Л12. Xamarin.Forms. Розробка користувацького інтерфейсу засобами C#. Клас ToolbarItem, NavigationPage

1. Контейнер Frame

Контейнер Frame використовується для оформлення або створення фону для вкладеного елемента. Серед властивостей класу Frame слід виділити такі:

- BorderColor: колір кордону кадру за допомогою структури Color;
- CornerRadius: представляє радіус кордону кадру як значення типу float;
- HasShadow: зберігає значення типу bool, яке вказує, чи кадр відкидає тінь.

Кадр може вміщати лише один елемент. Приклад створення кадру в XAML:

```
<Frame>
  <Label Text="Hello Xamarin" />
</Frame>
```

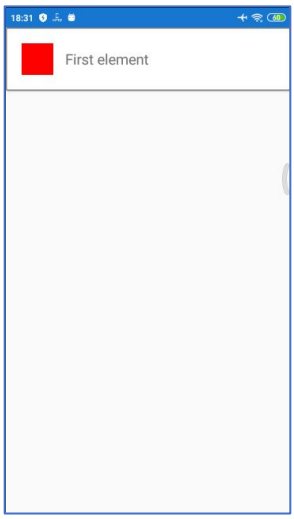
Створення кадру в коді C#:

```
Frame frame = new Frame
{
    Content = new Label { Text = "Hello Xamarin" }
};
```

Створюємо на базі Empti шаблону проєкт зі списку елементів. Кожний елемент списку складається з BoxView та Label

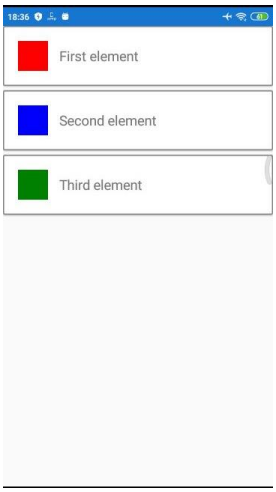
```
<StackLayout>
        <Frame BorderColor="Gray">
            <StackLayout Orientation="Horizontal" Spacing="15">
                <BoxView Color="Red"></BoxView>
                <Label Text="First element" VerticalOptions="Center"></Label>
            </StackLayout>
        </Frame>
    </StackLayout>
```

Запускаємо проєкт на виконання



Дублюємо Frame

```
<StackLayout>
        <Frame BorderColor="Gray">
            <StackLayout Orientation="Horizontal" Spacing="15">
                <BoxView Color="Red"></BoxView>
                <Label Text="First element" VerticalOptions="Center"></Label>
            </StackLayout>
        </Frame>

        <Frame BorderColor="Gray">
            <StackLayout Orientation="Horizontal" Spacing="15">
                <BoxView Color="Blue"></BoxView>
                <Label Text="Second element" VerticalOptions="Center"></Label>
            </StackLayout>
        </Frame>

        <Frame BorderColor="Gray">
            <StackLayout Orientation="Horizontal" Spacing="15">
                <BoxView Color="Green"></BoxView>
                <Label Text="Third element" VerticalOptions="Center"></Label>
            </StackLayout>
        </Frame>
    </StackLayout>
```



Реалізуємо теж саме в коді C#. Видаляємо усе з ContentPage та дописуємо код в метод OnAppearing()

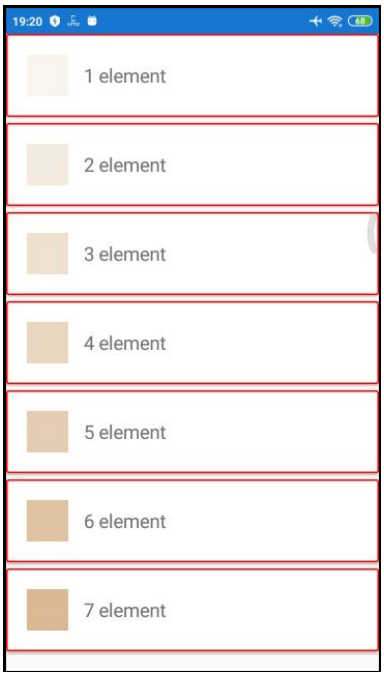
```
protected override void OnAppearing()
{
   StackLayout stackLayout1 = new StackLayout();

   Frame frame1 = new Frame()
   {
       Content = new StackLayout()
       {
           Children =
           {
              new BoxView() { Color = Color.Red},
              new Label() { Text="First element", VerticalOptions=LayoutOptions.Center}
           },
            Orientation = StackOrientation.Horizontal,
            Spacing = 15
        },
        BorderColor = Color.Red
     };
  }
```

Додаємо, ще 2 Frame, додаємо frame1, frame2, frame3 до stackLayuot1 (копіювання Ctrl+D) та Content= stackLayuot1

```
protected override void OnAppearing()
{
  StackLayout stackLayout1 = new StackLayout();
  Frame frame1 = new Frame()
  {
       Content = new StackLayout()
       {
          Children =
          {
             new BoxView() { Color = Color.Red},
             new Label() { Text="First element,VerticalOptions=LayoutOptions.Center }
          },
          Orientation = StackOrientation.Horizontal,
          Spacing = 15
       },
       BorderColor = Color.Red
  };

  Frame frame2 = new Frame()
  {
       Content = new StackLayout()
       {
          Children =
          {
            new BoxView() { Color = Color.Green},
            new Label() {Text="Second element", VerticalOptions=LayoutOptions.Center}
          },
         Orientation = StackOrientation.Horizontal,
         Spacing = 15
       },
       BorderColor = Color.Red
    };

    Frame frame3 = new Frame()
    {
        Content = new StackLayout()
        {
           Children =
           {
             new BoxView() { Color = Color.Blue},
             new Label() {Text="Third element",
VerticalOptions=LayoutOptions.Center}
           },
           Orientation = StackOrientation.Horizontal,
           Spacing = 15
        },
        BorderColor = Color.Red
     };
     stackLayout1.Children.Add(frame1);
     stackLayout1.Children.Add(frame2);
     stackLayout1.Children.Add(frame3);

     Content = stackLayout1;
}
```

Розглянемо як додавати елементи в циклі з одним frame

```
protected override void OnAppearing()
{
    StackLayout stackLayout1 = new StackLayout();
    Frame frame1 = null; //otimizated Memory

    for (int i=1; i<8; i++)
    {
        frame1 = new Frame()
        {
           Content = new StackLayout()
           {
               Children =
               {
                  new BoxView() { Color = Color.FromRgb(255-i*5, 255-i*10, 255-i*15)},
                  new Label() { Text=$"{i} element", VerticalOptions=LayoutOptions.Center}
                },
                Orientation = StackOrientation.Horizontal,
                Spacing = 15
             },
                    BorderColor = Color.Red
         };

     stackLayout1.Children.Add(frame1);
     }
        Content = stackLayout1;
  }
```



2. Клас Xamarin.Forms ToolbarItem

Клас Xamarin.Forms ToolbarItem – це особливий тип кнопки, яку можна додати до колекції Page об'єкта ToolbarItems. Кожен ToolbarItem об'єкт відображатиметься у вигляді кнопки на панелі навігації програми. Екземпляр ToolbarItem може мати значок і відображатись як основний або додатковий

пункт меню. Клас ToolbarItem успадковується від MenuItem. На рис. 1 показано ToolbarItem об'єкти на панелі навігації в iOS та Android.

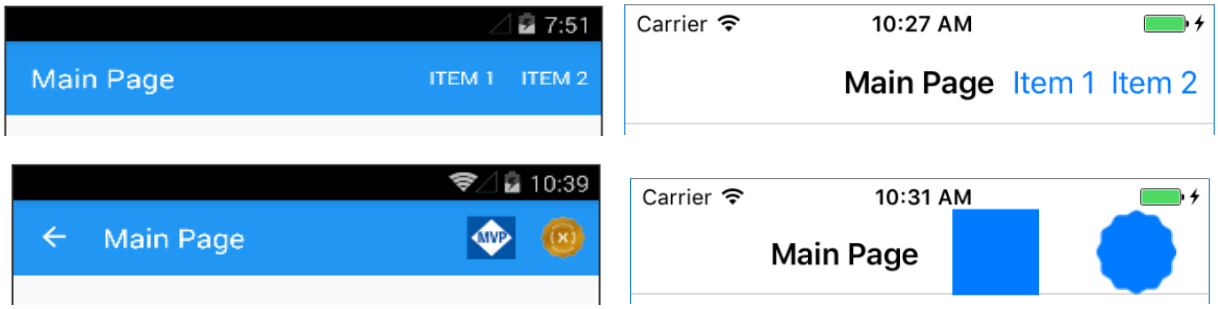


Рисунок 1 – ToolbarItem об'єкти на панелі навігації Android та iOS (обидва Primary)

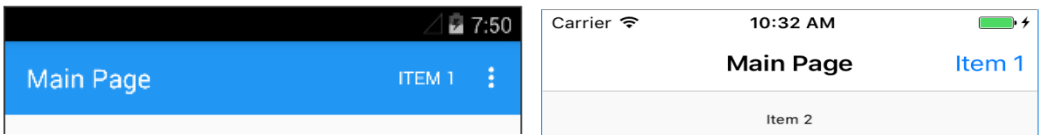


Рисунок 2 – ToolbarItem об'єкти на панелі навігації Android та iOS (один Primary, один Secondary)

Клас ToolbarItem визначає такі властивості:

– **Order** (ToolbarItemOrder) – це значення переліку, яке визначає, чи відображається ToolbarItem екземпляр в основному або додатковому меню;
– **Priority** integer – це значення, що визначає порядок відображення елементів у Page колекції об'єкта ToolbarItems.

Клас ToolbarItem успадковує такі властивості, що зазвичай використовуються від MenuItem класу:

– **Command** – це об'єкт ICommand, що дозволяє прив'язувати дії користувача, такі як торкання пальця або клацання, до команд, визначених у моделі подання;
– **CommandParameter** object – це значення, що вказує параметр, який повинен бути переданий у Command;
– **IconImageSource** ImageSource – значення, що визначає піктограму відображення об'єкта ToolbarItem;
– **Text** – це об'єкт string, що визначає ToolbarItem текст, що відображається.

Об'єкт ToolbarItem можна створити у XAML. Властивості Text можна IconImageSource встановити, щоб визначити, як кнопка відображається на панелі навігації. У цьому прикладі показано, ToolbarItem як створити екземпляр з деякими загальними наборами властивостей і додати його до колекції ContentPageToolbarItems:

```
<ContentPage.ToolbarItems>
    <ToolbarItem Text="Example Item"
        IconImageSource="example_icon.png"
        Order="Primary"
        Priority="0" />
</ContentPage.ToolbarItems>
```

У ToolbarItem буде показано об'єкт із текстом, значком та першим в основній області навігаційної панелі. Можна ToolbarItem також створити в коді та додати до колекції ToolbarItems:

```
ToolbarItem item = new ToolbarItem
{
    Text = "Example Item",
    IconImageSource = ImageSource.FromFile("example_icon.png"),
    Order = ToolbarItemOrder.Primary,
    Priority = 0
};

// "this" refers to a Page object
this.ToolbarItems.Add(item);
```

Клас ToolbarItem успадковує Clicked подію від MenuItem класу. Обробник подій можна підключити до події Clicked, щоб реагувати на торкання або клацати ToolbarItem екземпляри в XAML:

```
<ToolbarItem ...
             Clicked="OnItemClicked" />
```

Обробник подій також можна приєднати до коду:

```
ToolbarItem item = new ToolbarItem { ... }
item.Clicked += OnItemClicked;
```

У наступному коді показаний приклад реалізації OnItemClicked обробника події (рис. 3):

```
void OnItemClicked(object sender, EventArgs e)
{
    ToolbarItem item = (ToolbarItem)sender;
    messageLabel.Text = $"You clicked the \"{item.Text}\" toolbar
item.";
}
```

Якщо для властивості Order встановлено значення Primary, ToolbarItem об'єкт буде відображатися на головній навігаційній панелі на всіх платформах.

Якщо для властивості Order встановлене значення Secondary, то поведінка залежить від платформ. У UWP та Android Secondary меню елементів відображається як три точки, які можна торкнутися або клацнути, щоб відобразити елементи у вертикальному списку. У iOS Secondary меню елементів відображається під навігаційною панеллю у вигляді горизонтального списку.

Рисунок 3 – Secondary меню елементів для Android

Розглянемо проект ToolBarMenu, який демонструє роботу з ToolbarItem

```
public MainPage()
        {
            InitializeComponent();
            ToolbarItem tb = new ToolbarItem
            {
                Text = "Зателефонувати",
                Order = ToolbarItemOrder.Primary,
                Priority = 0,
                IconImageSource = "phone.png"
            };

            tb.Clicked += Tb_Clicked;
            ToolbarItem tb1 = new ToolbarItem
            {
                Text = "Створити новий контакт",
                Order = ToolbarItemOrder.Secondary,
                Priority = 1,
            };

            ToolbarItem tb2 = new ToolbarItem
            {
                Text = "Видалити",
                Order = ToolbarItemOrder.Secondary,
                Priority = 2,
            };

            ToolbarItem tb3 = new ToolbarItem
            {
                Text = "Про програму",
                Order = ToolbarItemOrder.Secondary,
                Priority = 3,
            };
            ToolbarItems.Add(tb);
            ToolbarItems.Add(tb1);
```

```
            ToolbarItems.Add(tb2);
            ToolbarItems.Add(tb3);
        }

        private async void Tb_Clicked(object sender, EventArgs e)
        {
           await DisplayAlert("Виклик", "Йде набір номера ...",
          "Скинути");
        }
}
```

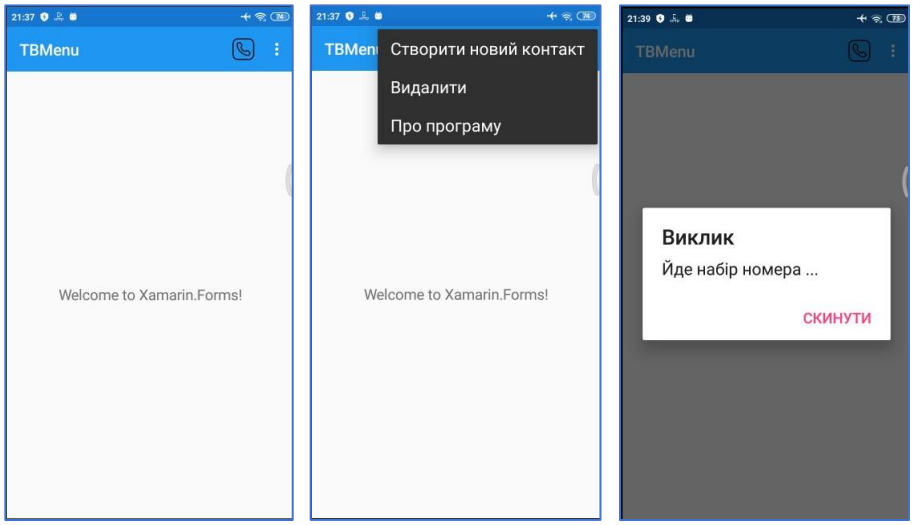


Рисунок 4 – Результати роботи додатку ToolBarMenu

3. Клас NavigationPage

Клас NavigationPage забезпечує ієрархічну навігацію, коли користувач може переходити по сторінках вперед та назад за своїм бажанням. Цей клас реалізує навігацію на основі стека об'єктів Page за методом LIFO (останнім надійшов – першим обслужений). Для переходу з однієї сторінки на іншу програма поміщає нову сторінку у стек навігації, де вона стає активною сторінкою, як показано на рис. 5.

Рисунок 5 – Додавання нової сторінки у стек навігації

Щоб повернутися до попередньої сторінки, програма витягне поточну сторінку зі стеку навігації, а нова найвища сторінка стане активною сторінкою, як показано на рис. 6.

Рисунок 6 – Робота стеку навігації при поверненні до попередньої сторінки

Методи навігації надаються властивістю Navigation будь-яких Page похідних типів. Ці методи надають можливість надсилати сторінки в стек навігації, витягувати сторінки зі стеку навігації та виконувати маніпуляції зі стеком.

Перша сторінка, додана в стек навігації, називається кореневою сторінкою програми, що демонструється в наступному прикладі коду:

```
public App ()
{
  MainPage = new NavigationPage (new Page1Xaml ());
}
```

У результаті в стекі навігації міститься екземпляр Page1Xaml ContentPage, де він стає активною сторінкою та кореневою сторінкою програми. Ці дії відображаються на рис. 7.

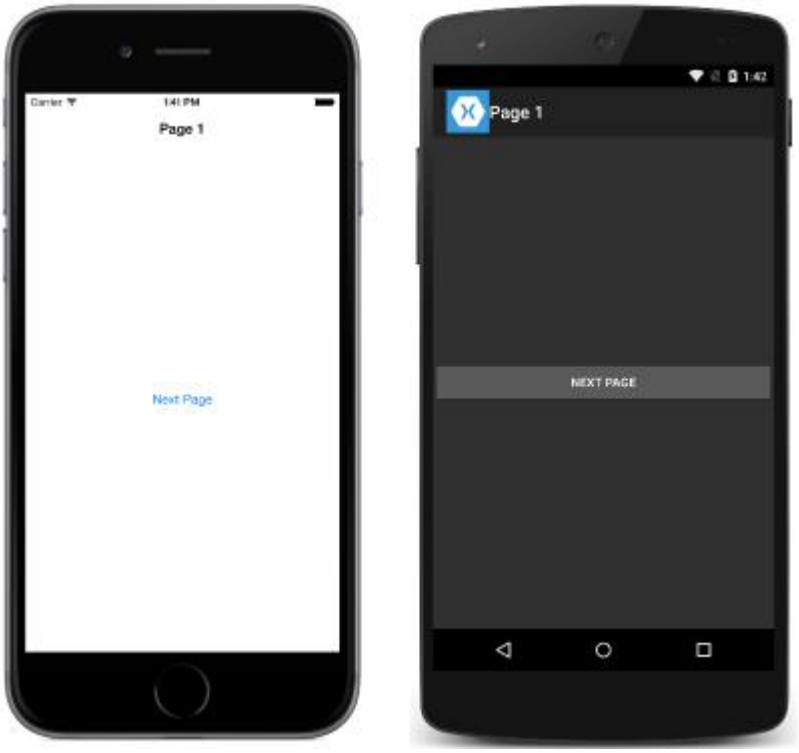


Рисунок 7 – Відображення кореневої сторінки Page1Xaml стеку навігації

Для додавання нової сторінки до стеку навігації використовується код:

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

Для вилучення нової сторінки зі стеку навігації використовується код:

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

Властивість Navigation надає методи InsertPageBefore та RemovePage керувати стеком шляхом вставки сторінок або їх видалення (рис. 8, рис. 9):

```
Navigation.InsertPageBefore (new MainPage (), this);
Navigation.RemovePage (new MainPage);
```

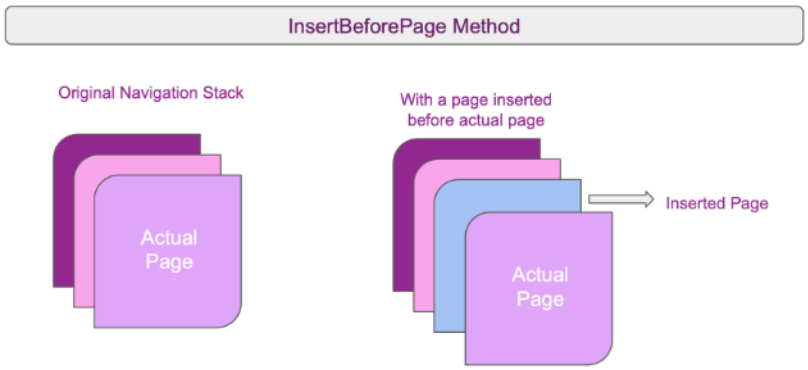


Рисунок 8 – Метод InsertPageBefore

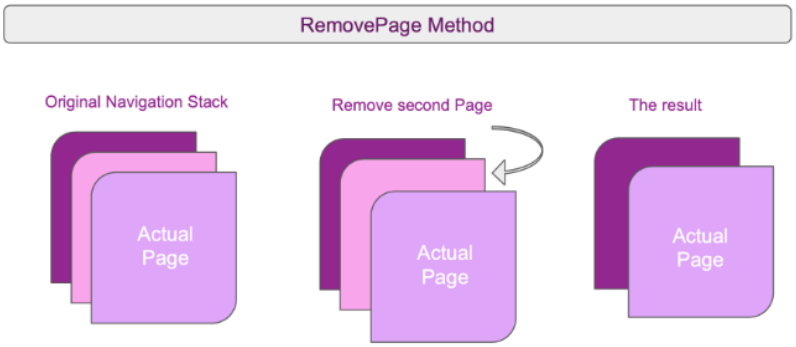


Рисунок 9 – Метод RemovePage

Метод PopToRootAsync видаляє всі сторінки, що містяться в навігаційному стеку, крім кореневої сторінки, роблячи її активною сторінкою.

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

Передати дані між сторінками стеку навігації можливо через constructor або BindingContext об’єкт.

Використовуючи конструктор, дані можемо отримати в два кроки:

Крок 1: Додайте потрібний параметр на сторінці конструктора.

```
public MainPage (string YourName)
{
    InitializeComponent ();
    Name = YourName;
}
```

Крок 2: Передача даних під час виклику сторінки, яка використовувалася раніше. (MainPage)

```
await Navigation.PushAsync(new YourPageName("MariaWhite"));
```

Розглянемо приклад використання об’єкта Binding Context. Створимо метод FillDataClicked та додамо всередину нього дані об’єкта класу Contact. Далі створений об’єкт передаємо на сторінку через об’єкт BindingContext.

```
async void FillDataClicked (object sender, EventArgs e)
{
    var contact = new Contact {
        Name     = "Merrie",
        LastName = "White",
        Country  = "USA"
  };

   var contactsPage = new ContactsPage ();
   contactsPage.BindingContext = contact;
   await Navigation.PushAsync (contactsPage);
}
```

Щоб видалити кнопку «Назад», необхідно виконати фрагмент коду у файлі XAML:

```
NavigationPage.HasBackButton="false".
```

Щоб приховати панель навігації, необхідно виконати фрагмент коду у файлі XAML:

```
NavigationPage.HasNavigationBar="false"
```