Opis końcowy projektu z Akademii C# MS 2017

Autorzy projektu: Robert Kaczmarek (project manager) oraz Jakub Stencel

Tytuł projektu: Aplikacja w WPF dla nauczycieli akademickich

Film: https://youtu.be/BdUMH7wKnlI

Możliwości aplikacji

Aplikacji umożliwia użytkownikowi wprowadzenie danych studenta (tj. imienia, nazwiska oraz numeru indeksu) a następnie dodanie go do listy. Następnie po wybraniu kalkulatora filtru bądź wzmacniacza oraz numeru indeksu studenta, możliwe jest obliczenie parametrów wybranych funkcji. Parametry obwodu są unikalne dla danego studenta i są losowane automatycznie przez program. Dzięki zastosowaniu w programie wtyczki LiveCharts, może podejrzeć wyniki obliczeń na wykresach a tym samym sprawdzić rozwiązania studentów. Umożliwiona została również generacja pliku PDF zawierającego wybrany schemat, przydzielone parametry oraz dane wprowadzonego studenta. Dodatkowo można wprowadzić treść samego zadania. Aplikacja zwiera również zakładkę z materiałami dydaktycznymi.

Opis użytych metod celem uzyskania wyżej wymienionych funkcjonalności

Do zbudowania szkieletu programu wykorzystano paczkę nuget Material Design in XAML Toolkit oraz kody źródłowe programu demo autora paczki. Zdecydowano się wykorzystać wzorzec projektowy MVVM. Wymusiło to podzielenie aplikacji na trzy płaszczyzny. W kolejnych punktach omówiono kolejne klasy oraz metody stworzone w celu osiągnięcia założonej funkcjonalności.

Model

Składa się m.in. z klas Data, gdzie zawierają się statyczne prywatne pola odpowiadające potrzebnym w programie zmiennym, właściwości z publicznym modyfikatorem dostępu oraz akcesoriami get i set. Takie podejście umożliwia dostęp do właściwości pól a nie samych pól, z poziomu innych klas, dzięki czemu zachowana jest hermetyzacja. We właściwościach występują również operatory lambda „=>”, które określają metody get oraz set i zwracają przypisane po lewej stronie elementy. Klasa model zawiera również statyczną listę klas, gdzie przechowywane są dodawane obiekty Student.

```
        public static List<Student> Students
        { get => students;
          set => students = value; }

        private static List<Student> students = new List<Student>()
        {
            new Student(123456, "Adam", "Kowalski")
        };
```

Do modelu należy także klasa Student, która zawiera takie pola jak imię, nazwisko, numer indeksu oraz parametry obwodu. Kolejno występuje metoda konstruktora Student, która przyjmuje jako parametry wartości indexu studenta, jego imienia oraz nazwiska.

Ciało metody zawiera przypisanie losowych wartości dla zmiennych unikalnych dla każdego studenta. Za samo randomowe przydzielenie wartości odpowiedzialna jest klasa Random.

```
public class Student
    {

        private int _index;
                                •
                                •
                                •
        private int _R6;

        public int Index { get => _index; set => _index = value; }
                                •
                                •
                                •
        public int R6 { get => _R6; set => _R6 = value; }

        System.Random x = new Random(System.DateTime.Now.Millisecond);

        public Student(int Index, string FirstName, string LastName)
        {
            this.Index = Index;
                                •
                                •
                                •
            R6 = x.Next(1, 100);
        }
    }
```

Ciało metody zawiera przypisanie losowych wartości dla zmiennych unikalnych dla każdego studenta. Za samo randomowe przydzielenie wartości odpowiedzialna jest klasa Random. Z racji większej liczby randomowych liczb, konieczne jest utworzenie instancji klasy, która umożliwia ponowne jej wykorzystanie:

```
        System.Random x = new Random
```

Istnieje również klasa DataModelForChart, która jest klasą pomocniczą wykorzystywaną do rysowania wykresów. Ponadto stworzono klasę Panels, która reprezentuje poszczególne widoki. Zawiera pola z nazwą, zawartością widoku, ustawieniami widoku paska do przewijania oraz marginesu.

View

Zawiera piki widoków stworzone w języku XAML, gdzie wszystkie widoki zostały stworzone dziedzicząc po klasie UserControl poza widokiem głównym (ramką programu), który dziedziczy po klasie Window. Obiekty poszczególnych widoków tworzone są w MainViewViewModel, który tworzy obiekty klasy Panels.

```
        public Panels[] Panels { get; }
        public MainWindowViewModel()
        {
            Panels = new[]
            {
                new Panels("Start", new Home()),
                                    •
                                    •
                                    •
                new Panels("O aplikacji", new About())
            };
        }
```

Widoki bindowane są do kontrolki ListBox w widoku ramki programu (MainView). ListBox wyświetlany jest w formie rozwijanego Menu.

```
<ListBox x:Name="PanelsListBox" Margin="0 16 0 16" SelectedIndex="0"
              ItemsSource="{Binding Panels}"
              PreviewMouseLeftButtonUp="UIElement_OnPreviewMouseLeftButtonUp">
       <ListBox.ItemTemplate>
              <DataTemplate DataType="Panels">
                     <TextBlock Text="{Binding Name}" Margin="32 0 32 0" />
              </DataTemplate>
       </ListBox.ItemTemplate>
</ListBox>
```

ViewModel

Każdy widok posiada swój ViewModel. Każdorazowo wraz z tworzeniem na początku programu wszystkich widoków, tworzone są obiekty ViewModeli w code-behind poszczególnych modeli:

```
    public partial class Filter : UserControl
    {
        public Filter()
        {
            InitializeComponent();
            DataContext = new FilterViewModel();
        }
    }
```

W powyższym przykładzie zaimplementowano w code-behind DataContext dla widoku Filter. Dane do bindowanych danych pobierane będą z obiektu FilterViewModel. Podejście takie odpowiada wzorcowi projektowemu MVVM.

Klasy ViewModel zawierają publiczne pola oraz metody odpowiadające za obliczenia.

```
<Binding Path="R1" UpdateSourceTrigger="PropertyChanged">
```

Powyższa linijka kodu umożliwia łączenie źródeł danych do wizualizacji z właściwościami obiektów z danych klas. Jest to bindowanie parametru R1, gdzie możliwy jest do niej dostęp poprzez publiczną właściwość:

```
       public double R1
        {
            get { return _R1; }
            set
            {
                _R1 = value;
                NotifyPropertyChanged("R1");
            }
        }
```

Dzięki wykorzystaniu interfejsu INotifyPropertyChanged możliwe jest automatyczne odświeżanie kontrolki, dzięki czemu w przypadku zmiany wartości w ViewModel, wartość zmieni się w widoku.

Złożone obliczenia wymagane do narysowania przebiegów dla danego schematu obsługiwane są w osobnym wątku – wykonywane są poprzez BackgroudWorker'a. Taki zabieg był konieczny by interfejs użytkownika był w dalszym ciągu interaktywny.

```
private void Worker_DoWork(object sender, DoWorkEventArgs e)
        {
            worker.WorkerSupportsCancellation = true;

            if (worker.CancellationPending)
            {
                e.Cancel = true;
                return;
            }
            else
            {

                for (Data.Fakt = Data.F1; Data.Fakt <= Data.F2; Data.Fakt++)
                {
                                •
                                •
                                •
                }
            }
        }
```

Walidację wprowadzanych danych zapewniono poprzez klasy NotEmptyValidationRule oraz IntValidationRule. Są to klasy dziedziczące po klasie ValidationRule:

```
    public class NotEmptyValidationRule : ValidationRule
    {
        public override ValidationResult Validate(object value, CultureInfo cultureInfo)
        {
            return string.IsNullOrWhiteSpace((value ?? "").ToString())
                ? new ValidationResult(false, "Pole jest wymagane.")
                : ValidationResult.ValidResult;
        }
    }
```

Poszczególne metody walidujące wywoływane są poprzez bindowanie w poszczególnych TextBoxach w widoku Home.

Do wydruku arkuszy PDF wykorzystano paczkę nuget o nazwie PdfSharp. Odpowiednie metody drukujące stworzono w klasie PrintViewModel. Treść zadania jest wprowadzana w oknie Print oraz zapisywana do statycznego pola w klasie Data. Dokładny przebieg działania metody drukującej arkusz PDF Filtru:

```
        public void PrintFilterTask()
        {
            Student tempStudent = SelectedStudent; // obiekt klasy Student
            PdfDocument document = new PdfDocument(); // nowy obiekt klasy PdfDocument
            document.Info.Title = "Filtr pasywny"; // tytuł dokumentu poprzez właściowość
            PdfPage page = document.AddPage(); // nowa strona dokumentu
              XGraphics gfx = XGraphics.FromPdfPage(page); // tworzenie obiektu gfx
              niezbędnego do rysowania na stronie page
            XPen pen = new XPen(XColors.Black, 5); // obiekt pen do rysowania lini
            XTextFormatter tf = new XTextFormatter(gfx); // obiekt tf do formatowania
            XFont fontBold = new XFont("Verdana", 12, XFontStyle.Bold);
            XFont fontRegular = new XFont("Verdana", 10, XFontStyle.Regular);

            XImage image = XImage.FromFile("RLC.jpg"); // inicjowanie obiektu klasy
            XImage poprzez metodę FromFile
            gfx.DrawImage(image, 15, 100, 260, 110); // metoda do wrysowania image na
            stronę

            gfx.DrawString(tempStudent.FirstName.ToString(), fontBold, XBrushes.Black,
              new XRect(20, 20, page.Width, page.Height),
              XStringFormats.TopLeft); // metoda do przepisania obiektu string na stronę
                                       •
                                       •
                                       •

            XRect rect = new XRect(300, 20, 250, 80); // obiekt kwadratu
            gfx.DrawRectangle(XBrushes.White, rect); // wrysowanie kwadratu na stronę
            tf.DrawString(Data.ContentFilterTask.ToString(),
                          fontRegular, XBrushes.Black, rect, XStringFormats.TopLeft);
                          // metoda wrysowania tekstu w kanwę kwadratu rect

            string filename = "Filtr_" + tempStudent.Index.ToString() + ".pdf"; // obiekt
            nazwy z numerem indeksu oraz rozszerzeniem
            document.Save(filename); // wywołanie metody zapisu pliku z nazwą filename

            Process.Start(filename); //wywołanie metody otworzenia stworzonego pliku
        }
```