrapping="Wrap"/>
15  </WrapPanel>

```

Wszystkie pola tekstowe kontrolera EventControlCard są przywiązane do pól Context, czyli do przesłanej klasy Event:

```

public partial class EventControlCard : UserControl
{
    1 reference
    public EventControlCard(Event ev)
    {
        InitializeComponent();
        DataContext = ev;
    }
}

```