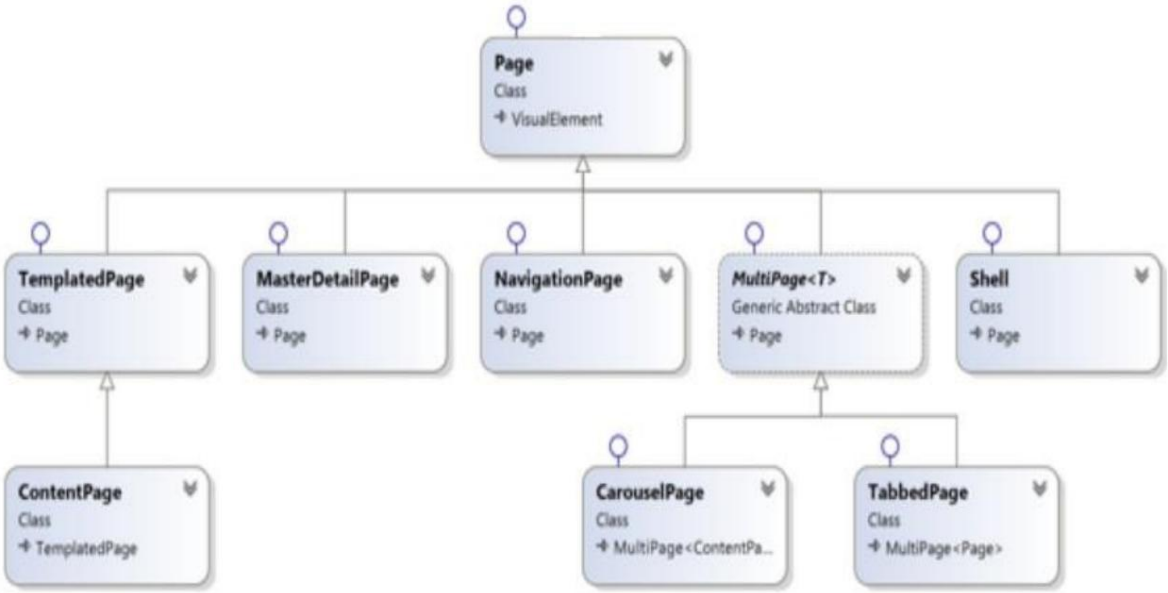
User interface: pages like ContentPage, FlyoutPage, TabbedPage, CarouselPage and TemplatedPage

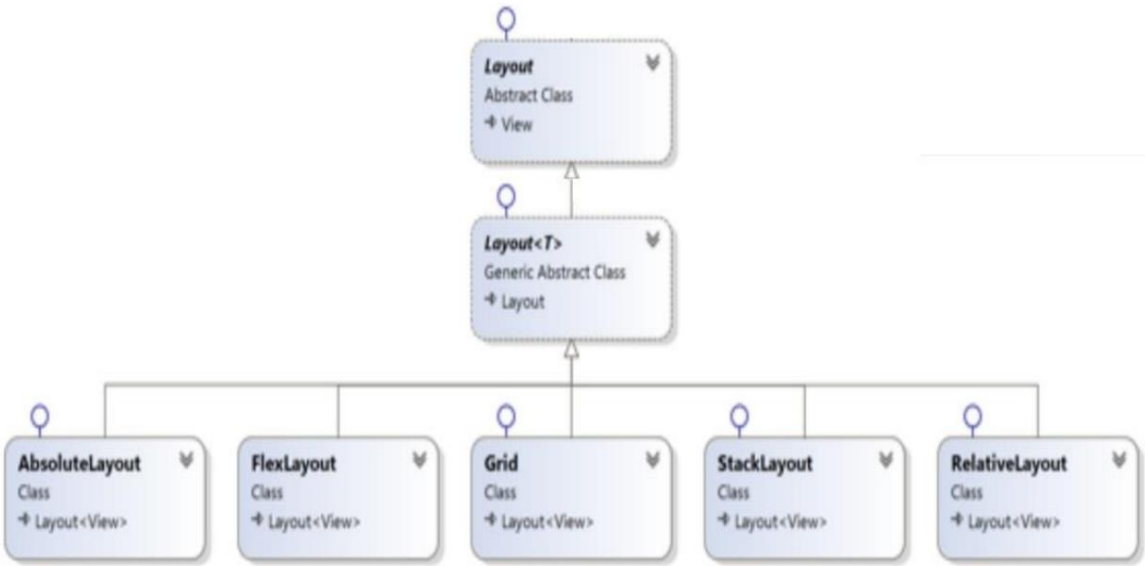
Below are the four main groups of elements used for creating the user interface of a Xamarin.Forms application.

- Pages
- Layouts
- Controls
- Cells

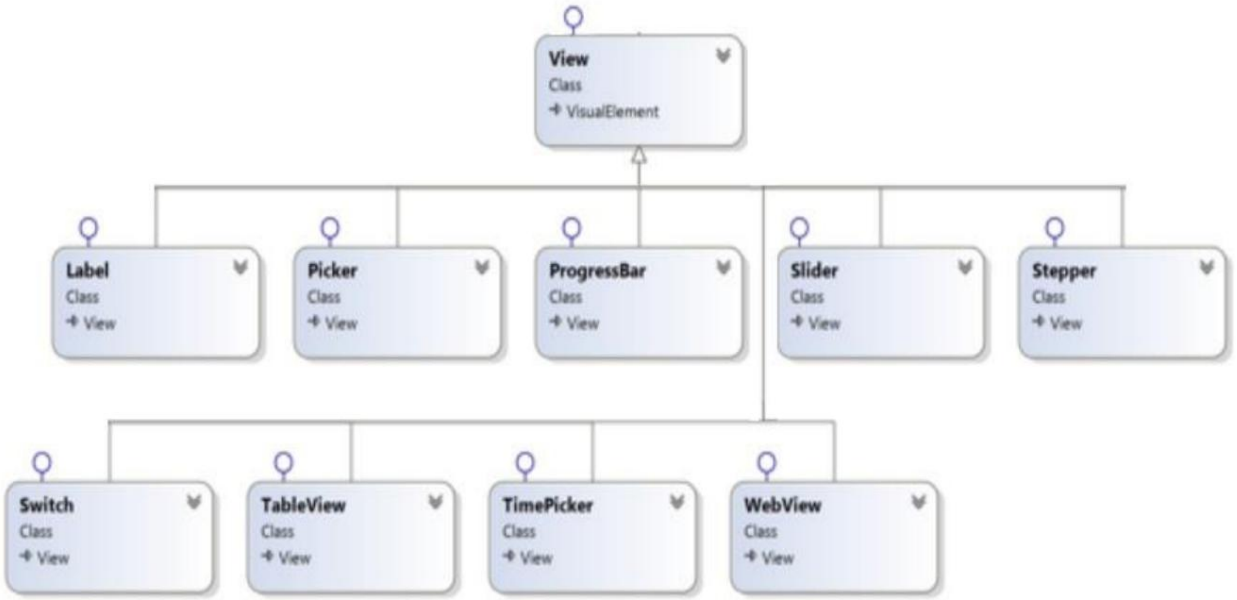
A Xamarin.Forms application page typically takes up the entire screen. All page types derive from the Page class.



A page typically contains a layout, which contains controls and possibly other layouts. All layout types derive from the Layout class.

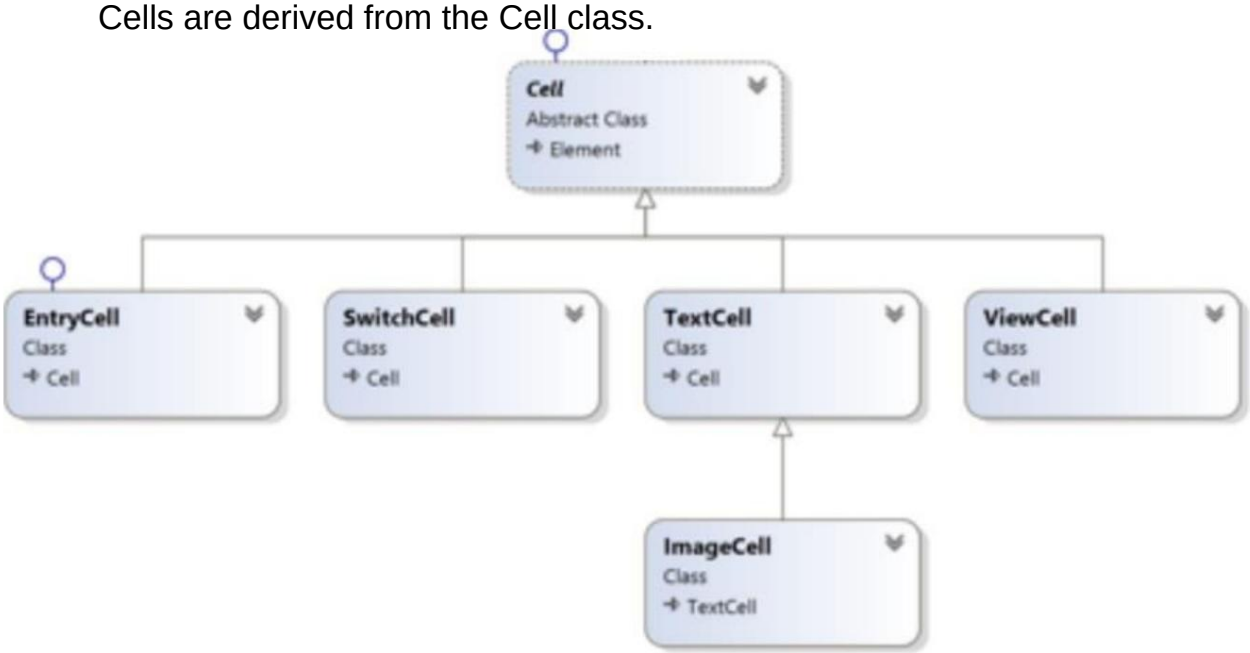


The layout contains controls or other layouts. All types controls are derived from the View class.



Cells are specialized controls, which are used when displaying data in TableView and ListView.

Cells are derived from the Cell class.



Pages, layouts, views, and cells ultimately derive from the Element class.

ContentPage, FlyoutPage, NavigationPage, TabbedPage, TempatedPage, CarouselPage are cross-platform mobile app screens. All page types are derived from the Page class.

These visual elements take up the entire screen or most of it. screen.

On iOS, a Page class object is represented as a ViewController, while on Android, each Page is an Activity, but is not an Activity object.

Let's look at working with pages. A ContentPage is the simplest and most common type of page, which can contain one control or one layout. To work with a ContentPage, you need to create a project named

ContentPageDemoPage for a multiplatform Xamarin.Forms application using the "Empty" template.

Delete automatically created and located in the file MainPage.xaml is the user interface code.

To do this, right-click the MainPage.xaml file in Solution Explorer and select Delete. This action also deletes the

MainPage.xaml.cs code file.

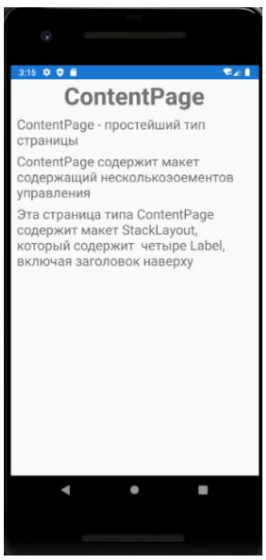
Create a class in a new file called MainPage.cs. Make the class derive from the ContentPage class. Add a using Xamarin.Forms statement since ContentPage is in the namespace Xamarin.Forms.

```
using System;
using System.Collections.Generic;
using System.Text;
using Xamarin.Forms;

namespace ContentPageDemoPage
{
    Ссылок: 2
    public class MainPage: ContentPage
    {
        ссылка: 1
        public MainPage()
        {
            {
                Label header = new Label
                {
                    Text = "ContentPage",
                    FontSize = 40,
                    FontAttributes = FontAttributes.Bold,
                    HorizontalOptions = LayoutOptions.Center
                };

                Label label1 = new Label
                {
                    Text = "ContentPage - простейший тип страницы",
                    FontSize = Device.GetNamedSize(NamedSize.Large, typeof(Label)),
                };
```

```
            Label label2 = new Label
            {
                Text = "ContentPage содержит макет " +
                       "содержащий несколько" +
                       "эоементов управления",
                FontSize = Device.GetNamedSize(NamedSize.Large, typeof(Label)),
            };

            Label label3 = new Label
            {
                Text = "Эта страница типа ContentPage содержит " +
                   "макет StackLayout, который содержит  " +
                   "четыре Label, включая заголовок наверху",
                FontSize = Device.GetNamedSize(NamedSize.Large, typeof(Label)),
            };

            // Build the page.
            Title = "ContentPage Demo";
            Padding = new Thickness(10, 0);
            Content = new StackLayout
            {
                Children =
                {
                header,
                label1,
                label2,
                label3
                }
                };
            }
        }
    }
}
```

Next, you need to compile the software application.

When you launch the software application, the following user interface should be displayed



The FlyoutPage page controls two pages, which are defined by the Flyout and Detail properties. The Flyout property is used to specify the main page that displays the list or menu. The Detail property is used to specify a secondary page that displays in more detail the information that is selected on the main page in the list or menu.

menu.

The IsPresented property determines which page is being displayed – primary (IsPresented property is true) or secondary (IsPresented property is false).

Let's move on to working with the FlyoutPage page. To do this, we'll create a project named FPageCS for a Xamarin.Forms multiplatform application using the "Empty" template.

Let's create an auxiliary class NamedColor, which is necessary for working with colors and their names, which is performed in the program file MainPage.cs, which will be created later. Open the file NamedColor.cs and type in the file the program code in C Sharp language, given in the presentation. In the given program code, the constructor of the NamedColor class has two input parameters of type string (for receiving and outputting the color name) and Color (for receiving and outputting the color itself).

```
using System;
using System.Collections.Generic;
using System.Text;
using Xamarin.Forms;

namespace FPageCS
{
    public class NamedColor
  {
      public NamedColor()
      {
         ;
      }
      public NamedColor(string name, Color color)
      {
         Name = name;
         Color = color;
      }
      public string Name { set; get; }
      public Color Color { set; get; }
      public override string ToString()
      {
         return Name;
      }
  }
}
```

Let's create an auxiliary class NamedColorPage, which is necessary later for working with the auxiliary page, which corresponds to the Detail property of the MainPage page.

Open the NamedColorPage.cs file and type the following code in the file in C Sharp language, given below.

The code below creates a ContentPage type page that contains two Layout type layouts and a BoxView control.

In the program code, a BoxView control is first created, into which the SetBinding method is used to bind to the color selected on the main page in the ListView control.

Next, the CreateLabel function is formed, which is designed to form elements of six Label controls that display data about the color selected on the main page in the ListView control.

Next, the Content property is used to generate pages that contain:

a StackLayout type layout, inside which there are three Label controls, which display the components of the color selected on the main page (red, green, blue), control;

a BoxView control that displays the color selected on the home page;

a StackLayout containing three Label controls that display the color's hue, saturation, or intensity, and luminosity on a scale from white to black.

```
using System;
using System.Collections.Generic;
using System.Text;
using Xamarin.Forms;

namespace FPageCS
{
    class NamedColorPage: ContentPage
  {
    public NamedColorPage()
    {
      // BoxView control to display color
      BoxView boxView = new BoxView
      {
        WidthRequest = 100,
        HeightRequest = 100,
        HorizontalOptions = LayoutOptions.Center
      };
      boxView.SetBinding(BoxView.ColorProperty, "Color");
```

```
            // Function to generate six Label controls
            Func<string, string, Label> CreateLabel = (string source, string fmt) => {

                Label label = new Label {

                    FontSize = Device.GetNamedSize(NamedSize.Large, typeof(Label)),
                    HorizontalTextAlignment = TextAlignment.End };

                label.SetBinding(Label.TextProperty, new
                    Binding(source, BindingMode.OneWay, null, null, fmt));

                return label; };


            // Build a page (Stacklayout, BoxView, Stacklayout)
            Content = new StackLayout {

                Children = {

                    new StackLayout {

                        HorizontalOptions = LayoutOptions.Center,
                        VerticalOptions = LayoutOptions.CenterAndExpand, Children
                        = {

                            CreateLabel("Color.R", "R = {0:F2}"),
                            CreateLabel("Color.G", "G = {0:F2}"),
                            CreateLabel("Color.B", "B = { 0:F2}"),


                    } },
                    boxView, new
                    StackLayout {
                        HorizontalOptions = LayoutOptions.Center,
                        VerticalOptions = LayoutOptions.CenterAndExpand, Children
                        = {

                            CreateLabel("Color.Hue", "Hue = {0:F2}"),
                            CreateLabel("Color.Saturation", "Saturation = {0:F2}"),
                            CreateLabel("Color.Luminosity", "Luminosity = { 0:F2}")
                        }
                    }

                } };
        }

} }
```

We will form the user interface using
C Sharp program code.

To do this, you must first delete the automatically generated UI code in the MainPage.xaml file. To do this, right-click the MainPage.xaml file in Solution Explorer and select Delete. This action also deletes the MainPage.xaml.cs code file.

Next, you need to create the MainPage class, for which you need to move the cursor to the project without a suffix, right-click, select the Add menu, select Class and type the class name MainPage.cs at the bottom of the dialog box. This will create a file named MainPage.cs. Open the MainPage.cs file.

Add a using Xamarin.Forms statement at the top of your code since FlyoutPage is in a namespace
Xamarin.Forms.

In your code, make the MainPage class derive from the FlyoutPage class. To do this, after the MainPage class name, put a colon and type FlyoutPage.

First, a Label control named header is created, which contains the page title - the text "FlyoutPage Demo". Then, an array namedColors is created,

containing the colors and their
names.

The color names will be displayed as a list in a ListView control that will be located on the main page, that is, the page that corresponds to the Flyout property.

Next, a ListView control is created. The ListView

object is populated with data using the ItemsSource property,
which specifies that the color names from the namedColors array will be used as the list items in the ListView control.

The Margin property of the ListView control specifies the distance
between the edges of the page and the edges of the ListView.

The distance is specified using the Thickness structure, which in the program code has two parameters, corresponding to the horizontal and vertical distance.

The horizontal value will be applied symmetrically to the left and right sides of the ListView, and the vertical value will be applied symmetrically to the top and bottom sides of the ListView control.

Next, using the Flyout property, the main page of the type is formed
ContentPage titled "Color List".

Using the Content property, a StackLayout type layout is placed in the FlyoutPage, in which the previously created Label controls named header, as well as the ListView control, are placed using the Children property.

Next, using the Detail property of the MainPage page, an auxiliary detailPage is created as an instance of the NamedColorPage class,
whose code was discussed earlier. The helper page
displays more detailed information about the color selected in the color list displayed in the ListView control on the main page.

The following code handles the ItemSelected event when a color is selected from a list displayed in a ListView control. When a color is selected, the SelectedItem object is retrieved as
an argument to the ItemSelected event. The sender parameter of the ItemSelected event corresponds to the

ListView control within which the color selection was made.
The helper page detailPage, which is an instance of the NamedColorPage class, is passed information about the selected color (BindingContext) using the Detail property.
The IsPresented property of the MainPage page of the FlyoutPage type is responsible for displaying the main or auxiliary page. If the
main page needs to be displayed first when launching a software application on the screen of a mobile device, then the IsPresented property is assigned the value true, and if the auxiliary page, then the IsPresented property is assigned the value false.
Depending on the type of mobile device on which the software application is running and the orientation of the mobile device screen, several methods are used to control the transition from the main page to the auxiliary page.

The property is responsible for managing the transition between pages. **FlyoutLayoutBehavior,** which can take several values:

- Default - pages are displayed by default according to the platform used
- Popover - an auxiliary page partially overlaps the main one page.
- Split - the main page is displayed on the left and the subpage is displayed on the right.
- SplitOnLandscape — the mobile device screen is divided into two parts in case the screen orientation is landscape
- SplitOnPortrait — the mobile device screen is divided into two parts in case the screen orientation is portrait

```
using System;
using System.Collections.Generic;
using System.Text;
using Xamarin.Forms;

namespace FPageCS
```

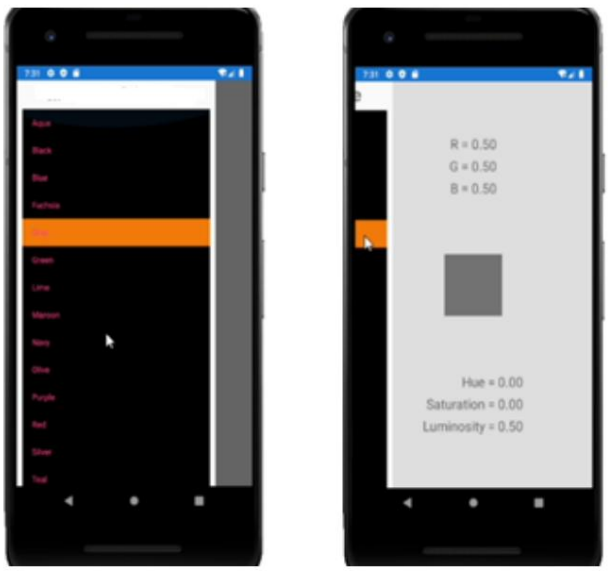
```
{
    class MainPage: FlyoutPage {

        public MainPage() {

            Title = "FlyoutPage Demo";

            Label header = new Label

            { Text = "FlyoutPage",
                FontSize = 30,
                FontAttributes = FontAttributes.Bold,
                HorizontalOptions = LayoutOptions.Center };


            // Array of NamedColor objects
            NamedColor[] namedColors = { new
                NamedColor("Aqua", Color.Aqua), new
                NamedColor("Black", Color.Black), new
                NamedColor("Blue", Color.Blue), new
                NamedColor ( "Fuchsia", Color.Fuchsia), new
                NamedColor ("Gray", Color.Gray), new
                NamedColor ("Green", Color.Green), new
                NamedColor ("Lime", Color.Lime), new
                NamedColor ("Maroon ", Color.Maroon), new
                NamedColor ("Navy", Color.Navy), new
                NamedColor ("Olive", Color.Olive), new
                NamedColor ("Purple", Color.Purple), new
                NamedColor ("Red", Color.Red), new
                NamedColor("Silver", Color.Silver), new
                NamedColor("Teal", Color.Teal), new
                NamedColor("White", Color.White), new
                NamedColor("Yellow", Color. Yellow) };


            // Create a ListView control for the flyout page.
            ListView listView = new ListView {

                ItemsSource = namedColors,
                Margin = new Thickness(10, 0),
                BackgroundColor=Color.Black,

            };

            // Generate a flyout page using the ListView control.
            Flyout = new ContentPage {

                Title = "Color List",
                Content = new StackLayout
```

```
            {
                Children = {
                    header,
                    listView
                }
            }
        };

        // Create a detail page using the NamedColorPage class
        NamedColorPage detailPage = new NamedColorPage();
        Detail = detailPage;

        // Define a selected handler for the ListView.
        listView.ItemSelected += (sender, args) => {
            // Set the BindingContext of the detail page.
            Detail.BindingContext = args.SelectedItem;
            // Show the detail page.
            IsPresented = true;
        };
        // Initialize the ListView selection.
        listView.SelectedItem = namedColors[5];
        FlyoutLayoutBehavior = FlyoutLayoutBehavior.Popover;
    }
  }
}
```

After launching the software application, the main page is displayed, which displays a list of colors, which is equipped with the ability to scroll. At first, gray is selected in the list of colors.

To view the auxiliary page, which contains more detailed information about the selected color, you need to shift the main page to the left. Next, select the red color and also to view the auxiliary page, shift the main page to the

left.



Next, we will consider working with a page of the NavigationPage type, which is used to manage navigation between pages (dialog boxes) using a stack-based architecture. When navigating between pages in a software application, an instance of the page must be passed to the constructor of the NavigationPage object. Class

NavigationPage provides hierarchical navigation where the user can move forward and backward through pages. Class
NavigationPage implements navigation based on a last-in, first-out (LIFO) stack of Page objects. When navigating from one page to another, the application pushes the new page (the one being navigated to) onto the navigation stack, where it is placed at the top and becomes the active page.

To return from the current page to the previous page, the application removes the current page from the top of the navigation stack, after which the page that is on the top of the stack becomes active.

Navigation methods are provided by the Navigation property for any page type that derives from the Page class. For hierarchical navigation, the NavigationPage

class is used to navigate a stack of ContentPage objects. The first page added to the navigation stack is called the root page of the application.

To work with a page of type NavigationPage, a project with the name is created NavPage for a Xamarin.Forms multiplatform application using the "Empty" template. The user interface will be generated using C Sharp code. To do this, you must first delete the automatically generated user interface code located in the MainPage.xaml file. To do this, right-click on

MainPage.xaml file and select the "Delete" menu. This also deletes the MainPage.xaml.cs file associated with the MainPage.xaml file.

Next, you need to open the App.xaml.cs file in the project and add the program code to it, which adds the root page Page1 to the navigation stack using the NavigationPage class. As a result, the page Page1

is placed on the top of the stack, becomes active, and is displayed first when the application starts.

```
using System;
using Xamarin.Forms;
using Xamarin.Forms.Xaml;

namespaceNavPage
{
   public partial class App: Application
   {
      public App()
      {
        MainPage = new NavigationPage(new Page1());
      }
   }
}
```

Next, the LabelPage page is created, to which the transition from Page1 will be made. Also, the transition from Page1 to BoxViewPage will be made.

The LabelPage class is created using the steps for creating a class discussed earlier. Next, you need to open the created file LabelPage.cs and make the LabelPage class a child of the ContentPage class. Since ContentPage and NavigationPage pages are supported by Xamarin.Forms, you need to add a using Xamarin.Forms statement at the top of your code to access the appropriate namespace.

Next, you need to type the program code in the C Sharp language in the file, given below. The

page contains a StackLayout type layout with two labels and two buttons for going to the BoxViewPage page and returning to the Page1 page. Going to the

BoxViewPage page is done by clicking the NextPage button. The event handler that occurs when the NextPage button is clicked calls the PushAsync method of the Navigation property of the LabelPage page. Returning to the Page1 page is done by clicking the PrevPage button. The event handler that occurs when the PrevPage button is clicked calls the PopAsync method.

As a result, the LabelPage page is removed from the navigation stack, and the top page in the stack, i.e. Page1, becomes active. **namespace**

```
NavPage
{
   class LabelPage: ContentPage
   {
      public LabelPage()
      {
        Label header = new Label
        {
```

```
    Text = "Label",
    FontSize = 50,
    FontAttributes = FontAttributes.Bold,
    HorizontalOptions = LayoutOptions.Center
};

Label label = new Label
{
   Text = "Xamarin.Forms allows you to create "user " +
 interfaces that are "common across Android and " +
 iOS platforms",
   FontSize = Device.GetNamedSize(NamedSize.Large,
                                  typeof(Label)),
    VerticalOptions = LayoutOptions.CenterAndExpand,
    Margin = new Thickness(10, 0)
};

Button nextPageButton = new Button
{ Text = "Next Page",
  VerticalOptions = LayoutOptions.CenterAndExpand };

nextPageButton.Clicked += NextPage;

Button previousPageButton = new Button
{ Text = "Previous Page",
  VerticalOptions = LayoutOptions.CenterAndExpand };

previousPageButton.Clicked += PrevPage;

// Page construction
Title = "Label Demo";
Content = new StackLayout
{
    Children =
    {
        header,
        label,
        nextPageButton,
        previousPageButton
    }
};

async void NextPage(object sender, EventArgs e)
{
    await Navigation.PushAsync(new BoxViewPage());
```

```
            }
            async void PrevPage(object sender, EventArgs e)
            {
                await Navigation.PopAsync();
            }
        }
    }
}
```

The BoxViewPage page is created, to which the transition from the Page1 and LabelPage pages will be made. To do this, you need to create a class BoxViewPage, open the created BoxViewPage.cs file and make a class BoxViewPage is a child of the ContentPage class.

Next, you need to add the using Xamarin.Forms statement at the top of the program code. After that, you need to type the program code in the C Sharp language in the file, given below.

The page contains a StackLayout type layout with controls Label and BoxView and a button to return to the LabelPage and Page1 pages.

Return to previous pages is performed by clicking the NextPage button. This calls the PopAsync method. As a result, the BoxViewPage page is removed from the navigation stack, and the top page in the stack, i.e. Page1 or LabelPage, becomes active.

```
namespaceNavPage
{
    class BoxViewPage: ContentPage
    {
        public BoxViewPage()
        {
            Label header = new Label
            {
                Text = "BoxView",
                FontSize = 50,
                FontAttributes = FontAttributes.Bold,
                HorizontalOptions = LayoutOptions.Center
            };
            BoxView boxView = new BoxView
            {
                Color = Color.Orange,
                WidthRequest = 200,
                HeightRequest = 200,
                HorizontalOptions = LayoutOptions.Center,
                VerticalOptions = LayoutOptions.CenterAndExpand
            };
            Button previousPageButton = new Button
            {
                Text = "Previous Page",
```

```
            VerticalOptions = LayoutOptions.CenterAndExpand
        };
        previousPageButton.Clicked += PrevPage;
        // Page construction
        Title = "BoxView Demo";
        Content = new StackLayout
        {
            Children =
            {
                header,
                boxView,
                previousPageButton
            }
        };
        async void PrevPage(object sender, EventArgs e)
        {
            await Navigation.PopAsync();
        }
    }
  }
}
```

Let's move on to creating Page1. Create a Page1 class, open the Page1.cs file you created, and make Page1 a child of the ContentPage class. Add the using Xamarin.Forms

statement at the top of the code.

code, since ContentPage is in the Xamarin.Forms namespace.

Type in the file the program code in C Sharp language, given in the presentation. Inside the page there is a layout with two buttons for going to the LabelPage and BoxViewPage pages. Also in the

program code there are handlers of events of pressing the buttons. The handlers use the PushAsync method of the Navigation property of the Page1 page to go to the LabelPage and BoxViewPage pages.

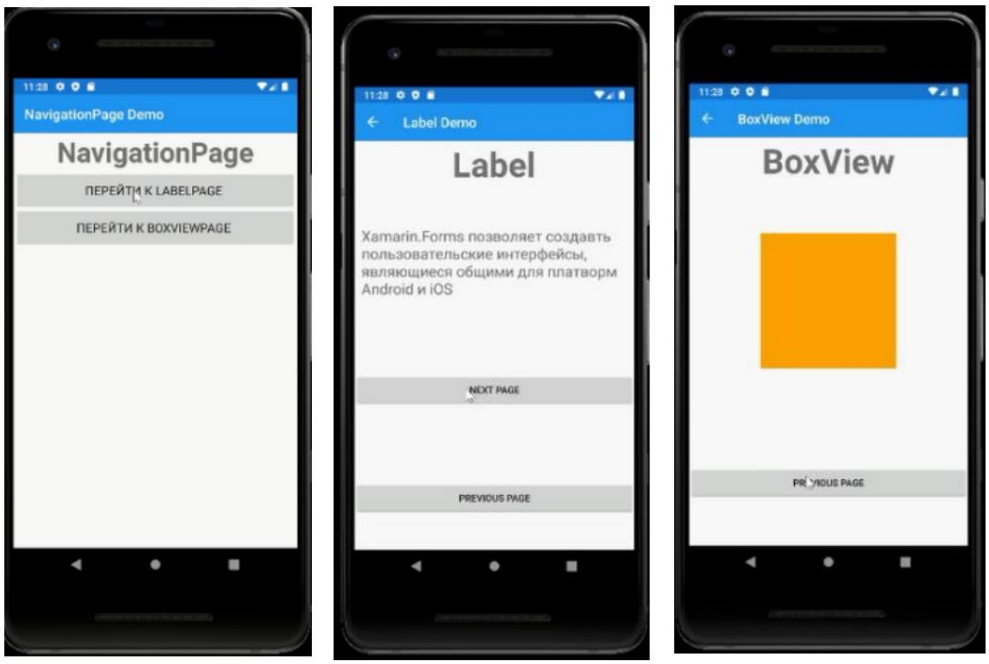
```
namespaceNavPage
{
  class Page1: ContentPage
  {
    public Page1()
    {
      Label header = new Label
      {
        Text = "NavigationPage",
        FontSize = 40,
        FontAttributes = FontAttributes.Bold,
        HorizontalOptions = LayoutOptions.Center
      };
```

```
            Button button1 = new Button {

                Text = "Go to LabelPage ", Font =
                Font.SystemFontOfSize(NamedSize.Large), BorderWidth = 1,
                WidthRequest=400 };
                button1.Clicked +=

            async (sender, args) =>
                await Navigation.PushAsync(new LabelPage());

            Button button2 = new Button {

                Text = "Go to BoxViewPage ", Font =
                Font.SystemFontOfSize(NamedSize.Large), BorderWidth = 1,
                WidthRequest = 400 };
                button2.Clicked += async

            (sender, args) =>
                await Navigation.PushAsync(new BoxViewPage());
            // Page construction
            Title = "NavigationPage Demo";
            Content = new StackLayout {

                Children = {

                    header,
                    new StackLayout {

                        HorizontalOptions = LayoutOptions.Center, Children =
                        {

                            button1,
                            button2
                        }
                    }

                } };
        }
    }
```

} When the software application is launched, the Page1 page is displayed on the mobile device screen with two buttons “Go to LabelPage” and “Go to BoxViewPage”.

Clicking the "Go to LabelPage" button takes you to the LabelPage page, from which you can go to the BoxViewPage page or

return back to the root page Page1. If you go to the BoxViewPage page, you can return to the LabelPage page.



Next, let's look at working with the **TabbedPage page,** which is a child of the abstract MultiPage class and allows you to navigate between pages using a list of tabs.

On iOS, a list of tabs appears at the bottom of the screen, as well as in the data area at the top.

Each tab has a title and can also have an icon, which must be a PNG file.

In portrait orientation, the tab bar icons appear above tab titles.

In landscape orientation, icons and titles appear side by side.

Additionally, depending on your device and orientation, you may see either the regular or compact tab bar.

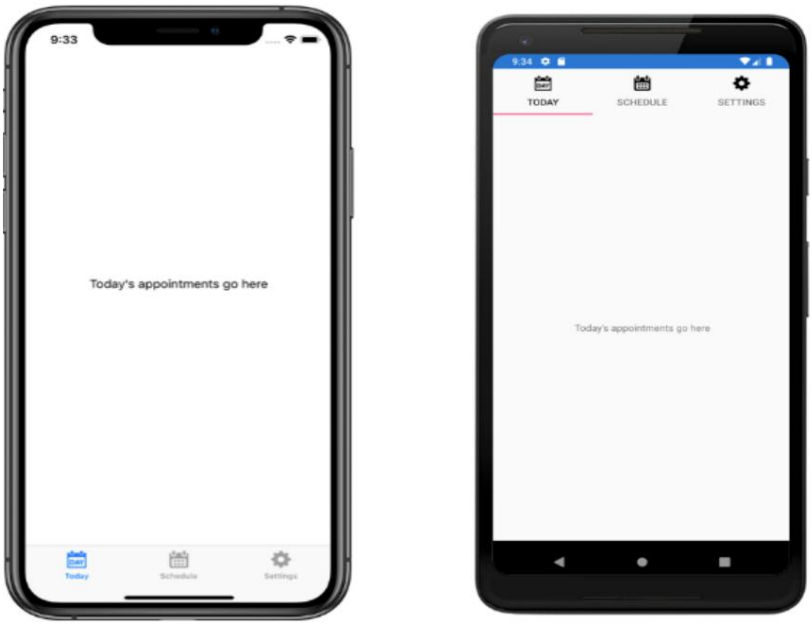
If you have more than five tabs, a More tab appears, which is used to access additional tabs.

On Android, a list of tabs appears at the top of the screen, as well as in the data area below.

Each tab has a title and an icon, which must be a PNG file with an alpha channel.

But these tabs can be moved to the bottom of the screen in depending on the specific platform.

If you have more than five tabs listed at the bottom of the screen, a More tab appears that you can use to access additional tabs.



There are two ways to create a TabbedPage:

1. The TabbedPage is filled with a collection of child objects.
type Page, for example, pages of type ContentPage.

2. Assign the Page collection to the ItemsSource property of the TabbedPage page, and then assign the DataTemplate to the ItemTemplate property to display the pages included in the page uniformly.
composition of the collection.

With both approaches, TabbedPage displays a page that corresponds to the selected tab.

A TabbedPage has the following properties:

BarBackgroundColor (type Color) — background color of the tab bar;

BarTextColor (Color type) — text color on the tab bar;

SelectedTabColor (type Color) — the color of the selected tab;

UnselectedTabColor (type Color) - the color of the tab if it is not selected.

Let's create a TabPage project for a multiplatform Xamarin.Forms project using the "Empty" template. Open the file MainPage.xaml.cs and adjust the program code in accordance with the presentation. From this file it is clear that the MainPage page is a TabbedPage type page.

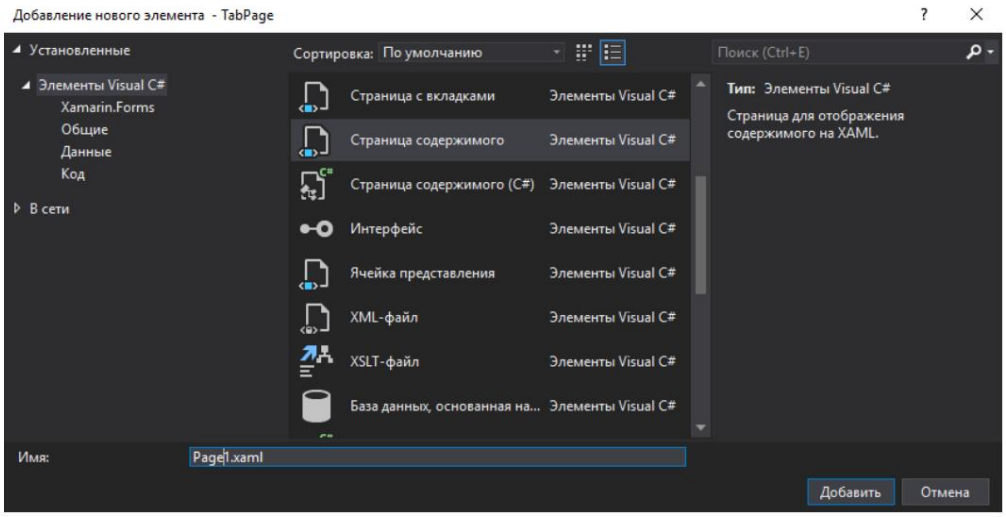
```
using Xamarin.Forms;
namespace TabPage
{
	public partial class MainPage : TabbedPage
	{
		public MainPage()
		{
			InitializeComponent();
		}
	}
}
```

Let's open the MainPage.xaml file and type the program code given below.
in the presentation.

In the MainPage XAML code, Page1, Page2, and Page3 are added to the TabbedPage collection using the Children property.

```
<TabbedPage xmlns="http://xamarin.com/schemas/2014/forms"
        xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
        xmlns:local="clr-namespace:TabPage ;assembly=TabPage"
        x:Class="TabPage.MainPage">
   <TabbedPage.Children>
      <local:Page1 />
      <local:Page2 />
      <local:Page3 />
   </TabbedPage.Children>
</TabbedPage>
```

Let's add the Page1 page to the project. To do this, move the mouse cursor to the general TabPage project, right-click and select "Add" - "New Element". In the "Add New Element" dialog box, select "Content Page", which creates a ContentPage type page. Name the page Page1.



You need to open the Page1.xaml file and type the xaml program code Page1, which is referenced from the program code
MainPage page. The page contains a StackLayout type layout, inside which there is a Label control with explanatory text and an orange-red BoxView control.

```
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
        xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
        xmlns:d="http://xamarin.com/schemas/2014/forms/design"
        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
        mc:Ignorable="d"
```

```
        x:Class="TabPage.Page1"
        BackgroundColor="Coral"
        Title=" BoxView "">
  <ContentPage.Content>
    <StackLayout>
      <Label Text="BoxView is for
          display a rectangle with a given width, height
          and color"
        VerticalTextAlignment="Center"
        HorizontalTextAlignment="Center"
        WidthRequest="300" BackgroundColor="Brown"
        HeightRequest="300" TextColor="Yellow"
        FontAttributes="Bold"
        FontSize="30" /
        >
      <BoxView
        WidthRequest="250"
        HeightRequest="250"
        VerticalOptions="CenterAndExpand"
        HorizontalOptions="Center"
        BackgroundColor="OrangeRed"/
    > </StackLayout>
  </ContentPage.Content>
</ContentPage>
```

Let's add Page2 to the project. You need to open the file Page2.xaml and type the xaml program code of the Page2 page, which is referenced from the program code of the MainPage page. The page contains a StackLayout type layout, inside which there is a Label control with explanatory text and an Editor control of the color "Bisque".

```
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
        xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
        xmlns:d="http://xamarin .com/schemas/2014/forms/design"
        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
        mc:Ignorable="d"
        x:Class="TabPage.Page2"
        BackgroundColor="DarkGray"
        Title="Editor">
  <ContentPage.Content>
    <StackLayout>

      <Label Text="Editor allows the user
          enter and edit multiple lines of text"

        VerticalTextAlignment="Center"
        HorizontalTextAlignment="Center"
        WidthRequest="300" BackgroundColor="Brown"
```

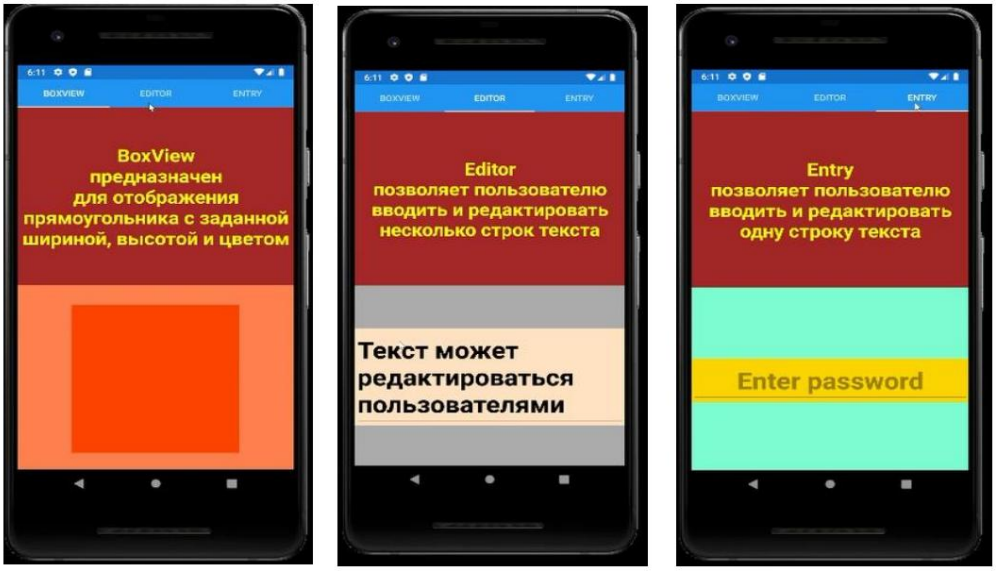
```
            HeightRequest="300" TextColor="Yellow"
            FontAttributes="Bold"
            FontSize="30"/>

        <Editor
            BackgroundColor="Bisque"
            VerticalOptions="CenterAndExpand"
                TextColor="Black"
            Text="Text can be edited by users"
            FontSize="40" FontAttributes="Bold"/>
    </StackLayout>
  </ContentPage.Content>
</ContentPage>
```

Let's add the page Page3 to the project. You need to open the file Page3.xaml and type the xaml program code of the Page3 page, which is referenced from the program code of the MainPage page. The page contains a StackLayout type layout, inside which there is a Label control with explanatory text and an Entry control of the color "Gold".

```
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
        xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
        xmlns:d="http://xamarin .com/schemas/2014/forms/design"
        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
        mc:Ignorable="d"
        x:Class="TabPage.Page3"
        BackgroundColor="Aquamarine"
        Title="Entry">
  <ContentPage.Content>
    <StackLayout>

        <Label Text="Entry allows the user
            enter and edit one line of text."

            VerticalTextAlignment="Center"
            HorizontalTextAlignment="Center"
            WidthRequest="300" BackgroundColor="Brown"
            HeightRequest="300" TextColor="Yellow"
            FontAttributes="Bold"
            FontSize="30"/>

        <Entry Keyboard="Text"
            BackgroundColor="Gold"
            HorizontalTextAlignment="Center"
            Placeholder="Enter password"
            VerticalOptions="CenterAndExpand"
            FontSize="40" FontAttributes="Bold"/>
```

```
    </StackLayout>
  </ContentPage.Content>
</ContentPage>
```

Let's run the software application. A page with three tabs is displayed. It is possible to switch between the tabs, which display the names of the controls used to form the user interface and explain the purpose of the controls.



The **CarouselPage** in Xamarin.Forms allows you to navigate between pages using a swipe gesture, which is when a user swipes their finger across the touchscreen to the right to move forward through a collection of pages, or to the left to move backwards.

If a CarouselPage object is embedded in the Detail page of a FlyoutPage, the FlyoutPage.IsGestureEnabled property must be set to false to prevent conflicts between pages.

CarouselPage and FlyoutDetailPage.

There are two ways to create a CarouselPage:

1. It is necessary to assign a collection of pages to the Children property of the CarouselPage page. In this case, the collection can only include pages of the ContentPage type or objects of the ContentPage type.

2. Assign the ItemsSource property of the CarouselPage page a collection of pages. Then assign the DataTemplate (a description of the template that specifies the same display for each page from the collection specified in the ItemsSource property) to the ItemTemplate property.

Both methods result in the pages in the CarouselPage collection being displayed one by one as you swipe across the screen.

Let's create a CarPage project for a multiplatform Xamarin.Forms project using the "Empty" template. Open the file MainPage.xaml.cs and adjust the program code in accordance with

presentation. From this file it is clear that the MainPage page is a CarouselPage type page.

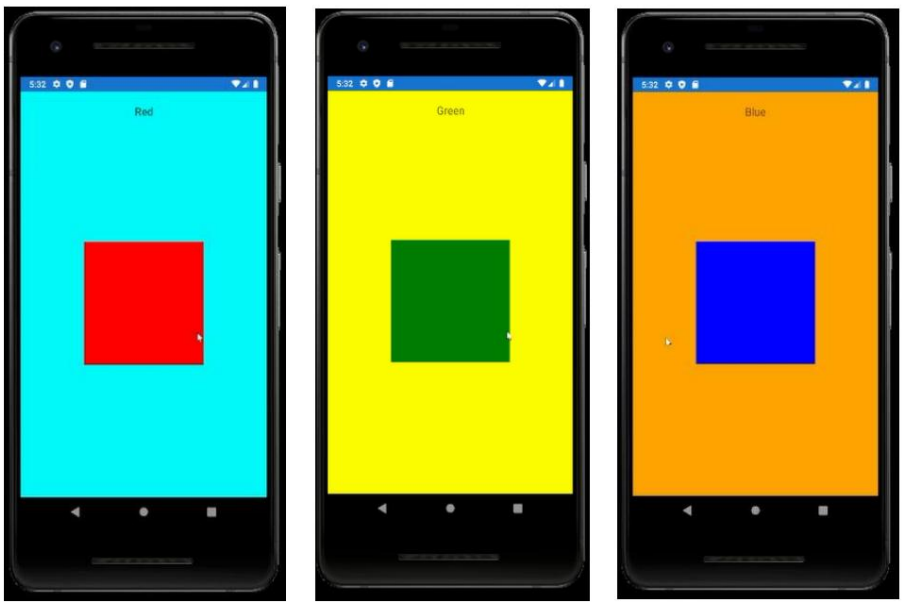
```
using Xamarin.Forms;
namespaceCarPage
{
        public partial class MainPage : CarouselPage
        {
                public MainPage()
                {
                        InitializeComponent();
                }
        }
}
```

Let's open the MainPage.xaml file. This file contains the program code XAML that corresponds to the CarouselPage containing a collection of three ContentPages. In this software application, the CarouselPage is created using the first of the above methods (i.e., a collection of ContentPages). The collection consists of three ContentPages. Each page contains a StackLayout with Label and BoxView controls inside. The first page has a Cyan background and a red BoxView control. The second page has a yellow background and a green BoxView control. The third page has an orange background and a blue BoxView control.

```
<?xml version="1.0" encoding="UTF-8"?>
<CarouselPage xmlns="http://xamarin.com/schemas/2014/forms"
          xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
                x:Class="CarPage.MainPage">
  <ContentPage
    BackgroundColor="Cyan" Padding="20">
    <StackLayout>
      <Label Text="Red"
          FontSize="Medium" FontAttributes="Bold"
          HorizontalOptions="Center" />
      <BoxView Color="Red"
           WidthRequest="200" HeightRequest="200"
           HorizontalOptions="Center"
           VerticalOptions="CenterAndExpand" />
    </StackLayout>
  </ContentPage>
  <ContentPage
    BackgroundColor="Yellow" Padding="20">
    <StackLayout>
      <Label Text="Green"
          FontSize="Medium"
          HorizontalOptions="Center" />
      <BoxView Color="Green"
```

```
                WidthRequest="200" HeightRequest="200"
                HorizontalOptions="Center"
                VerticalOptions="CenterAndExpand" />
        </StackLayout>
    </ContentPage>
    <ContentPage
        BackgroundColor="Orange" Padding="20">
        <StackLayout>
            <Label Text="Blue"
                FontSize="Medium"
                HorizontalOptions="Center" />
            <BoxView Color="Blue"
                WidthRequest="200" HeightRequest="200"
                HorizontalOptions="Center"
                VerticalOptions="CenterAndExpand" />
        </StackLayout>
    </ContentPage>
</CarouselPage>
```

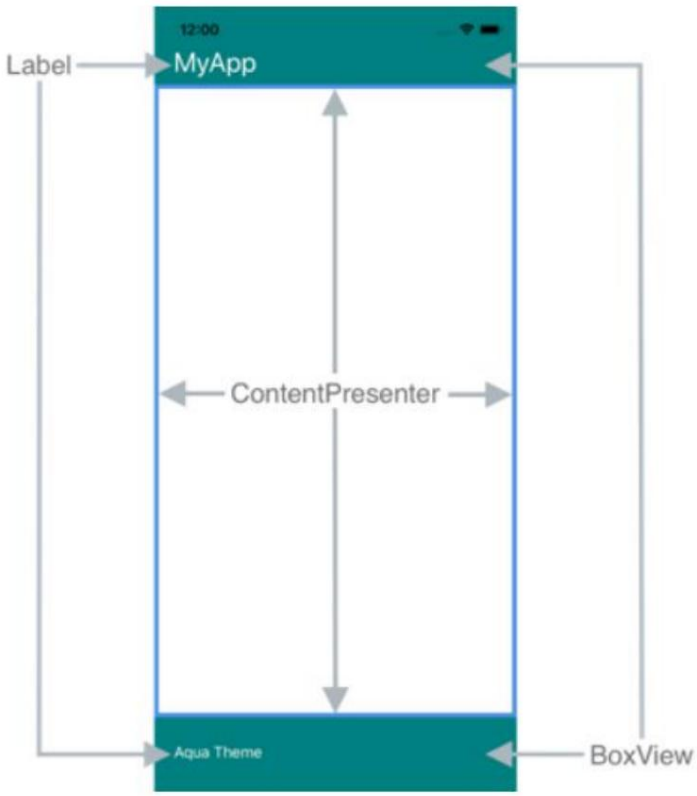
After launching the software application, the first page with a red filled BoxView control is displayed. As you scroll, pages with green and blue filled BoxView controls will appear.



The **TemplatedPage** page allows you to define a template to set a standard visual structure for pages of the ContentPage type. In this way, a page template can be used to display multiple pages in the same way. In this case, the controls in the pages will be displayed uniformly, taking into account the settings contained in the template.

For a template page that contains multiple controls, the ControlTemplate template is used, which specifies

controls included in the template. In the template, you can specify the ContentPresenter control, which contains the elements
controls passed to a template from the page to which the template is applied. The figure shows a ControlTemplate for a page that contains Label controls, BoxView controls, and a rectangle ContentPresenter.



Let's create a TemPage project for a multiplatform Xamarin.Forms project using the "Empty" template. Let's create a file HeaderFooterPage.xaml, which defines the "TealTemplate" templates for formation of the main page MainPage, the program code of which refers to the template and will be created later. We will type the program code in accordance with the presentation.

At the beginning of the page, settings are set for the Entry controls located in the MainPage page. The settings are set using Style and the TargetType property. The visual display style of the controls is configured when working with them (that is, when the focus is transferred to them).

Typically, a ControlTemplate is declared as a resource. If a ControlTemplate is declared as a resource, it must have a key specified using the x:Key attribute.

The ControlTemplate template has only one control as its root element. However, the root control contains other controls.

The program code shows the templates OrangeTemplate and GreenTemplate. The root element of each template is the element

Grid controls. The combination of controls within the root control makes up the visual structure of the template page. Within the root element are three cells arranged

one below the other. The first cell contains the page title.

The first cell has a height of 10% of the page height and contains a BoxView control that fills the entire cell, as well as a Label control that displays the name of the template being used.

These controls have been customized for placement, fill color, font color, font size, and margins from the edge of the control.

The second cell contains the ContentPresenter control, which will contain the controls that are located on the page that accesses the template. The second cell has a height of 80% of the page height. The third cell is the page header, has a height of 10% of the template page height, and contains a BoxView control that

fills the entire cell, as well as a Label control that must be clicked if the template used needs to be switched. These controls are also configured for placement, fill color, font color and size, text style, and margins from the edge of the control.

The TemplateBinding markup extension, using the HeaderText property, enables the Text property of a templated Label control to be set from code in the MainPage.xaml file.

It is also worth noting that the Label control located at the bottom of the page has an event handler attached to it.
OnChangeTheme, which is designed to handle the touch gesture and is listed in the HeaderFooterPage.xaml.cs file, which will be discussed later. The control is also associated with the OnApplyTemplate procedure, which is responsible for changing the page display template and is listed in the MainPage.xaml.cs file.

```
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
        xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
        xmlns:controls="clr-namespace: TemPage"
        x:Class="TemPage.HeaderFooterPage">
  <ContentPage.Resources>
    <Style TargetType="Entry">
      <Setter Property="VisualStateManager.VisualStateGroups">
        <VisualStateGroupList>
          <VisualStateGroup x:Name="CommonStates">
            <VisualState x:Name="Normal">
              <VisualState.Setters>
                <Setter Property="FontSize"
                    Value="18" />
              </VisualState.Setters>
```

```
            </VisualState>
            <VisualState x:Name="Focused">
               <VisualState.Setters>
                  <Setter Property="FontSize"
                          Value="36" /> </
               VisualState.Setters> </
            VisualState> </
         VisualStateGroup> </
      VisualStateGroupList> </
   Setter> </
Style>

<ControlTemplate x:Key="OrangeTemplate">
   <Grid>
      <Grid.RowDefinitions>
         <RowDefinition Height="0.1*" />
         <RowDefinition Height="0.8*" />
         <RowDefinition Height="0.1*" /> </
      Grid.RowDefinitions>
      <BoxView Color="Orange" />
      <Label Margin=" 20.0.0.0"
          Text="{TemplateBinding HeaderText}"
          TextColor="White"
          FontSize="20"
          HorizontalOptions="Center"
          VerticalOptions="Center" />
      <ContentPresenter Grid.Row="1" />
      <BoxView Grid.Row="2"
           Color="Orange" />
      <Label x:Name="ChangeThemeLabel"
          Grid.Row="2"
          Margin="20,0,0,0"
          Text="Change Theme"
          TextColor="White"
          FontSize="20"
          HorizontalOptions="Center"
          VerticalOptions="Center">
         <Label.GestureRecognizers>
            <TapGestureRecognizer
               Tapped="OnChangeTheme" />
         </Label.GestureRecognizers> </
      Label> </
   Grid> </
ControlTemplate>

<ControlTemplate x:Key="GreenTemplate">
   <Grid>
      <Grid.RowDefinitions>
         <RowDefinition Height="0.1*" />
```

```
            <RowDefinition Height="0.8*" />
            <RowDefinition Height="0.1*" />
          </Grid.RowDefinitions>
          <BoxView Color="Green" />
          <Label Margin="20,0,0,0"
              Text="{TemplateBinding HeaderText}"
              TextColor="Yellow"
              FontSize="20"
              HorizontalOptions="Center"
              VerticalOptions="Center" />
          <ContentPresenter Grid.Row="1" />
          <BoxView Grid.Row="2"
               Color="Green" />
          <Label x:Name="ChangeThemeLabel"
              Grid.Row="2"
              Margin="20,0,0,0"
              Text="Change Theme"
              FontSize="20" FontAttributes="Bold"
              TextColor="Yellow"
              HorizontalOptions="Center"
              VerticalOptions="Center">
            <Label.GestureRecognizers>
              <TapGestureRecognizer
                Tapped="OnChangeTheme" />
            </Label.GestureRecognizers>
          </Label>
        </Grid>
      </ControlTemplate>
    </ContentPage.Resources>
</ContentPage>
```

Let's open the HeaderFooterPage.xaml.cs file. Type the program code in the file code according to presentation.

In the program code, the previously discussed ones are first declared OrangeTemplate and GreenTemplate templates.

Then the HeaderTextProperty property is declared to write the value of the HeaderText template property. The HeaderFooterPage page of type ContentPage is initialized.

The OriginalTemplate property and the auxiliary variable original are used to track switching between the OrangeTemplate and GreenTemplate templates. The handler is designed

to handle the gesture of pressing the Label control, which is intended to change the page template.
OnChangeTheme, which is referenced by the xaml code of the page HeaderFooterPage.

```
using Xamarin.Forms;
using Xamarin.Forms.Xaml;
```

```
namespaceTemPage
{
   [XamlCompilation(XamlCompilationOptions.Compile)]
   public partial class HeaderFooterPage : ContentPage
   {
      ControlTemplate OrangeTemplate;
      ControlTemplate GreenTemplate;

      public static readonly BindableProperty
         HeaderTextProperty =
         BindableProperty.Create(nameof(HeaderText),
                        typeof(string),
                        typeof(HeaderFooterPage),
                        default(string));
      public string HeaderText
      {
         get => (string)GetValue(HeaderTextProperty);
         set => SetValue(HeaderTextProperty, value);
      }
      bool original = true;
      public bool OriginalTemplate
      {
         get { return original; }
      }

      public HeaderFooterPage()
      {
         InitializeComponent();
         OrangeTemplate =
            (ControlTemplate)Resources["OrangeTemplate"];
         GreenTemplate =
            (ControlTemplate)Resources["GreenTemplate"];
      }
      void OnChangeTheme(object sender, EventArgs e)
      {
         original = !original;
         ControlTemplate =
            (original) ? OrangeTemplate: GreenTemplate;
      }
   }
}
```

Let's open the MainPage.xaml file and type the program code according to the presentation. It should

be noted that the program code makes a reference to the HeaderFooterPage page, which stores templates, as well as settings for the Entry controls.

The HeaderText property is used to pass the text string "TemplatedPage" to the Text property value of the Label control located in the header of the templated page.

The MainPage to which the template is applied passes a StackLayout layout with two Entry controls and a Button control to the template page's ContentPresenter control. The Entry controls are given the default display settings specified in the HeaderFooterPage code.

```
<controls:HeaderFooterPage xmlns="http://xamarin.com/schemas/2014/forms"
        xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
        xmlns:controls="clr-namespace:TemPage"
        x:Class="TemPage.MainPage"
        Title="Access template"
        HeaderText="TemplatedPage Page"
        ControlTemplate="{StaticResource OrangeTemplate}">
  <StackLayout Margin="10">
    <Entry Placeholder="Enter username" />
    <Entry Placeholder="Enter password"
          IsPassword="True" />
    <Button Text="Login" />
  </StackLayout>
</controls:HeaderFooterPage>
```

Let's open the MainPage.xaml.cs file and type the program code in according to the presentation.

The TemLabel variable is used to link to the Label control located in the header of the template page and named ChangeThemeLabel according to the x:Name attribute.

The OnApplyTemplate procedure is designed to change the text in the Label control on the template page when this control is clicked when the page display template is changed.

```
using Xamarin.Forms;
namespaceTemPage
{
  public partial class MainPage : HeaderFooterPage
  {
    Label TemLabel;
    public MainPage()
    {
      InitializeComponent();
    }
    protected override void OnApplyTemplate()
    {
      base.OnApplyTemplate();
```

```
            TemLabel = (Label)GetTemplateChild("ChangeThemeLabel");
            TemLabel.Text =
                OriginalTemplate? "Orange Theme" : "Green Theme";
        }
    }
}
```

We will build a software application, after which there should be no there will be errors, and we will launch the software application.

First the page is displayed according to the template OrangeTemplate.

Inside the page are two Entry controls and a Button control, which are passed to the ContentPresenter template control as part of a StackLayout layout.

Clicking on the Label control in the header, which displays the name of the template being used, changes the display template for the MainPage page.

When you start working with Entry controls, they change their visual state (font size changes).

Once you've finished working with the Entry controls, they return to their original state.